Minimality in a Linear Calculus with Iteration

Sandra Alves[1], Mário Florido[1], Ian Mackie[2], and François-Régis Sinot[1]

[1] University of Porto, Department of Computer Science & LIACC,
R. do Campo Alegre 823, 4150-180, Porto, Portugal
[2] LIX, École Polytechnique, 91128 Palaiseau Cedex, France

Gödel's System $\mathcal{T}$ is an extremely powerful calculus: essentially anything that we want to compute can be expressed [3]. A *linear* variant of this well-known calculus, called System $\mathcal{L}$, was introduced in [1], and shown to be every bit as expressive as System $\mathcal{T}$. The novelty of System $\mathcal{L}$ is that it is based on the linear λ-calculus, and all duplication and erasing can be done through an encoding using the iterator.

There are many well-known, and well-understood, strategies for reduction in the (pure) λ-calculus. When investigating deeper into the structure of terms, we get a deeper understanding of reduction. For instance, calculi with explicit resource management or explicit substitution allow a finer control over reduction. In a similar way, System $\mathcal{L}$ splits the usual λ in two different constructs: a binder, able to generate a substitution, and an iterator able to erase or copy its argument. This entails a finer control of these fundamentally different issues, which are intertwined in the λ-calculus. Having a calculus which offers at the same time a lot of freedom in reduction and a lot of information about resources makes it an ideal framework to start a fresh attempt at studying reduction strategies in λ-calculi.

We present a first step towards a thorough study of reduction strategies for System $\mathcal{L}$. In particular:

1. we present, and compare, different ways of writing the reduction rules associated to iterators;
2. we define a weak reduction relation for System $\mathcal{L}$ (we call this new system *weak System $\mathcal{L}$*) similar to weak reduction used in the implementation of functional programming languages, where reduction is forbidden inside abstractions;
3. we present reduction strategies for the weak reduction relation: call-by-name, call-by-value, and call-by-need (emphasising this last one), proving that they are indeed strategies in a technical sense. Since neededness is usually undecidable, extra features (like sharing graphs, environments, explicit substitutions) are generally added to actually implement call-by-need. In contrast, for System $\mathcal{L}$, we can define call-by-need *within the calculus* in an *effective* way.
4. we give a proof of minimality of the call-by-need strategy. It is well-known that there exists no computable minimal strategy for the λ-calculus [2]. One of our main contributions is a (family of) computable minimal strategies for weak System $\mathcal{L}$.

References

1. Sandra Alves, Maribel Fernández, Mário Florido, and Ian Mackie. The power of linear functions. In Z. Ésik, editor, *Proceedings of the 15th EACSL Conference on Computer Science Logic (CSL'06)*, volume 4207 of *Lecture Notes in Computer Science*, pages 119–134. Springer-Verlag, 2006.
2. Henk P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland Publishing Company, second, revised edition, 1984.
3. Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and Types*, volume 7 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1989.