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 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.