Dive into *Algebraic Effects*

びしょ〜じょ

ML Days #2

September 16, 2018

■ やること

- Algebraic Effects を伝道
  - Algebraic Effects is 何
  - Algebraic Effects が使える言語
  - Algebraic Effects の活用事例
- 研究のご紹介

  先日 JSSST でポスター発表した内容を紹介シマス

■目次

Dive into *Algebraic Effects*

- 自己紹介
- Algebraic Effects
  - effect の型
  - デモ
  - ポイント
- AE Implimentations
  - Multicore OCaml
- 研究紹介
  - oneshot AE →
  - oneshot s/r
  - oneshot s/r → AC
  - 課題、今後の予定
- まとめ

■自己紹介

こんにちは、びしょ〜じょです。

- T 大学大学院で M1 をやっている
  プログラム言語や型とか検証などの研究室
- 少し前は Lua にお熱だった

Nymphium

Dive into *Algebraic Effects*

■Algebraic Effects

イメージとしては**継続を持てる例外**

■Algebraic Effects

イメージとしては**継続を持てる例外**
例 (Eff language implementation):

```
effect Say    : string → unit
effect Twice  : unit   → unit

handle (
  perform (Say "Hello");
  perform (Twice ());
  perform (Say "world")
) with
| x → x
| effect (Twice ()) k → k (); k ()
| effect (Say msg)  k →
    print_endline msg; k ()
```

■Algebraic Effects

イメージとしては**継続を持てる例外**
例 (Eff language implementation):

effect definition

```
effect Say    : string → unit
effect Twice  : unit   → unit

handle (
  perform (Say "Hello");
  perform (Twice ());
  perform (Say "world")
) with
| x → x
| effect (Twice ()) k → k (); k ()
| effect (Say msg)  k →
    print_endline msg; k ()
```

■Algebraic Effects

イメージとしては**継続を持てる例外**

例 (Eff language implementation):


```
effect Say   : string → unit
effect Twice : unit   → unit

handle (
  perform (Say "Hello");
  perform (Twice ());
  perform (Say "world")
) with
| x → x
| effect (Twice ()) k → k (); k ()
| effect (Say msg)  k →
    print_endline msg; k ()
```

■Algebraic Effects

イメージとしては**継続を持てる例外**
例 (Eff language implementation):

```
effect Say    : string → unit
effect Twice  : unit   → unit

handle (
   perform (Say "Hello");
   perform (Twice ());
   perform (Say "world")
) with
| x → x
| effect (Twice ()) k → k (); k ()
| effect (Say msg)  k →
    print_endline msg; k ()
```

effect definition

effect invocation

effect handler

■Algebraic Effects

イメージとしては**継続を持てる例外**

例 (Eff language implementation):

```
effect Say    : string → unit
effect Twice  : unit   → unit

handle (
   perform (Say "Hello");
   perform (Twice ());
   perform (Say "world")
) with
| x → x
| effect (Twice ()) k → k (); k ()
| effect (Say msg)  k →
    print_endline msg; k ()
```

effect definition

effect invocation

effect handler

continuation

●effectの型

なにこれ🤔

```
effect Say : string → unit
```

●effectの型

なにこれ🤔

```
effect Say : string → unit
```

例:

```
E[perform (Say "Hello")]
```

●effectの型

なにこれ🤔

```
effect Say : string → unit
```

例:

```
E[perform (Say "Hello")]
                  : string
```

●effectの型

なにこれ🤔

```
effect Say : string → unit
```

例:

```
E[perform (Say "Hello")]
                  : string
        : unit
```

●effectの型

なにこれ🤔

```
effect Say : string → unit
```

例:

```
        : unit → 'a (継続の型)
E[perform (Say "Hello")]
                  : string
        : unit
```

●effectの型

なにこれ🤔

```
effect Say : string → unit
```

例:

```
          : unit → 'a (継続の型)
E[perform (Say "Hello")]
                   : string
          : unit
```

