Extending partial deduction to tabled execution: some results and open issues

Konstantinos Sagonas and Michael Leuschel

ACM Computing Surveys, Vol. 30, No. 3es, September 1998

Article 16

Extending Partial Deduction to Tabled Execution: Some Results and Open Issues

KONSTANTINOS SAGONAS and MICHAEL LEUSCHEL
Katholieke Universiteit Leuven

1. TABLED-BASED EXECUTION OF LOGIC PROGRAMS

Resolution methods based on *tabling* [Tamaki and Sato 1986; Warren 1992; Chen and Warren 1996] evaluate programs by recording subgoals (referred to as *calls*) and their provable instances (referred to as *answers*) in a global store called a *table*. Predicates are designated *a priori* as either *tabled* or *nontabled*, and execution proceeds as follows. For nontabled predicates the call is resolved against program clauses. For tabled predicates, if the call is new to the evaluation, it is entered in the table and Prolog-style program clause resolution is used to compute its *set* of answers which are also recorded in the table. If, on the other hand, the call is already present in the table, then it is resolved against its recorded answers. By using answer tables for resolving subsequent invocations of the same call, tabled execution prevents many cases of infinite looping which normally occur in Prolog-style SLD evaluation and avoids performing redundant subcomputations. As a result, tabling significantly extends the range of applications of logic programming (LP). Moreover, because of the better termination and complexity properties of tabled execution, tabled-based systems such as XSB [Sagonas et al. 1994] make a good first step in fulfilling the promise of declarativeness: allow a programmer to write problem specifications, and then use meaning-preserving transformations or use tabled execution selectively to turn these specifications into effective programs.

2. PARTIAL DEDUCTION AND TABLING

So, tabling makes more specifications executable. Obviously, it is of great interest, that these specifications are executed efficiently. *Partial evaluation* [Jones et al. 1993; Jones 1996; Danvy et al. 1996] aims at exactly this: improve the performance by carrying out part of the program's execution (for input of a particular form) at

Authors' address: Departement Computerwetenschappen, K.U.Leuven, Celestijnenlaan 200A, B-3001 Heverlee, Belgium. E-mail : {kostis,michael}@cs.kuleuven.ac.be
Authors are Post-doctoral Fellows of the K.U. Leuven Research Council and the Fund for Scientific Research — Flanders Belgium, respectively.

compile-time, thus specialise the program for this input and leave less computation for run-time. In the LP context, where computation is logical deduction, partial evaluation is logically called *partial deduction* (PD) [Lloyd and Shepherdson 1991; Komorowski 1992]. It is natural then, to try to extend PD to tabling, i.e., to improve efficiency of tabled execution through (fully) automatic program transformation [Pettorossi and Proietti 1994] or specialisation [Gallagher 1993].

2.1 Partial Deduction of Tabled Execution: Some Results

Unfortunately, no methodology for the partial deduction of tabled logic programs currently exists. All techniques stay within the context of Prolog-style untabled execution. Although one may expect that results established in the "classic" LP setting more or less carry over, this turned out to be far from obviously true as the differences between the execution models are significant. Therefore, providing such a methodology is still a challenging research problem.

In [Leuschel et al. 1997], we provided a first investigation of the specialisation and transformation of tabled logic programs through *unfolding*. We mainly concentrated on issues related to the safety of the unfolding rule since it the major transformation rule applied by PD. We showed that — surprisingly — unfolding, even determinate, can worsen the termination behaviour in the context of tabling. To reason about the safety of unfolding, we also developed in [Decorte et al. 1997] a framework that captures their (*quasi*-) *termination* (under the left-to-right selection rule) and showed that termination of tabled execution is *not* closed under substitution. Using this framework, we defined applicability conditions that ensure the intended equivalence property between the original and the transformed program. In summary, results regarding unfolding in the context of tabled execution are:

—As in SLD, unfolding preserves the set of computed answer substitutions (c.a.s.).
—Left-most unfolding or unfolding without any left-propagation of bindings preserves termination of tabled logic programs.
—Even though left-propagation of bindings through unfolding can worsen the termination characteristics of tabled programs, left-propagation of *grounding* substitutions is safe w.r.t. termination.

