The TALP Tool for Termination Analysis of Logic Programs

Enno Ohlebusch[1], Claus Claves[2], and Claude Marché[3]

[1] Faculty of Technology, University of Bielefeld
P.O. Box 10 01 31, 33501 Bielefeld, Germany
Email: enno@techfak.uni-bielefeld.de
[2] Bremer Str. 39, 33613 Bielefeld, Germany
claus.claves@bertelsmann.de
[3] LRI (UMR 8623 du CNRS) & INRIA Futurs
Bât. 490, Université de Paris-Sud, Centre d'Orsay
91405 Orsay Cedex, France
Email: marche@lri.fr

1 Introduction

In the last decade, the automatic termination analysis of logic programs has been receiving increasing attention. Among other methods, techniques have been proposed that transform a well-moded logic program into a term rewriting system (TRS) so that termination of the TRS implies termination of the logic program under Prolog's selection rule. In [8] it has been shown that the two-stage transformation obtained by combining the transformations of [6] into deterministic conditional TRSs (CTRSs) with a further transformation into TRSs [3] yields the transformation proposed in [2], and that these three transformations are equally powerful. In most cases simplification orderings are not sufficient to prove termination of the TRSs obtained by the two-stage transformation. However, if one uses the dependency pair method [1] in combination with polynomial interpretations instead, then most of the examples described in the literature can automatically be proven terminating. Based on these observations, we have implemented a tool for proving termination of logic programs automatically. This tool consists of a front-end which implements the two-stage transformation and a back-end, the C*i*ME system [4], for proving termination of the generated TRS.

As in [5], we have tested the TALP system on well-known examples (the benchmarks are collected in [7]). A Web interface for TALP is available at http://bibiserv.techfak.uni-bielefeld.de/talp/. Overall, our results are comparable to those reported in [5] but there are examples for which TALP succeeds and other tools don't (the example in Sect. 3 for instance) and vice versa. During our experiments we made the following observations. For the class of TRSs generated from logic programs, a termination proof using the standard Manna-Ness criterion is usually not possible. On the other hand, the dependency pairs criterion is quite powerful, using moreover very simple polynomial interpretations: linear polynomial interpretation with coefficients in the interval $[0;2]$. The restriction to linear polynomial interpretations seems to be a very good heuristic because whenever searching for a linear interpretation fails, then searching for a more general one, like simple-mixed, does not succeed either.

Moreover, for programs of a larger size, with many predicates, being able to perform an *incremental* proof [9, 10] appears to be very important.

2 Transformation of Programs into TRSs

TALP takes a Prolog program and a query as input and proceeds in four steps:

1. The Prolog program is translated into a logic program P. In this process, clauses with `if-then`-structures, disjunctions, or negated atoms are translated into new clauses. For instance, the clause $A \leftarrow B, \texttt{not}\, C, D$ is replaced with the clauses $A \leftarrow B, C, \texttt{fail}$ and $A \leftarrow B, D$. Cuts are ignored.
2. The query determines which of the arguments in its predicates are used as input and output, respectively. According to this information, the tool tries to generate a *moding* for the logic program such that the program is *well-moded*. If this step is successfully completed, the logic program will be transformed into a TRS as follows.
3. Every atom $A = p(t_1, \ldots, t_n)$ with input positions $i_1, \ldots, i_k$ and output positions $i_{k+1}, \ldots, i_n$ associates with a rewrite rule

$$\rho(A) = p_{in}(t_{i_1}, \ldots, t_{i_k}) \to p_{out}(t_{i_{k+1}}, \ldots, t_{i_n})$$

   and every program clause $C = A \leftarrow B_1, \ldots, B_m$ is transformed into a conditional rewrite rule $\rho(C) = \rho(A) \Leftarrow \rho(B_1), \ldots, \rho(B_m)$. The CTRS $R_P = \{\rho(C) \mid C \in P\}$ obtained in this way is deterministic because the logic program P is well-moded.
4. Every rule $l \to r \Leftarrow c \in R_P$ with n conditions in c is transformed into $n+1$ unconditional rewrite rules by operator U defined inductively by :

$$U(l \to r) = \{l \to r\}$$
$$U(l \to r \Leftarrow s \to t, c) = \{l \to u(s, x)\} \cup U(u(t, x) \to r \Leftarrow c)$$

   where u is a fresh symbol and $x = Var(l) \cap (Var(t) \cup Var(c) \cup Var(r))$

3 Example

If TALP gets the following Prolog program with query $\texttt{flat}(in, out)$ as input

```
flat(niltree, nil).
flat(tree(X, niltree, T), cons(X, L)) :- flat(T, L).
flat(tree(X, tree(Y, T1, T2), T3), L) :-
  flat(tree(Y, T1, tree(X, T2, T3)), L).
```

