

限定継続に着目した代数的効果から 非対称コルーチンへの変換

201820631 河原 悟

October 11, 2018

限定継続に着目した代数的効果から 非対称コルーチンへの変換

背景と動機

プログラム言語の理想と
現実
代数的効果

目的、内容

コルーチン

代数的効果

ワンショット

研究紹介

oneshot AE \rightarrow
oneshot s/r
oneshot s/r \rightarrow AC

応用

抽象化

ハンドラの合成

コントロールフロー操作

課題、今後の予定

● プログラム言語の理想と現実

- 理想 モジュール性の高さ
機能ごとに独立した部品 再利用可能性
- 現実 必要ドリブンの設計
言語の美しさは二の次

● プログラム言語の理想と現実

e.g.) Lua

- ▶ 組み込みでよく使われる比較的新しい言語
ゲーム、解析ツール、組版システムなど
- ▶ コルーチン
 - 😊 並行、非同期プログラミングを支援
 - 😓 コントロールの表現としては極めて低いレベル

● プログラム言語の理想と現実

e.g.) Lua

- ▶ 組み込みでよく使われる比較的新しい言語
ゲーム、解析ツール、組版システムなど
- ▶ コルーチン
 - 😊 並行、非同期プログラミングを支援
 - 😓 コントロールの表現としては極めて低いレベル

⇒ Lua でも“高レベルコントロール”を記述したい

● 代数的効果

Plotkin らにより圏論に基づいて 2003 年に提案
複雑な制御を構造化する機構を持つ

- ▶ 抽象化
- ▶ 合成

● 代数的効果

Plotkinらにより圏論に基づいて2003年に提案
複雑な制御を**構造化**する機構を持つ

- ▶ 抽象化
- ▶ 合成

⇒ Luaでも代数的効果を用いれば
複雑な制御を高レベルで記述できると予想

限定継続に着目した代数的効果から 非対称コルーチンへの変換

背景と動機

プログラム言語の理想と
現実
代数的効果

目的、内容

コルーチン

代数的効果

ワンショット

研究紹介

oneshot AE \rightarrow
oneshot s/r
oneshot s/r \rightarrow AC

応用

抽象化

ハンドラの合成

コントロールフロー操作

課題、今後の予定

■ 目的、内容

- ▶ 目的:
代数的効果を Lua に導入
- ▶ 鍵となる考察:
コルーチンは本質的にワンショット (状態コピー無し)
⇒ 代数的効果をワンショットに制限
- ▶ 研究内容:
ワンショットの代数的効果をプログラム変換により
Lua のコルーチンに変換する
より一般に、代数的効果とコルーチンの関係を明らかにする