Regarding *folding*, we conjecture that contrary to Prolog-style evaluation, in tabled execution folding is always safe w.r.t. termination (but may change the set of c.a.s.).

2.2 Tabled Execution for Partial Deduction

Note that tabled execution has a lot to offer to partial deduction. Firstly, it can provide (part of) the *control* necessary to ensure termination. In an *off-line* setting tabling can even be used to obtain a *binding time analysis* by executing an abstract version of the program to be specialised as shown recently in [Bruynooghe et al. 1998]. Secondly, by creating a partial deduction methodology based upon tabled resolution, one can achieve an elegant *integration of partial deduction and abstract interpretation*: program clause resolution performs top-down unfolding steps while answer clause resolution performs bottom-up abstract interpretation steps. Prospects of such an integration are discussed in e.g. [Leuschel and De Schreye 1996; Jones 1997; Puebla et al. 1997]. A technique to use tabling for controlling

program transformation can be found in [Boulanger and Bruynooghe 1993].

Moreover, a particular instance of tabled resolution, *SLG resolution* [Chen and Warren 1996], can be viewed as a partial deduction procedure where given a program P, a query Q is transformed step by step into a *residual program* P' consisting of all (conditional) answers of Q plus all answer clauses upon which these answers (conditionally) depend upon. P' is such that all dependencies between atoms in its clauses have been simplified w.r.t. the well-founded partial model of P. Thus, a direct way of performing partial deduction of LPs is to enforce the truth value *undefined* to atoms that cannot be partially evaluated and evaluate the rest of the atoms under the well-founded semantics.

3. SKETCH OF SOME OPEN ISSUES

Several open issues in the PD of tabled LPs exist and most of them have to do with guaranteeing that PD will result in no performance loss. An extended discussion can be found in [Leuschel et al. 1997], but we identify some of them here. To ensure the independence condition of [Lloyd and Shepherdson 1991] and allow unlimited polyvariance without the use of abstraction, most specialisers use a technique called *renaming*. In SLD execution, renaming might increase the code size, but it is always beneficial in terms of the number of primitive operations executed. As tabling recognizes and avoids redundant subcomputations based on mainly syntactic identity of atoms, through renaming this possibility can be lost and the specialised program might thus be less efficient than the original one. Similar difficulties can arise when performing *conjunctive partial deduction* [De Schreye et al. 1997] (as well as *tupling* or *deforestation*). Ensuring that PD will cause no loss in performance is a general issue, but because tabled execution adds a global store where results are materialised and can then be efficiently retrieved without recomputation, the problem appears particularly severe in this context.

Finally, an independent but perhaps more promising and challenging problem is to obtain automatic partial deducers that understand and fully exploit the properties of tabled execution. Tabled execution transforms a set of program clauses into a *set* of answers; clauses or derivation paths that produce answers *identical* to those obtained through other clauses are obviously redundant, and the corresponding derivation paths are effectively treated as *fail* branches. So can be some branches of the trees constructed by the PD of tabled programs. In order to obtain an "optimal" residual program, a partial deducer (for tabling) needs to be able to recognize this dead-wood and remove it from (the forest and) the residual program. For this it has to understand termination of tabled resolution and detect *completion* of tabled goals (i.e., when all answers to the goals have been produced) to know that it can already *completely* evaluate parts of the computation at compile-time and transform these computations to just (answer) table lookups. Tabling might thus also provide a general mechanism to perform powerful optimisations — outside the reach of current partial deduction techniques — which view the c.a.s. of logic programs as a set rather than as a sequence.

ACKNOWLEDGMENTS

We thank Bern Martens and David S. Warren for interesting ideas and discussions.

REFERENCES

Boulanger, D. and Bruynooghe, M. 1993. Deriving Fold/Unfold Transformations of Logic Programs Using Extended OLDT-Based Abstract Interpretation. *Journal of Symbolic Computation 15*, 5 & 6, 495–521.

