Preliminary Report on Polynomial-Time Program Staging by Partial Evaluation

Robert Glück

DIKU, Dept. of Computer Science, University of Copenhagen, Denmark

Summary

Maximally-polyvariant partial evaluation is a strategy for program specialization that propagates static values as accurate as possible [4]. The increased accuracy allows a maximally-polyvariant partial evaluator to achieve, among others, the Bulyonkov effect [3], that is, constant folding while specializing an interpreter.

The polyvariant handling of return values avoids the monovariant return approximation of conventional partial evaluators [14], in which the result of a call is dynamic if one of its arguments is dynamic. But multiple return values are the "complexity generators" of maximally polyvariant partial evaluation because multiple continuations need to be explored after a call, and this degree of branching is not bound by a program-dependent constant, but depends on the initial static values. In an offline partial evaluator, a recursive call has at most one return value that is either static or dynamic.

A conventional realization of a maximally-polyvariant partial evaluator can take exponential time for specializing programs. The online partial evaluator [10] achieves the same precision in time polynomial in the number of partial-evaluation configurations. This is a significant improvement because no fast algorithm was known for maximally-polyvariant specialization. The solution involves applying a polynomial-time simulation of nondeterministic pushdown automata [11].

For an important class of quasi-deterministic specialization problems the partial evaluator takes linear time, which includes Futamura's challenge [8]: (1) the linear-time specialization of a naive string matcher into (2) a linear-time matcher. This is remarkable because both parts of Futamura's challenge are solved. The second part was solved in different ways by several partial evaluators, including generalized partial computation by employing a theorem prover [7], perfect supercompilation based on unification-based driving [13], and offline partial evaluation after binding-time improvement of a naive matcher [5]. The first part remained unsolved until this study, though it had been pointed out [1] that manual binding-time improvement of a naive matcher could expose static functions to the caching of a hypothetical memoizing partial evaluator.

Previously, it was unknown that the *KMP test* [15] could be passed by a partial evaluator without sophisticated binding-time improvements. Known solutions to the KMP test include Futamura's generalized partial computation utilizing a theorem prover [8], Turchin's supercompilation with unification-based driving [13], and various binding-time-improved matchers [1, 5].

As a result, a class of specialization problems can now be solved faster than before with high precision, which may enable faster Ershov's generating extensions [6,9,12], *e.g.*, for a class similar to Bulyonkov's analyzer programs [2]. This is significant because *super-linear program staging* by partial evaluation becomes possible: the time to run the partial evaluator *and* its residual program is linear in the input, while the original program is not, as for the naive matcher.

This approach provides fresh insights into fast partial evaluation and accurate program staging. This contribution summarizes [10, 11], and examines applications to program staging, and the relation to recursive pushdown systems.

References

1. M. S. Ager, O. Danvy, H. K. Rohde. Fast partial evaluation of pattern matching in strings. *ACM TOPLAS*, 28(4):696–714, 2006.
2. M. A. Bulyonkov. Polyvariant mixed computation for analyzer programs. *Acta Informatica*, 21(5):473–484, 1984.
3. M. A. Bulyonkov. Extracting polyvariant binding time analysis from polyvariant specializer. In *Symposium on Partial Evaluation and Semantics-Based Program Manipulation*, 59–65. ACM Press, 1993.
4. N. H. Christensen, R. Glück. Offline partial evaluation can be as accurate as online partial evaluation. *ACM TOPLAS*, 26(1):191–220, 2004.
5. C. Consel, O. Danvy. Partial evaluation of pattern matching in strings. *Information Processing Letters*, 30(2):79–86, 1989.
6. A. P. Ershov. On the partial computation principle. *Information Processing Letters*, 6(2):38–41, 1977.
7. Y. Futamura, Z. Konishi, R. Glück. Program transformation system based on generalized partial computation. *New Generation Computing*, 20(1):75–99, 2002.
8. Y. Futamura, K. Nogi. Generalized partial computation. In D. Bjørner, A. P. Ershov, N. D. Jones (eds.), *Partial Evaluation and Mixed Computation*, 133–151. North-Holland, 1988.
9. R. Glück. A self-applicable online partial evaluator for recursive flowchart languages. *Software – Practice and Experience*, 42(6):649–673, 2012.
10. R. Glück. Maximally-polyvariant partial evaluation in polynomial time. In M. Mazzara, A. Voronkov (eds.), *Perspectives of System Informatics. Proceedings*, LNCS 9609. Springer-Verlag, 2016.
11. R. Glück. A practical simulation result for two-way pushdown automata. In Y.-S. Han, K. Salomaa (eds.), *Implementation and Application of Automata. Proceedings*, LNCS 9705. Springer-Verlag, 2016.
12. R. Glück, J. Jørgensen. Generating transformers for deforestation and supercompilation. In B. Le Charlier (ed.), *Static Analysis. Proceedings*, LNCS 864, 432–448. Springer-Verlag, 1994.
13. R. Glück, A. V. Klimov. Occam's razor in metacomputation: the notion of a perfect process tree. In P. Cousot, et al. (eds.), *Static Analysis. Proceedings*, LNCS 724, 112–123. Springer-Verlag, 1993.
14. N. D. Jones, C. K. Gomard, P. Sestoft. *Partial Evaluation and Automatic Program Generation.* Prentice-Hall, 1993.
15. M. H. Sørensen, R. Glück, N. D. Jones. A positive supercompiler. *Journal of Functional Programming*, 6(6):811–838, 1996.