Normalization by evaluation

Samuel Mimram
samuel.mimram@lix.polytechnique.fr
http://lambdacat.mimram.fr

November 22, 2021

Implementing an evaluator for λ-calculus (or, more generally, for a functional programming language) is painful because one has to explicitly handle α-conversion. Techniques such as de Bruijn indices exist but they are quite error prone. We present here a technique called *normalization-by-evaluation* which allows easy implementation of normalization of λ-terms when the host language is itself functional and test for β-equivalence.

1. A term is *normal* when it cannot reduce. Give a grammar describing all terms in normal form.

2. A term is *neutral* when it is normal, and remains normal when applied to a normal form. Intuitively, this corresponds to a computation which is either finished or "stuck". Describe those by a grammar and use it to simplify the previous characterization of normal forms.

3. Define a function $[\![-]\!]_\rho$ which computes the normal form a term (we suppose that it is strongly normalizing) in an environment ρ which associates a normal form to free variables.

4. In OCaml define types corresponding to λ-terms, normal terms and neutral terms. If necessary, modify your implementation so that abstractions in neutral terms are implemented by OCaml abstractions. Finally, define a function `eval` which associates a normal term to every λ-term.

5. Suppose given a function `fresh` which generates fresh variable names. Implement a function `readback` which translates a normal form back to a λ-term.

6. Use this to implement a normalization function from λ-terms to λ-terms. Can we use it to easily test for β-conversion?

7. Transform your implementation in order to canonically generate variable names, so that the result is deterministic.