Towards Scalable Partial Evaluation of Declarative Programs[⋆]

Germán Vidal

DSIC, Universidad Politécnica de Valencia, Spain
gvidal@dsic.upv.es

1 Introduction

Partial evaluation is a well-known technique for program specialization [4]. Essentially, given a program and *part* of its input data—the so-called *static* data—a partial evaluator returns a new, *residual* program which is specialized for the given data. The residual program is then used for performing the remaining computations—those that depend on the so-called *dynamic* data.

There are two main approaches to partial evaluation, depending on the way termination issues are addressed. On the one hand, *online* partial evaluators take decisions on the fly while the constructs of the source code are partially evaluated and the corresponding residual program is built. *Offline* partial evaluators, on the other hand, require a *binding-time analysis* (BTA) to be run before specialization, which annotates the source code to be specialized. Basically, every call of the source program is annotated as either unfold (to be executed by the partial evaluator) or memo (to be executed at run time, i.e., memoized), and every argument is annotated as static (known at specialization time) or dynamic (only definitely known at run time). Offline partial evaluators are usually faster but less accurate than online ones since the BTA phase is performed—and also termination issues are addressed—using an *approximation* of the static data.

There are several basic properties of a partial evaluator that can be addressed:

– *correctness:* is the specialized program equivalent to the original one for the considered static data?
– *accuracy:* is the residual program a good specialization of the original program for the static data? is it fast enough compared to a hand-written specialization?
– *efficiency:* is the partial evaluator fast? does it scale up well to large source programs?
– *predictability:* is it possible to determine the achievable run time speedup *before* partial evaluation starts?

Here, we are mainly concerned with efficiency issues in offline partial evaluation.

[⋆] This work has been partially supported by the Spanish *Ministerio de Ciencia e Innovación* under grant TIN2008-06622-C03-02, by *Generalitat Valenciana* under grant ACOMP/2009/017, and by *UPV* (programs PAID-05-08 and PAID-06-08).

2 Accuracy vs Efficiency

Clearly, there is a trade-off between accuracy and efficiency. For instance, some accurate BTAs are rather inefficient because the termination analysis and the algorithm for propagating static information should be interleaved, so that every time a call is annotated as memo, the termination analysis has to be re-executed to take into account that some bindings will not be propagated anymore.

Consider, for instance, the following logic programming clause:

$$p(s(X), s(Y)) \leftarrow q(X, Z),\ p(Z, Y).$$

with variable X static[1] and variable Y dynamic. A traditional BTA initially marks every predicate call as unfold and proceeds as follows:

- First, it runs a procedure for propagating static information (i.e., a sort of groundness analysis). Let us assume that, in every successful computation for $q(X, Z)$ with X ground, variable Z becomes ground too. Therefore, by assuming a fixed left-to-right selection strategy, this procedure may conclude that variable Z in $p(Z, Y)$ is static too.
- Now, a termination analysis—that takes into account which variables are marked as static—is used to infer annotations for predicate calls. We also consider a fixed left-to-right selection strategy (i.e., a so called left-termination analysis). Let us now assume that this analysis is not able to ensure termination for predicate q and, thus, it is now marked as memo.
- Since q is annotated as memo, the call to q will not be unfolded at partial evaluation time. Therefore, we should run again the procedure for propagating static information, now assuming that q is not unfolded. Clearly, this will imply that we cannot ensure that variable Z is static in $p(Z, Y)$ anymore. Therefore, since the static/dynamic annotations have changed, the termination analysis should also be run again, and so forth. This iterative process is computationally very expensive, so it does seem a good candidate as a basis for designing a scalable partial evaluator.

Our recent work [1, 6–9] shows that this drawback can be overcome by using instead a *strong* termination analysis [3], i.e., an analysis that considers termination for every possible selection or evaluation strategy. In this case, both tasks—termination analysis and propagation of static information—can be kept independent, so that the termination analysis is done once and for all before the propagation phase, resulting in major efficiency improvements over previous approaches.

For instance, given the previous clause, $p(s(X), s(Y)) \leftarrow q(X, Z),\ p(Z, Y).$, the BTA would now proceed as follows:

- First, the strong termination analysis is executed. For this purpose, we have adapted the size-change analysis originally introduced for functional programs by Lee, Jones and Ben-Amram [5]. Roughly speaking, this analysis

[1] For simplicity, here we assume that a static variable is ground.

traces the size changes of arguments when going from one call to another by means of so called *size-change graphs*.
The strong termination analysis for logic programs was introduced in [9] and later refined and extended in [7, 6]. Basically, for every program clause $H \leftarrow B_1, \ldots, B_n$, the analysis constructs n size-change graphs, each of them stating the relation between the sizes of the arguments of H and the sizes of the arguments of every atom B_i in the body. For instance, for the above clause, the analysis constructs two size-change graphs, one that relates the sizes of $(s(X), s(Y))$ and (X, Z), and another one that relates the sizes of $(s(X), s(Y))$ and (Z, Y).[2] The output of the analysis is then a set of conditions for the termination of every predicate call that depends on which variables are marked as static.
– Then, in a second step, the BTA applies a standard procedure for the propagation of static/dynamic information that uses the output of the strong termination analysis to infer the right annotations for both predicate calls and their arguments. The details of this procedure can be found in [7].

