TermiLog: A System for Checking Termination of Queries to Logic Programs *

N. Lindenstrauss, Y. Sagiv, A. Serebrenik
Dept. of Computer Science
Hebrew University
Jerusalem, Israel
Email: {naomil,sagiv,alicser}@cs.huji.ac.il

Abstract. *TermiLog* is a system implemented in SICStus Prolog for automatically checking termination of queries to logic programs. Given a program and query, the system either answers that the query terminates or that it cannot prove termination. The system can handle automatically 82% of the 120 programs we tested it on.

1 Introduction

TermiLog is a system, implemented in SICStus Prolog [SICS95], for automatic termination analysis of logic programs. The system accepts as input a Prolog program and a query, and returns as the answer either that the query terminates or that it cannot prove termination. In contrast to some other systems the program does not have to satisfy any condition in order to be analyzed by *TermiLog* (e.g., in the system of [Plu90], the program has to be well-moded). Most predefined predicates of Prolog may appear in the program and are handled directly or by suitable transformations.

The type of termination analyzed by the system is the termination of computing all the answers to the given query, using Prolog's computation rule. As pointed out in [O'K90], this is the relevant notion of termination for Prolog, because even when one is interested only in a single answer, it is still important to know that the computation of all answers terminates, due to the possibility of backtracking.

We have applied *TermiLog* to 120 programs, taken from the literature on termination and some benchmarks. 82% of these programs were analyzed correctly by *TermiLog*, completely automatically. The largest program that was analyzed is the 57-clause credit-evaluation expert system from [StSh86].

* This research was supported in part by grants from the Israel Science Foundation

2 Overview of the system

Termination is proved by using well-founded orderings on terms. Formally, we define a *norm* for each term as follows:

$$||f(T_1, \ldots T_n)|| = c + \sum_{i=1}^{n} a_i ||T_i||$$

where c and $a_1, \ldots, a_n$ are non-negative integers that depend only on f/n. The norm of a variable X is denoted by X itself. In general, the norm is a linear expression. To be used in a termination proof, however, the norm of the term must be an integer (such a term will be called *instantiated enough*). Note that the norm of a non-ground term may be an integer, since some of the a_i may be zero. Our definition of norm includes, as special cases, the term-size norm [VanG91] and the list-size norm [UV88].

The system consists of three main parts — see [LiSa96, LiSa97] for details. The first does the *instantiation analysis* — that is, it determines which argument positions of predicates are instantiated enough and which are not. The instantiation analysis is done by means of a bottom-up abstract interpretation similar to groundness analysis (cf. [Cous92]).

The second part is inference of constraints among argument sizes. The types of constraints are the *monotonicity* and *equality constraints* of [BrSa89], but the inference is done in a more accurate way. Since inferred constraints are not always needed to show termination, the system provides the option of restricting the constraint inference just to some parts of the given program.

The constraint inference also tells us whether a constraint is recursive or non-recursive. Non-recursive constraints can often be "factored out" from the termination analysis by automatic unfolding. This suggests a completely automatic way of handling, for example, the *mergesort* program, that previously was shown to terminate only by first applying some ad hoc transformation.

The third part consists of constructing the *query-mapping pairs* and applying the test of [Sag91], which was originally intended for Datalog programs and is here extended to general logic programs.

3 Benchmarks

This section sums up the results of applying our system to 120 programs taken from papers on termination [DSD94, Plu90, AP94, Ver92], the benchmark collection of [BGH94], and some other sources. *TermiLog* has analyzed correctly 82% of them. The results are given in the following table—explanations follow. The detailed results may be found in the tables in [LiSaExp].

The *TermiLog* system has analyzed correctly all the examples from the survey of [DSD94] on termination, including the mutually recursive *bool*. It is worth noting that mutual recursion does not require any special consideration in our system, while in earlier work [Plu90, VanG91] special transformations were needed to eliminate mutual recursion.

Source	Number of Programs	Handled Correctly Automatically
[DSD94]	**7**	**7** (100%)
[Plu90]	**17**	**16** (94%)
[AP94]	**19**	**17** (89%)
[BGH94]	**24**	**11** (46%)
[Ver92]	**32**	**26** (81%)
Other	**21**	**21**(100%)
Total	**120**	**98** (82%)

TermiLog can handle all the examples of [Plu90] that Plümer's own system can handle, except for the program *perm. TermiLog* can also handle the program *mult,* which Plümer's system cannot handle. The program *perm* would be handled by our system once linear equalities among argument sizes are added.