完全に理解した

● デモ

```
effect Say    : string → unit
effect Twice  : unit   → unit

handle (
  perform (Say "Hello");
  perform (Twice ());
  perform (Say "world")
) with
| x → x
| effect (Twice ()) k → k (); k ()
| effect (Say msg)  k →
    print_endline msg; k ()
```

● ポイント

- 定義と実装の**分離**
- ハンドラの**合成**

} モジュラーなプログラミングを支援

- **限定継続**が使える

} 例外より強力

Dive into *Algebraic Effects*

■AE Implimentations

- Eff[*1]
  - ML ベースの syntax、型推論
  - 処理系やライブラリ (EDSL) など実装多数
- Koka[*2]
  - MS Research 製
  - effect が型で表される
    (ex: `println :a -> console ()`)
  - Algebraic Effects 以外にもおもしろ機能
- Multicore **OCaml**[*3]
  - OCaml Labs が OCaml を fork
  - 継続が oneshot の Algebraic Effets を持つ

[*1] https://www.eff-lang.org/

[*2] https://koka-lang.github.io/koka/

[*3] http://ocamllabs.io/doc/multicore.html

Multicore OCaml

State モナド風

```
module State(S : sig type t end)
= struct
  type t = S.t

  effect Put : t → unit;;
  effect Get : t

  let run init f =
    init |> match f () with
    | x → (fun s → (s, x))
    | effect (Put s') k →
      (fun s → continue k () s')
    | effect Get k →
      (fun s → continue k s s)
end

effect Log : int → unit
let log msg = perform @@ Log msg
```

```
try begin
  let module S1 =
    State(struct
      type t = int
    end) in
  let open S1 in
  let incr () =
   perform (Put (perform Get + 1))
  in
  run 0 @@ fun () →
      incr ();
      log @@ perform Get;
      incr ();
      log @@ perform Get
end
with effect (Log msg) k →
  print_int msg; continue k ()
```

●Multicore OCaml

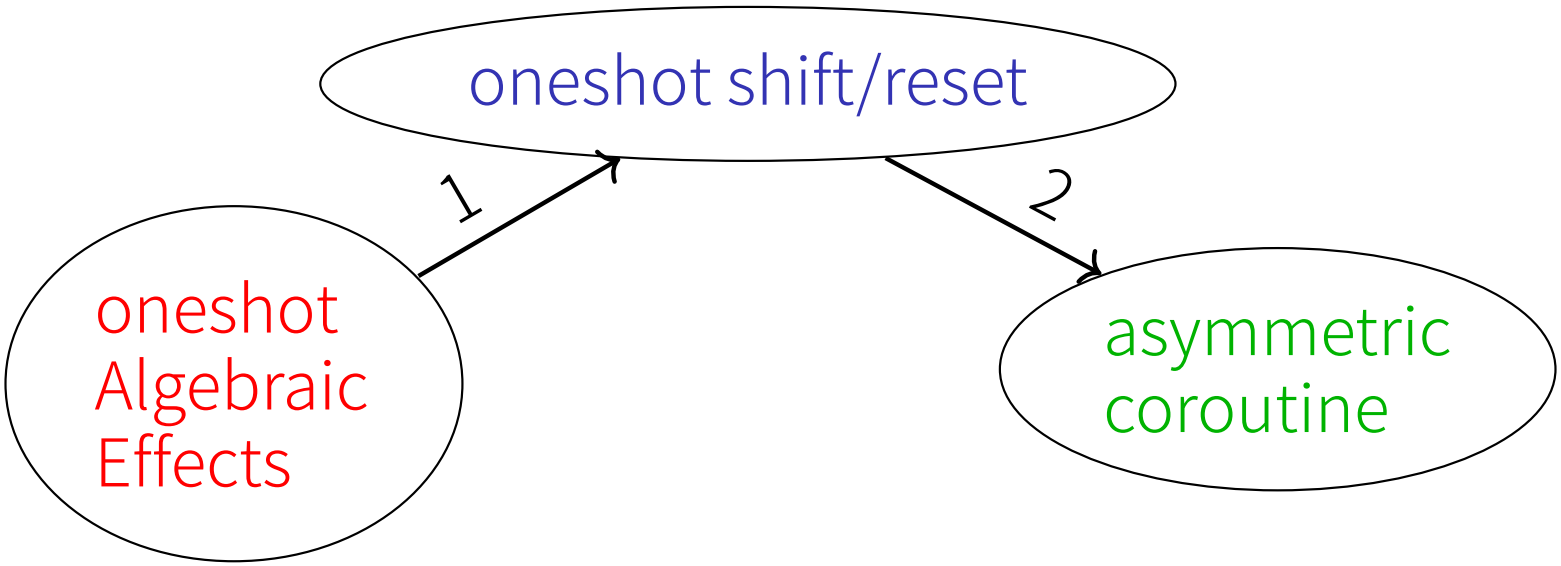
shift/reset も実装できるぞ!!

```
module DelimccOne = struct
  type 'a prompt = {
    take : 'b. (('b → 'a) → 'a) → 'b;
    push : (unit → 'a) → 'a;
  }

  let new_prompt (type a) : unit → a prompt = fun () →
    let module M = struct
      effect Prompt : (('b → a) → a) → 'b
    end in
    let take f = perform (M.Prompt f) in
    let push th =
      try th () with effect (M.Prompt f) k → f @@ continue k
    in
    {take; push}

  let push_prompt {push} = push
  let take_subcont {take} = take

  let shift0 p f = take_subcont p @@ fun k → f k
end
```

