Termination of Programs using Term Rewriting and SAT Solving*

J. Giesl[1], P. Schneider-Kamp[1], R. Thiemann[1], S. Swiderski[1], M. T. Nguyen[2], D. De Schreye[2], and A. Serebrenik[3]

[1] LuFG Informatik 2, RWTH Aachen, Ahornstr. 55, 52074 Aachen, Germany, {giesl,psk,thiemann,swiderski}@informatik.rwth-aachen.de
[2] Dept. of Computer Science, K. U. Leuven, Belgium, {ManhThang.Nguyen,Danny.DeSchreye}@cs.kuleuven.be
[3] Dept. of Mathematics and Computer Science, TU Eindhoven, P.O. Box 513, 5600 MB Eindhoven, The Netherlands, a.serebrenik@tue.nl

1 Introduction

There are many powerful techniques and tools for automated termination analysis of term rewrite systems (TRSs). However, up to now they have hardly been used for real programming languages. Therefore, our goal is to apply existing methods and systems from term rewriting in order to prove termination of programs automatically. We discuss two possible approaches:

1. One could *transform* programs into TRSs and then use existing tools to verify termination of the resulting TRSs.
2. One could adapt TRS-techniques to the respective programming languages in order to analyze programs *directly*.

We have developed such approaches for the functional language Haskell and the logic language Prolog. More precisely, we developed

- a *direct* method for termination analysis of *logic programs* (Sect. 2) where we adapted TRS-techniques in order to apply them to logic programs directly
- a *transformational* method for termination analysis of *logic programs* (Sect. 3), where we transform logic programs into TRSs
- a *transformational* method for termination analysis of *functional programs* in the language Haskell (Sect. 4)

Our results have been implemented in the termination provers AProVE [7] and Polytool [11, 12]. In order to handle termination problems resulting from real programs, these provers had to be coupled with modern *SAT solvers*, since the automation of the TRS-termination techniques had to be improved significantly, cf. Sect. 5. Our resulting termination analyzers are currently the most powerful ones for Haskell and Prolog.

* Supported by the Deutsche Forschungsgsmeinschaft DFG under grant GI 274/5-1.

2 Direct Termination of Logic Programs

Termination analysis of logic programs is a widely studied subject which has been investigated for decades. Nevertheless, we showed that its power can be increased dramatically by adapting termination techniques from term rewriting to the logic programming domain. More precisely, we modified the concepts of *polynomial orderings* [10] and of *dependency pairs* and *dependency graphs* [1] in order to apply them to logic programs directly.

The main idea of this new approach is that termination conditions for a program are established based on the decomposition of its dependency graph into its strongly connected components. These conditions can then be analyzed separately by possibly different well-founded orders. We developed a constraint-based approach for automating this framework. Then, for example, termination techniques based on polynomial interpretations can be plugged in as a component to generate well-founded orders. For details on this approach, we refer to [13].

This approach has been implemented in the Polytool system [11, 12] which obtained the second place in the *International Competition of Termination Tools* 2007 [9]. This shows that the use of term rewriting techniques indeed improves the performance of termination analyzers for logic programs significantly. Indeed, Polytool is now the most powerful direct termination analyzer for logic programs.

3 Transformational Termination of Logic Programs

There are two kinds of approaches for termination analysis of logic programs: "transformational" and "direct" ones. Direct approaches (as in Sect. 2) prove termination directly on the basis of the logic program. Transformational approaches transform a logic program into a TRS and then analyze termination of the resulting TRS instead. Thus, transformational approaches make all methods previously developed for TRSs available for logic programs as well. However, the applicability of most existing transformations is quite restricted, as they can only be used for certain subclasses of logic programs. (Most of them are restricted to *well-moded* programs.) We improved these transformations such that they become applicable for *any* definite logic program. Moreover, our transformation results in TRSs which are indeed suitable for *automated* termination analysis. In contrast to most other methods for termination of logic programs, our technique is also sound for logic programming *without occur check*, which is typically used in practice. For details on these contributions, we refer to [16].

We implemented our approach in the termination prover AProVE [7]. AProVE was already the most powerful tool for termination analysis of TRSs and the winner of the *International Competitions of Termination Tools* [9] for TRSs in all years 2004 - 2007. But due to this new transformational approach, AProVE is now also the most powerful system and the winner of the *International Competition of Termination Tools* for logic programs. This shows that TRS-techniques can really be used for existing programming languages and can clearly compete with specialized termination methods developed for specific programming languages.

4 Transformational Termination of Functional Programs

In addition to our work on logic programming, we also developed a new approach which permits the application of existing techniques from term rewriting in order to prove termination of programs in the functional language Haskell. Adapting TRS-techniques for termination of Haskell is challenging for the following reasons:

- Haskell has a *lazy evaluation* strategy. However, most TRS-techniques ignore such evaluation strategies and try to prove that *all* reductions terminate.
- Defining equations in Haskell are handled from top to bottom. In contrast for TRSs, *any* rule may be used for rewriting.
- Haskell has polymorphic types, whereas TRSs are untyped.
- In Haskell-programs with infinite data objects, only certain functions are terminating. But most TRS-methods try to prove termination of *all* terms.
- Haskell is a *higher-order* language, whereas most automatic termination techniques for TRSs only handle first-order rewriting.

