effectful subtyping

@型システム祭りオンライン

びしょ〜じょ

俺 is 誰

こんにちは、びしょ〜じょです。

🏢 株式会社 HERP

つくばでエンジニアリングしてる

先月まで代数的効果の研究してた

👤 Nymphium

この発表 is 何

エフェクトに
部分型があると
楽しい!!!

エフェクトに部分型のある代数的効果
https://nymphium.github.io/2019/12/22/effsub.html

代数的効果 is 何

説明しよう**代数的効果**とは近年研究者のみならずプログラマからも注目を集めつつある計算エフェクトをモジュラーに扱う新たな言語機能である。特徴を**端的に**説明するならば**復帰可能な例外**もう少し言うとエフェクトの発生位置からハンドラで区切られた部分までの限定継続をハンドラが利用することができるコントロールオペレータなのだ。

代数的効果 is 何

代数的効果は**復帰可能**な例外

エフェクトの定義

```
effect Error : () → int

let default_zero th =
 handle th () with
 | Error () k → k 0

handle_defer (fun () →
 (#Error ()) + 3)
```

代数的効果 is 何

代数的効果は**復帰可能**な例外

エフェクトの定義

```
effect Error : () → int

let default_zero th =
 handle th () with
 | Error () k → k 0

handle_defer (fun () →
 (#Error ()) + 3)
```

例外のようにエフェクトを発生、ハンドラが捕捉

代数的効果 is 何

代数的効果は**復帰可能**な例外

```
effect Error : () → int

let default_zero th =
 handle th () with
 | Error () k → k 0

handle_defer (fun () →
 (#Error ()) + 3)
```

エフェクトの定義

`[0] + 3`

例外のようにエフェクトを発生、ハンドラが捕捉
継続を利用してエフェクト発生位置から復帰!!

代数的効果 is 何

ほかにもいろいろ

- 実装をハンドラで記述するので……

  構造と見た目の分離ができる!

  例: DI、hook

- 継続が使えるので……

  ユーザレベルで
  コントロールフローを操作できる!

  例: 例外、async/await、shift/reset、
  バックトラッキング

例外ならば

いろんな例外を定義して
細かくコントロールしたい!

例外といえば

そういえば Java の例外には
継承を利用した階層があるよな🤔

- Exception
  - RuntimeException
    - ArithmeticException
  - IOException
  - ……

例外なので

代数的効果でも

継承のような順序関係を

導入したい!

例外なので

代数的効果でも
継承のような順序関係を
導入したい!

(Java などの) 例外	代数的効果
例外オブジェクト	エフェクト
(例外) オブジェクトに関する部分型	

例外なので

代数的効果でも
継承のような順序関係を
導入したい!

(Java などの) 例外	代数的効果
例外オブジェクト	エフェクト
(例外) オブジェクトに関する部分型	エフェクトに関する部分型?

$\lambda_{\sigma<:}$

1. **エフェクトに順序関係のある**
   代数的効果を持つ言語 $\lambda_{\sigma<:}$ を定義
2. 代数的効果の自作 Ruby ライブラリに導入

1. エフェクトに順序関係のある
   代数的効果を持つ言語 $\lambda_{\sigma<:}$ を定義
2. 代数的効果の自作 Ruby ライブラリに導入

$\lambda_{\sigma<:}$

$$
\begin{array}{rcl}
x & \in & \mathit{Variables} \\
\Sigma & ::= & \{\sigma_1 : \tau_1 \hookrightarrow \tau_1', \cdots\cdots, \sigma_l : \tau_l \hookrightarrow \tau_l'\} \\
\sigma & \in & \Sigma \\
 & & \\
v & ::= & x \mid h \mid \lambda x.e \mid \sigma \\
e & ::= & v \mid v\ v \mid \texttt{let}\ x = e\ \texttt{in}\ e \\
 & \mid & \texttt{with}\ v\ \texttt{handle}\ e \\
 & \mid & \texttt{perform}\ v\ v \\
h & ::= & \texttt{handler}\ v\ \ (\texttt{val}\ x \to e)\,((x, x) \to e) \\
 & & \\
\tau & ::= & \tau \to \underline{\tau} \mid \underline{\tau} \Rightarrow \underline{\tau} \\
\underline{\tau} & ::= & \tau ! \Delta \\
\Delta & ::= & \emptyset \mid \Delta, \sigma \\
\Gamma & ::= & \emptyset \mid \Gamma, (x : \tau)
\end{array}
$$

syntactic に目立つところはない

- ハンドラの型
  $\underline{\tau} \Rightarrow \underline{\tau}$
- エフェクトの集合
  Δ

エフェクト上の順序関係

エフェクト上の順序関係は予め定められている仮定

$$\frac{}{\sigma <: \sigma}\ (\text{S-Refl})$$

$$\frac{(\sigma_1 : \tau \hookrightarrow \tau') <: (\sigma_2 : \tau \hookrightarrow \tau') \qquad (\sigma_2 : \tau \hookrightarrow \tau') <: (\sigma_3 : \tau \hookrightarrow \tau')}{(\sigma_1 : \tau \hookrightarrow \tau') <: (\sigma_3 : \tau \hookrightarrow \tau')}\ (\text{S-Trans})$$

エフェクト上の順序関係であって型上の順序関係ではない!