Dive into *Algebraic Effects*

■ 研究紹介

- Asymmetric Coroutine による Oneshot Algebraic Effects の実装 [*4]



- (asymmetric) coroutine のある言語で AE が使える
- 1 の類似の変換、2 の既存の変換を参考にする

[*4] http://logic.cs.tsukuba.ac.jp/~sat/pdf/jssst2018.pdf

oneshot Algebraic Effects → oneshot shift/reset

[KS16] に基づき先程の実装の逆をやる
直感としては effect instance→prompt,
effect invocation → shift, effect handler → reset

```
module Translate(D: DELIMCC) : sig
  type ('a, 'b) free
  type 'a thunk = unit → 'a
  val newi : unit → 'a D.prompt
  val op : ('a, 'b) free D.prompt → 'a → 'b
  val handler : ('g, 'g) free D.prompt →
    ('g → 'o) → ('g * ('g → 'o) → 'o) → 'g thunk → 'o
  val handle : ('a thunk → 'b) → 'a thunk → 'b
end = struct
  ......
end
```

Q. 継続の oneshotness を AE が引き継いでることの証明は?

A. まだ無い。affine type で型によって示す予定

oneshot shift/reset → Asymmetric Coroutine

[Usu17] による変換を使う (Lua による実装)

Lua の `coroutine` + 現在の thread を表す `current`

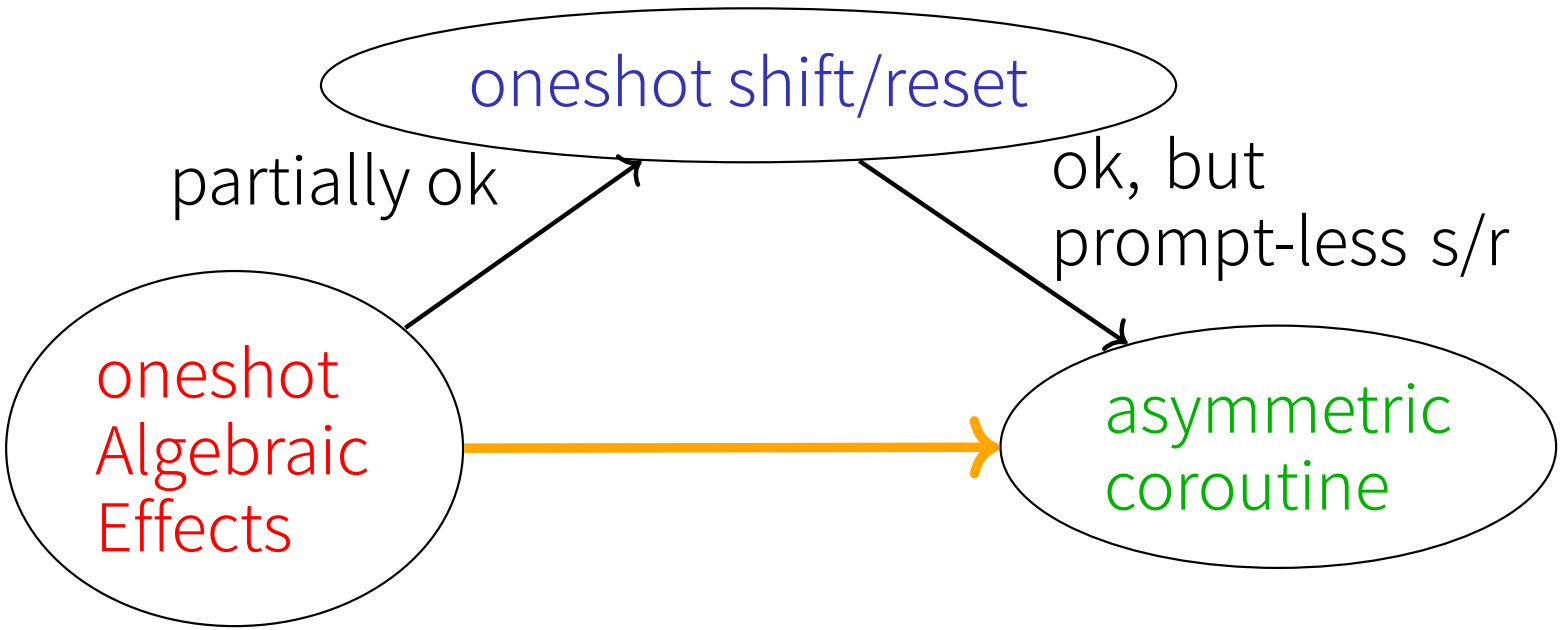
※ ただし prompt はない (shift が持つ限定継続は直前の reset)

```
function sr.reset(th)
  local l = coro.create(
    function(_)
      return (function(y)
        return function(_)
          return y
        end
      end)(th())
    end)

