ISSN 2186-7437

NII Shonan Meeting Report

No. 146

Programming and Reasoning with Algebraic Effects and Effect Handlers

Sam Lindley
Nicolas Wu
Oleg Kiselyov
Gordon Plotkin

March 25–29, 2019

National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-Ku, Tokyo, Japan

Programming and Reasoning with Algebraic Effects and Effect Handlers

Organizers:
Oleg Kiselyov (Tohoku University, Japan)
Sam Lindley (The University of Edinburgh, UK)
Gordon Plotkin (The University of Edinburgh, UK)
Nicolas Wu (University of Bristol, UK)

March 25–29, 2019

Algebraic effects [11, 12] and effect handlers are becoming an increasingly popular approach for expressing and composing computational effects. There are implementations of algebraic effects and effect handlers in Clojure, F#, Haskell [6, 7], Idris [2], Javascript[1], OCaml PureScript, Racket, Scala[2], Scheme, Standard ML, and even C [8]. There are full-fledged languages built around effects such as Eff [1], Frank [10], Links [4], Koka [9], and Multicore OCaml [3]. Moreover, there is growing interest from industry in effect handlers. For instance, Facebook's React library for JavaScript UI programming is directly inspired by effect handlers, and Uber's recently released Pyro tool for probabilistic programming and deep learning relies on effect handlers. Such interest arises from the ease of combining in the same program independently developed components using algebraic effects and effect handlers.

The increased adoption and use of algebraic effects and effect handlers reveal and make pressing three main problems: *reasoning*, *performance*, and *typing*. These problems may appear disparate, but we believe there are in fact deep connections that bring them together.

Reasoning Algebraic effects are defined by a signature of operations and an equational theory that describes how the operations interact, providing direct support for reasoning. Effect handlers are modular interpreters for algebraic effects, giving meaning to such operations. Existing implementations of effect handlers dispense with equations, largely because many open questions remain about how to incorporate them into a programming language. A key question that this meeting will seek to address is how to reintroduce equations and other forms of reasoning back into the effect handlers picture. An important consideration here is how to combine equational theories for several interacting effects.

[1] `https://www.humblespark.com/blog/extensible-effects-in-node-part-1`

[2] `https://github.com/atnos-org/eff`, `https://github.com/m50d/paperdoll`, among others

Performance The dominant implementation method, the free monad, is notably slower than the direct execution of side-effects where available. A range of approaches for improving performance are under active investigation. These include direct stack manipulation, in the case that continuations are used linearly, selective CPS translations, and fusion transformations. The jury is still out on which techniques work best in which situations.

Typing Programming in the large involves working with complex and interacting systems. Effect type systems are a powerful means of taming this complexity, in a way that is amenable for practical programming. Several different effect type systems have been introduced for algebraic effects and effect handlers. It is not clear yet precisely what the tradeoffs are between the different approaches. Many open questions remain over how best to support features such as generative effects, and how to leverage effect type systems to support reasoning and to improve performance.

Given the complexity of these problems and their importance, we believed the face-to-face meeting of main community representatives would promote their solution.

We identified five specific application areas to be discussed at the meeting in the context of the three main problem areas:

- Effect handlers for concurrent and distributed programming;
- Effect handlers for generative effects (ML references, renaming effects, scoped effects, modularity, runST, existentials);
- Effect handlers with behavioral types (parameterized monads, graded monads, type state, session types, answer type modification, dependent types);
- Effect handlers and resource management;
- Effect handlers for probabilistic programming.

To promote mutual understanding, we have planned for the workshop to have substantial time available for discussion. Our hope has been to emphasize tutorials, brainstorming, and working-group sessions, rather than mere conference-like presentations.

List of Participants

The following people participated in the seminar, besides the organizers.

1. Danel Ahman, INRIA, Paris (UK)
2. Robert Atkey, University of Strathclyde (UK)
3. Oliver Bračevac, TU Darmstadt (Germany)
4. Edwin Brady, University of St Andrews (UK)
5. Youyou Cong, Ochanomizu University (Japan)
6. Stephen Dolan, University of Cambridge (UK)