then the first transformation yields the CTRS

$$\texttt{flat}_{in}(\texttt{niltree}) \to \texttt{flat}_{out}(\texttt{nil})$$
$$\texttt{flat}_{in}(\texttt{tree}(x, \texttt{niltree}, t)) \to \texttt{flat}_{out}(\texttt{cons}(x, l)) \Leftarrow \texttt{flat}_{in}(t) \to \texttt{flat}_{out}(l)$$
$$\texttt{flat}_{in}(\texttt{tree}(x, \texttt{tree}(y, t_1, t_2), t_3)) \to \texttt{flat}_{out}(l) \Leftarrow \texttt{flat}_{in}(\texttt{tree}(y, t_1, \texttt{tree}(x, t_2, t_3))) \to \texttt{flat}_{out}(l)$$

and the second transformation yields the unconditional TRS

$$\begin{aligned}
\mathtt{flat}_{in}(\mathtt{niltree}) &\to \mathtt{flat}_{out}(\mathtt{nil})\\
\mathtt{flat}_{in}(\mathtt{tree}(x,\mathtt{niltree},t)) &\to u_1(\mathtt{flat}_{in}(t),x)\\
u_1(\mathtt{flat}_{out}(l),x) &\to \mathtt{flat}_{out}(\mathtt{cons}(x,l))\\
\mathtt{flat}_{in}(\mathtt{tree}(x,\mathtt{tree}(y,t_1,t_2),t_3)) &\to u_2(\mathtt{flat}_{in}(\mathtt{tree}(y,t_1,\mathtt{tree}(x,t_2,t_3))))\\
u_2(\mathtt{flat}_{out}(l)) &\to \mathtt{flat}_{out}(l)
\end{aligned}$$

Subsequently C*i*ME is asked to find a linear polynomial interpretation with coefficients in the interval $[0;2]$. It generates the following interpretation

$$\begin{array}{llllll}
[\![\mathtt{nil}]\!] & = 0 & [\![\mathtt{flat}_{out}]\!](x_0) & = 0 & [\![u_1]\!](x_0,x_1) & = 0\\
[\![\mathtt{niltree}]\!] & = 0 & [\![u_2]\!](x_0) & = 0 & [\![\mathtt{tree}]\!](x_0,x_1,x_2) & = x_2+2x_1+1\\
[\![\mathtt{flat}_{in}]\!](x_0) & = 0 & [\![\mathtt{cons}]\!](x_0,x_1) & = 0 & [\![\mathtt{FLAT}_{in}]\!](x_0) & = x_0
\end{array}$$

and the induced polynomial ordering satisfies all constraints obtained from the cycles in the estimated dependency graph.

References

1. T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133–178, 2000.
2. T. Arts and H. Zantema. Termination of logic programs via labelled term rewrite systems. In *Proceedings of Computing Science in the Netherlands*, pages 22–34, 1995.
3. M. Chtourou and M. Rusinowitch. Méthode transformationnelle pour la preuve de terminaison des programmes logiques. Unpublished manuscript, 1993.
4. E. Contejean, C. Marché, B. Monate, and X. Urbain. CiME version 2, 2000. Available at `http://cime.lri.fr/`.
5. S. Decorte, D. D. Schreye, and H. Vandecasteele. Constraint-based termination analysis of logic programs. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 21(6):1137–1195, 1999.
6. H. Ganzinger and U. Waldmann. Termination proofs of well-moded logic programs via conditional rewrite systems. In *3rd International Workshop on Conditional Term Rewriting Systems*, volume 656 of *Lecture Notes in Computer Science*, pages 113–127, Berlin, 1993. Springer-Verlag.
7. N. Lindenstrauss and Y. Sagiv. Automatic termination analysis of logic programs (with detailed experimental results). Technical report, Hebrew University, Jerusalem, 1997.
8. E. Ohlebusch. Transforming conditional rewrite systems with extra variables into unconditional systems. In *6th International Conference on Logic for Programming and Automated Reasoning*, volume 1705 of *Lecture Notes in Artificial Intelligence*, pages 111–130, Berlin, 1999.
9. X. Urbain. Automated incremental termination proofs for hierarchically defined term rewriting systems. In R. Goré, A. Leitsch, and T. Nipkow, editors, *International Joint Conference on Automated Reasoning*, volume 2083 of *Lecture Notes in Artificial Intelligence*, pages 485–498, Siena, Italy, june 2001. Springer-Verlag.
10. X. Urbain. Modular and incremental automated termination proofs. *Journal of Automated Reasoning*, 2003. to appear.