STRATEGIES FOR BINDING

Abstract. We discuss various strategies for practically managing binding your compiler, including explicit variable names, De Bruijn indicies, and locally nameless form. We also offer various formalizations of substitution, a critical issue in implementing binding.

Question from last time: in normalize and compare, we reduce expression under λ abstractions. This offers a critical difference between normalization of F^ω's kind structure and the dynamics of the simply typed lambda calculus.

We have two main strategies: explicit variables and De Bruijn indices. Explicit variables use variable names either as given by the programmer or as by generated by the compiler. Experience has suggested that, there are pro's and con's to each strategy. Hybrid strategies may also be used, placing different strategies for types and terms, or different strategies for free and bound variables. The latter is commonly called *locally nameless form*.

1. Explicit Variables

Substitution can be hazardous. Consider what happens when we move underneath a binder: one may be tempted to write

$$[M/x](\lambda y.N) = (\lambda y.[M/x]N)$$

but if y is not free in N, this is incorrect. For instance, $[y/x](\lambda y.x) \neq (\lambda y.y)$.

Consequently, substitution using explicit variables may be expressed as

$$[M/x](\lambda y.M) = \begin{cases} \lambda y.[M/x]N & \text{if } y \notin FV(M) \\ \lambda y'.[M/x][y'/y]N & \text{otherwise, } (y' \notin FV(M)) \end{cases}$$

To simplify the implementation (and avoid repeated expensive checks or extra memory usage), one may simply always perform the second case, generating a fresh variable name regardless. However, this can negatively influence other parts of the compiler, including alpha-equivalent memoization. Addiionally, this strategy can be quite difficult to debug, as errors arise in non-obvious places.

2. De Bruijn Indices

Using De Bruijn Indices, variable names are replaced by integers. Variable i refers to the variable bound i binding sites above. For instance, the term $\lambda x.\lambda y.x$ may be written $\lambda.\lambda.1$, assuming 0-indexing.

Conveniently, α equivalence is now structural identity. De Bruijn also suggested a concept called "levels," which counts from the top down, rather than the bottom up. However, this strategy ruins various nice algebraic properties of substitution.

We define substitution inductively. For cases where there is no binding occuring at this level, we may simply move into the subterms. For instance, application:

$$[M/i]NP = [M/i]N[M/i]P$$

The case for variables is as follows

$$[M/i]j = \begin{cases} M & j = i \\ j-1 & i < j \\ j & i > j \end{cases}$$

for binding sites, we'll need an auxilary judgement. We write

$$[M/i]\lambda.N = \lambda[\uparrow_{\geq 0} M/(i+1)]N$$

where

$$\uparrow_{\geq i} j = \begin{cases} j+1 & j \geq i \text{ (free)} \\ j & j < i \text{ (bound)} \end{cases} \qquad \uparrow_{\geq_i} MN = \uparrow_{\geq i} M \uparrow_{\geq i} N$$

$$\uparrow_{\geq_i} \lambda.N = \lambda.\uparrow_{\geq_i+1} N$$

with this, we've presented a formal definition of substitution for De Bruijn Indices. However, this is not how we tend to think of substitutions.

3. Explicit Substitutions

As usual, this tooling is not necassary at this point, but will soon pay dividends. Most of this will be revisited once we get to phase-splitting.

We denote explicit substitutions by σ, defined as

$$\sigma \cdots = M.\sigma | \uparrow^i$$

$[M/0, N/1]$ becomes $[M.N.\mathtt{id}]$, where $\mathtt{id}$ is syntactic sugar for $\uparrow^0$. As an example,

$$0[M.N.\mathtt{id}] = M$$
$$1[M.N.\mathtt{id}] = N$$
$$2[M.N.\mathtt{id}] = 0$$

Note that we apply explicit substitutions on the right. Our full definition is as follows (recall that we are denoting variables by integers):

$$\begin{aligned} 0[M.\sigma] &= M \\ (i+1)[M.\sigma] &= i[\sigma] \\ n[\uparrow^i] &= n+i \\ (MN)[\sigma] &= M[\sigma]N[\sigma] \\ (\lambda A.M)[\sigma] &= \lambda A[\sigma].M[0.(\sigma \circ \uparrow^1)] \end{aligned}$$

we can thinking of the λ case as saying "leave 0 (the newly bound variable) alone, and reach up one binding site more for every variable."

Rewriting our $\uparrow_i$ notation from the previous section in this form, we have

$$\begin{aligned} \uparrow_{\geq 0} M &= M[\uparrow^1] \\ \uparrow_{\geq 0}\uparrow_{\geq 0} M &= M[\uparrow^2] \\ \uparrow_{\geq 1} M &= M[0.\uparrow^1] \end{aligned}$$

We haven't actually defined what it means to compose two explicit substitutions yet $(\sigma \circ \sigma')$, so let's do that now. It should be the case that $M[\sigma \circ \sigma'] = M[\sigma][\sigma']$.

$$(M \circ \sigma) \circ \sigma' = M[\sigma].(\sigma \circ \sigma')$$
$$\uparrow^0 \circ \sigma' = \sigma$$
$$\uparrow^0 \circ \uparrow'_j = \uparrow^{i+j}$$
$$\uparrow^{i+1} \circ (M.\sigma) = \uparrow^i \circ \sigma$$

This gives us *nouns* for substitutions, allowing us to reason more formally about them. In type theory, apparently all sorts of crazy stuff can happen in *the calculus of explicit substitutions*, including a sort of "lazy substitution," though that's beyond the scope of this class.

4. Typing Judgements for Explicit Substitutions

Types are nice. We want to make sure that explicit substitutions do not ruin the types of our terms, so let's convince ourselves of that now. We should first formally define our context as

$$\Gamma \cdots = \epsilon \mid \Gamma, A$$

where A is some "type-ish" thing.

We would like it to be the case that if $\Gamma \vdash \sigma : \Gamma'$ and $\Gamma' \vdash M : B$, then $\Gamma \vdash M[\sigma] : B[\sigma]$. Note that $B[\sigma]$ represents dependent types secretly sneaking into lecture. Our first typing judgement is

$$\frac{}{\Gamma, A_1, \ldots A_i \vdash \uparrow^i : \Gamma}$$

which says that something typechecks in Γ, it should typecheck in $\Gamma, A_1, \ldots, A_i$ after $\uparrow^i$ is applied to it. We also have

$$\frac{\Gamma \vdash \sigma : \Gamma' \quad \Gamma \vdash M : A[\sigma]}{\Gamma \vdash M.\sigma : \Gamma', A}$$

To see why the $A[\sigma]$ is required, consider the following example. If we did not apply $[\texttt{string.id}]$ to 0 in the premise of the final inference, 0 would be garbage.

$$\dfrac{\dfrac{\Gamma \vdash \texttt{id} : \Gamma \qquad \Gamma \vdash \texttt{string} : \texttt{type}[\texttt{id}]}{\Gamma \vdash [\texttt{string.id}] : \Gamma, \texttt{type}} \qquad \Gamma \vdash \texttt{"Hello World"} : 0[\texttt{string.id}]}{\Gamma \vdash [\texttt{"Hello World".string.id}] : \Gamma, \texttt{type}, 0}$$