Künstl Intell (2010) 24: 79–81
DOI 10.1007/s13218-010-0015-5

DISSERTATIONEN UND HABILITATIONEN

Static Termination Analysis for Prolog Using Term Rewriting and SAT Solving

Peter Schneider-Kamp

Published online: 9 February 2010

Abstract The dissertation "Static Termination Analysis for Prolog using Term Rewriting and SAT Solving" (Schneider-Kamp in Dissertation, RWTH Aachen University, 2008) presents a fresh approach to automated termination analysis of Prolog programs. This approach is based on the following three main concepts: the use of program transformations to obtain simpler termination problems, a framework for modular termination analysis, and the encoding of search problems into satisfiability of propositional logic (SAT) for efficient generation of ranking functions.

1 Overview

The most fundamental decision problem in computer science is the halting problem, i.e., given a program and an input, decide whether the program terminates after finitely many steps. While Turing showed this problem to be undecidable, developing static analysis techniques that can automatically prove termination for many programs is of great practical interest.

This is true in particular for logic programming (LP). Due to the lack of direction in the computation, any non-trivial program terminates only for certain classes of inputs. Thus, termination of LPs is widely studied and significant advances have been made during the last decades. Still, there remain many LPs that cannot be handled by current termination tools.

The main challenges in the termination analysis of Prolog programs include the analysis of programs with extra-logical features such as cuts and negation-as-failure, the combination of constraint-based and local approaches, the better use of termination techniques and tools developed for term rewrite systems (TRSs), and the efficient generation of ranking functions.

To handle these challenges, we introduce a program transformation to analyze the effect of cuts and negation-as-failure, a modular framework that allows for constraint-based local analysis, a program transformation from LPs to TRSs that allows to reuse existing powerful tools for TRSs, and an encoding of the search for ranking functions into SAT (cf. Fig. 1).

The above contributions are all implemented—mostly in our fully automated termination prover AProVE, which has reached the highest score both for LPs and TRSs at all annual International Termination Competitions since 2004.

2 Analyzing the Effect of Cuts

Virtually all existing approaches for automated termination analysis of LPs are limited to definite LPs. There are several major differences between this notion and Prolog programs: In definite LPs, the only method of computation is left-to-right, depth-first search (SLD resolution). In practice, Prolog programs make use of additional extra-logical constructs to cut the search space (! operator) or implement negation-as-failure (\+ operator).

Consider for example the following Prolog program which checks if the list given in the second argument can be obtained by concatenating the list in the first argument to itself repeatedly:

$$\mathsf{star}(U, [\,]) \leftarrow !.$$
$$\mathsf{star}([\,], W) \leftarrow !, W = [\,].$$
$$\mathsf{star}(U, W) \leftarrow \mathsf{append}(U, V, W), \mathsf{star}(U, V).$$

P. Schneider-Kamp (✉)
http://imada.sdu.dk/~petersk/
e-mail: petersk@imada.sdu.dk

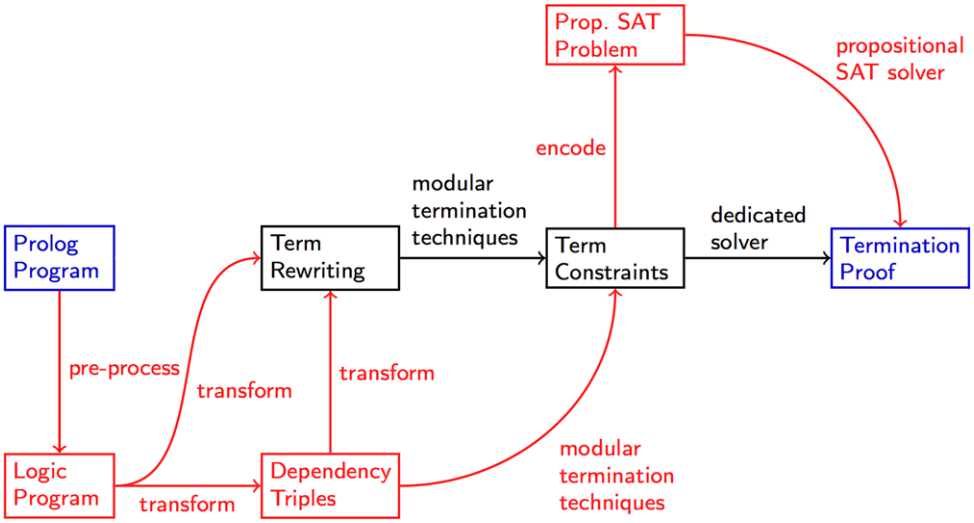

Fig. 1 Overview of contributions.

Now consider the input query star([ ], [a]). The second clause introduces a cut and thus prevents the third clause from being applied. If we ignored the cut, using the third clause and applying the built-in rules for append we would again obtain our initial query and thereby result in non-termination.

In order to analyze the effect of cuts, we introduce a novel pre-processing step for LPs. This program transformation transforms an LP with cuts into a definite LP such that the termination of the cut-free program guarantees the termination of the original program. In this way, we can also handle LPs with negation-as-failure as this can readily be expressed using a cut. For the example above we obtain the following cut-free program:

$$\mathsf{star}(U, [\,]).$$
$$\mathsf{star}([X|U], [X|W]) \leftarrow \mathsf{append}(U, V, W), \mathsf{star}([X|U], V).$$

The idea of the transformation above is to symbolically evaluate the (possibly) infinite set of inputs specified by the user using symbolic evaluation rules that capture both SLD resolution as well as the effect of cuts. The result of applying one such evaluation step to a set of inputs is a new set of inputs whose termination guarantees the termination of the previous set of inputs. Overall, we obtain a (usually infinite) tree labeled by sets of inputs. To allow for an automated analysis, we turn the infinite tree into a finite graph by introducing special "back-edges" whenever we find an instance of a previous node.

