06-02552 *Principles of Programming Languages*
Spring Semester 2009-10

The University of Birmingham
School of Computer Science
© Uday Reddy2009-10

Solutions to Exercise Sheet 2

Exercise 1: Free variables and substitution

i. *List the free variables of the following lambda terms.*

   i. $FV(\lambda x.\, \lambda y.\, x\ z) = \{z\}$
   ii. $FV(\lambda r.\, f\ r\ 1) = \{f, 1\}$
   iii. $FV(\lambda y.\, *\ y\ 2) = \{*, 2\}$
   iv. $FV(\lambda y.\, \lambda z.\, g\ (x\ z)\ (y\ z)) = \{g, x\}$.
   v. $FV(\lambda x.\, \lambda x.\, x) = \{\}$

ii. *Write down the result of the following substitution operations.*

   i. $(\lambda y.\, \lambda z.\, g\ (x\ z)\ (y\ z))\,[(\lambda r.\, f\ r\ 1)/g] = \lambda y.\lambda z.(\lambda r.fr1)(xz)(yz)$
   ii. $(\lambda x.\, \lambda y.\, x\ z)\,[(\lambda y.\, 2)/z] = \lambda x.\lambda y.x(\lambda y.2)$
   iii. $(\lambda x.\, \lambda x.\, x)\,[(\lambda y.\, *\ y\ 2)/x] = \lambda x.\lambda x.x$
   iv. $(\lambda x.\, \lambda y.\, x\ z)\,[(y\ 2)/z] = \lambda x.\lambda y.x(y\ 2)$

iii. *For each of the above substitutions, mention if it is valid or not.*

   Only (iv) is invalid; the rest are valid.

iv. *Explain in a few sentences why invalid substitutions can be problematic.*

   An invalid substitution $M[N/x]$ brings in free variables of N into a context where they become bound variables. This violates the idea that the bound variables are purely "local"; they do not affect the variables used elsewhere.

Exercise 2: Types for lambda terms

Write down possible types for the following lambda terms.

i. $\lambda x.\, \lambda y.\, y\ x$. The overall type should be of the form $\ldots \to \ldots \to \ldots$ where the first ... is the type of x, the second ... is the type y and the final ... is the type of the result of the function. To get the actual type, we should find possible types for x and y that allow us to type check the function body type, and then calculate the type of the result.

   We can assume that the argument x is of some unknown type A, because there are no constraints on the type of x. The argument y should then be of some type $A \to \ldots$ so that the function application $y\ x$ can be type checked. Suppose y is of type $A \to B$. Then $y\ x$ type checks and has type B. So, $\lambda y.\, y\ x$ is of type $(A \to B) \to B$ and $\lambda x.\, \lambda y.\, y\ x$ is of type $A \to (A \to B) \to B$.

ii. $\lambda y.\, *\ y\ 2$. For the application $*\ y\ 2$ to type check, y must be of type **int**. So, this term is of type **int** $\to$ **int**.

iii. $\lambda f.\, \lambda y.\, f\ y\ y$. If we assume that y is of type A, then the application $f\ y\ y$ requires that f should be of type $A \to A \to \ldots$. Suppose f is of type $A \to A \to B$. Then the whole term is of type $(A \to A \to B) \to A \to B$.

iv. $\lambda g.\, \lambda x.\, \lambda y.\, \lambda z.\, g\ (x\ z)\ (y\ z)$. Suppose z is of type A. Then x must have a type of the form $A \to B$ (or $x\ z$ won't type check). Similarly, y must have a type of the form $A \to C$ (or $y\ z$ won't type check). There is no need for x and y to have the same return type. So, we used different letter B and C for the two return types. The function g must then take two successive arguments of type B and C and return results of some type D. So, it must have a type of the form $B \to C \to D$. So, the overall type of the term is

$$(B \to C \to D) \to (A \to B) \to (A \to C) \to A \to D$$