Concepts of Programming Language Design

Polymorphism Exercises

Gabriele Keller

November 6, 2024

1. Find the MGU of the following pairs of type terms, if it exists:
   (a)
   - $(\texttt{a} \to \texttt{a}) * \texttt{Int}$ and $\texttt{b} * \texttt{c}$
   - $(\texttt{a} \to \texttt{a}) * \texttt{Int}$ and $\texttt{b} \to \texttt{c}$
   - $(\texttt{a} \to \texttt{a}) * \texttt{a}$ and $\texttt{b} * \texttt{b}$
   - $\texttt{Rec t.}(\texttt{t} + \texttt{Int})$ and $\texttt{Rec t.}(\texttt{t} + \texttt{a})$
   - $\texttt{Rec t.}(\texttt{t} + \texttt{Int})$ and $\texttt{Rec t.}(\texttt{a} + \texttt{Int})$

2. **Parametric polymorphism** In the lecture, we introduced two sets of typing rules for an implicitly typed variant of MinHs. While the first set of rules specified when a certain polymorphic type can be inferred, it did not describe an algorithm to infer the principal type of an expression, in contrast to the second set, which is a formalisation of the Hindley-Milner type inference algorithm.
   (a) For the non-algorithmic typing rules, the rule for the `Pair` constructor is exactly the same as for explicitly typed MinHs. For each set of rules, show that

   $$\{\} \vdash \texttt{Apply}(\texttt{Recfun}\ (f.x.\texttt{Pair}\ x\ x))\ (\texttt{Num 5}) :: (\texttt{Int} * \texttt{Int})$$

   is derivable.
   (b) Consider this (explicitly typed) program (we're using Haskell notation here):

   ```
   g :: forall b. ((forall a. (a -> a)) -> b -> Int)
   g f x = if f True
             then (f 1)
             else 2
   ```

   The type of `g` is not a legal type in polymorphic MinHs as discussed in the lecture. Which restriction did we put on MinHs which excludes this type?
   (c) In a language which permits this type, what can you tell about the behaviour of the function `g`?
   (d) The higher-order abstract syntax term below represents this program:

   ```
   Recfun (g.f. (Recfun (h.x. (If (Apply f (Const True)) (Num 1) (Num 2)))))
   ```

   Is the type $\forall b.((\forall a.(a \to a)) \to b \to Int)$ for this expression inferable with the non-algorithmic or the algorithmic typing rules? Explain why.
   (e) What is the type of the expression as derived by Hindley-Milner?

3. **Coercions and Subtyping**

   You are given the type Rectangle, parameterised by its height and width, and the type Square parameterised by the length of one of its sides. Neither type is mutable.
   (a) Which type is the subtype, which type is the supertype?
   (b) Give a subtype/supertype ordering of the following set of function types:

   ```
   Rectangle -> Rectangle
   Rectangle -> Square
   Square -> Rectangle
   Square -> Square.
   ```

   (c) Define a data type Square and a data type Rectangle in Haskell. Then define a coercion function from elements of the subclass to elements of the superclass.
   (d) Show that the ordering you have given in the previous question is correct by defining coercion functions for each pair of types in a subtyping relationship in part (b).

4. **Constructor variance**

   List some examples of a covariant, contravariant and invariant type constructor.