Logical Relations for Effect Capabilities

PATRYCJA BALIK and PIOTR POLESIUK, University of Wrocław, Poland

Algebraic effects with handlers [9] provide a convenient mechanism for managing multiple computational effects. Now, they are well-established in the research community, and are making their way into industry-grade programming languages [10]. Originally, handlers had a dynamic semantics, in which effect operations were handled by the dynamically closest handler. However, this semantics leads to problems similar to those from the world of dynamic variable binding, such as accidental effect capture. To overcome these problems, Zhang and Myers [12] and Biernacki et al. [3] independently proposed two different calculi of lexical handlers, where the use of an effect is connected to its handler via a special value (called *effect instance*) bound by the handler. In their work this value was second-class in order to statically ensure that an effect instance will never be used outside its handler. Xie et al. [11] observed that the same static guarantees can be obtained using simpler means, just by creating a fresh abstract effect at each handler. This is similar to how Rank-2 types are used to implement the `ST` monad in Haskell.

One of the possible approaches to effect instances is based on the capability-passing style [4, 5]. In this approach, the handler creates a first-class value, called a capability, which can be used to invoke the effect. Using our syntax we can write the following simple example of the standard reader effect.

```
handle ask = effect () / k => k 21 in ask () + ask ()
```

The handler defines the value `ask` which is used as a regular function in `ask () + ask ()`. However, once called with a unit argument, the computational effect takes place. Since it was defined using the **`effect`** construct, its body (`k 21`) is evaluated in the handler's context, but the caller's context can be resumed by calling the continuation `k`. We follow Xie et al. and use a type-and-effect system to forbid the use of `ask` outside the handler. The type of `ask` is `Unit ->[E] Int`, where `E` is a fresh effect variable introduced by the handler.

Inspired by examples of embedding of algebraic effects in object-oriented languages [5] we observed that it is useful to allow other constructs besides **`effect`** to be used in the definition of a capability. For example, effects with multiple operations can be expressed using a pair or record of effectful functions. If we additionally permit let-definitions, we can cleanly separate the interface of an effect from its implementation. Consider the following slightly modified standard handler of state.

```
handle st =
  let get = effect () / k => fn s => k s  s
  let put = effect s  / k => fn _ => k () s
  let update f = put (f (get ())) in
    { get, put, update }
  return  x => fn _ => x
  finally c => c initState in ...
```

In this example, we define two primitive operations `get` and `put` using **`effect`**, and additionally expose the function `update` that composes these two operations. As another example, we could implement the `yield` and `spawn` operations for lightweight threads similar to those in [1] by using the more primitive `fork` and `exit`, which are not exposed at all.

We tackle the problem of designing logical relations for a calculus with the discussed features. Among the prior work on logical relations for effect handlers [2, 3, 12], only Zhang and Myers [12] study a calculus with effect capabilities. However, their logical relation is defined intensionally, assuming all capabilities are of a specific shape (`effect x / k =>` e using our syntax), and therefore, it does not easily scale to our calculus. An extensional definition turned out to be a significant challenge, in part due to the unusual behavior of the variable representing the handled effect. Logical relations are usually extended to open terms by universally quantifying over the interpretations of variables in the environment, but the handled effect variable behaves more existentially.

During the talk we will describe two approaches to logical relations for effect capabilities. The first approach generalizes the extension of logical relations to open terms by using existential quantification over the interpretation of the distinguished effect variable. This introduces a new quirk in the definition in the form of alternation of quantifiers. The second gets rid of the existential quantifier by using a concrete, maximal semantic effect instead. Our construction of the second model requires us to make certain restrictions on the type system. At the time of writing, we do not know which restrictions are essential, and which could still be relaxed. However, we still believe this model is interesting, as it also happens to be a model for the delimited control operators shift_0 and reset_0. This observation provides us a new, more semantic perspective on the well-known connection between those operators and effect handlers [6–8].

