

Background Literature and Designs

Michael Buch

December 10, 2018

1 About this document

This is a collection of references and summaries to research in the field of meta-programming/supercompilation/partial evaluation

2 Examples

Examples drawn from paper on collapsing towers [1]:

- Regular expression matcher <- Evaluator <- Virtual Machine
 - Generate low-level VM code for a matcher specialized to one regex (through arbitrary number of intermediate interpreters)
- Modified evaluator <- Evaluator <- Virtual Machine
 - Modified for tracing/counting calls/be in CPS
 - Under modified semantics "interpreters become program transformers". E.g. CPS interpreter becomes CPS transformer

3 Methodologies

- Stage polymorphism [2]: "abstract over staging decisions" i.e. single program generator can produce code that is specialized in many different ways (instance of the Fourth Futamura Projection? [3])
- Multi-level base evaluator written in $\lambda \uparrow \downarrow$: supports staging operators (**polymorphic Lift**)
- Modify other interpreters: make them **stage polymorphic**, i.e. commands either evaluate code (like an interpreter) or generate code (like a translator)
- Stage only user-most interpreter: *wire tower* such that the **staging commands in L_n are interpreted directly in terms of staging commands in L_0** i.e. staging commands pass through all other layers handing down commands to layers below without performing any staging commands
- Non-reflective method: meta-circular evaluator **Pink** \Rightarrow collapse arbitrary levels of "self-interpretation"
- $\lambda \uparrow \downarrow$ features:

- *run residual code*
- binding-time/stage polymorphism [4]
- preserves execution order of future-stage expressions
- does not require type system or static analysis
 - * TDPE [5]: **polymorphic Lift** operator turns static values into dynamic (future-stage) expressions

3.1 Towers of Interpreters Project Overview

3.1.1 Scala

- `base.scala`: implements definitional interpreter for $\lambda \uparrow \downarrow$

References

- [1] N. Amin and T. Rompf, “Collapsing towers of interpreters,” *Proceedings of the ACM on Programming Languages*, vol. 2, no. POPL, p. 52, 2017.
- [2] G. Ofenbeck, T. Rompf, and M. Püschel, “Staging for generic programming in space and time,” in *ACM SIGPLAN Notices*, vol. 52, no. 12. ACM, 2017, pp. 15–28.
- [3] R. Glück, “Is there a fourth futamura projection?” in *Proceedings of the 2009 ACM SIGPLAN workshop on Partial evaluation and program manipulation*. ACM, 2009, pp. 51–60.
- [4] F. Henglein and C. Mossin, “Polymorphic binding-time analysis,” in *European Symposium on Programming*. Springer, 1994, pp. 287–301.
- [5] O. Danvy, “Type-directed partial evaluation,” in *Partial Evaluation*. Springer, 1999, pp. 367–411.