A Statement Describing My Research Agenda

Alex Hubers

Writing code can be tedious; every coder knows this. In particular, we are told to "not repeat ourselves", a maxim most strive to follow. Yet somehow we find ourselves doing just that. Why? One reason, as observed by Wadler [8], is that data types are notoriously difficult to extend in both their *cases* and *behaviors*. We may fix one, and let the other freely extend, but to simultaneously extend both has proven challenging for the language designer in both object-oriented and functional programming languages. This is infamously known as the *expression problem.*

The expression problem frequently surfaces in two dual scenarios of compiler design: that of extending *abstract syntax trees* (ASTs) with new cases and, dually, in writing modular nano-passes over ASTs (e.g., a pass that desugars syntax). This can have serious implications when blown to scale. Consider Xavier Leroy's compcert compiler [4], which translates from C source code through eight intermediate languages—many of which have substantial syntactic overlap—on its way to assembly. This project was estimated to have take three person-years, and involved extreme cases of "copy-and-paste" code reuse.

Many solutions to the *expression problem* resort to esoteric or obscure encodings—such as the *visitor pattern* or *two level types* [7]—or imprecise and fragile tools, like the C preprocessor, to achieve reusability in practice. These solutions are in wont of an ergonomic alternative.

My research offers such an alternative by achieving code reuse with the aid of

the type system itself. In particular, I (and co-authors) use *row types* [9, 6, 10] to record the structure of records and variants as typed information. By encoding this data in a qualified type system (i.e., that of Jones [2]), we have also been able to safely type not only long sought-after operations on records, such as record- and row-concatenation, but to abstract differing notions of label overlap resolution into what we call *row theories* [5]. By parameterizing our calculi by row theories, we are able to capture the behavior of many extant row type systems, such as: *overwriting* rows [9]; *non-overlapping* rows [6]; and *scoped rows* [3].

Our work has appeared in two premier programming languages theory conferences—POPL (Morris and McKinna [5]) and ICFP (Hubers and Morris [1]). Although we have shown row type systems to be theoretically promising, they have had limited practical adoption. My first research thrust aims to remedy this. In particular, while there exists much work modeling structural inheritance of objects (a là object-oriented programming) by way of record calculi, very little attention has been placed on the precise dual: extensibility of variants. Further, there exists no known account of *variant recursion*. This omission in the literature is perplexing, as recursion is mandatory to express all inductive data, which constitutes most of the interesting types a programmer may use (e.g. trees and lists). So, we will first fill this gap.

My second research thrust will be to show that our account of recursive variants in fact offers a straightforward method to extend data types in both their cases and behaviors. This can be demonstrated by elaboration of Haskell-like data types to extensible variants. Then we may ask: what does *extensible* Haskell look like? Can the extensibility we demonstrate be reproduced in GHC, Haskell's most prominent compiler? The result will be a *first-class* treatment of extensiblity, which, in line with our thesis, will lead to more code reuse and less code abuse.

References

[1] Alex Hubers and J. Garrett Morris. Generic programming with extensible data types; or, making ad hoc extensible data types less ad hoc. ICFP 23, 2023.

[2] Mark P. Jones. A theory of qualified types. In Bernd K. Bruckner, editor, *Proceedings of the 4th European symposium on programming*, volume 582 of *ESOP'92*. Springer-Verlag, Rennes, France, 1992.

[3] Daan Leijen. Extensible records with scoped labels. In *Revised Selected Papers from the Sixth Symposium on Trends in Functional Programming, TFP 2005, Tallinn, Estonia, 23-24 September 2005*, pages 179–194, 2005.

[4] Xavier Leroy. Formal verification of a realistic compiler. *Commun. ACM*, 52(7):107–115, 2009. doi: 10.1145/1538788.1538814. URL `https://doi.org/10.1145/1538788.1538814`.

[5] J. Garrett Morris and James McKinna. Abstracting extensible data types: or, rows by any other name. *Proc. ACM Program. Lang.*, 3(POPL):12:1–12:28, 2019. doi: 10.1145/3290325. URL `https://www.youtube.com/watch?v=5rDfyB2udKA`.

[6] Didier Rémy. Typechecking records and variants in a natural extension of ML. In *Conference Record of the Sixteenth Annual ACM Symposium on Principles of Programming Languages, Austin, Texas, USA, January 11-13, 1989*, pages 77–88. ACM Press, 1989.

[7] Tim Sheard and Emir Pasalic. Two-level types and parameterized modules. *Journal of Functional Programming*, 14(5):547–587, 2004. doi: 10.1017/S095679680300488X.

[8] Philip Wadler. The expression problem, 1998. URL `https://homepages.inf.ed.ac.uk/wadler/papers/expression/expression.txt`.

[9] Mitchell Wand. Complete type inference for simple objects. In *Proceedings of the Symposium on Logic in Computer Science (LICS '87), Ithaca, New York, USA, June 22-25, 1987*, pages 37–44. IEEE Computer Society, 1987.

[10] Mitchell Wand. Type inference for record concatenation and multiple inheritance. *Inf. Comput.*, 93(1):1–15, 1991.