TERMINATION OF SIMPLY MODED WELL-TYPED LOGIC PROGRAMS UNDER TABLED EXECUTION MECHANISM

Sofie Verbaeten and Danny De Schreye
Department of Computer Science, K.U.Leuven,
Celestijnenlaan 200A, B-3001 Heverlee, Belgium.
{sofie.verbaeten, danny.deschreye}@cs.kuleuven.ac.be

Tabled logic programming [2, 4, 12, 14] is receiving increasing attention in the Logic Programming community. It avoids many of the shortcomings of SLD(NF) execution [10] and provides a more flexible and often extremely efficient execution mechanism for logic programs. In particular, tabled execution of logic programs terminates more often than execution based on SLD-resolution. So, if a program can be proven to terminate under SLD-resolution (by one of the existing automated techniques surveyed in [5]), then the program will trivially also terminate under SLG-resolution, the resolution principle of tabulation [4]. But, since there are SLG-terminating programs which are not SLD-terminating, better proof techniques can be found.

The ideas underlying *tabling* are very simple. Essentially, under tabled execution mechanism, answers for selected atoms are stored in a table. When a variant of such an atom is recursively called, the selected atom is not resolved against program clauses, instead, all corresponding answers computed so far are looked up in the table and the corresponding answer substitutions are applied to the atom. This process is repeated for all subsequent computed answer substitutions that correspond to the atom.

We study *universal* termination of definite tabled logic programs executed under SLG using a fixed left-to-right selection rule (we drop the "S" in SLD and SLG whenever we refer to the left-to-right selection rule). We refer to the full paper [13] for the formal definitions, theorems and motivating examples. Note that, whenever SLD-computation from an initial goal leads to infinitely many different, non-variant calls, the SLG-computation from this initial goal will also be infinite. This leads to the first basic notion of termination under tabled execution mechanism: *quasi-termination* (this term is borrowed from [9], defining a similar notion in the context of termination of off-line partial evaluation of functional programs). For a program P and a set of queries Q, let $Call(P,Q)$ denote the set of atoms (modulo variant relation) which occur as selected atom in an LD-derivation of some query in Q. This set is also called the call set of the program P w.r.t. Q. With this notation, a program P is said to quasi-terminate w.r.t. a set of queries S iff for all $A \in S$, $\sharp Call(P,\{A\}) < \infty$. Even when tabling, quasi-termination only partially corresponds to our intuitive notion of a "terminating computation". This is because an atom can have infinitely many computed answers (note that this does not have to lead to infinitely many new calls). Therefore, the stronger notion of *LG-termination* is introduced: P LG-terminates w.r.t. S iff P quasi-terminates w.r.t. S and for all $A \in S$ the set of all computed answers for atoms in $Call(P,\{A\})$ is finite.

One of the few approaches studying termination of tabled logic programs was developed by Decorte et al [8]. They present necessary and sufficient conditions for quasi-termination and LG-termination. The main differences with characterizations (of e.g.[6]) of LD-terminating programs and queries are:

- level mappings are assumed to be *finitely partitioning* on the call set, i.e. only finitely many atoms of the call set can be mapped to the same natural number,
- decreases of the level mapping are *not* assumed to be *strict* (but are imposed on all body atoms).

Starting from the necessary and sufficient conditions of [8], we introduce sufficient, easy to *automatize* conditions for quasi-termination and LG-termination. We show how our termination conditions can be automatized by extending the recently developed constraint-based automatic termination analysis for SLD-resolution by Decorte and De Schreye [7]. In order to automatically prove termination under SLG-resolution, our termination conditions:

- reason fully at the *clause level* (and not on "calls" as in [8]),
- give a *syntactical* condition on a level mapping (we will consider variants of semi-linear level mappings, see [3]) to be *finitely partitioning*.

In order to be able to reason fully at the clause level, most automatic termination analyses (like [7]) require that the level mapping is *rigid* on the call set. If a level mapping is rigid on the call set, the atoms in the call set can be considered as ground w.r.t. the level mapping (their value is invariant under substitutions). In this way, the problem of backpropagation of bindings in the calls is dealt with, and the conditions can be stated at the clause level. It turned out that a straightforward combination of the notions of rigid level mapping (as it appears in automatic termination analyses) and finitely partitioning level mapping (as it appears in termination analyses of tabled logic programs) offers some difficulties (being rigid and being finitely partitioning are in a way contradictory requirements). But, having some mode and type information at our disposal, we are able to solve these problems.
For *simply moded, well-moded* programs and queries (see for instance [1]), we show how to combine these two requirements on level mappings. Namely, for a simply moded, well-moded program P and set of queries S, a level mapping is rigid and finitely partitioning on $Call(P,S)$ if it measures all and only input positions in $Call(P,S)$. The condition to measure all and only input positions in a set of atoms is a syntactical condition (on the kind of level mappings we consider, namely variants of semi-linear level mappings [3]). In the case of quasi-termination, this leads to one of the main results in the paper, namely:

Proposition 1 *Let P be a simply moded, well-moded program and S be a set of simply moded, well-moded queries. Let $|.|$ be a level mapping which measures all and only the input positions in $Call(P,S)$, such that*

- *for any clause $H \leftarrow B_1, \ldots, B_n$ in P,*
- *for any atom B_i, $i \in \{1,..,n\}$,*
- *for any correct answer substitution ψ for $\leftarrow B_1, \ldots, B_{i-1}$, such that $(H\psi)^{In}$, $(B_1\psi)^{In}, \ldots, (B_i\psi)^{In} \in Call(P,S)^{In}$,*

$$|H\psi| \geq |B_i\psi|.$$

Then P quasi-terminates w.r.t. S.