限定継続に着目した代数的効果から 非対称コルーチンへの変換

背景と動機

プログラム言語の理想と
現実
代数的効果

目的、内容

コルーチン

代数的効果

ワンショット

研究紹介

oneshot AE \rightarrow
oneshot s/r
oneshot s/r \rightarrow AC

応用

抽象化
ハンドラの合成
コントロールフロー操作

課題、今後の予定

■ コルーチン

一時停止、リジュームのできるサブルーチン

- ▶ **非対称コルーチン**

- 一時停止するとリジューム元に戻る

- ▶ **対称コルーチン**

- 任意のコルーチンに移動できる

■ コルーチン

一時停止、リジュームのできるサブルーチン

▶ 非対称コルーチン

一時停止するとリジューム元に戻る

(main) ----->

Cor1 ----->

Cor2 ----->

▶ 対称コルーチン

任意のコルーチンに移動できる

■ コルーチン

一時停止、リジュームのできるサブルーチン

▶ 非対称コルーチン

一時停止するとリジューム元に戻る



▶ 対称コルーチン

任意のコルーチンに移動できる

■ コルーチン

一時停止、リジュームのできるサブルーチン

▶ 非対称コルーチン

一時停止するとリジューム元に戻る



▶ 対称コルーチン

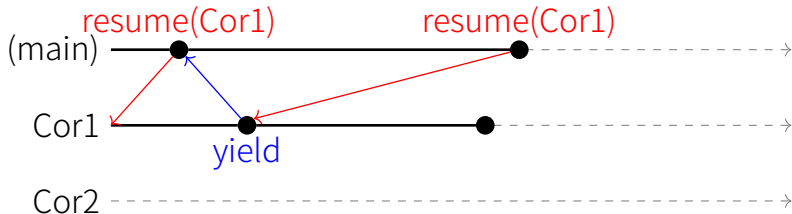
任意のコルーチンに移動できる

■ コルーチン

一時停止、リジュームのできるサブルーチン

▶ 非対称コルーチン

一時停止するとリジューム元に戻る



▶ 対称コルーチン

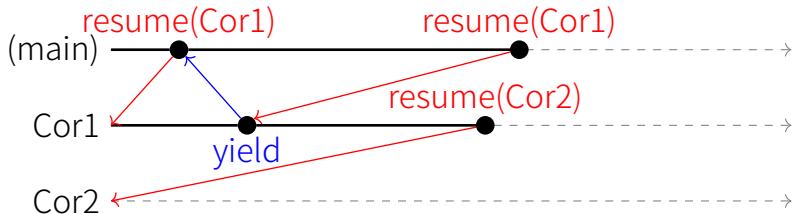
任意のコルーチンに移動できる

■ コルーチン

一時停止、リジュームのできるサブルーチン

▶ 非対称コルーチン

一時停止するとリジューム元に戻る



▶ 対称コルーチン

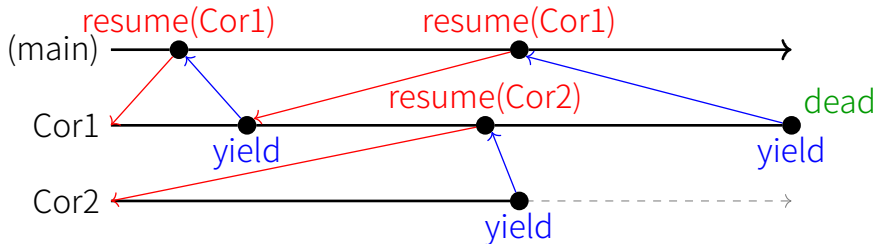
任意のコルーチンに移動できる

■ コルーチン

一時停止、リジュームのできるサブルーチン

▶ 非対称コルーチン

一時停止するとリジューム元に戻る



▶ 対称コルーチン

任意のコルーチンに移動できる

限定継続に着目した代数的効果から 非対称コルーチンへの変換

背景と動機

プログラム言語の理想と
現実
代数的効果

目的、内容

コルーチン

代数的効果

ワンショット

研究紹介

oneshot AE \rightarrow
oneshot s/r
oneshot s/r \rightarrow AC

応用

抽象化

ハンドラの合成

コントロールフロー操作

課題、今後の予定

■ 代数的効果

計算エフェクト (I/O, 例外, etc.) を代数的に扱う言語機能

- ▶ エフェクトの抽象化
- ▶ ハンドラの合成
- ▶ コントロールの状態 (継続) のキャプチャ

```
effect Print : string → unit

let print_string str =
  handle perform (Print str) with
  | effect (Print str) k →
    write stdout str; k ()
```

限定継続に着目した代数的効果から 非対称コルーチンへの変換

背景と動機

プログラム言語の理想と
現実
代数的効果

目的、内容

コルーチン

代数的効果

ワンショット

研究紹介

oneshot AE \rightarrow
oneshot s/r
oneshot s/r \rightarrow AC

応用

抽象化
ハンドラの合成
コントロールフロー操作

課題、今後の予定

■ ワンショット

リソースを使えるのは高々 1 回という性質

■ ワンショット

リソースを使えるのは高々 1 回という性質

- ▶ コルーチンはワンショット
コルーチンの状態のコピー操作は無い

■ ワンショット

リソースを使えるのは高々 1 回という性質

- ▶ コルーチンはワンショット
コルーチンの状態のコピー操作は無い
- ▶ 代数的効果におけるワンショット
キャプチャしたコントロールのリジュームは 1 回だけ
e.g.) Multicore OCaml

限定継続に着目した代数的効果から 非対称コルーチンへの変換

背景と動機

プログラム言語の理想と
現実
代数的効果

目的、内容

コルーチン

代数的効果

ワンショット

研究紹介

oneshot AE \rightarrow
oneshot s/r
oneshot s/r \rightarrow AC

応用

抽象化

ハンドラの合成

コントロールフロー操作

課題、今後の予定

■ 研究紹介

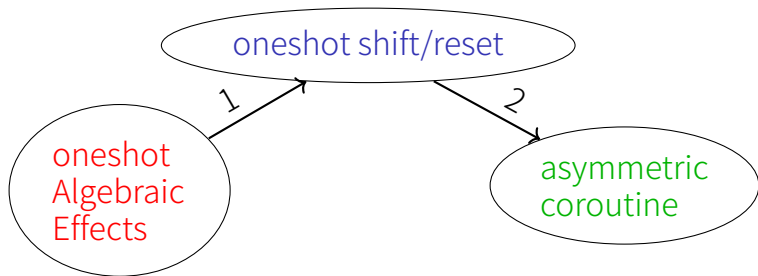
非対称コルーチンによるワンショットの代数的効果の実装

▶ shift/reset を経由

1. ワンショットの代数的効果からワンショットの shift/reset
2. ワンショットの shift/reset から非対称コルーチン

▶ ワンショットは保証しにくい動的な性質

⇒ 保守的近似として affine type system を導入



■ ワンショット代数的効果 → ワンショット shift/reset

[KS16] による変換 (ワンショット制限のない代数的効果
→ shift/reset) に基づいて実装した

TODO: 変換前の継続がワンショットならば
変換後もワンショットであることの証明

方針: 保守的近似である *affine type system* を
用いる

```
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
```

