Some applications of context-sensitive rewriting*

Salvador Lucas

DSIC, Universitat Politècnica de València, Spain

In programming languages design, a clear rule to select the arguments of functions that are evaluated in function calls is essential to *understand* the program behavior. From the implementation point of view, such a rule is also important to improve the evaluation of function calls in a number of different ways: efficiency, speed, termination behavior, etc. For instance, in the evaluation of conditional expressions if b then e else e' involving the if-then-else operator, evaluating the boolean guard first is essential. Furthermore, if the boolean guard b does not yield either *true* or *false*, most languages discard the full expression as *meaningless*; thus, evaluating the second and third arguments e and e' without a successful evaluation of the first argument b is wasteful. Other operators like *sequencing* (;) or *choice* (+) that are used in concurrent and/or imperative languages require a similar treatment. The (lazy) list constructor cons of functional languages is another well-known example.

At the *syntactic* level we can specify this by just associating a set $\mu(f)$ of indices of *evaluable arguments* to each function symbol f by means of a mapping μ which we call a *replacement map*. For instance, we let $\mu(\text{if-then-else}) = \{1\}$ to specify that only the boolean argument b of a conditional expression if b then e else e' is necessarily evaluated. We can write $\mu(;) = \{1\}$ to avoid computations on S_2 in a sequence $S_1;S_2$, and $\mu(+) = \varnothing$ to say that processes should not be executed as part of a choice expression.

In the realm of term rewriting, *context-sensitive rewriting* [6, 7] is the restriction of rewriting that arises when these *syntactic replacement restrictions* are taken into account. It has been used to improve the termination behavior of reduction-based computation systems and programs. It has been shown useful as an operational notion to model or simulate the executions of various formalisms and calculi. Some computational properties of context-sensitive rewriting (remarkably termination) have been used to characterize or verify computational properties of important rewriting strategies like innermost, outermost, demand-driven, and lazy rewriting. Context-sensitive rewriting has also been shown useful to develop verification techniques and tools for variants of rewriting like order-sorted or conditional rewriting. Consequently, it is also useful for analyzing computational properties of programs written in sophisticated rewriting-based programming languages such as CafeOBJ [3], Clean [9], Haskell [5], Lisp [8], Maude [1], OBJ2 [2], OBJ3 [4], etc., where related language constructions are used. This tutorial provides an overview of the theory of context-sensitive rewriting and some of its applications.

Keywords: context-sensitive rewriting, infinitary normalization, normalization, replacement restrictions, rewriting semantics, termination.

References

[1] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. All About Maude – A High-Performance Logical Framework. Lecture Notes in Computer Science 4350, 2007.

*Partially supported by the EU (FEDER), Spanish MINECO project TIN2015-69175-C4-1-R, and GV project PROMETEOII/2015/013.

Submitted to:
PROLE 2017

[2] K. Futatsugi, J. Goguen, J.-P. Jouannaud, and J. Meseguer. Principles of OBJ2. In *Proc. of Conference Record of the 12th Annual ACM Symposium on Principles of Programming Languages, POPL'85*, pages 52-66, ACM Press, 1985.

[3] K. Futatsugi and A. Nakagawa. An Overview of CAFE Specification Environment – An algebraic approach for creating, verifying, and maintaining formal specification over networks –. In *Proc. of the 1st International Conference on Formal Engineering Methods, ICFEM'97*, 1997.

[4] J.A. Goguen, T. Winkler, J. Meseguer, K. Futatsugi, and J.-P. Jouannaud. Introducing OBJ. In J. Goguen and G. Malcolm, editors, *Software Engineering with OBJ: algebraic specification in action*, Kluwer, 2000.

[5] P. Hudak, S.J. Peyton-Jones, and P. Wadler. Report on the Functional Programming Language Haskell: a non–strict, purely functional language. *Sigplan Notices*, 27(5):1-164, 1992.

[6] S. Lucas. Context-sensitive computations in functional and functional logic programs. *Journal of Functional and Logic Programming* 1998(1):1-61, January 1998.

[7] S. Lucas. Context-Sensitive Rewriting Strategies. *Information and Computation* 178(1):293–343, 2002.

[8] J. McCarthy. Recursive Functions of Symbolic Expressions and their Computations by Machine, Part I. *Communications of the ACM*, 3(4):184-195, 1960.

[9] E.G.J.M.H. Nöcker, J.E.W. Smetsers, M.C.J.D. van Eekelen, and M.J. Plasmeijer. Concurrent Clean. In E.H.L. Aarts, J. Leeuwen, and M. Rem, editors, *Proc. of Parallel Architectures and Languages Europe, PARLE'91*, LNCS 506:202-219, 1992.