Where the notation $A^{In} \in Call(P,S)^{In}$ stands for A is input-correct w.r.t. $Call(P,S)$, i.e. there is an atom $A' \in Call(P,S)$ such that the input positions of A and A' are equal (modulo variable renaming). In the case of LG-termination, we obtain a similar result. However, in this case, the conditions are not on the program P, but on the program P^a, which is obtained by applying the answer-transformation on P. We refer to [13] for more details.
In order to deal with non-ground input, we present also a solution for *simply moded, well-typed* [1] programs and queries, which generalizes our result for simply moded, well-moded programs and queries. Instead of trying to combine the rigidity requirement on the level mapping with the requirement to be finitely partitioning, we abandon the rigidity requirement and provide syntactical conditions on the program and set of queries which allow us to reason fully at the clause level in the termination condition. Namely, these syntactical conditions will ensure that the input arguments of the queries get never instantiated during a derivation. The conditions are based on concepts introduced in [1]. The results for quasi-termination and LG-termination are of a similar nature as Proposition 1.
Besides the work of [8], there are only relatively few works studying termination under tabled execution mechanism: in [11], in the context of well-moded programs, a sufficient condition is given for the bounded term-size property, which implies LG-termination; [9] provides another sufficient condition for quasi-termination in the context of functional programming.
Finally, the study of termination of *normal* logic programs under tabled execution mechanism is an interesting topic for future research.

References

[1] K. R. Apt and S. Etalle. On the unification free prolog programs. In A. Borzyszkowski and S. Sokolowski, editors, *Proceedings of the Conference on Mathematical Foundations of Computer Science (MFCS 93)*, pages 1–19. Lecture Notes in Computer Science, Springer-Verlag, 1993.

[2] R. Bol and L. Degerstedt. The underlying search for magic templates and tabulation. In D. S. Warren, editor, *Proceedings of the Tenth International Conference on Logic Programming*, pages 793–811, Budapest, Hungary, june 1993. The MIT Press.

[3] A. Bossi, N. Cocco, and M. Fabris. Norms on terms and their use in proving universal termination of a logic program. *Theoretical Computer Science*, 124(2):297–328, februari 1994.

[4] W. Chen and D. S. Warren. Tabled Evaluation with Delaying for General Logic Programs. *Journal of the ACM*, 43(1):20–74, january 1996.

[5] D. De Schreye and S. Decorte. Termination of logic programs: the never-ending story. *Journal of Logic Programming*, 19 & 20:199–260, may/july 1994.

[6] D. De Schreye, K. Verschaetse, and M. Bruynooghe. A framework for analysing the termination of definite logic programs with respect to call patterns. In *Proc. FGCS'92*, pages 481–488, ICOT Tokyo, 1992. ICOT.

[7] S. Decorte and D. De Schreye. Demand-driven and constraint-based automatic termination analysis for logic programs. In L. Naish, editor, *Proc. 14th International Conference on Logic Programming*, pages 78–92, Leuven, Belgium, july 1997.

[8] S. Decorte, D. De Schreye, M. Leuschel, B. Martens, and K. Sagonas. Termination analysis for tabled logic programs. In *Proceedings of LOPSTR'97: Logic Program Synthesis and Transformation*, Leuven, Belgium, july 1997.

[9] C. K. Holst. Finiteness Analysis. In John Hughes, editor, *Proceedings of the 5th ACM Conference on Functional Programming Languages and Computer Architecture (FPCA)*, number 523 in LNCS, pages 473–495. Springer-Verlag, august 1991.

[10] J.W. Lloyd. *Foundations of logic programming*. Springer-Verlag, 1987.

[11] L. Plümer. *Termination proofs for logic programs*. Number 446 in LNAI. Springer-Verlag, 1990.

[12] H. Tamaki and T. Sato. OLD Resolution with Tabulation. In *Proceedings ICLP'86*, Lecture Notes in Computer Science 225, pages 84–98. Springer Verlag, 1986.

[13] S. Verbaeten and D. De Schreye. Termination analysis of tabled logic programs using mode and type information. Technical Report 277, Department of Computer Science, K.U.Leuven, 1999.

[14] L. Vieille. Recursive query processing: the power of logic. *Theoretical computer science*, 69(1):1–53, 1989.