



# Effect Systems in Haskell - Part II

Sanchayan Maity



- ▶ Cover two papers on Effect Systems.





- ▶ Cover two papers on Effect Systems.
  - ▶ Generalized Evidence Passing for Effect Handlers<sup>1</sup>

---

<sup>1</sup>Generalized Evidence Passing for Effect Handlers



- ▶ Cover two papers on Effect Systems.
  - ▶ Generalized Evidence Passing for Effect Handlers<sup>1</sup>
  - ▶ Effect Handlers in Haskell, Evidently<sup>2</sup>

---

<sup>1</sup>Generalized Evidence Passing for Effect Handlers

<sup>2</sup>Effect Handlers in Haskell, evidently



- ▶ Cover two papers on Effect Systems.
  - ▶ Generalized Evidence Passing for Effect Handlers<sup>1</sup>
  - ▶ Effect Handlers in Haskell, Evidently<sup>2</sup>
- ▶ Some sections today's discussion isn't going to cover

---

<sup>1</sup>Generalized Evidence Passing for Effect Handlers

<sup>2</sup>Effect Handlers in Haskell, evidently



- ▶ Cover two papers on Effect Systems.
  - ▶ Generalized Evidence Passing for Effect Handlers<sup>1</sup>
  - ▶ Effect Handlers in Haskell, Evidently<sup>2</sup>
- ▶ Some sections today's discussion isn't going to cover
  - ▶ Efficiency/Performance of the library or effect system itself

---

<sup>1</sup>Generalized Evidence Passing for Effect Handlers

<sup>2</sup>Effect Handlers in Haskell, evidently



- ▶ Cover two papers on Effect Systems.
  - ▶ Generalized Evidence Passing for Effect Handlers<sup>1</sup>
  - ▶ Effect Handlers in Haskell, Evidently<sup>2</sup>
- ▶ Some sections today's discussion isn't going to cover
  - ▶ Efficiency/Performance of the library or effect system itself
  - ▶ How to use effect systems

---

<sup>1</sup>Generalized Evidence Passing for Effect Handlers

<sup>2</sup>Effect Handlers in Haskell, evidently



- ▶ Cover two papers on Effect Systems.
  - ▶ Generalized Evidence Passing for Effect Handlers<sup>1</sup>
  - ▶ Effect Handlers in Haskell, Evidently<sup>2</sup>
- ▶ Some sections today's discussion isn't going to cover
  - ▶ Efficiency/Performance of the library or effect system itself
  - ▶ How to use effect systems
  - ▶ Comparison of effect system libraries or how to choose one

---

<sup>1</sup>Generalized Evidence Passing for Effect Handlers

<sup>2</sup>Effect Handlers in Haskell, evidently



## Recap, what's it all about



- ▶ **Separate syntax from semantics**

## Recap, what's it all about



- ▶ **Separate syntax from semantics**
- ▶ **Interpret your abstract syntax tree in various ways**

## Recap, what's it all about



- ▶ **Separate syntax from semantics**
- ▶ **Interpret your abstract syntax tree in various ways**
- ▶ **Not losing performance while having both**

## Recap, why effect systems



- ▶ Monads to model effects but monads don't compose<sup>3</sup>

---

<sup>3</sup>Composing Monads by Mark Jones and Luc Duponcheel

## Recap, why effect systems



- ▶ Monads to model effects but monads don't compose<sup>3</sup>
- ▶ transformers/mtl has limitations

```
class Monad m => MonadReader r m | m -> r
```

---

<sup>3</sup>Composing Monads by Mark Jones and Luc Duponcheel

## Recap, why effect systems



- ▶ Monads to model effects but monads don't compose<sup>3</sup>
- ▶ transformers/mtl has limitations
  - ▶ Monad transformer stacks are rigid

```
class Monad m => MonadReader r m | m -> r
```

---

<sup>3</sup>Composing Monads by Mark Jones and Luc Duponcheel



- ▶ Monads to model effects but monads don't compose<sup>3</sup>
- ▶ transformers/mtl has limitations
  - ▶ Monad transformer stacks are rigid
  - ▶ Doesn't allow handling something like `Reader Int (Reader String)` due to functional dependencies

```
class Monad m => MonadReader r m | m -> r
```

