Notes on W-types

Heman Gandhi April 2020

Lists

From 5.1, we says lists are:

- $nil : List(A)$
- $cons : A \to List(A) \to List(A)$

With W-types, this is simplified to $List(A) :\equiv W_{x:1+A}\ \mathrm{rec}_{1+A}(\mathcal{U}, 0, \lambda a.1, x)$. This means that lists are either a null-ary empty thing or something that takes one (1) argument. This seems at odds with "cons" above, but is reconciled by "sup": $cons(a, l) = sup(inr(a), \lambda \star .l)$.

If we have a list $< a, b, c >: List(A)$, we can write:

$$sup(inr(a), \lambda \star .sup(inr(b), \lambda \star .sup(inr(c), \lambda \star .sup(inl(\star), \lambda x.empty))))$$

which is suitably ugly and where *empty* is the 0-induction to get us the nil at the end of the list.

Binary Trees

With bullet-points, we'd probably write:

- $nil : BinTree(A)$
- $node : A \to BinTree(A) \to BinTree(A) \to BinTree(A)$

With W-types, we'd get something very similar to lists:

$$BinTree(A) :\equiv W_{a:1+A}\ \mathrm{rec}_{1+A}(\mathcal{U}, 0, 2, a)$$

Where the sort of "cons"-ing operator would combine two trees under a root with:

$$\begin{gathered} cons'(a, l, r) = sup(inr(a), C) \\ where\ C\ 0_2 = l;\ C\ 1_2 = r \end{gathered} \tag{1}$$

(Writing out an actual tree is hecking ugly.)

Proofs over W-types

(I don't think this bit actually matters – will just grab from the book.)

All Proofs of the Same Thing Over a W-type are the Same

"Proofs of the same thing" means that $g, h : \prod_{w:W_{x:A}B(X)} E(w)$ and where $t = g, h$ have $G_t : \prod_{a,f} t(sup(a, f)) = e(a, f, \lambda b.t(f(b)))$.

In order to prove this, we consider our own dependent type over our W-type. We put $D : \prod_{w:W_{x:A}B(x)} g(w) = h(w)$ which would prove $g = h$ by function extensionality. In order to prove this, by induction on W-types, we need only inhabit

$$d : \prod_{a:A} \prod_{f:B(a)\to W_{x:A}B(x)} \prod_{g':\prod_{b:B(a)} D(f(b))} D(sup(a, f))$$

To do so, consider e above. g' gives us that $g(f(b)) = h(f(b))$ and then function extensionality makes $p : \lambda b.g(f(b)) = \lambda b.h(f(b))$. $ap_{e(a,f)}(p)$ gives us that $e(a, f, \lambda b.g(f(b))) = e(a, f, \lambda b.h(f(b)))$ and by the given G_t, we get $g(sup(a, f)) = h(sup(a, f))$, which gives us d as desired.