エフェクト集合の部分型関係

$$\frac{\tau_1' \leqslant \tau_1 \qquad \underline{\tau_2} \leqslant \underline{\tau_2'}}{\tau_1 \to \underline{\tau_2} \leqslant \tau_1' \to \underline{\tau_2'}}\ (\text{S-FUN})$$

$$\frac{\underline{\tau_1'} \leqslant \underline{\tau_1} \qquad \underline{\tau_2} \leqslant \underline{\tau_2'}}{\underline{\tau_1} \Rightarrow \underline{\tau_2} \leqslant \underline{\tau_1'} \Rightarrow \underline{\tau_2'}}\ (\text{S-HANDLER})$$

$$\frac{}{\tau!\emptyset \leqslant \tau}\ (\text{S-PURE})$$

$$\frac{\tau \leqslant \tau' \qquad \forall \sigma \in \Delta.\exists \sigma' \in \Delta'.\sigma <: \sigma'}{\tau!\Delta \leqslant \tau'!\Delta'}\ (\text{S-DIRT})$$

- ハンドラは contravariant(S-Handler)
  - ハンドラが許容するエフェクトを過大近似
  - ハンドルされる項のエフェクトを最小化
- エフェクト自体は covariatn(S-Dirt)
  - 集合のサイズが小さく
  - 各エフェクトも小さい

型システム (一部)

$$\frac{\Gamma \vdash e : \underline{\tau} \qquad \underline{\tau'} \leqslant \underline{\tau}}{\Gamma \vdash e : \underline{\tau'}} \text{ (T-SubComp)}$$

$$\frac{\sigma = (\sigma : \tau_1 \hookrightarrow \tau_2) \qquad \sigma \in \Sigma \qquad \Gamma, (x : \tau) \vdash e : \tau'!\Delta \qquad \Gamma, (y : \tau_1, k : \tau_2 \to \tau'!\Delta) \vdash e' : \tau!\Delta}{\Gamma \vdash \texttt{handler}\ (\texttt{val}\ x \to e)\ ((y, k) \to e') : \tau!\Delta \cup \{\sigma\} \Rightarrow \tau'!\Delta} \text{ (T-Handler)}$$

$$\frac{\sigma = (\sigma : \tau_1 \hookrightarrow \tau_2) \qquad \sigma \in \Sigma \qquad \Gamma \vdash v : \tau_1}{\Gamma \vdash \texttt{perform}\ \sigma\ v : \tau_2!\{\sigma\}} \text{ (T-Perform)}$$

$$\frac{\Gamma \vdash h : \tau!\Delta \Rightarrow \tau'!\Delta' \qquad \Gamma \vdash e : \tau!\Delta}{\Gamma \vdash \texttt{with}\ h\ \texttt{handle}\ e : \tau'!\Delta'} \text{ (T-With)}$$

意味論(一部)

エフェクトに対応するハンドラを探すときにエフェクトの順序関係を利用する

$$((\texttt{with}\ w\ \texttt{handle}\ \square)^{\sigma} :: K)\ //\ \sigma' = ([]\,, (\texttt{with}\ w\ \texttt{handle}\ \square)^{\sigma}\,, K) \quad \text{iff}\ \sigma' <: \sigma$$

$$(F :: K)\ //\ \sigma = (F :: K', F', K'')$$

$$\text{where}\ F\ \neq (\texttt{with}\ w\ \texttt{handle}\ \square)^{\sigma}$$

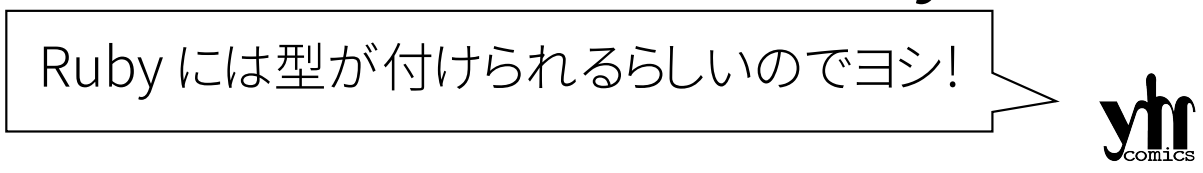
$$\text{and}\ (K', F', K'') = K\ //\ \sigma$$

$\lambda_{\sigma<:}$

1. **エフェクトに順序関係のある**
   代数的効果を持つ言語 $\lambda_{\sigma<:}$ を定義
2. 代数的効果の自作 Ruby ライブラリに導入

1. エフェクトに順序関係のある
   代数的効果を持つ言語 $\lambda_{\sigma<:}$ を定義
2. 代数的効果の自作 Ruby ライブラリに導入

1. エフェクトに順序関係のある代数的効果を持つ言語 $\lambda_{\sigma<:}$ を定義
2. 代数的効果の自作 Ruby ライブラリに導入



ライブラリ

```
Exception = Ruff::Effect.new
ZeroDivisionExceiption =
  Ruff::Effect.new Exception

handler =
  Ruff::Handler.new
    .on(ZeroDivisionExceipt) {
      puts 'ZeroDivisionExceiption'
    }
    .on(Exception) {
      puts 'Exception'
    }
```

ZeroDivisionExceiption <: Exception

部分型特有のやつは?

部分型といえば型情報が消えるやつで
すが……

```
(((pixel: Pixel) -> pixel)(colorPixel)).color // error
```

- エフェクトを発生させる側は……
  coariantなので情報は失われない
- ハンドラ側は……
  ハンドルされないエフェクトは詳細化されるので情報は失われない!

まとめ・課題

- エフェクトにも部分型のような関係があると嬉しい
- とりあえず順序関係を導入した
- いい感じなんで実装もした
- 🤔 多相性について
  列多相、有界量化、etc.

おわり

 Nymphium/ruff

もう少し詳細:

nymphium.github.io/2019/12/22/effsub.html