We Need Real Tools for Generating Type Inferencers

William E. Byrd
University of Utah
Will.Byrd@utah.edu

Nada Amin
EPFL
nada.amin@epfl.ch

For decades programmers have had access to production-quality tools for generating lexers and parsers from high-level declarative specifications. Recent work on nano-pass compiler frameworks [3] makes it possible to develop a many-pass compiler using a grammar-based domain-specific language. Yet we still do not have mature, flexible tools for generating efficient type checkers and type inferencers from a high-level description (ideally, directly from the typing judgements).

Although researchers use logic programming, term writing, and proof assistants to express type systems in a declarative fashion, in practice developers—including computer science researchers—write production type inferencers and type checkers by hand using general-purpose programming languages. This point was driven home for one of us during development of the compiler for Harlan [2] (a language for high-level GPGPU programming). After defining the type system for Harlan's region-based memory management system, we wanted to generate a type inferencer from the typing judgements. We knew of no tool to do this, so our team developed a type inferencer in the pure logic programming language miniKanren [1]. Although the resulting code matched the typing judgements, the inferencer was slow, and the error messages non-existent. However, this led us to start thinking about a special purpose tool, perhaps based on logic programming, logic meta-interpreters, or term-rewriting, specifically for generating usable type inferencers from the logical inference rules.

In our talk we will outline the challenges of building such a tool, and the opportunities the tool might expose.

References

[1] Friedman, D. P., Byrd, W. E., and Kiselyov, O. *The Reasoned Schemer*. The MIT Press, Cambridge, MA, 2005.

[2] Holk, E., Byrd, W., Mahajan, N., Willock, J., Chauhan, A., and Lumsdaine, A. Declarative programming for GPUs. In *Proceedings of the 2011 International Conference on Parallel Computing (ParCo 2011)* (sep 2011).

[3] Keep, A. W., and Dybvig, R. K_1 A nanopass framework for commercial compiler development. In *Proceedings of the 18th ACM SIGPLAN International Conference on Functional Programming* (New York, NY, USA, 2013), ICFP '13, ACM, pp. 343–350.