7. Jeremy Gibbons, University of Oxford (UK)
8. Daniel Hillerström, The University of Edinburgh (UK)
9. Atsushi Igarashi, Kyoto University (Japan)
10. Mauro Jaskelioff, Universidad Nacional de Rosario (Argentina)
11. Yukiyoshi Kameyama, University of Tsukuba (JP)
12. Ohad Kammar, University of Oxford (UK)
13. Shin-ya Katsumata, National Institute of Informatics (Japan)
14. Daan Leijen, Microsoft Research (USA)
15. Conor McBride, University of Strathclyde (UK)
16. James McKinna, The University of Edinburgh (UK)
17. Craig McLaughlin, The University of Edinburgh (UK)
18. Shin-Cheng Mu, Academia Sinica (Taiwan)
19. Max S. New, Northeastern University (US)
20. Maciej Piróg, University of Wrocław (Poland)
21. Andreas Rossberg, DFINITY (Germany)
22. Tom Schrijvers, University of Leuven (Belgium)
23. Philipp Schüster, Universität Tübingen (Germany)
24. Leo White, Jane Street (UK)
25. Jeremy Yallop, OCaml Labs, Cambridge (UK)
26. Yizhou Zhang, Cornell University (US)

Themes

The seminar covered different aspects of programming and reasoning with algebraic effects and effect handlers. Here we outline some of the main themes.

Encapsulating effects. Several different speakers discussed approaches to encapsulating effects. Yizhou Zhang presented an approach called "tunneling" based around capabilities and implicit arguments. Maciej Piróg presented an approach based on an effect type system derived from Koka's row-based effect type system, extended with *coercions* and constructs for fresh effects and abstract effects. Craig McLaughlin presented an equivalent to coercions, called *adaptors*, in the Frank programming language, but generalised to support all finite remappings of effects. Stephen Dolan presented yet another approach, similar to that of Maciej Piróg, but using nominal techniques for generating fresh effects.

Equations. Algebraic effects come equipped with equational theories. Though the theory of effect handlers arose in the context of algebraic effects [12], their success in practical programming has typically been as a programming feature founded on free algebras. This is partly because equations (and proofs in general) require special treatment for incorporating into most standard programming languages, and partly because it can be useful to interpret the same effects using different equational theories. Nevertheless, equations offer a much promise for supporting reasoning, and reasoning with algebraic effects was a pervasive theme at the meeting. Jeremy Gibbons presented old work on equational reasoning with effects. Tom Schrijvers presented new work building on the ideas of Jeremy Gibbons and collaborators, systematically studying different ways of combining state with nondeterminism. Danel Ahman presented work in progress by his colleagues Matija Pretnar and Žiga Luksic (neither who could attend, unfortunately) on combining equations with effect handlers in a practical programming language.

Non-linear continuations and external resources. Effect handlers allow continuations to be invoked more than once, which is useful for applications such as backtracking search and probabilistic programming. However, non-linear use of continuations presents a problem if the continuation performs operations on external resources, such as file I/O — typically we expect such operations to happen exactly once, and if, for instance, we try to close an already closed file then non-linear invocation results in an error. Steven Dolan presented a simple example of the problem via a handler involving a choice operation interpreted by running the continuation twice, in which the continuation performs file I/O. Daniel Hillerström and Sam Lindley exhibited essentially the same problem in Links when one combines effect handlers with session types (both of which are built into Links). A number of techniques for ruling out these kind of examples were discussed. One option is to simply rule out non-linear use of continuations — this still allows the use of effect handlers for applications such as concurrency, for instance, but rules out applications such as backtracking. The dynamic wind construct of Scheme provides a way of initialising and finalising resources each time they are accessed by the same continuation. Another idea, is to mark certain effects as linear, and statically disallow them from being used in non-linear continuations. Daan Leijen has begun to explore this approach in Koka. At the meeting, Bob Atkey sketched a variation of linear logic extended with support for linear effect annotations. Conor McBride presented an indexed version of effects that carefully tracks resources. In this setting it is not obvious how to even write examples that make non-linear use of the continuation. Related to this discussion, Danel Ahman presented early work with Andrej Bauer on using comodels to account for external resources.

Tangible Outcomes

The seminar provided a fruitful forum for discussion and an ideal setting for fostering future collaboration. Here we enumerate some of the tangible outcomes of the seminar.

- Danel Ahman, Amal Ahmed, Sam Lindley, and Andreas Rossberg, have

submitted a follow-on Dagstuhl seminar proposal on "Scalable Handling of Effects".

- Daan Leijen and Ohad Kammar initiated a collaboration to develop a probabilistic programming library for Koka.
- Mauro Jaskelioff, Maciej Piróg, Tom Schrijvers, and Nicolas Wu wrote a paper together on the relationship between monad transformers and effect handlers, which they have submitted to the Haskell Symposium.
- The proposal for adding effect handlers to WebAssembly, presented by Andreas Rossberg, was refined. Emerging from this discussion, Daniel Hillerström has arranged a post-doctoral internship with Daan Leijen at Microsoft to explore efficient compilation of effect handlers.
- Informed by feedback following Daniel Hillerström's presentation at the meeting, Daniel Hillerström and Sam Lindley (in collaboration with John Longley) are working on a POPL submission.
- The effect handlers rosetta stone is an open repository for examples of programming with effect handlers. At the meeting, we supplemented the effect handlers rosetta stone with several new examples.

