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!

You may either submit on paper to the ITO, or electronically by submit itcs 2 answers.pdf

If you write your answers by hand, please submit either the original paper to the ITO, or a clean scan electronically – no mobile phone photos.

1 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.

(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 \( a \)s are all in \( \{0, 1\} \), it is enough to consider assignments where the \( x \)s are all in \( \{0, 1\} \).)

(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.)
2 Untyped and simply typed lambda-calculus

(a) Evaluate the following expression as far as possible, showing your working. (Remember to $\alpha$-convert variables if necessary)

\[
(\lambda m. \lambda n. \lambda f. \lambda x. m (f (n f x))) (\lambda f. \lambda x. f x) (\lambda f. \lambda x. f (f x))
\]

Does this suggest anything to you? [4]

(b) Consider the following expression $Y'$ given by

\[
Y' \overset{\text{def}}{=} \lambda F. (\lambda X. X X) (\lambda X. F (X X))
\]

Show that $Y' G = G (Y G)$ (where $Y$ is the combinator from lectures).
Show that $Y'$ reduces to $Y$ by an internal $\beta$-reduction. [3]

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

\[
\lambda f : \text{nat} \to \text{nat}. \lambda x : \text{nat}. f (f x)
\]

(d) Consider the 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 $o$.) [4]

3 Extensions and polymorphism

(a) Suppose that \texttt{zerop} : \text{nat} $\to$ \text{bool} is a function returning $true$ iff its argument is zero, and that \texttt{pred} : \text{nat} $\to$ \text{nat} is the predecessor function.

Using the \texttt{if} and \texttt{letrec} extensions, define the function \texttt{evenp} : \text{nat} $\to$ \text{bool} which returns $true$ iff its argument is even, when it is applied to 1; that is, fill in the dots in the expression

\[
\text{letrec evenp} = \ldots \text{ in evenp}(1)
\]

(b) Expand the definitions of \texttt{letrec} and \texttt{let} in your previous answer, and then carry out one step of evaluation of the resulting $\lambda$-expression. [4]

(c) In the Church encoding of booleans, the constant $true$ can be represented by $\lambda t. \lambda f. t$, and $false$ by $\lambda t. \lambda f. f$.

In the let-polymorphic typed $\lambda$-calculus, what is the type of these terms? [3]

(d) In the let-polymorphic calculus, sketch the inference of a type for the expression

\[
(\lambda f. \lambda x. f (f x))(\texttt{succ})(3)
\]

where \texttt{succ} : \text{nat} $\to$ \text{nat} is the successor function. It is sufficient to annotate each subterm with a type and say how the variables are unified – you need not write any formal derivations. [4]