Types

Give the syntax of (types and terms of) the second-order polymorphic lambda calculus \( \lambda_2 \) whose five ways of constructing terms, \( M \), are: identifiers, lambda abstraction, application, type abstraction and type application. (The last two are sometimes known as generalisation and specialisation.) Make it clear which, if any, sub-phrases of terms represent types or type variables. \[ 4 \text{ marks} \]

Give a term \( M \) conforming to the syntax of \( \lambda_2 \) which is not well-typed according to the usual inference rules for \( \lambda_2 \). \[ 2 \text{ marks} \]

Let \( \lambda U \) be the untyped lambda calculus whose terms \( N \) have syntax:

\[ N ::= x \mid \lambda x. N^1 \mid N^1 N^2. \]

Define a function \( \text{erase} : \lambda_2 \rightarrow \lambda U \) which removes all types from a \( \lambda_2 \) term, but which preserves the rest of it.

[Hint: \( \text{erase}(\Lambda \alpha. M) = \text{erase}(M). \)] \[ 3 \text{ marks} \]

Now find (or briefly justify why this is impossible):

(a) two well-typed \( \lambda_2 \) terms \( M_1 \) and \( M_2 \) without free type variables such that \( \text{erase}(M_1) = \text{erase}(M_2) = \lambda x.x \) and that \( M_1 \) and \( M_2 \) differ by more than type variable renaming;

(b) a well-typed \( \lambda_2 \) term \( M_3 \) such that \( \text{erase}(M_3) = \lambda x.xx \);

(c) a well-typed \( \lambda_2 \) term \( M_4 \) such that \( \text{erase}(M_4) = (\lambda x.xx)(\lambda x.xx) \);

(d) a well-typed \( \lambda_2 \) term \( M_5 \) such that \( N_5 = \text{erase}(M_5) \) has no ML type;

(e) a \( \lambda U \) term \( N_6 \) which has an ML type, but such that there is no well-typed \( \lambda_2 \) term \( M_6 \) with \( \text{erase}(M_6) = N_6. \)

[11 marks]