  return coro.resume(l)()
end
```

```
function sr.shift(f)
  local k = coro.current

  return coro.yield(function(_)
    return sr.reset(function()
      return f(k)
    end)
  end)
end

function sr.throw(k, e)
  return coro.resume(k, e)()
end
```

課題、今後の予定

- oneshotness の保証 (先述)
- Usui による変換の対象の shift/reset を multi-prompt で再定義
- oneshot shift/reset を経由しないダイレクトな変換を考える



Dive into *Algebraic Effects*

■まとめ

- 🌋 Algebraic Effects が楽しい
  ICFP 2018 や ML Workshop2018 にも
  AE 関連のトピック [*5][*6]
- ㊗ Algebraic Effects 使おう
  - 🛠 インプリいろいろ
  - 💪 なければ自作も可
- 👨‍💻 研究やってます

[*5] https://icfp18.sigplan.org/event/mlfamilyworkshop-2018-papers-programming-with-abstract-algebraic-effects

[*6] https://icfp18.sigplan.org/event/icfp-2018-papers-versatile-event-correlation-with-algebraic-effects

おわり

参考文献

[BP15] Andrej Bauer and Matija Pretnar. Programming with algebraic effects and handlers. In: *Journal of Logical and Algebraic Methods in Programming* 84.1 (2015), pp. 108–123.

[KS16] Oleg Kiselyov and KC Sivaramakrishnan. Eff directly in OCaml. In: *ML Workshop*. 2016.

[Usu17] Chiharu Usui. One-shot Delimited Continuations as Coroutines. MA thesis. University of Tsukuba, 2017.