

Higher-Order Effects with Implicitly Resolved Elaborations

Terts Diepraam






April 11, 2023

This table of contents is of course tentative.

Contents

1	Introduction	2
2	Basic Syntax and Semantics	4
3	Algebraic Effects	4
4	Higher-Order Effects and Hefty Algebras	4
5	Elaboration Resolution	4
6	Compiling Elaborations	4
7	Conclusion	4
A	Elaine Specification	4
A.1	Syntax	4
A.2	Typing Judgments	4
A.3	Semantics	4
A.4	Standard Library	4

Todo list

	This table of contents is of course tentative.	1
	This is currently mostly a short history of effects, not quite an introduction yet.	2
	Set up the basic syntax and semantics of Elaine, used throughout the thesis.	4
	Introduce algebraic effects, effect rows, and handlers.	4
	Introduce higher-order effects, the composability problem, scoped effects, hefty algebras etc.	4

1 Introduction

This is currently mostly a short history of effects, not quite an introduction yet.

In standard λ -calculus, there is no model for effects of the computation, only of the result. Speaking broadly, effects concern the aspects of the program besides the pure computation. [5]

In their seminal paper, Moggi [5] unified monads with computational effects, or notions of computation as they call it. Functional programming languages often rely on monads to perform effectful computations. Haskell, for example, uses an `IO` monad for printing output and reading input. The computation is then kept pure and impure operations (like reading and printing) are delegated to the system. However, a limitation of treating effects as monads is that monads do not compose well.

Plotkin and Power [6] then showed that effects can be represented as equational theories. The effects that can be represented in this framework are called algebraic effects. The `State` and the `Maybe` monads can for example be expressed in this framework.

Plotkin and Pretnar [7] then introduced effect handlers, allowing the programmer to destructure effects by placing handlers around effectful expressions. This provides a way to treat exception handling using effects. However, the scope in which an effect is handled can only be changed by adding handlers. Effect operations cannot define their own scope. To support this, a system for higher-order effects is required, which are effects that take effectful operations as parameters.

A solution to this was the framework of scoped effects [10]. However, scoped effects require a significant increase in complexity and cannot express effects that are neither algebraic nor scoped, such as lambda abstractions [9]. Latent effects [9] were subsequently introduced as an alternative that encapsulates a larger set of effects.

As an alternative approach to latent effects, Bach Poulsen and van der Rest [1] introduced hefty algebras. With hefty algebras, higher-order effects are treated separately from algebraic effects. Higher-order effects are not handled, but elaborated into algebraic effects, which can then be handled. The advantage is that the treatment of algebraic effects remains intact and that the process of elaboration is relatively simple.

In parallel with the work to define theoretical frameworks for effects, several libraries and languages have been designed that include effects as first-class concepts, allowing the programmer to define their own effects and handlers. For example, there are some libraries available for Haskell, like `fused-effects`¹, `polysemy`², `freer-simple`³ and `eff`⁴, each encoding effects in a slightly

¹<https://github.com/fused-effects/fused-effects>

²<https://github.com/polysemy-research/polysemy>

³<https://github.com/lexi-lambda/freer-simple>

⁴<https://github.com/hasura/eff>

different way.

Notable examples of languages with support for algebraic effects are Eff [2], Koka [3] and Frank [4]. OCaml also gained support for effects [8]. By building algebraic effects into the language, instead of delegating to a library, is advantageous because the language can provide more convenient syntax. In these languages, some concepts that were traditionally only available with language support, can be expressed by the programmer. This includes exception handling and asynchronous programming.

These languages either only support algebraic effects or scoped effects. This means that their support for higher-order effects is limited. The exception is `heft`⁵, which is produced in conjunction with the work in [1].

The aim of this thesis is to build on the work by Bach Poulsen and van der Rest and create a new language which

- supports higher-order effects using hefty algebras, with separate elaborations and handlers,
- implicitly resolves elaborations, and
- can be compiled to a representation with only algebraic effects.

To this end, we need to show that the compilation of implicitly resolved effects is equivalent to the operational semantics for elaborations. By showing that elaborations can be removed at compile-time, we show that it is possible to compile programs in our language using the techniques from [3].

In addition, we provide an implementation of this language, which is published on GitHub under the name `elaine`⁶ (sharing a prefix with “elaboration”) as part of the repository that hosts this thesis.

⁵<https://github.com/heft-lang/heft>

⁶<https://github.com/tertsdiepraam/thesis/tree/main/elaine>

2 Basic Syntax and Semantics

Set up the basic syntax and semantics of Elaine, used throughout the thesis.

