エフェクトに部分型のある代数的効果

びしょ〜じょ

2019 年 12 月 24 日

1　はじめに

例えば、代数的効果で例外を定義するときは Exception エフェクトのようなものを定義する。より細かい粒度の例外を定義したい場合、新たに ZeroDivisionException、 IndexOutOfBoundsException などを定義する。しかし、一般に例外は、例えばイベントループを内側で回す例外ハンドラなど、粗い粒度でハンドルしたい場合がある。 OCaml などの代数的な例外を持つ言語ではワイルドカードであらゆる例外を捕捉することができる。しかし代数的効果による例外のエミュレーションでワイルドカードを使ってしまうと、例外でない他のエフェクトも捕捉してしまい、うまく例外として動作させることが難しい。

本記事ではエフェクトが部分型を持つ代数的効果のある言語 $\lambda_{\sigma_{<:}}$ について述べる。エフェクトに部分型関係 (<:) を導入することで、上記で述べた問題を、あらゆる例外の親となるエフェクト型を定義することで、例外としてのエフェクトを全て捕捉し、なおかつ他のエフェクトと干渉しないハンドラを定義することができる。

$\lambda_{\sigma_{<:}}$ の定義に基づき、代数的効果を提供する Ruby 言語上のライブラリのエフェクトに部分型を導入した。

2　examples

エフェクトに部分型のある代数的効果を用いた例を、 Ruby 言語上に実装したライブラリ *Ruff*[*1]を用いて書く。

Java のような継承のあるオブジェクト指向言語では、例外もオブジェクトであり、継承関係を持つ。例えば Java では、 Exception が主な例外の基本型となり、それを継承する詳細な例外が作られ、階層を成していく。いま、部分型のある代数的効果を用いて階層のある例外を定義していく。はじめに、例外を表すエフェクト Exception を定義する。

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

Ruff は Ruffモジュールを提供し、そこにぶら下がるクラス等にアクセスできる。エフェクトオブジェクト Ruff::Effectのインスタンスを生成することで、エフェクトが作られる。次に、 Java の例外の階層に倣い、実行時例外を表す RuntimeException を以下に定義する。

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

[*1] 2019/12/22 現在、 master にまだマージしてない。 https://github.com/Nymphium/ruff/tree/subtyping

Ruff::Effectのコンストラクタにエフェクトインスタンスを渡すことで、渡されたエフェクトの部分型となるエフェクトを生成することができる。つまり、 Java のクラス階層同様に、 RunTimeException <: Exception という部分型関係がここで定義された。

あるエフェクトの部分型となるエフェクトから、さらに派生させることができる。

```
ZeroDivisionException = Ruff::Effect.new RunTimeException
IndexOutOfBoundsException = Ruff::Effect.new RunTimeException
```

RunTimeException同様に、 ZeroDivisionException <: RunTimeException、さらに部分型関係の遷移律に基づき、 ZeroDivisionException <: Exception という関係が成り立つ。

本章に続く例のため、 2 つのメソッド、 divと accessを定義する。 divは 2 つの引数から商を計算するメソッドである。第 2 引数が分母となり、 0の場合は ZeroDivisionExceptionを発生させる。

```
def div(x, y)
  if y.zero?
    ZeroDivisionException.perform
  else
    x / y
  end
end
```

Ruff におけるエフェクトの発生は、発生するエフェクトのオブジェクトのメソッド performを呼ぶことで表される。 accessは配列へのアクセスのラッパーメソッドであり配列外のインデックスへアクセスしようとすると IndexOutOfBoundsExceptionが発生する。

```
def access(arr, idx)
  if arr.length <= idx || idx.negative?
    IndexOutOfBoundsException.perform idx
  else
    arr[idx]
  end
end
```

閑話休題、あるエフェクトから派生したエフェクトは、元のエフェクトをハンドルするハンドラによって捕捉することができる。いま、 “例外” を粗い粒度で捕捉する rough_handlerを次のように定義する。

```
rough_handler = Ruff::Handler.new
  .on(RunTimeException) {
    puts 'RTE'
  }
```

Ruff は Ruff::Handlerがハンドラを表すクラスとなっており、 onメソッドでハンドラオブジェクトにエフェクトハンドラを設定できる。このハンドラは RunTimeExceptionエフェクトおよびそれから派生したエフェ

クトを捕捉できる。“例外” が捕捉されると、継続および引数は破棄され、 RTEを出力してハンドラのコントロールを抜ける。

```
rough_handler.run {
  div(10, 0) + 3
}
# ==> prints `RTE`
```

次は “例外” を細かい粒度でキャッチする acc_handlerを定義する。

```
acc_handler = Ruff::Handler.new
  .on(ZeroDivisionException) {
    puts 'ZeroDivisionException'
  }
  .on(IndexOutOfBoundsException) {|_k, x|
    puts "IndexOutOfBoundsException(#{x})"
  }
```

