Proving termination with adornments

Alexander Serebrenik, Danny De Schreye

Department of Computer Science, K.U. Leuven
Celestijnenlaan 200A, B-3001, Heverlee, Belgium
E-mail: {Alexander.Serebrenik,Danny.DeSchreye}@cs.kuleuven.ac.be

Developing reliable software is one of the most important challenges posed by modern society. In this context verifying correctness of the software becomes crucial. In our work we consider one aspect of program correctness, namely, termination. Logic programming provides a framework with a strong theoretical basis for tackling the problem of termination. On the other hand, due to the declarative formulation of programs, the danger of non-termination may increase. As a result, termination analysis received considerable attention in logic programming (see e.g. [1, 5, 7, 9]). Termination proofs often are based on the concept of a *norm* (a *level mapping*), a function mapping terms (atoms) of the program to the natural numbers.

Recently, using *type information* became a prominent trend in termination analysis for symbolic computations [2, 3, 13]. On the other hand, the study of termination of numerical programs led to the emerging of the *adorning technique* [11, 12]. Both approaches are based on the idea of distinguishing between different subsets of values for variables, and deriving norms and level mappings based on these subsets. Therefore, we would like to investigate these similarities and propose a framework for integration. In the current paper we discuss some preliminary results in this direction.

The key notion of the integrated framework is the notion of *a set of adornments*, partitioning the domain of the predicates arguments into a finite set of pairwise disjoint segments. A set of adornments is usually derived from the program itself. As a basis for a set of adornments one can use types [2], sets of integers [11], and even interargument relations [8]. Given a program and a set of adornments, the program can be transformed such that a predicate p^a in the transformed program P^a is called if and only if the corresponding predicate p is called in the original program P and the arguments of the call are in the segment a. One can show that under certain conditions this transformation preserves termination. In this way (implicit) information on the domain is made explicit in the transformed program. It should be observed, that instead of one level mapping required for P that is supposed to decrease along all possible computations, in order to prove termination of P^a one can find a number of potentially less sophisticated different level mappings for each one of the "cases".

In the examples we have considered, such level-mappings can be constructed automatically, and thus, they play a key role in automation of the approach.

After the transformation step is performed, existing termination analysers can be applied to infer termination of the transformed program. Since the transformation above preserved termination, termination of the original program is implied as well. The importance of the transformation can be illustrated by observing that for such examples as *dist* [6], none of the termination analysers available (cTI [9], TALP [10], TermiLog [7], TerminWeb [4], and Hasta-La-Vista [11]) is powerful enough to prove termination of the original program, while all of them succeed in proving termination of the transformed one.

We summarise the discussion above, by claiming that the main contribution of this work is twofold. First, it provides a link between two different but related approaches to termination analysis based on type information by integrating both of them in one framework. Second, the integrated approach turned out to be powerful enough to prove termination of such example as *dist* [6], that could not have been analysed by the previous techniques. As a future work we consider implementing the approach and evaluating its robustness experimentally.

References

1. K. R. Apt, E. Marchiori, and C. Palamidessi. A declarative approach for first-order built-in's in Prolog. *Applicable Algebra in Engineering, Communication and Computation*, 5(3/4):159–191, 1994.
2. M. Bruynooghe, M. Codish, S. Genaim, and W. Vanhoof. Reuse of results in termination analysis of typed logic programs. In M. V. Hermenegildo and G. Puebla, editors, *Static Analysis, 9th International Symposium*, volume 2477 of *Lecture Notes in Computer Science*, pages 477–492. Technical University of Madrid (Spain), School of Computer Science, Springer Verlag, 2002.
3. M. Bruynooghe, W. Vanhoof, and M. Codish. Pos(T): Analyzing dependencies in typed logic programs. In D. Bjørner, M. Broy, and A. V. Zamulin, editors, *Perspectives of System Informatics, 4th International Andrei Ershov Memorial Conference, PSI 2001, Akademgorodok, Novosibirsk, Russia, July 2-6, 2001, Revised Papers*, volume 2244 of *Lecture Notes in Computer Science*. Springer Verlag, 2001.
4. M. Codish and C. Taboch. A semantic basis for termination analysis of logic programs. *Journal of Logic Programming*, 41(1):103–123, 1999.
5. S. Decorte, D. De Schreye, and H. Vandecasteele. Constraint-based termination analysis of logic programs. *ACM TOPLAS*, 21(6):1137–1195, November 1999.
6. N. Dershowitz and C. Hoot. Topics in termination. In C. Kirchner, editor, *Rewriting Techniques and Applications, 5th International Conference*, volume 690 of *Lecture Notes in Computer Science*, pages 198–212. Springer Verlag, 1993.
7. N. Dershowitz, N. Lindenstrauss, Y. Sagiv, and A. Serebrenik. A general framework for automatic termination analysis of logic programs. *Applicable Algebra in Engineering, Communication and Computing*, 12(1-2):117–156, 2001.
8. N. Lindenstrauss, Y. Sagiv, and A. Serebrenik. Unfolding mystery of the *mergesort*. In N. Fuchs, editor, *Proceedings of the* 7^{th} *International Workshop on Logic Program Synthesis*

and Transformation, volume 1463 of *Lecture Notes in Computer Science*. Springer Verlag, 1998.

9. F. Mesnard. Inferring left-terminating classes of queries for constraint logic programs. In M. Maher, editor, *Proceedings of the 1996 Joint International Conference and Syposium on Logic Programming*, pages 7–21. The MIT Press, 1996.
10. E. Ohlebusch. Automatic termination proofs of logic programs via rewrite systems. *Applicable Algebra in Engineering, Communication and Computing*, 12(1-2):73–116, 2001.
11. A. Serebrenik and D. De Schreye. Inference of termination conditions for numerical loops in Prolog. In R. Nieuwenhuis and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning, 8th International Conferencerence, Proceedings*, volume 2250 of *Lecture Notes in Computer Science*, pages 654–668. Springer Verlag, 2001.
12. A. Serebrenik and D. De Schreye. On termination of logic programs with floating point computations. In M. V. Hermenegildo and G. Puebla, editors, *9th International Static Analysis Symposiumsium*, volume 2477 of *Lecture Notes in Computer Science*, pages 151–164. Springer Verlag, 2002.
13. W. Vanhoof and M. Bruynooghe. When size does matter - Termination analysis for typed logic programs. In A. Pettorossi, editor, *Logic-based Program Synthesis and Transformation, 11th International Workshop, LOPSTR 2001, Selected Papers*, volume 2372 of *Lecture Notes in Computer Science*, pages 129–147. Springer Verlag, 2002.