The paper of [AP94] does not deal with automatic termination analysis, but develops a theoretical basis for studying termination of logic programs as well as Prolog programs. Our system can handle all the examples of [AP94], except for program *perm* of [Plu90] and the map-coloring program of [StSh86].

The benchmark collection of [BGH94] has more complex programs than those usually found in the literature on termination. Out of the 24 programs in that collection, our system could handle 11 (46%) programs. Some programs of that benchmark could not be handled because the algorithms we use are not powerful enough to show their termination, while others were too big and caused memory problems.

The examples from [Ver92] are handled automaticallly except for six, three of the latter being programs in which termination depends on the differentiation between *constants* (cf. [Llo87]), which is not made in our abstraction (cf. [LiSa96]).

The *TermiLog* system has analyzed correctly 21 further examples, including
* Four programming examples from the SICStus manual [SICS95].
* Ackermann's function (from [StSh86]).
* Greatest common divisor.
* Huffman codes computation.
* Quicksort using difference lists (from [StSh86]).
* 8 queens.
* Rewriting system for normalizing expressions with an associative operator.
* A game program from [AP93].
* The Yale shooting problem from [AB91].
* The credit-evaluation expert system from [StSh86] (this 57-clause program is the biggest among all those analyzed).

It should be emphasized that all the experimental results reported in this section were obtained by using only the basic algorithms implemented in the system, and *without* any additional program transformations or other ad hoc features intended to increase the power of the system.

An example session with *TermiLog* is given in [LSS97].

References

[AB91] K. R. Apt and M. Bezem. Acyclic Programs. *New Generation Computing*, 9:335-363, 1991.

[AP93] K. R. Apt and D. Pedreschi. Reasoning about Termination of Pure Prolog Programs. *Information and Computation*, 106:109-157, 1993.

[AP94] K. R. Apt and D. Pedreschi. Modular Termination Proofs for Logic and Pure Prolog Programs. In *Advances in Logic Programming Theory*, 183-229. Oxford University Press, 1994.

[BrSa89] A. Brodsky and Y. Sagiv. Inference of monotonicity constraints in Datalog programs. *Proceedings of the Eighth ACM SIGACT-SIGART-SIGMOD Symposium on Principles of Database Systems*, 1989, 190-199.

[BGH94] F. Bueno, M. García de la Banda and M. Hermenegildo. Effectiveness of Global Analysis in Strict Independence-Based Automatic Program Parallelization. *International Symposium on Logic Programming*, 320-336. MIT Press, 1994.

[Cous92] P. Cousot and R. Cousot. Abstract interpretation and application to logic programs. *J. Logic Programming*, 13:103-179, 1992.

[DSD94] D. De Schreye and S. Decorte. Termination of Logic Programs: the Never-Ending Story. *J. Logic Programming*, 19/20:199-260, 1994.

[LiSa96] N. Lindenstrauss and Y. Sagiv. Checking Termination of Queries to Logic Programs. http://www.cs.huji.ac.il/~naomil/

[LiSa97] N. Lindenstrauss and Y. Sagiv. Automatic Termination Analysis of Logic Programs. ICLP'97. MIT Press, 1997.

[LiSaExp] N. Lindenstrauss and Y. Sagiv. Automatic Termination Analysis of Logic Programs (with Detailed Experimental Results). http://www.cs.huji.ac.il/~naomil/

[LSS97] N. Lindenstrauss ,Y. Sagiv and A. Serebrenik. An Example Session with *TermiLog*. http://www.cs.huji.ac.il/~naomil/

[Llo87] J. W. Lloyd. *Foundations of Logic Programming*. Springer Verlag, second edition, 1987.

[O'K90] R. A. O'Keefe. *The Craft of Prolog*. MIT Press, 1990.

[Plu90] L. Plümer. *Termination Proofs for Logic Programs*. Springer Verlag, LNAI 446, 1990.

[Sag91] Y. Sagiv. A termination test for logic programs. In *International Logic Programming Symposium*. MIT Press, 1991.

[SICS95] *SICStus Prolog User's Manual*. Release 3. Swedish Institute of Computer Science, 1995.

[StSh86] L. Sterling and E. Shapiro. *The Art of Prolog*. MIT Press, 1986.

[UV88] J. D. Ullman and A. Van Gelder. Efficient tests for top-down termination of logical rules. JACM 35:2(1988), 345-373.

[VanG91] A. Van Gelder. Deriving constraints among argument sizes in logic programs. *Annals of Mathematics and Artificial Intelligence*, 3:361–392, 1991.

[Ver92] C. Verschaetse. Static Termination Analysis for Definite Horn Clause Programs. Ph.D. Thesis, K.U. Leuven, 1992.