Meeting Schedule

May 25 (Monday)

- Self-introductions
- Tutorial on foundations of algebraic effects (Gordon Plotkin)
- Tutorial on pragmatics of effect handlers (Daan Leijen)
- Just do It (Jeremy Gibbons)
- Implementing local state with global state (Tom Schrijvers)
- Making equations great again (Danel Ahman, for Ziga Lukšič and Matija Pretnar)

May 26 (Tuesday)

- Abstracting Algebraic Effects (Maciej Piróg)
- Abstraction-safe effect handling via tunneling (Yizhou Zhang)
- Nominal Effects (Stephen Dolan)
- Effect handlers for a low-level stack machine (Andreas Rossberg)
- Working groups
- Cooking concurrency for algebraic effects (Mauro Jaskelioff)
- Ambient Parameters (Daan Leijen)
- Cubical type theory 101 (Conor McBride)

May 27 (Wednesday)

- Syntax and Semantics for Operations with Scopes (Nicolas Wu)
- Graded monads (Shin-ya Katsumata)
- One monad to the tune of another (Robert Atkey)
- Talking to Frank (Craig McLaughlin)
- Excursion and banquet

May 28 (Thursday)

- Equational theories and monads from polynomial Cayley representations (Maciej Piróg)
- Handling polymorphic algebraic effects (Atsushi Igarashi)
- Handlers and multihandlers (Gordon Plotkin)
- Comodels as a gateway for interacting with the external world (Danel Ahman)
- Asymptotic improvement through delimited control (Daniel Hillerström)
- Working Groups

May 29 (Friday)

- Breaking Links (Daniel Hillerström, Sam Lindley, Leo White)
- Working Groups

Selected abstracts

Comodels as a gateway for interacting with the external world

Danel Ahman, Ljubljana, Slovenia

As much as we might try to convince ourselves, functional programming is rarely completely pure as one often cannot escape having to interact with the external world, e.g., by needing to write the results of one's program to a terminal window or a file, or perhaps also consult an external source of randomness for making nondeterministic and probabilistic choices. In this work we are exploring programming abstractions for orchestrating such interactions in a principled way.

In languages with algebraic effects (resp. monads), the usual way of modeling interactions with the external world is to define a dedicated effect (resp. monad) and then treat it specially in the compiler, such as the RandomInt effect in the Eff language and the IO monad in Haskell. This however has various drawbacks. For instance, such languages often lack enforcement mechanisms to ensure that a program does not write to an already closed file. And what is even worse, in

languages with algebraic effects, where all effects can be handled, it is very easy to accidentally cause one's program not to even reach file closing, by discarding a continuation somewhere in the handler. The common denominator here is the lack of linearity guaranteed by these languages.

In this talk, I will present some of our initial findings on the use of comodels of algebraic effects as a programming abstraction for (i) modeling the external world and interactions with it, (ii) ensuring the linearity of these interactions, and (iii) controlling which capabilities of the external world different parts of programs have access to. Regarding (ii), the novel aspect of our work is that we do not ensure linearity of these interactions by the means of a linear type system, but instead we leave the external world implicit and interact with it through a combination of algebraic operations and (under the hood) a linear state-passing translation similar to that of Møgelberg and Staton. Regarding (i) and (ii), we do not limit the programmer to a single external world, but instead allow them to modularly build their own intermediate external worlds. In the talk I will demonstrate these ideas through small programming examples, and also comment on the opportunities and challenges involved in combining such use of comodels with effect handlers.

(This is joint work with Andrej Bauer.)

Making equations great again

Danel Ahman, for Ziga Lukšič and Matija Pretnar, Ljubljana, Slovenia

Algebraic effects are computational effects that can be described with a set of basic operations and equations between them. As many interesting effect handlers do not respect these equations most approaches assume a trivial theory sacrificing both reasoning power and safety. We present an alternative approach where the type system tracks equations that are observed in subparts of the program yielding a sound and flexible logic and paving a way for practical optimizations and reasoning tools.

