Confluence of an Explicit Substitutions Calculus Formalized

Flávio L. C. de Moura[*], Leandro O. Rezende[†]
Departamento de Ciência da Computação
Universidade de Brasília, Brasília, Brazil

Abstract

Rewriting theory is a well established model of computation equivalent to the Turing machines, and the best-known rewriting system is the λ-calculus, the theoretical foundation of the functional paradigm of programming. Confluence is an important property related to the determinism of the results given by a rewriting system. In this work, which is still in progress, we formalize the confluence property of an extension of the λ-calculus with explicit substitutions following the steps in [4, 5]. Confluence is obtained through the Z property [3], and the first challenge of this work was to prove that an abstract rewriting system, i.e. a binary relation over an arbitrary set, that satisfies the Z property is confluent. The difficulty relies on the precise structure of the nested induction that needs to be done on the reflexive transitive closure generated by the divergence in the definition of confluence. The formalization is done in the Coq proof assistant [8], a system based on a constructive higher-order logic with a well developed extraction mechanism [6].

In the λ-calculus, terms that only differ by the name of its bound variables are considered equal. This notion is known as α-equivalence, which is a costly computational equivalence relation. Alternatives to α-equivalence include the so called De Bruijn indexes [2], where variables are represented by natural numbers. In De Bruijn notation terms have a unique representation, and hence there is no need of α-equivalence. Nevertheless, defining a reduction in De Bruijn notation requires a non-trivial algebra for referencing and updating indexes. The Locally Nameless Representation (LNR) [1] is a framework that takes the advantages of the two notations: bound variables are represented as De Bruijn indexes, while free variables uses names. The original framework uses classical logic and was built for representing pure λ-terms, therefore we decided to take some of its constructions (which are not based on classical logic) and extend it with a new operator for the substitution operation in such a way that our framework is constructive. This is important because one of the goals of this work is the generation of certified code via the extraction mechanism of Coq.

Our formalization is based on the paper [4], where the λex-calculus is defined. Another challenging step of this formalization is that the λex-calculus defines an equational theory based on the fact that reduction is done modulo permutation of independent substitutions. In order to avoid an explicit manipulation of permutation of independent substitutions, we use the generalized rewriting facilities of Coq [7]. Nevertheless, the generated equivalence relation needs to be defined over every expression defined by the LNR grammar, and not only over λ-terms with explicit substitutions. In order to circumvent this problem, we proved that the reduction relation defined by the calculus in LNR modulo permutations of independent substitutions is restricted to λ-terms with explicit substitutions.

The formalization is available at `https://github.com/flaviodemoura/Zproperty.git` and is divided in two files:

1. The file `ZtoConfl.v` contains the proof that an abstract rewriting system R that satisfies the Z property is confluent;
2. The file `lex.v` contains the current status of the formalization showing that the calculus in LNR satisfies the Z-property, and hence is confluent.

[*]flaviomoura@unb.br
[†]L-ordo.ab.chao@hotmail.com

References

[1] A. Charguéraud. The Locally Nameless Representation. *Journal of Automated Reasoning*, pages 1–46, 2011.

[2] N.G. de Bruijn. Lambda-Calculus Notation with Nameless Dummies, a Tool for Automatic Formula Manipulation, with Application to the Church-Rosser Theorem. *Indag. Mat.*, 34(5):381–392, 1972.

[3] B. Felgenhauer, J. Nagele, V. van Oostrom, and C. Sternagel. The Z property. *Archive of Formal Proofs*, 2016, 2016.

[4] D. Kesner. Perpetuality for full and safe composition (in a constructive setting). In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II - Track B: Logic, Semantics, and Theory of Programming & Track C: Security and Cryptography Foundations*, volume 5126 of *Lecture Notes in Computer Science*, pages 311–322. Springer, 2008.

[5] D. Kesner. A Theory of Explicit Substitutions with Safe and Full Composition. *Logical Methods in Computer Science*, 5(3:1):1–29, 2009.

[6] P. Letouzey. Coq Extraction, an Overview. In C. Dimitracopoulos A. Beckmann and B. Löwe, editors, *Logic and Theory of Algorithms, Fourth Conference on Computability in Europe, CiE 2008*, volume 5028 of *Lecture Notes in Computer Science*. Springer-Verlag, 2008.

[7] M. Sozeau. A new look at generalized rewriting in type theory. *J. Formalized Reasoning*, 2(1):41–62, 2009.

[8] The Coq Development Team. The coq proof assistant, version 8.7.2, February 2018.