Logical Relations for Algebraic Effects

Dariusz Biernacki Maciej Piróg Piotr Polesiuk
Filip Sieczkowski

Institute of Computer Science
University of Wrocław

Algebraic effects, introduced 15 years ago by Plotkin and Power [5], as well as algebraic effect handlers of Plotkin and Pretnar [6], are making a breakthrough in the programming language design. The reason for this is the promise of effectful programming that is modular at the same time, driven by algebraic effects' inherent separation between effect interfaces and their implementations. Thus, a programmer can express the effectful programs by only referring to the interface, while multiple handlers can give diverse interpretations to the operations provided by the interface. In the last few years, the languages that employ this technique to allow effectful computations are gaining in complexity, routinely utilising type-and-effect systems of varying sophistication (e.g., Bauer and Pretnar's *Eff* [7], Hillerström and Lindley's extensions to *Links* [2], Lindley *et al.*'s *Frank* [4] or Leijen's *Koka* [3]). However, novel reasoning techniques have not yet followed these developments in language design.

In this talk, we would like to present a logical relation for reasoning about contextual equivalence and contextual approximation of programs in a language with algebraic effect handlers and row-based polymorphic type-and-effect system. We start with a state-of-the-art programming language with algebraic effects, Leijen's *Koka*, identify the challenging subset of the calculus and use the standard technique of biorthogonality to build a step-indexed relational interpretation (cf., for instance, Benton and Hur [1]) that allows us to reason about approximation and equivalence, as well as show soundness of the type system. We are then able to use the relation to show equivalence of various programs (including equivalence of pure and effectful code), as well as some interesting type-directed equivalences.

The challenge in building a biorthogonal relational interpretation of a language with algebraic effects is twofold. Firstly, in such a language values are not the only sensible irreducible expressions: the second kind of normal forms are effect operations applied to a value. Thus, to check that two evaluation contexts are related it would no longer suffice to observe their behaviour when plugged with related values. The challenge, then, is to extend the framework to handle the effects of a computation properly. Moreover, since the type system includes quantification over arbitrary effect rows, the extension has to be uniform enough to allow for universal quantification over the interpretations of effect rows.

An interesting choice in *Koka*, wholly in keeping with the idea of algebraic effects, is to allow multiple occurrences of the same effect name in the row. However, our interpretation shows that for this idea to be fully realised, the calculus has to be extended with an additional operation, which we introduce and show to behave well with the rest of the calculus. We are also able to strengthen the effect subsumption rules significantly: as usual, the soundness of these extensions follows as a corollary from the correctness of the logical relation.

Finally, we demonstrate that the relation is a useful tool for proving program equivalence. To this end, we use type of polymorphic higher-order computations that can receive various effectful and non-effectful implementations of certain operations, and show that such implementations can easily be shown equivalent by choosing an appropriate interpretation, regardless of the number and names of effects used by the implementations. We also show, as an example of a general type-directed equivalence, that the usual *return* clauses in handlers are not necessary in our calculus (i.e., that we can always transform the program to an equivalent one which does not use a non-trivial *return* clause). Thus, we believe that our technique of interpreting algebraic effects and row polymorphism has the potential to be a useful tool for showing program equivalence.

In conclusion, we feel that the work on proving contextual equivalence in presence of algebraic effects and row polymorphism falls squarely within the scope of the HOPE workshop, and we feel it would be particularly interesting to its audience. We also expect that the discussion at the workshop would help us develop this line of work to its fullest potential.

References

[1] N. Benton and C.-K. Hur. *Biorthogonality, Step-indexing and Compiler Correctness.* In A. Tolmach, editor, ICFP. ACM, 2009

[2] D. Hillerström and S. Lindley. *Liberating effects with rows and handlers.* In J. Chapman and W. Swierstra, editors, TyDe , pages 15–27. ACM, 2016.

[3] D. Leijen. *Type directed compilation of row-typed algebraic effects.* In A. D. Gordon, editor, POPL. ACM, 2017.

[4] Sam Lindley, Connor McBride, and Craig McLaughlin. *Do Be Do Be Do.* In A. D. Gordon, editor, POPL. ACM, 2017.

[5] G. D. Plotkin and J. Power. *Semantics for algebraic operations.* Electr. Notes Theor. Comput. Sci., 45:332–345, 2001.

[6] G. D. Plotkin and M. Pretnar. *Handling algebraic effects.* Logical Methods in Computer Science, 9(4), 2013.

[7] M. Pretnar. *Inferring algebraic effects.* Logical Methods in Computer Science, 10(3), 2014.