There is no problem of variables

Randall Holmes

May 5, 2015

Against comments by unnamed persons, we illustrate the fact that standard notation with bound variables, which might be open to criticism because it is not clear that its structure parallels its meaning in a technical sense, can readily be converted to a different syntax, precisely parallel in structure with additional decorations, in which it is clear for example that a bound variable can be thought of as referring to a projection operator (with possible additional manipulations involving the K combinator to handle levels of binding).

We do this by first providing an intermediate syntax in which expressions containing free variables are seen to represent functions, then transforming the intermediate syntax suitably.

An expression E all of whose variables are taken from a list $(x_1, \ldots, x_n)$ is to be converted to an expression $[E[X_1^n/x_1, \ldots, X_n^n/x_n]]$ – enclose the expression in brackets then replace the ith variable in the list of n variables with the formal constant X_i^n. The referent of this expression is the function which we might more usually denote by $(x_1, \ldots, x_n \mapsto E)$, the function which takes any list of arguments $(a_1, \ldots, a_n)$ to $E[a_1/x_1, \ldots, a_n/x_m]$, the result of replacing each x_i with the value a_i. Please note that this account depends in no way on variables actually being indexed: if the variables in question are a, b, c, u, v, w the argument list might be (a, b, c, u, v, w) and we would transform c for example to X_6^3. One does not carry out the bracketing construction on an expression which contains a free variable enclosed in brackets!

We then indicate how to eliminate the brackets. We suppose for the sake of simplicity that all the underlying operators of our language are unary or binary. Each unary operator f gets variants f_n with each natural number superscript n, and similarly for binary operators. f_0 is synonymous with f; the meaning of the higher indexed f_n's will become clear.

Transform any expression $[f_n t]$ (f a unary operator, t a term) to $f_{n+1}[t]$. Transform any expression $[u o_n v]$ where u and v are terms and o is a binary operator to $[u]o_{n+1}[v]$. In this way, all brackets will eventually enclose atomic terms. $[X^i_n]$ transforms to π^i_n. Any term $[t]$ where t does not contain any X^i_n can be transformed to Kt (and $[K^n t]$ to $K^{n+1}t$, K^1 being synonymous with K).

Variable binding operators $(Bx_1, \ldots, x_n.E)$ translate to applications of B to E expressed as above as a function of the x_i's.

$(\forall a.a = b)$ is $\forall[X^1_1 = b]$, which is $\forall([X^1_1] =_1 [b])$ or $\forall(\pi^1_1 =_1 (Kb))$, which is then $[\forall(\pi^1_1 =_1 (KX^1_1))]$, which is then $\forall^1((K\pi^1_1) =_2 (K_1\pi^1_1))$. This approach doesnt preserve the letter names, but this can be restored by other changes in format. We could get the form $\forall^2_1((K\pi^2_1) =_2 (K_1\pi^2_2))$ by using the same argument list (a, b) in both contexts, and also using a version $\forall^2$ of the quantifier which quantifies on the first variable of an argument list of two variables.

The point here is that once the debracketing is carried out, every subterm of the resulting notation actually has reference to a specific object. Subterms of the original notation containing free occurrences of bound variables refer to suitable functions. Bound variables in the original notation end up referring to projection operators with some K^m_n (always read as $(K^m)_n$ applied to them. The additional notational cost, which is admittedly annoying and not desirable for practical use, is effectively linear (technically it may be $n\log(n)$). The point is not that we should use this notation. The point is that there is no problem with the usual notation: it is a fairly straightforward transformation of a pure mathematical notation with no variable binding features at all.