■ ワンショット shift/reset → 非対称コルーチン

[Usu17] による変換を用いる

問題点: 代数的効果 → shift/reset で使われる
shift/reset との互換がない

プロンプトが無い体系
(コントロールの型情報などを
保持するオブジェクト)

方針: Usui らの変換で対象となる shift/reset
をプロンプトのある体系で再定義する

限定継続に着目した代数的効果から 非対称コルーチンへの変換

背景と動機

プログラム言語の理想と
現実
代数的効果

目的、内容

コルーチン

代数的効果

ワンショット

研究紹介

oneshot AE \rightarrow
oneshot s/r
oneshot s/r \rightarrow AC

応用

抽象化

ハンドラの合成

コントロールフロー操作

課題、今後の予定

● 抽象化

実装をハンドラに任せる (e.g. 依存性の注入)

- エフェクトの発生 → インタフェース
- エフェクトハンドラ → 具体的な実装


```
let filter p =  
  let list = perform (GetAccountList ()) in  
  List.filter p list
```

● 抽象化

実装をハンドラに任せる (e.g. 依存性の注入)

- エフェクトの発生 → インタフェース
- エフェクトハンドラ → 具体的な実装

```
let filter p =  
  let list = perform (GetAccountList ()) in  
  List.filter p list
```




```
(* for test *)  
handler  
| effect GetAccountList k →  
  k test_db
```

● 抽象化

実装をハンドラに任せる (e.g. 依存性の注入)

- エフェクトの発生 → インタフェース
- エフェクトハンドラ → 具体的な実装

```
let filter p =  
  let list = perform (GetAccountList ()) in  
  List.filter p list
```



(* for test *)

handler

```
| effect GetAccountList k →  
  k test_db
```

(* for production *)

handler

```
| effect GetAccountList k →  
  k prod_db
```

● ハンドラの合成

ハンドルできるエフェクトを増やしたり、
特定のエフェクトの処理を変更できる
cf. monad, pimp my library pattern, open class

```
(* Foo をハンドル *)
```

```
let foohandler e =  
  handle e () with  
  | effect (Foo x) k → k x
```

```
(* foohandlerの取り逃がしたBarもハンドル *)
```

```
let foobarhandler e =  
  handle (foohandler e) with  
  | effect (Bar y) k → k y
```


● コントロールフロー操作

直接形式で簡潔に記述できる (cf. CPS, callback hell)

```
file_read_async filename (fun content →  
  traverse_text regexp content)
```

● コントロールフロー操作

直接形式で簡潔に記述できる (cf. CPS, callback hell)

```
handle
  let content = perform (RadAsync filename) in
  traverse_text regexp content
with
| effect (ReadAsync filename) k →
  read_async filename k
```

```
file_read_async filename (fun content →
  traverse_text regexp content)
```

限定継続に着目した代数的効果から 非対称コルーチンへの変換

背景と動機

プログラム言語の理想と
現実
代数的効果

目的、内容

コルーチン

代数的効果

ワンショット

研究紹介

oneshot AE \rightarrow
oneshot s/r
oneshot s/r \rightarrow AC

応用

抽象化

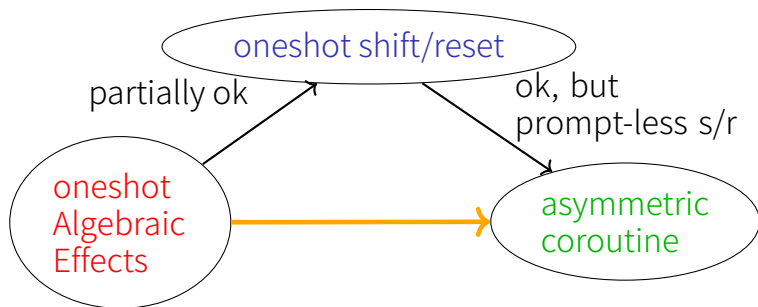
ハンドラの合成

コントロールフロー操作

課題、今後の予定

■ 課題、今後の予定

- ▶ ワンショット性の保証
affine types を用いて変換後のワンショット性を示す
- ▶ Usui による変換の対象の shift/reset をプロンプトありで再定義
- ▶ ワンショットの shift/reset を経由しない **ダイレクトな変換** を考える



■ 参考文献

- [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.