Dijkstra Monads: One Monad to the Tune of Another

Robert Atkey, Strathclyde, UK

I'll talk about a generic way of reasoning about monadic computations by mapping one monad, the "computational" monad, to another, the "specification" monad via a monad morphism. Such monad morphisms allow us to reconstruct existing program logics for state, I/O, nondeterminism, and exceptions. Monad morphisms can be packaged up neatly as a "Dijkstra Monad" – essentially a monad graded by another monad – yielding a useful technique for simultaneous programming and verification.

(This is joint work with Kenji Maillard, Danel Ahman, Guido Martinez, Catalin Hritcu, Exequiel Rivas and Eric Tanter.)

Just Do It

Jeremy Gibbons, Oxford, UK

I summarized my paper "Just Do It: Simple Monadic Equational Reasoning"

(ICFP 2011, co-authored with Ralf Hinze). I had two main points. The first, and the main point of the paper, was that although Haskell's comprehension-like "do" notation looks more like imperative programming than it does traditional functional programming, nevertheless it is still eminently amenable to the kind of equational reasoning we love in FP. The second point, perhaps of more interest to this meeting, was the emphasis on modeling effects by combinations of algebraic theories (as opposed to stacks of monad transformers, which was the prevailing approach in 2011). I paid particular attention to combining state and nondeterminism: there is a nice model of *local* state (in which each nondeterministic branch works separately on its own copy of the state), but it is not so obvious how to do *global* state (in which the branches work on a single shared state). This served as an introduction to Tom Schrijvers' talk.

Asymptotic improvement through delimited control

Daniel Hillerström, Edinburgh, UK

I will provide an overview of a recent expressivity result for effect handlers that I have established in collaboration with John Longley and Sam Lindley.

Using a variation of generic search, that counts all solutions to a given search problem, we show that a PCF flavoured language with effect handlers admits an implementation of generic search that runs in $O(2^n)$ time, whilst there is no implementation of generic search in a PCF flavoured language without effect handlers that runs better than $O(n2^n)$ time. As corollary we obtain that there exists a class of programs for which effect handlers provide more efficient implementations.

Intuitively the efficiency gap is due to control operators, such as effect handlers, providing the ability to backtrack, meaning that it is possible to revert a past decision whilst remembering the outcome of that decision, whereas in a purely functional setting repeating past decisions is necessary to explore an alternative decision.

Handling Polymorphic Algebraic Effects

Atsushi Igarashi, Kyoto, Japan

Algebraic effects and handlers are a powerful abstraction mechanism to represent and implement control effects. In this work, we study their extension with parametric polymorphism that allows abstracting not only expressions but also effects and handlers. Although polymorphism makes it possible to reuse and reason about effect implementations more effectively, it has long been known that naive combination of polymorphic effects and let-polymorphism breaks type safety. While type safety can often be gained by restricting let-bound expressions—e.g., by adopting value restriction or weak polymorphism—we propose a complementary approach, which restricts, instead of let-bound expressions, handlers. Our key observation is, informally speaking, that a handler is safe if resumptions from the handler do not interfere with each other. To formalize our idea, we define a call-by-value lambda calculus $\lambda_{\text{eff}}^{\text{let}}$ that supports let-polymorphism and polymorphic algebraic effects and handlers, design a type system that rejects interfering handlers, and prove type safety of our calculus.

Talking to Frank

Craig McLaughlin, Edinburgh, UK

Frank is a strict functional language supporting algebraic effects (a la Plotkin & Power) and handlers (a la Plotkin & Pretnar) within an effect type system. Key to the design is the generalisation of functions to *operators* which may handle effects by pattern matching on computation trees. Operators are n-ary allowing the simultaneous handling of multiple computations. I will describe the key features of Frank by way of example, showing its similarities to regular functional programming and its differences. I will highlight the conveniences afforded by some of the design choices, in particular achieving effect polymorphism while avoiding the need to mention effect variables. Finally, I describe the effect pollution problem and how this is overcome in Frank.

(This is Joint work with Lukas Convent, Sam Lindley and Conor McBride.)

Effect handlers in a low-level stack machine

Andreas Rossberg, DFINITY Foundation, Germany

WebAssembly (Wasm) is a general-purpose low-level code format for environments where portability and safety are mandatory, such as the Web, content delivery networks, or decentralised cloud computation. In order to enable the compilation of a wide range of programming languages to Wasm, we are considering to extend it with effect handlers as a way to compile the growing zoo of control abstractions that exist in those languages. I present a preliminary design for an extension to the Wasm instruction set and type system that provides a low-level representation of effect handlers.

(This work originates from the Dagstuhl workshop last year, and is joint work with multiple other participants.)