ZeroDivisionExceptionを捕捉した場合は ZeroDivisionExceiptと出力して終了し、IndexOutOfBoundsExceptionを捕捉した場合は、ハンドラの第 1 引数の継続を無視し、第 2 引数 x を文字列補間中に使い IndexOutOfException#{x}を出力して終了する。

```
acc_handler.run do
  div(10, 0)
end
# ==> prints `ZeroDivisionException`

acc_handler.run do
  arr = [1, 2]

  puts access(arr, 2)
end
# ==> prints `IndexOutOfException(2)`
```

3 $\lambda_{\sigma_{<:}}$

本章ではエフェクトに部分型のある代数的効果を持つ言語 $\lambda_{\sigma_{<:}}$ について説明する。 $\lambda_{\sigma_{<:}}$ はラムダ計算に加え、 let 式および代数的効果に関する構文を持つ言語である。構文および型システムは主に Effy 言語 [1] を参考にた。

3.1 構文

$\lambda_{\sigma_{<:}}$ の構文を Figure 1に示す。 エフェクトの集合 Σ は前もって、エフェクト間の部分型関係も含めて定義

$$
\begin{array}{rcl}
x & \in & \textit{Variables} \\
\Sigma & ::= & \{\sigma_1 : \tau_1 \hookrightarrow \tau_1', \sigma_2 : \tau_2 \hookrightarrow \tau_2', \cdots\cdots \sigma_l : \tau_l \hookrightarrow \tau_l'\} \\
\sigma & \in & \Sigma \\
 & & \\
v & ::= & x \mid h \mid \lambda x.e \mid \sigma \\
e & ::= & v \mid v\ v \mid \texttt{let}\ x = e\ \texttt{in}\ e \\
 & \mid & \texttt{with}\ v\ \texttt{handle}\ e \\
 & \mid & \texttt{perform}\ v\ v \\
h & ::= & \texttt{handler}\ v\ (\texttt{val}\ x \to e)\ ((x, x) \to e) \\
 & & \\
\tau & ::= & \tau \to \underline{\tau} \mid \underline{\tau} \Rightarrow \underline{\tau} \\
\underline{\tau} & ::= & \tau!\Delta \\
\Delta & ::= & \emptyset \mid \Delta, \sigma \\
\Gamma & ::= & \emptyset \mid \Gamma, (x : \tau)
\end{array}
$$

図 1　$\lambda_{\sigma_{<:}}$ の構文

されているものと仮定する。これはモデルを簡単にするためで、実際の実装では、エフェクトの生成および部分型関係の定義はユーザによって動的におこなうことができる。エフェクトの型は $\tau \hookrightarrow \tau'$ で表され、エフェクト発生時に τ 型の引数を受け取り、限定継続に τ' を渡す。

代数的効果のコンポーネントは他に、ハンドラを生成する構文、ハンドル式、エフェクトの発生の 3 つがある。 $\texttt{handler}\ \sigma\ (\texttt{val}\ x \to e)\ ((y, k) \to e')$ はエフェクト σ に対するハンドラを作る。 $(y, k) \to e'$ は σ に対するエフェクトハンドラである。 σ を捕捉したとき、エフェクトの引数を y に、エフェクト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 k に束縛し、 e' を評価する。 $\texttt{val}\ x \to e$ はハンドルされている式全体に対する値ハンドラ (value handler) である。ハンドルされている式が値を返した場合、その値を x に束縛して e を評価する。 $\lambda_{\sigma_{<:}}$ では、エフェクトの生成と同様に簡単のため、 1 つのハンドラは 1 つのエフェクトしかハンドルできない。この制約は、やはり同様に実装の段階では取り払われている。 $\texttt{with}\ h\ \texttt{handle}\ e$ はハンドル式であり、ハンドラ h のもとで e を評価する。エフェクトの発生は $\texttt{perform}\ \sigma\ v$ で表され、エフェクト σ を発生し、引数に v が渡される。

型は関数型 $\tau \to \underline{\tau'}$、ハンドラ型 $\underline{\tau} \Rightarrow \underline{\tau'}$ から成る。 $\underline{\tau}$ は $\tau!\Delta$ のエイリアスであり、エフェクト (の集合)Δ が発生しうる τ を返す。ハンドラ型 $\tau!\Delta \Rightarrow \tau'!\Delta'$ は、 $\tau!\Delta$ 型の式をハンドルし、 Δ' がハンドル式から計算中に発生しうる τ' 型の値を返す。

3.2 型システム

3.2.1 部分型

$\lambda_{\sigma_{<:}}$ には 2 種類の部分型がある。はじめに、本記事で導入するエフェクト上の部分型関係 (<:) を Figure 2 に示す。 改めて、 $\lambda_{\sigma_{<:}}$ においてこの部分型関係は予め定義されているものとする。また、注意したいのはエ

