Introduction to Theoretical Computer Science
Coursework 2

Questions have varying numbers of marks. They are not necessarily of equal difficulty, but I expect the entire coursework to take around 10–15 hours.

You are **strongly** recommended to do these exercises as the material is taught. The exercises are designed to prompt the reading you should be doing after the lectures!

Submission boxes can be found on Learn. **Formats other than PDF will not be accepted, and will count as a non-submission.**

You are welcome to typeset your answers, but you do not have to. Manucripts should be scanned if possible, otherwise photographed carefully – you must submit work sized for A4 paper.

1 Partial functions

When we defined computable functions, we were talking about standard mathematical functions, which are by definition total.

A *partial function* $f : \mathbb{N} \rightharpoonup \mathbb{N}$ is a function $\mathbb{N} \to \mathbb{N} \cup \{\perp\}$, or equivalently a function that may be undefined on some values (we write $f(n) = \perp$, or sometimes $f(n)\uparrow$).

A partial function f is *computable* iff there is a register machine which, given n in R_0, halts with $f(n)$ in R_0 *whenever* $f(n) \neq \perp$.

(a) Let $\hat{H} : \mathbb{N} \times \mathbb{N} \rightharpoonup \mathbb{N}$ be the partial function given by

$$\hat{H}(m, n) = \begin{cases} 0 & \text{if } m \text{ is not the code of any RM program } P \\ 1 & \text{if } m = \ulcorner P \urcorner \text{ for some } P\text{, and } P \text{ halts on input } n \\ \perp & \text{otherwise} \end{cases}$$

Show that $\hat{H}$ is computable. Deduce that it is undecidable whether a computable partial function is total. *[4]*

(b) It is (easily) decidable whether a number n is the code $\ulcorner P \urcorner$ of a machine. Therefore we can computably list machines in some order $P_0, P_1, \ldots$, where $\ulcorner P_0 \urcorner$ is the first valid code of a program, and so on.

Consider the partial function $d : \mathbb{N} \rightharpoonup \mathbb{N}$ given by

$$d(n) = \begin{cases} P_n(n) + 1 & \text{if } P_n \text{ returns a result on input } n \\ \perp & \text{otherwise} \end{cases}$$

Is d computable? Justify your answer. *[2]*

(c) Now suppose that f is a *total* function which agrees with d (i.e. $f(n) = d(n)$) wherever d is defined. Show that f is not computable. *[4]*

2 P, NP

(a) (Revision of big-O notation.) Which of the following are true:
$n^2 = O(n \lg n)$; $n \lg n = O(n^2)$; $3^n = O(2^n)$; $3^n = 2^{O(n)}$. [*1*]

(b) Suppose that X, Y are both decision problems over the same domain, and both in P. Show that $X \cup Y$, $X \cap Y$ and $\neg X$ are also in P. [*2*]

(c) Suppose that L_1, L_2 are languages of strings over some finite alphabet, whose decision problems are in NP. Let $L = L_1 L_2 = \{ x_1 x_2 \mid x_1 \in L_1 \text{ and } x_2 \in L_2 \}$. Show that $L \in \text{NP}$. [*2*]

3 Reductions for NP-completeness

(a) EXACT-3SAT is a special case of 3SAT where every clause must have exactly three literals. Prove that EXACT-3SAT is NP-complete. [*3*]

(b) If $x_1, \ldots, x_n$ are variable over the integers, then a *3-product* is an expression $\pm(a_1 - x_i)(a_2 - x_j)(a_3 - x_k)$, where a_1, a_2, a_3 are 0 or 1.

3PRODEQNS is the problem where an instance is variables $x_1, \ldots, x_n$ and a finite number of 3-products over those variables, and the query is: is there an assignment of (integer) values to the variables such that *all* the 3-products evaluate to zero?

Give a polynomial reduction from EXACT-3SAT to 3PRODEQNS. (*Hint:* because the as are all in $\{0, 1\}$, it is enough to consider assignments where the xs are all in $\{0, 1\}$.) [*4*]

(c) The independent set problem INDSET is the following: given a graph G and an integer k, does G have a set I of k vertices such that no two vertices are joined by an edge?

Show that INDSET is NP-complete. (*Hint:* We know CLIQUE is NP-complete.) [*4*]

4 Untyped and simply typed lambda-calculus

(a) Evaluate the following expression as far as possible, showing your working. (Be careful with different variables having the same name; if you are in doubt about how to handle them, α-convert them to be different before you reduce.) After you have evaluated as far as possible in our standard call-by-name strategy, you may also wish to do some internal β-reductions to 'optimize' the result.

$$(\lambda m.\lambda n.\lambda f.\lambda x.mf(nfx))(\lambda f.\lambda x.fx)(\lambda f.\lambda x.f(fx))$$

Does this suggest anything to you? [*4*]

(b) Consider the following expression Y' given by

$$Y' \stackrel{\mathsf{def}}{=} \lambda F.(\lambda X.XX)(\lambda X.F(XX))$$

Show that $Y'G = G(YG)$ (where Y is the combinator from lectures).

Show that Y' reduces to Y by an internal β-reduction. *[3]*

(c) Give a formal derivation for the type of the simply typed expression

$$\lambda f{:}\mathsf{nat} \to \mathsf{nat}.\lambda x{:}\mathsf{nat}.f(fx)$$

[4]

(d) Consider the *original* untyped expression from part (a). Assign simple types to the variables so as to make the expression well typed. (*Hint:* start by giving the variables x the base type o.) *[3]*