3 Algebraic Effects

Introduce algebraic effects, effect rows, and handlers.

4 Higher-Order Effects and Hefty Algebras

Introduce higher-order effects, the composability problem, scoped effects, hefty algebras etc.

5 Elaboration Resolution

6 Compiling Elaborations

7 Conclusion

A Elaine Specification

A.1 Syntax

A.2 Typing Judgments

A.3 Semantics

A.4 Standard Library

References

- [1] C. Bach Poulsen and C. van der Rest. “Hefty Algebras: Modular Elaboration of Higher-Order Algebraic Effects”. In: *Proceedings of the ACM on Programming Languages* 7 (POPL Jan. 9, 2023), pp. 1801–1831. ISSN: 2475-1421. DOI: 10.1145/3571255. URL: <https://dl.acm.org/doi/10.1145/3571255> (visited on 01/26/2023).
- [2] A. Bauer and M. Pretnar. “Programming with algebraic effects and handlers”. In: *Journal of Logical and Algebraic Methods in Programming* 84.1 (Jan. 2015), pp. 108–123. ISSN: 23522208. DOI: 10.1016/j.jlamp.2014.02.001. URL: <https://linkinghub.elsevier.com/retrieve/pii/S2352220814000194> (visited on 04/03/2023).

- [3] D. Leijen. “Type directed compilation of row-typed algebraic effects”. In: *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*. POPL ’17: The 44th Annual ACM SIGPLAN Symposium on Principles of Programming Languages. Paris France: ACM, Jan. 2017, pp. 486–499. ISBN: 978-1-4503-4660-3. DOI: 10.1145/3009837.3009872. URL: <https://dl.acm.org/doi/10.1145/3009837.3009872> (visited on 04/08/2023).
- [4] S. Lindley, C. McBride, and C. McLaughlin. *Do be do be do*. Oct. 3, 2017. arXiv: 1611.09259[cs]. URL: <http://arxiv.org/abs/1611.09259> (visited on 04/08/2023).
- [5] E. Moggi. “Computational lambda-calculus and monads”. In: *[1989] Proceedings. Fourth Annual Symposium on Logic in Computer Science*. [1989] Proceedings. Fourth Annual Symposium on Logic in Computer Science. Pacific Grove, CA, USA: IEEE Comput. Soc. Press, 1989, pp. 14–23. ISBN: 978-0-8186-1954-0. DOI: 10.1109/LICS.1989.39155. URL: <http://ieeexplore.ieee.org/document/39155/> (visited on 04/08/2023).
- [6] G. Plotkin and J. Power. “Adequacy for Algebraic Effects”. In: *Foundations of Software Science and Computation Structures*. Ed. by F. Honsell and M. Miculan. Red. by G. Goos, J. Hartmanis, and J. van Leeuwen. Vol. 2030. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 1–24. ISBN: 978-3-540-41864-1 978-3-540-45315-4. DOI: 10.1007/3-540-45315-6_1. URL: http://link.springer.com/10.1007/3-540-45315-6_1 (visited on 04/08/2023).
- [7] G. Plotkin and M. Pretnar. “Handlers of Algebraic Effects”. In: *Programming Languages and Systems*. Ed. by G. Castagna. Vol. 5502. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 80–94. ISBN: 978-3-642-00589-3 978-3-642-00590-9. DOI: 10.1007/978-3-642-00590-9_7. URL: http://link.springer.com/10.1007/978-3-642-00590-9_7 (visited on 04/08/2023).
- [8] K. C. Sivaramakrishnan et al. “Retrofitting Effect Handlers onto OCaml”. In: *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*. June 19, 2021, pp. 206–221. DOI: 10.1145/3453483.3454039. arXiv: 2104.00250[cs]. URL: <http://arxiv.org/abs/2104.00250> (visited on 04/08/2023).
- [9] B. van den Berg et al. “Latent Effects for Reusable Language Components”. In: *Programming Languages and Systems*. Ed. by H. Oh. Vol. 13008. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2021, pp. 182–201. ISBN: 978-3-030-89050-6 978-3-030-89051-3. DOI: 10.1007/978-3-030-89051-3_11. URL: https://link.springer.com/10.1007/978-3-030-89051-3_11 (visited on 01/12/2023).
- [10] N. Wu, T. Schrijvers, and R. Hinze. *Effect Handlers in Scope*. June 10, 2014.