---

<sup>3</sup>Composing Monads by Mark Jones and Luc Duponcheel



- ▶ Monads to model effects but monads don't compose<sup>3</sup>
  - ▶ transformers/mtl has limitations
    - ▶ Monad transformer stacks are rigid
    - ▶ Doesn't allow handling something like `Reader Int (Reader String)` due to functional dependencies
- ```
class Monad m => MonadReader r m | m -> r
```
- ▶ More than a few effects in stack become unwieldy

---

<sup>3</sup>Composing Monads by Mark Jones and Luc Duponcheel





- ▶ Monads to model effects but monads don't compose<sup>3</sup>
  - ▶ transformers/mtl has limitations
    - ▶ Monad transformer stacks are rigid
    - ▶ Doesn't allow handling something like `Reader Int (Reader String)` due to functional dependencies
- ```
class Monad m => MonadReader r m | m -> r
```
- ▶ More than a few effects in stack become unwieldy
  - ▶ n-square instances problem

---

<sup>3</sup>Composing Monads by Mark Jones and Luc Duponcheel

# Evidence passing based effect system libraries



▶ EvEff

# Evidence passing based effect system libraries



- ▶ EvEff
- ▶ MpEff

# Evidence passing based effect system libraries



- ▶ EvEff
- ▶ MpEff
- ▶ speff

# Evidence passing based effect system libraries



- ▶ EvEff
- ▶ MpEff
- ▶ speff
- ▶ cleff based on ReaderT IO

# Evidence passing based effect system libraries



- ▶ EvEff
- ▶ MpEff
- ▶ speff
- ▶ cleff based on ReaderT IO
- ▶ effectful based on ReaderT IO



- ▶ EvEff
- ▶ MpEff
- ▶ speff
- ▶ cleff based on ReaderT IO
- ▶ effectful based on ReaderT IO
- ▶ others?



- ▶ Many extensible effects libraries are implemented with `free(r)` monads (True)



## Some myths<sup>4</sup>



- ▶ Many extensible effects libraries are implemented with `free(r)` monads (True)
  - ▶ Therefore extensible effects = `free(r)` monads (False)
-

## Some myths<sup>4</sup>



- ▶ Many extensible effects libraries are implemented with `free(r)` monads (True)
  - ▶ Therefore extensible effects = `free(r)` monads (False)
  - ▶ `Free(r)` monads require certain mathematical concepts to grasp (True)
-



- ▶ Many extensible effects libraries are implemented with `free(r)` monads (True)
  - ▶ Therefore extensible effects = `free(r)` monads (False)
  - ▶ `Free(r)` monads require certain mathematical concepts to grasp (True)
  - ▶ `Free(r)` monads don't have very good performance (True, to some extent)
-



- ▶ Many extensible effects libraries are implemented with `free(r)` monads (True)
- ▶ Therefore extensible effects = `free(r)` monads (False)
- ▶ `Free(r)` monads require certain mathematical concepts to grasp (True)
- ▶ `Free(r)` monads don't have very good performance (True, to some extent)
- ▶ Therefore extensible effects are slow, ivory-towerish toys (False)

---

<sup>4</sup>ReaderT pattern is just extensible effects

Effects can be implemented in various way



- ▶ free monads

## Effects can be implemented in various way



- ▶ free monads
- ▶ ReaderT IO

## Effects can be implemented in various way



- ▶ free monads
- ▶ ReaderT IO
- ▶ CPS

## Effects can be implemented in various way



- ▶ free monads
- ▶ ReaderT IO
- ▶ CPS
- ▶ delimited continuations



## What's the gist



- ▶ How do you pass the handler for the effect?



- ▶ Languages that expose a yield primitive actually have a way to access delimited continuations! Central result of the paper by James-Sabry <sup>56</sup>.