Up to now, there were only few techniques for automated termination analysis of functional programs. These were all "stand-alone" methods which did not allow the use of modern termination techniques from term rewriting. In our approach we built upon the method of [14], but adapted it in order to make TRS-techniques applicable. For details on our technique we refer to [6].

Again, we implemented our results in the termination prover AProVE [7]. It accepts the full Haskell 98 language defined in [15] and we successfully evaluated our implementation with standard Haskell-libraries from the Hugs-distribution such as Prelude, Monad, List, FiniteMap, etc. It turns out that AProVE can automatically prove termination of around 80 % of the functions in these libraries. Hence, these experiments confirm that TRS-termination techniques and tools can be successfully used for real-life programs in existing languages.

5 SAT Solving

In the preceding three sections, we showed how to apply TRS-techniques for termination analysis of programs in different languages. However, termination problems resulting from real programs are often *large*. For example, the transformation of logic programs in Sect. 3 often generates TRSs with function symbols of high arity and the transformation of Haskell programs in Sect. 4 typically results in TRSs with a large number of rules. Here, existing previous algorithms to automate TRS-termination techniques often turned out to be too inefficient.

Therefore, recently several researchers have studied alternative ways to automate TRS-termination techniques using *SAT solving* (e.g., [2–5, 8, 17–19]). Indeed, AProVE uses powerful SAT solvers for several different tasks and Polytool applies AProVE's SAT-based implementation for synthesizing polynomial orderings [5] as a back-end. So by SAT solving, the TRS-techniques can indeed be automated efficiently and used successfully for real programs.

References

1. T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133-178, 2000.
2. M. Codish, V. Lagoon, and P. Stuckey. Solving partial order constraints for LPO termination. In *Proc. RTA '06*, LNCS 4098, p. 4-18, 2006.
3. M. Codish, P. Schneider-Kamp, V. Lagoon, R. Thiemann, and J. Giesl. SAT solving for argument filterings. In *Proc. LPAR '06*, LNAI 4246, p. 30-44, 2006.
4. J. Endrullis, J. Waldmann, and H. Zantema. Matrix interpretations for proving termination of term rewriting. In *Proc. IJCAR '06*, LNAI 4130, p. 574-588, 2006.
5. C. Fuhs, J. Giesl, A. Middeldorp, P. Schneider-Kamp, R. Thiemann, and H. Zankl. SAT solving for termination analysis with polynomial interpretations. In *Proc. SAT '07*, LNCS 4501, p. 340-354, 2007.
6. J. Giesl, S. Swiderski, P. Schneider-Kamp, and R. Thiemann. Automated termination analysis for Haskell: From term rewriting to programming languages. In *Proc. RTA '06*, LNCS 4098, p. 297-312, 2006.
7. J. Giesl, P. Schneider-Kamp, and R. Thiemann. AProVE 1.2: Automatic termination proofs in the DP framework. *Proc. IJCAR '06*, LNAI 4130, p. 281-286, 2006.
8. A. Koprowski and A. Middeldorp. Predictive labeling with dependency pairs using SAT. In *Proc. CADE '07*, LNAI 4603, p. 410-425, 2007.
9. C. Marché and H. Zantema. The termination competition. In *Proc. RTA '07*, LNCS 4533, p. 303-313, 2007.
10. D. Lankford. On proving term rewriting systems are Noetherian. Technical Report MTP-3, Louisiana Technical University, Ruston, LA, USA, 1979.
11. M. T. Nguyen and D. De Schreye. Polynomial interpretations as a basis for termination analysis of logic programs. In *Proc. ICLP '05*, LNCS 3668, p. 311-325, 2005.
12. M. T. Nguyen and D. De Schreye. Polytool: Proving termination automatically based on polynomial interpretations. In *Proc. LOPSTR '06*, LNCS 4407, p. 210-218, 2007.
13. M. T. Nguyen, J. Giesl, P. Schneider-Kamp, and D. De Schreye. Termination analysis of logic programs based on dependency graphs. In *Proc. LOPSTR '07*, LNCS, 2007. To appear.
14. S. E. Panitz and M. Schmidt-Schauss. TEA: Automatically proving termination of programs in a non-strict higher-order functional language. In *Proc. SAS '97*, LNCS 1302, p. 345–360, 1997.
15. S. Peyton Jones (ed.). Haskell 98 *Languages and Libraries: The revised report*. Cambridge University Press, 2003.
16. P. Schneider-Kamp, J. Giesl, A. Serebrenik, R. Thiemann. Automated termination analysis for logic programs by term rewriting. In *Proc. LOPSTR '06*, LNCS 4407, p. 177-193, 2007.
17. P. Schneider-Kamp, R. Thiemann, E. Annov, M. Codish, and J. Giesl. Proving termination using recursive path orders and SAT solving. In *Proc. FroCoS '07*, LNAI 4720, p. 267-282, 2007.
18. H. Zankl, N. Hirokawa, and A. Middeldorp. Constraints for argument filterings. In *Proc. SOFSEM '07*, LNCS 4362, p. 579-590, 2007.
19. H. Zankl and A. Middeldorp. Satisfying KBO constraints. *Proc. RTA '07*, LNCS 4533, p. 389-403, 2007.