Fully Normalizing Lambda Calculus Machines

Werner E. Kluge
E–mail: wk@informatik.uni-kiel.de

April 25, 2005

Abstract

Almost all implementations of functional languages known today realize a weakly (head) normalizing λ-calculus which rules out substitutions and reductions under abstractions, restricting what can be computed with these languages basically to ground terms, which is what can be accomplished with imperative languages as well. One could be content with this situation, arguing that for the majority of real-life applications this is all there is needed.

However, one could also argue that too much is given up too easily. Supporting a fully normalizing λ-calculus that employs full-fledged β-reductions as the basic substitution mechanism is the key to symbolic computations involving free variables, i.e., to treat both variables and functions (abstractions) truly as first class objects. Full normalization is for instance required in proof assistant systems where terms must be compared for β-equivalence, but it may also be applied to advantage for high-level program optimizations such as converting partial applications into new, specialized abstractions and normalizing abstraction bodies, or to unroll recursive abstractions in order to eliminate repeated parameter passing steps.

Time permitting, we will outline two machine concepts that use fairly conventional machinery to reduce λ-terms to weak head normal forms and then employ special mechanisms to push delayed substitutions (or environments) over abstractors, thereby resolving in an orderly form potential naming conflicts, in order to be able to continue with weak normalizations inside abstraction bodies. One approach is based on the so-called $\lambda\sigma$-calculus that separates substitutions from the λ-terms proper, the other treats substitutions conceptually as integral parts of λ-terms.