Continuation Passing Style for Effect Handlers

Mateusz Urbańczyk

7 January 2020

Abstract

We provide an implementation of algebraic effects and handlers by doing continuation passing style tranformation of the functional programming language Freak, which is based on the existing Links langauge [7].

1 Introduction

Implementation of Continuation Passing Style for Effect Handlers paper [7]

2 State of the art

Frank [10], Koka [9], Helium [4]

3 Syntax

4 Operational semantics

4.1 Integers

4.2 Binary operators

5 Usage guide

All commands are available within `src` directory.

5.1 Build and install

- Install dependencies: `make install`
- Compile: `make build`
- Link to PATH: `sudo make link`

- Remove artiacts: `make clean`
- Run tests: `make tests`
- Run code linter: `make lint`

After compiling and linking program to PATH, one may evaluate program as follows: `freak programs/choicesList.fk`. The actual code is described in subsection 7.1 Choice

5.2 Running tests

Test cases are available here, they include both inline and file-based tests. For more details about writing tests, one may refer to HUnit documentation [8].

- Run tests: `make tests`
- Run code linter: `make lint`
- Compile, run linter and tests: `make check`

6 Implementation

The Freak implementation is available at https://github.com/Tomatosoup97/freak. CPS based on [1]

6.1 Row types

6.2 Type inference

Type system as of this day is not implemented, as the focus has been put on CPS transformation. Further work is required.

7 Examples

In this section we present a few examples to show the capabilities of the language. The ideas has been based on [2], and thus will not be described in great details.

7.1 Choice

The first example will be based on modelling (nondeterministic) choice in the program. We will make two decisions, which will affect the computation result:

```
let c1 <- do Choice () in
let c2 <- do Choice () in
let x <- if c1 then return 10 else return 20 in
let y <- if c2 then return 0 else return 5 in
    return x - y
```

With that in hand, we may want to define effect handlers:

```
handle ... with {
    Choice p r ->
        let t <- r 1 in
        let f <- r 0 in
        <PLACEHOLDER> |
    return x -> return x
}
```

where in the `<PLACEHOLDER>` we can define on what to do with the computation. For example, min-max strategy for picking the minimum value:

```
if t < f then return t else return f
```

where the code evaluates to `5`. Another example is a handler that collects all possible results, which can be achieved by putting `return (t, f)` in the `<PLACEHOLDER>`, which evaluates to `((10, 5), (20, 15))`.

7.2 Exceptions

Exceptions are simply algebraic effect handlers which drop the resumption.

```
handle
    if x == 0 then do Raise ()
              else return 1 / x
with {
    Raise p r -> return 42 |
    return x -> return x
}
```

Where we imagine that x variable has been bound previously. More exemplary programs in Freak language can be found here.

8 Future work

8.1 Abstract machine

Implement abstract machine described in [5]

8.2 Multiple instances of algebraic effect

Proposed Helium [3], or Eff [2]

8.3 Make the langugae more usable

While the language is turing-complete, for convenient usage it requires more basic constructs and syntactic sugar for common patterns that would ease the programming.

8.4 Selective CPS

As in [9]

8.5 Exceptions as separate constructs

Exceptions are a trivial example of algebraic effect where the resumption is discarded, and as described in §4.5 [7], they can be modeled as a separate construct to improve performance.

8.6 Shallow handlers

X [6]

9 Conclusions

References

[1] Andrew W. Appel. *Compiling with Continuations*. New York, NY, USA: Cambridge University Press, 1992. ISBN: 0-521-41695-7.

[2] Andrej Bauer and Matija Pretnar. "Programming with Algebraic Effects and Handlers". In: *CoRR* abs/1203.1539 (2012). arXiv: 1203.1539. URL: http://arxiv.org/abs/1203.1539.

[3] Dariusz Biernacki et al. "Binders by Day, Labels by Night: Effect Instances via Lexically Scoped Handlers". In: *Proc. ACM Program. Lang.* POPL (2020).

[4] Dariusz Biernacki et al. "Handle with Care: Relational Interpretation of Algebraic Effects and Handlers". In: *Proc. ACM Program. Lang.* 2.POPL (Dec. 2017), 8:1–8:30. ISSN: 2475-1421. DOI: 10.1145/3158096. URL: http://doi.acm.org/10.1145/3158096.

[5] Daniel Hillerström and Sam Lindley. "Liberating Effects with Rows and Handlers". In: TyDe 2016. ACM, 2016.

[6] Daniel Hillerström and Sam Lindley. "Shallow Effect Handlers". In: *Programming Languages and Systems*. Springer International Publishing, 2018.

[7] Daniel Hillerström et al. “Continuation Passing Style for Effect Handlers”. In: *FSCD*. 2017.

[8] *HUnit: A unit testing framework for Haskell*. URL: `https://hackage.haskell.org/package/HUnit`.

[9] Daan Leijen. “Type Directed Compilation of Row-typed Algebraic Effects”. In: POPL. ACM, 2017.

[10] Sam Lindley, Conor McBride, and Craig McLaughlin. “Do be do be do”. In: *CoRR* abs/1611.09259 (2016). arXiv: `1611.09259`. URL: `http://arxiv.org/abs/1611.09259`.