---

<sup>5</sup>Yield: Mainstream Delimited Continuations

<sup>6</sup>Delimited Continuations are all you need

# WTH are delimited continuations



- ▶ Delimited Continuations for Everyone<sup>7</sup>

---

<sup>7</sup>Delimited Continuations for Everyone

## How does one define an effect



```
data Reader a e ans = Reader { ask :: !(Op () a e ans)
                               }
```

```
data State a e ans = State { get :: !(Op () a e ans)
                             , put :: !(Op a () e ans)
                             }
```

## Multi-prompt delimited control



```
data Ctl e a = Pure { result :: !a }
              | forall h b e' ans.
                Control {
  -- prompt marker to yield to (in type context `::ans`)
                marker :: Marker h e' ans,
  -- the final action, just needs the resumption (:: $b \rightarrow \text{Eff } e' \text{ ans}$ )
                op      :: !( $b \rightarrow \text{Eff } e' \text{ ans}$ )  $\rightarrow \text{Eff } e' \text{ ans}$ ),
  -- the (partially) build up resumption;
  -- ( $b \rightarrow \text{Eff } e' a$ )  $\sim$ : ( $b \rightarrow \text{Eff } e' \text{ ans}$ )` by the time
  -- we reach the prompt
                cont   :: !( $b \rightarrow \text{Eff } e' a$ ) }
```

```
data Context e where
```

```
  CCons :: !(Marker h e' ans)  $\rightarrow$  !(h e' ans)  $\rightarrow$  !(ContextT e e')
          $\rightarrow$  !(Context e)  $\rightarrow$  Context (h :* e)
```

```
  CNil  :: Context ()
```



- ▶ Multi Prompt

## Key ideas



- ▶ Multi Prompt
- ▶ Evidence Passing



- ▶ Multi Prompt
- ▶ Evidence Passing
- ▶ Tail Resumptive Operations





- ▶ Multi Prompt
- ▶ Evidence Passing
- ▶ Tail Resumptive Operations
- ▶ Bubbling Yields



- ▶ Multi Prompt
- ▶ Evidence Passing
- ▶ Tail Resumptive Operations
- ▶ Bubbling Yields
- ▶ Short cut resumptions



- ▶ Multi Prompt
- ▶ Evidence Passing
- ▶ Tail Resumptive Operations
- ▶ Bubbling Yields
- ▶ Short cut resumptions
- ▶ Monadic Translation



- ▶ Multi Prompt
- ▶ Evidence Passing
- ▶ Tail Resumptive Operations
- ▶ Bubbling Yields
- ▶ Short cut resumptions
- ▶ Monadic Translation
- ▶ Bind-inlining and Join-Point Sharing



- ▶ Dig in to the paper!



- ▶ Alexis King on “Delimited Continuations, Demystified” @ZuriHac2023



- ▶ Alexis King on “Delimited Continuations, Demystified” @ZuriHac2023
- ▶ GHC Proposal: Delimited continuation primops



- ▶ Alexis King on “Delimited Continuations, Demystified” @ZuriHac2023
- ▶ GHC Proposal: Delimited continuation primops
- ▶ Delimited Continuations





- ▶ Alexis King on “Delimited Continuations, Demystified” @ZuriHac2023
- ▶ GHC Proposal: Delimited continuation primops
- ▶ Delimited Continuations
- ▶ Efficient Compilation of Algebraic Effect Handlers - Ningning Xie



- ▶ Alexis King on “Delimited Continuations, Demystified” @ZuriHac2023
- ▶ GHC Proposal: Delimited continuation primops
- ▶ Delimited Continuations
- ▶ Efficient Compilation of Algebraic Effect Handlers - Ningning Xie
- ▶ From Folklore to Fact: Comparing Implementations of Stacks and Continuations



- ▶ Alexis King on “Delimited Continuations, Demystified” @ZuriHac2023
- ▶ GHC Proposal: Delimited continuation primops
- ▶ Delimited Continuations
- ▶ Efficient Compilation of Algebraic Effect Handlers - Ningning Xie
- ▶ From Folklore to Fact: Comparing Implementations of Stacks and Continuations
- ▶ Compiler and Runtime Support for Continuation Marks



- ▶ Alexis King on “Delimited Continuations, Demystified” @ZuriHac2023
- ▶ GHC Proposal: Delimited continuation primops
- ▶ Delimited Continuations
- ▶ Efficient Compilation of Algebraic Effect Handlers - Ningning Xie
- ▶ From Folklore to Fact: Comparing Implementations of Stacks and Continuations
- ▶ Compiler and Runtime Support for Continuation Marks
- ▶ Capturing the Future by Replaying the Past Functional Pearl



- ▶ Alexis King on “Delimited Continuations, Demystified” @ZuriHac2023
- ▶ GHC Proposal: Delimited continuation primops
- ▶ Delimited Continuations
- ▶ Efficient Compilation of Algebraic Effect Handlers - Ningning Xie
- ▶ From Folklore to Fact: Comparing Implementations of Stacks and Continuations
- ▶ Compiler and Runtime Support for Continuation Marks
- ▶ Capturing the Future by Replaying the Past Functional Pearl
- ▶ From Delimited Continuations to Algebraic Effects in Haskell