As expected, the accuracy of the resulting scheme is not comparable to that of previous approaches, but can nevertheless be improved in a number of ways:

– Firstly, the information gathered from a left-termination analysis (which would be run only once) can still be used to improve the accuracy in those cases where the order of evaluation is partially known. For instance, for the clause above, if we know that $q(X, Z)$ always terminates with a left-to-right selection rule (e.g., because q is not recursive), then one can safely mark q as unfold and take it into account in the size-change analysis to propagate some additional static information to the calls that occur to its right ($p(Z, Y)$ in the example, so that the size relation between $s(X)$ and Z can again be inferred, as in the traditional approach).
– Secondly, we could allow the user to provide manual annotations to improve the accuracy in some cases. We have experimentally checked that even for large programs, a few annotations suffice to get an optimal result [7].
– We may also replace some memo and/or dynamic annotations by online, a new annotation that delays the corresponding decisions to partial evaluation time. We note, however, that one should be very careful with these annotations since they may involve expensive computations at partial evaluation time (i.e., some heuristics is needed to decide when replacing memo/dynamic by online might be critical to get a good specialization).

3 Concluding Remarks

Although there is ample room for improving accuracy, we consider our approach a promising framework for developing scalable partial evaluators for declarative

[2] Observe that, in contrast to the traditional approach that uses a left-termination analysis, now we cannot infer any size relation between $s(X)$ and Z (and, as a consequence, the termination of p cannot be proved). Some possibilities to improve this situation are mentioned in the following.

programs. A promising line of research is based on the use of SAT solving techniques to improve the accuracy of the BTA while still keeping its scalability. For instance, one could extend the SAT-based approach to size-change analysis of [2] in order to compute unfold/memo annotations in polynomial time (despite the fact that a left-termination analysis is considered).

Finally, we note that the underlying techniques are essentially the same no matter the considered declarative programming language. Actually, we have applied similar principles to the partial evaluation of both functional (logic) programs [8, 1] and logic programs [9, 7, 6].

Acknowledgements. I would like to thank the participants of LOPSTR 2009 for their useful feedback. I would also like to thank Michael Codish for suggesting the use of SAT-based techniques to improve the accuracy of the BTA.

References

1. G. Arroyo, J.G. Ramos, J. Silva, and G. Vidal. Improving Offline Narrowing-Driven Partial Evaluation using Size-Change Graphs. In *Logic-Based Program Synthesis and Transformation. Revised and Selected Papers from LOPSTR'06*, pages 60–76. Springer LNCS 4407, 2007.
2. A. Ben-Amram and M. Codish. A SAT-Based Approach to Size Change Termination with Global Ranking Functions. In C.R. Ramakrishnan and Jakob Rehof, editors, *Proc. of the 14th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'07)*, pages 46–55. Springer LNCS 5028, 2008.
3. M. Bezem. Strong Termination of Logic Programs. *Journal of Logic Programming*, 15(1&2):79–97, 1993.
4. N.D. Jones, C.K. Gomard, and P. Sestoft. *Partial Evaluation and Automatic Program Generation.* Prentice-Hall, Englewood Cliffs, NJ, 1993.
5. C.S. Lee, N.D. Jones, and A.M. Ben-Amram. The Size-Change Principle for Program Termination. *SIGPLAN Notices (Proc. of POPL'01)*, 28:81–92, 2001.
6. M. Leuschel, S. Tamarit, and G. Vidal. Fast and Accurate Size-Change Strong Termination Analysis with an Application to Partial Evaluation. In S. Escobar, editor, *Proc. of the 18th Int'l Workshop on Functional and (Constraint) Logic Programming (WFLP'09)*, pages 111–127. Springer LNCS 5979, 2009.
7. M. Leuschel and G. Vidal. Fast Offline Partial Evaluation of Large Logic Programs. In *Logic-based Program Synthesis and Transformation (revised and selected papers from LOPSTR'08)*, pages 119–134. Springer LNCS 5438, 2009.
8. J.G. Ramos, J. Silva, and G. Vidal. Fast Narrowing-Driven Partial Evaluation for Inductively Sequential Systems. In *Proc. of the 10th ACM SIGPLAN Int'l Conf. on Functional Programming (ICFP'05)*, pages 228–239. ACM Press, 2005.
9. G. Vidal. Quasi-Terminating Logic Programs for Ensuring the Termination of Partial Evaluation. In *Proc. of the ACM SIGPLAN 2007 Workshop on Partial Evaluation and Program Manipulation (PEPM'07)*, pages 51–60. ACM Press, 2007.