Bruynooghe, M., Leuschel, M., and Sagonas, K. 1998. A Polyvariant Binding Time Analysis for Off-line Partial Deduction. In C. Hankin Ed., *Proceedings of the European Symposium on Programming*, LNCS (Lisbon, Portugal, April 1998). Springer-Verlag.

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

Danvy, O., Glück, R., and Thiemann, P. Eds. 1996. *Proceedings of the 1996 Dagstuhl Seminar on Partial Evaluation*, Number 1110 in LNCS (Schloß Dagstuhl, February 1996). Springer-Verlag.

De Schreye, D., Glück, R., Jørgensen, J., Leuschel, M., Martens, B., and Sørensen, M. H. 1997. Conjunctive Partial Deduction: Foundations, Control, Algorithms and Experiments. Submitted for Publication.

Decorte, S., De Schreye, D., Leuschel, M., Martens, B., and Sagonas, K. 1997. Termination Analysis for Tabled Logic Programming. In N. Fuchs Ed., *Proceedings of LOPSTR'97: Logic Program Synthesis and Transformation*, LNCS (Leuven, Belgium, July 1997). Springer-Verlag.

Gallagher, J. 1993. Tutorial on Specialisation of Logic Programs. In *ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation* (Copenhagen, Denmark, June 1993), pp. 88–98. ACM Press.

Jones, N. D. 1996. An Introduction to Partial Evaluation. *ACM Computing Surveys 28*, 3 (September), 480–503.

Jones, N. D. 1997. Combining Abstract Interpretation and Partial Evaluation (brief overview). In P. Van Henteryck Ed., *Proceedings of the Fourth International Symposium on Static Analysis*, Number 1302 in LNCS (Paris, France, September 1997), pp. 396–405.

Jones, N. D., Gomard, C. K., and Sestoft, P. 1993. *Partial Evaluation and Automatic Program Generation*. Prentice Hall International Series in Computer Science.

Komorowski, J. 1992. An Introduction to Partial Deduction. In A. Pettorossi Ed., *Proceedings of META-92: Meta-Programming in Logic*, Number 649 in LNCS (Uppsala, Sweden, June 1992), pp. 49–69. Springer-Verlag.

Leuschel, M. and De Schreye, D. 1996. Logic Program Specialisation: How To Be More Specific. In H. Kuchen and S. D. Swierstra Eds., *Proceedings of the International Symposium on Programming Languages, Implementations, Logics and Programs (PLILP'96)*, Number 1140 in LNCS (Aachen, Germany, September 1996), pp. 137–151. Springer-Verlag.

Leuschel, M., Martens, B., and Sagonas, K. 1997. Preserving Termination of Tabled Logic Programs While Unfolding. In N. Fuchs Ed., *Proceedings of LOPSTR'97: Logic Program Synthesis and Transformation*, LNCS (Leuven, Belgium, July 1997). Springer.

Lloyd, J. W. and Shepherdson, J. C. 1991. Partial Evaluation in Logic Programming. *Journal of Logic Programming 11*, 3 & 4 (October/November), 217–242.

Pettorossi, A. and Proietti, M. 1994. Transformation of Logic Programs: Foundations and Techniques. *Journal of Logic Programming 19 & 20*, 261–320.

Puebla, G., Gallagher, J., and Hermenegildo, M. 1997. Towards Integrating Partial Evaluation in a Specialisation Framework based on Generic Abstract Interpretation. In M. Leuschel Ed., *Proceedings of the International Workshop on Specialisation of Declarative Languages and its Applications* (Port Jefferson, N.Y., October 1997), pp. 29–38.

Sagonas, K., Swift, T., and Warren, D. S. 1994. XSB as an Efficient Deductive Database Engine. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data* (Minneapolis, Minnesota, May 1994), pp. 442–453. ACM Press.

Tamaki, H. and Sato, T. 1986. OLD Resolution with Tabulation. In E. Shapiro Ed., *Proceedings of the Third International Conference on Logic Programming*, Number 225 in LNCS (London, July 1986), pp. 84–98. Springer-Verlag.

Warren, D. S. 1992. Memoing for Logic Programs. *Commun. ACM 35*, 3 (March), 93–111.