- ▶ Alexis King on “Delimited Continuations, Demystified” @ZuriHac2023
- ▶ GHC Proposal: Delimited continuation primops
- ▶ Delimited Continuations
- ▶ Efficient Compilation of Algebraic Effect Handlers - Ningning Xie
- ▶ From Folklore to Fact: Comparing Implementations of Stacks and Continuations
- ▶ Compiler and Runtime Support for Continuation Marks
- ▶ Capturing the Future by Replaying the Past Functional Pearl
- ▶ From Delimited Continuations to Algebraic Effects in Haskell
- ▶ Concurrent System Programming with Effect Handlers



- ▶ Alexis King on “Delimited Continuations, Demystified” @ZuriHac2023
- ▶ GHC Proposal: Delimited continuation primops
- ▶ Delimited Continuations
- ▶ Efficient Compilation of Algebraic Effect Handlers - Ningning Xie
- ▶ From Folklore to Fact: Comparing Implementations of Stacks and Continuations
- ▶ Compiler and Runtime Support for Continuation Marks
- ▶ Capturing the Future by Replaying the Past Functional Pearl
- ▶ From Delimited Continuations to Algebraic Effects in Haskell
- ▶ Concurrent System Programming with Effect Handlers
- ▶ Eff Directly in OCaml



- ▶ Alexis King on “Delimited Continuations, Demystified” @ZuriHac2023
- ▶ GHC Proposal: Delimited continuation primops
- ▶ Delimited Continuations
- ▶ Efficient Compilation of Algebraic Effect Handlers - Ningning Xie
- ▶ From Folklore to Fact: Comparing Implementations of Stacks and Continuations
- ▶ Compiler and Runtime Support for Continuation Marks
- ▶ Capturing the Future by Replaying the Past Functional Pearl
- ▶ From Delimited Continuations to Algebraic Effects in Haskell
- ▶ Concurrent System Programming with Effect Handlers
- ▶ Eff Directly in OCaml
- ▶ Retrofitting Effect Handlers onto OCaml



# Questions?



▶ Reach out on

# Questions?



- ▶ Reach out on
  - ▶ Email: [sanchayan@sanchayanmaity.net](mailto:sanchayan@sanchayanmaity.net)

# Questions?



- ▶ Reach out on
  - ▶ Email: [sanchayan@sanchayanmaity.net](mailto:sanchayan@sanchayanmaity.net)
  - ▶ Mastodon: <https://sanchayanmaity.com/@sanchayan>

# Questions?



- ▶ Reach out on
  - ▶ Email: [sanchayan@sanchayanmaity.net](mailto:sanchayan@sanchayanmaity.net)
  - ▶ Mastodon: <https://sanchayanmaity.com/@sanchayan>
  - ▶ Blog: <https://sanchayanmaity.net>

# Questions?



- ▶ Reach out on
  - ▶ Email: [sanchayan@sanchayanmaity.net](mailto:sanchayan@sanchayanmaity.net)
  - ▶ Mastodon: <https://sanchayanmaity.com/@sanchayan>
  - ▶ Blog: <https://sanchayanmaity.net>
  - ▶ Telegram:

# Questions?



- ▶ Reach out on
  - ▶ Email: [sanchayan@sanchayanmaity.net](mailto:sanchayan@sanchayanmaity.net)
  - ▶ Mastodon: <https://sanchayanmaity.com/@sanchayan>
  - ▶ Blog: <https://sanchayanmaity.net>
  - ▶ Telegram:
    - ▶ [t.me/fpncr](https://t.me/fpncr)

# Questions?



- ▶ Reach out on
  - ▶ Email: [sanchayan@sanchayanmaity.net](mailto:sanchayan@sanchayanmaity.net)
  - ▶ Mastodon: <https://sanchayanmaity.com/@sanchayan>
  - ▶ Blog: <https://sanchayanmaity.net>
  - ▶ Telegram:
    - ▶ [t.me/fpncr](https://t.me/fpncr)
    - ▶ [t.me/SanchayanMaity](https://t.me/SanchayanMaity)