Well-Terminating, Input-Driven Logic Programs

Jan–Georg Smaus

University of Kent, Canterbury, CT2 7NF, United Kingdom,
phone +44/1227/827553, fax +44/1227/762811,
j.g.smaus@ukc.ac.uk

I will present what might be called the "second-most abstract" approach to termination of logic programs [6].

Termination is a problem in logic programming as much as in any other paradigm. For any paradigm, termination proofs rely on measuring the size of a state in the computation and showing that this size decreases during the computation. Measuring the size of a computation state usually involves measuring the size of the *data*, and this data originates from the *input* of the program.

To be more specific, let us look at functional programming and the well-known function $append :: (list, list) \rightarrow list$.[1] Consider the input $([1,2],[])$. Then the computation is as follows:

$$append([1,2],[]) \rightsquigarrow [1|append([2],[])] \rightsquigarrow [1,2|append([],[])] \rightsquigarrow [1,2].$$

At each step, the length of the list in the first argument of *append* decreases.

This introduction points us to a major if not the main problem for termination in logic programming: a priori, there is no notion of *input*. Corresponding to the above function would be a predicate *append* which has three arguments. Here are some computations for the predicate *append*:

$$\begin{array}{l}
append([1,2],[],Zs) \rightsquigarrow append([1],[],Zs') \rightsquigarrow append([],[],Zs'') \rightsquigarrow \epsilon. \\
append(Xs,Ys,[1,2]) \rightsquigarrow append(Xs',Ys,[2]) \rightsquigarrow append(Xs'',Ys,[]) \rightsquigarrow \epsilon. \\
append(Xs,Ys,Zs) \rightsquigarrow append(Xs',Ys,Zs') \rightsquigarrow append(Xs'',Ys,Zs'') \ldots
\end{array}$$

To prove termination of logic programs, it is useful to introduce a notion of input by declaring certain argument positions to be *input positions*.

To explain the effect of such a declaration, we must first explain what "evaluation strategy" means for logic programs. A computation state in a logic program is a *query*, which is a sequence of atoms. For example, $append([1,2],[],Zs)$ or $append([1,2],[],As), append(As,[],Bs)$ are queries. At each computation step, one atom in the query is selected and resolved with one clause in the program. The "evaluation strategy" (in logic programming terminology: the *selection rule*) states which atom is selected in each step.

[1] I avoid Currying to keep the notation similar to logic programming.

The most abstract approach is making no assumptions about the selection rule at all [2]. However, very few programs terminate for all selection rules. The most common selection rule is the rule that always selects the leftmost atom in a query.

My assumption about the selection rule is that an atom is selected for resolution only when it is instantiated to such a degree that its input arguments do not become instantiated any further in this resolution step. This is a minimal assumption if we want to give a natural meaning to "input".

We will see that many predicates terminate under such a weak assumption. Although implicitly present in previous work, my assumption has never been formulated in such an abstract way. Other authors have usually made stronger assumptions [1, 3, 4, 5].

The technical core of my work is establishing a dependency between the atoms of a query of the kind: the first atom cannot produce an infinite computation; the second atom cannot produce an infinite computation unless the first does; and so forth.

I believe that my approach is well suited for presentation at a general workshop because it will point out the similarities and differences between termination in logic programming and other paradigms.

References

[1] K. R. Apt and I. Luitjes. Verification of logic programs with delay declarations. In *Proceedings of AMAST'95*, LNCS, Berlin, 1995. Springer-Verlag. Invited Lecture.

[2] M. Bezem. Strong termination of logic programs. *Journal of Logic Programming*, 15(1 & 2):79–97, 1993.

[3] S. Lüttringhaus-Kappel. Control generation for logic programs. In D. S. Warren, editor, *Proceedings of the 10th International Conference on Logic Programming*, pages 478–495. MIT Press, 1993.

[4] E. Marchiori and F. Teusink. Proving termination of logic programs with delay declarations. In J. W. Lloyd, editor, *Proceedings of the 12th International Logic Programming Symposium*, pages 447–461. MIT Press, 1995.

[5] L. Naish. Coroutining and the construction of terminating logic programs. Technical Report 92/5, University of Melbourne, 1992.

[6] J.-G. Smaus. Well-terminating, input-driven logic programs. Technical Report 16-98, University of Kent at Canterbury, 1998.