To guarantee termination, i.e., to show that cycles in the graph cannot be traversed infinitely often, we read a new LP off the graph that includes a recursive clause for each cycle. If we can prove the termination of this new program, we have shown that the cycles cannot be traversed infinitely often and the original sets of inputs are indeed terminating.

3 A Modular Framework

Techniques for termination analysis of LPs can be divided into two groups: the global approach versus the local approach. In the global approach, we can use only one ranking function for all loops. In contrast, in the local approach we may apply different ranking functions for different loops. Typically only a given small set of fixed functions is used. In contrast, in the global approach a constraint-based framework is used to generate functions.

We introduce a novel "dependency triple" framework for termination analysis of LPs that is both local and constraint-based [2]. The basic idea is to reformulate the termination problem as a modular problem that can be split into subproblems and to generate constraints separately for each modular subproblem.

4 Transformation to TRSs

Another area where termination has been studied even more intensively than in logic programming is term rewriting. Significant advances during the last decade have yielded many powerful techniques and tools for proving termination of TRSs automatically. To use these also for LPs, we introduce a novel transformation from LPs to a specialized version of term rewriting based on infinite terms [3]. For the example above we obtain:

$$\mathsf{star}_{in}(U, [\,]) \to \mathsf{star}_{out}(U, [\,])$$
$$\mathsf{star}_{in}([X|U], [X|W]) \to \mathsf{u}(\mathsf{append}_{in}(U, V, W), X, U)$$
$$\mathsf{u}(\mathsf{append}_{out}(U, V, W), X, U) \to \mathsf{star}_{out}([X|U], V).$$

The idea of the transformation is to model each step of the LP by a number of steps in the TRS. To prove the termination of a set of inputs, it now suffices to prove the termination of the corresponding TRS on a corresponding set of

terms. We show that by adapting the most popular framework for termination analysis of TRSs, all existing techniques for TRSs can be used.

5 SAT for Efficient Automation

No matter if we use the modular framework or the transformational approach, in the end one only has to search for a suitable ranking function. In the context of LPs and TRSs, ranking functions are well-founded orders on terms. To find such orders, many tools search for polynomial orders, i.e., polynomial interpretations of terms into natural numbers. In recent work (cf. [4]) we show how to find polynomial orders using an encoding of NP-complete non-linear constraint solving to SAT.

Still, there are many natural examples where no tool based on polynomial orders can show termination. A popular class of orders that often succeeds in such cases is the class of syntactic path orders. We therefore introduce an encoding of the NP-complete search for such path orders into SAT [5]. To this end, we show how to encode partial orders, lexicographic and multiset extensions of a base order, and argument filters to SAT. By combining these encodings, the search for orders like the lexicographic path order (LPO), the multiset path order (MPO), or the recursive path order with status (RPO) can be achieved.

An empirical evaluation of the SAT-based solver on 865 inputs from the competition shows improvements of multiple orders of magnitude over existing dedicated solvers (cf. Fig. 2).

6 Summary and Impact

The dissertation has shown how to solve some of the main challenges in termination analysis of LPs. The contributions made to this end are useful in other contexts, too. The efficient search for path orders is also successfully used for TRSs and functional programs. The transformation from LPs to TRSs has also been adapted to analyze the termination of narrowing, i.e., the evaluation mechanism underlying functional logic programming. The symbolic evaluation approach for analyzing the termination behavior of programming languages has also been used for the functional language Haskell 98 [6] and for Java bytecode.

	LPO	MPO	RPO
dedicated	1426.3 / 123	2669.1 / 92	4708.2 / 158
SAT-based	44.7 / 123	74.2 / 92	85.3 / 162

Fig. 2 Total runtime in seconds/examples solved in 60 s.

References

1. Schneider-Kamp P (2008) Static termination analysis of prolog using term rewriting and SAT solving. Dissertation, RWTH Aachen University. Available from: http://sunsite.informatik.rwth-aachen.de/Publications/AIB/2008/2008-17.pdf
2. Schneider-Kamp P, Giesl J, Nguyen MT (2009) The dependency triple framework for termination of logic programs. In: LOPSTR'09, LNCS. Springer, Berlin (to appear)
3. Schneider-Kamp P, Giesl J, Serebrenik A, Thiemann R (2009) Automated termination proofs for logic programs by term rewriting. ACM Transactions on Computational Logic, 11(1)
4. Fuhs C, Giesl J, Middeldorp A, Schneider-Kamp P, Thiemann R, Zankl H (2007) SAT solving for termination analysis with polynomial interpretations. In: SAT'07. LNCS, vol 4501. Springer, Berlin
5. Schneider-Kamp P, Thiemann R, Annov E, Codish M, Giesl J (2007) Proving termination using recursive path orders and SAT solving. In: FroCoS'07. LNAI, vol 4720. Springer, Berlin
6. Giesl J, Swiderski S, Schneider-Kamp P, Thiemann R (2006) Automated termination analysis for Haskell: from term rewriting to programming languages. In: RTA'06. LNCS, vol 4098. Springer, Berlin

Peter Schneider-Kamp is an assistant professor at the University of Southern Denmark in Odense. His main research interests are in *automated reasoning* with a focus on *program verification* and, in particular, *automated termination analysis*. To facilitate this research, Peter Schneider-Kamp uses and develops methods from a diversity of areas including *program analysis*, *constraint solving*, *term rewriting*, and *theorem proving*.