The proposed relational models characterize contextual equivalence. We will present how our logical relations work in practice by showing a few non-trivial equivalences exploiting the fact that there exists only one capability for each effect. In particular, we can prove that a standard handler of state which exposes only the `get` operation is equivalent to a standard reader handler.

REFERENCES

[1] Andrej Bauer and Matija Pretnar. 2015. Programming with algebraic effects and handlers. *J. Log. Algebr. Methods Program.* 84, 1 (2015), 108–123. https://doi.org/10.1016/j.jlamp.2014.02.001

[2] Dariusz Biernacki, Maciej Piróg, Piotr Polesiuk, and Filip Sieczkowski. 2018. Handle with care: relational interpretation of algebraic effects and handlers. *PACMPL* 2, POPL (2018), 8:1–8:30. https://doi.org/10.1145/3158096

[3] Dariusz Biernacki, Maciej Piróg, Piotr Polesiuk, and Filip Sieczkowski. 2020. Binders by day, labels by night: effect instances via lexically scoped handlers. *PACMPL* 4, POPL (2020), 48:1–48:29. https://doi.org/10.1145/3371116

[4] Jonathan I. Brachthäuser, Philipp Schuster, and Klaus Ostermann. 2020. Effects as capabilities: effect handlers and lightweight effect polymorphism. *PACMPL* 4, OOPSLA (2020), 126:1–126:30. https://doi.org/10.1145/3428194

[5] Jonathan I. Brachthäuser, Philipp Schuster, and Klaus Ostermann. 2020. Effekt: Capability-passing style for type- and effect-safe, extensible effect handlers in Scala. *JFP* 30 (2020), e8. https://doi.org/10.1017/S0956796820000027

[6] Yannick Forster, Ohad Kammar, Sam Lindley, and Matija Pretnar. 2017. On the expressive power of user-defined effects: effect handlers, monadic reflection, delimited control. *PACMPL* 1, ICFP (2017), 13:1–13:29. https://doi.org/10.1145/3110257

[7] Kazuki Ikemori, Youyou Cong, and Hidehiko Masuhara. 2023. Typed Equivalence of Labeled Effect Handlers and Labeled Delimited Control Operators. In *International Symposium on Principles and Practice of Declarative Programming, PPDP 2023, Lisboa, Portugal, October 22–23, 2023*, Santiago Escobar and Vasco T. Vasconcelos (Eds.). ACM, 4:1–4:13. https://doi.org/10.1145/3610612.3610616

[8] Maciej Piróg, Piotr Polesiuk, and Filip Sieczkowski. 2019. Typed Equivalence of Effect Handlers and Delimited Control. In *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019, June 24–30, 2019, Dortmund, Germany (LIPIcs, Vol. 131)*, Herman Geuvers (Ed.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 30:1–30:16. https://doi.org/10.4230/LIPICS.FSCD.2019.30

[9] Gordon D. Plotkin and Matija Pretnar. 2013. Handling Algebraic Effects. *LMCS* 9, 4 (2013). https://doi.org/10.2168/LMCS-9(4:23)2013

[10] K. C. Sivaramakrishnan, Stephen Dolan, Leo White, Tom Kelly, Sadiq Jaffer, and Anil Madhavapeddy. 2021. Retrofitting effect handlers onto OCaml. In *PLDI '21: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual Event, Canada, June 20–25, 2021*, Stephen N. Freund and Eran Yahav (Eds.). ACM, 206–221. https://doi.org/10.1145/3453483.3454039

[11] Ningning Xie, Youyou Cong, Kazuki Ikemori, and Daan Leijen. 2022. First-class names for effect handlers. *PACMPL* 6, OOPSLA2 (2022), 30–59. https://doi.org/10.1145/3563289

[12] Yizhou Zhang and Andrew C. Myers. 2019. Abstraction-safe effect handlers via tunneling. *PACMPL* 3, POPL (2019), 5:1–5:29. https://doi.org/10.1145/3290318