Handling Local State with Global State

Tom Schrijvers, KU Leuven, Belgium

We consider the interaction between monadic renditions of the state and non-determinism effects. There are two prominent interactions in the literature: local state semantics and global state semantics. We characterise these with laws and show how to simulate local state semantics using global state semantics. To prove the correctness of this simulation we reconcile the typical denotational approach to semantics for monads with a syntactic approach to semantics that is convenient for inductive proofs that can be mechanised in a proof assistant. We do this by means of free monads that capture the syntax of effectful programs, but not their semantics. The semantics is obtained by mapping into a semantic domain, and requires us to express the effect laws on the free monad in terms of contextual equivalence. In order to make the simulation work, we require two non-contextual laws. These laws are not satisfied by conventional monadic models, but thanks to our free monad approach we get the monadic structure for free and can supply a non-monadic model.

Syntax and Semantics for Operations with Scopes

Nicolas Wu, University of Bristol, UK

Motivated by the problem of separating syntax from semantics in programming with algebraic effects and handlers, we propose a categorical model of abstract syntax with so-called scoped operations. As a building block of a term, a scoped operation is not merely a node in a tree, as it can also encompass a whole part of the term (a scope). Some examples from the area of programming are given by the operation catch for handling exceptions, in which the part in the scope is the code that may raise an exception, or the operation once, which selects a single solution from a nondeterministic computation. A distinctive feature of such operations is their behaviour under program composition, that is, syntactic substitution.

Our model is based on what Ghani et al. call the monad of explicit substitutions, defined using the initial-algebra semantics in the category of endofunctors. We also introduce a new kind of multi-sorted algebras, called scoped algebras, which serve as interpretations of syntax with scopes. In generality, scoped algebras are given in the style of the presheaf formalisation of syntax with binders of Fiore et al. As the main technical result, we show that our monad indeed arises from free objects in the category of scoped algebras

(This is joint work with Tom Schrijvers, Maciej Piróg, and Mauro Jaskelioff.)

Abstraction-Safe Effect Handlers via Tunneling

Yizhou Zhang, Cornell, US

Algebraic effect handlers offer a unified approach to expressing control-flow transfer idioms such as exceptions, iteration, and async/await. Unfortunately, previous attempts to make these handlers type-safe have failed to support the fundamental principle of modular reasoning for higher-order abstractions. We demonstrate that abstraction-safe algebraic effect handlers are possible by giving them a new semantics: effects tunnel through contexts polymorphic to them. We prove that our design is not only type-safe, but also abstraction-safe. Using a logical-relations model that we prove sound with respect to contextual equivalence, we derive previously unattainable program equivalence results. Our mechanism offers a viable approach for future language designs aiming for effect handlers with strong abstraction guarantees.

References

[1] Andrej Bauer and Matija Pretnar. Programming with algebraic effects and handlers. *J. Log. Algebr. Meth. Program.*, 84(1):108–123, 2015.

[2] Edwin Brady. Programming and reasoning with algebraic effects and dependent types. In ICFP [5], pages 133–144.

[3] Stephen Dolan, Leo White, KC Sivaramakrishnan, Jeremy Yallop, and Anil Madhavapeddy. Effective concurrency through algebraic effects. OCaml Workshop, 2015.

[4] Daniel Hillerström and Sam Lindley. Liberating effects with rows and handlers. In *TyDe@ICFP*, pages 15–27. ACM, 2016.

[5] *ICFP '13: Proceedings of the ACM International Conference on Functional Programming*. ACM Press, 2013.

[6] Ohad Kammar, Sam Lindley, and Nicolas Oury. Handlers in action. In ICFP [5], pages 145–158.

[7] Oleg Kiselyov and Hiromi Ishii. Freer monads, more extensible effects. pages 94–105. ACM, 2015.

[8] Daan Leijen. Implementing algebraic effects in C - "Monads for free in C". In *APLAS*, volume 10695 of *Lecture Notes in Computer Science*, pages 339–363. Springer, 2017.

[9] Daan Leijen. Type directed compilation of row-typed algebraic effects. In *POPL*, pages 486–499. ACM, 2017.

[10] Sam Lindley, Conor McBride, and Craig McLaughlin. Do be do be do. In *POPL*, pages 500–514. ACM, 2017.

[11] Gordon Plotkin and John Power. Algebraic operations and generic effects. *Applied Categorical Structures*, 11(1):69–94, 2003.

[12] Gordon Plotkin and Matija Pretnar. Handlers of algebraic effects. In Giuseppe Castagna, editor, *Programming Languages and Systems*, volume 5502 of *LNCS*, pages 80–94. Springer-Verlag, Berlin, Heidelberg, 2009.