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

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

図 2　$\lambda_{\sigma_{<:}}$ の部分型 (1): エフェク上の部分型関係

フェクト自体に対する部分型関係であり、エフェクトの引数は部分型関係を持たないことである (S-Refl は自明なので型を省略した)。 section 2で述べた `Exception` とその部分型に関する関係は、これら規則により満たされていることが確認できる。

もう 1 つの部分型関係 ($\leqslant$) は、代数的効果を持つ言語に多く見られる、エフェクトの集合に着目した型上の関係である (Figure 3)。 S-Handler は、ハンドルされる式が発生しうるエフェクトを過大近似し ($\underline{\tau_1'} \leqslant \underline{\tau_1}$)、

$$\frac{\tau_1' \leqslant \tau_1 \qquad \underline{\tau_2} \leqslant \underline{\tau_2'}}{\tau_1 \to \underline{\tau_2} \leqslant \tau_1' \to \underline{\tau_2'}}\ \text{(S-Fun)} \qquad \frac{\underline{\tau_1'} \leqslant \underline{\tau_1} \qquad \underline{\tau_2} \leqslant \underline{\tau_2'}}{\underline{\tau_1} \Rightarrow \underline{\tau_2} \leqslant \underline{\tau_1'} \Rightarrow \underline{\tau_2'}}\ \text{(S-Handler)} \qquad \frac{}{\tau!\emptyset \leqslant \tau}\ \text{(S-Pure)}$$

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

図 3　$\lambda_{\sigma_{<:}}$ の部分型 (2): エフェクトの集合に対する subtyping

ハンドラによって取り除かれる型を過小近似する ($\underline{\tau_2} \leqslant \underline{\tau_2'}$)。 S-Pure は、純粋な型 τ は空の Dirt の付いた型 $\tau!\emptyset$ の基本型であることを示している。 Effy 言語などでは `val` : $\forall \tau. \tau \to \tau!\emptyset$ 演算子などが提供されているが、 $\lambda_{\sigma_{<:}}$ では部分型関係による暗示的なダウンキャストをおこなう。 S-Dirt は、エフェクトの部分型関係を考慮した、発生しうるエフェクトの集合のサイズに着目した部分型関係である。 Effy 言語では Figure 4のような定義となっている。 我々は、 Δ の全ての要素 σ が、 Δ' のある要素 σ' の部分型になっていることを以て、

$$\frac{\tau \leqslant \tau' \qquad \Delta \subseteq \Delta'}{\tau!\Delta \leqslant \tau'!\Delta'} \qquad \text{(S-!)}$$

図 4　Effy の部分型関係の一部 ([1] より引用、一部改変)

Δ が Δ' の部分集合であることを示している。

3.2.2　型システム全体

$\lambda_{\sigma_{<:}}$ の型システムを Figure 5に示す。 T-Let は、束縛される項で発生しうるエフェクト Δ とボディで発生しうるエフェクト Δ' の和 $\Delta \cup \Delta'$ を、 `let` 式全体で発生しうるエフェクトと考える。 T-SubPure および

$$\frac{(x:\tau)\in\Gamma}{\Gamma\vdash x:\tau}\ \text{(T-Var)} \qquad \frac{\Gamma\vdash e:\tau!\Delta \qquad \Gamma,(x:\tau)\vdash e':\tau'!\Delta'}{\Gamma\vdash \mathtt{let}\ x=e\ \mathtt{in}\ e':\tau'!(\Delta\cup\Delta')}\ \text{(T-Let)}$$

$$\frac{\Gamma,(x:\tau)\vdash e:\underline{\tau'}}{\Gamma\vdash \lambda x.e:\tau\to\underline{\tau'}}\ \text{(T-Fun)} \qquad \frac{\Gamma\vdash v_1:\tau\to\underline{\tau'} \qquad \Gamma\vdash v_2:\tau}{\Gamma\vdash v_1\ v_2:\underline{\tau'}}\ \text{(T-App)}$$

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

$$\sigma=(\sigma:\tau_1\hookrightarrow\tau_2) \qquad \sigma\in\Sigma$$

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

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

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

図 5　$\lambda_{\sigma_{<:}}$ の型システム

T-SubComp は発生しうるエフェクトに関するダウンキャストをおこなう。 T-Handler はハンドラに関する規則である。値ハンドラとエフェクトハンドラの戻り値の型は同じ $\tau'!\Delta$ である。継続の answer type はハンドル式の戻り値の型と同じになるので、値ハンドラの戻り値の型同様に $\tau'!\Delta$ である。 T-Perform は、エフェクト $\sigma:\tau_1\hookrightarrow\tau_2$ に引数 $v:\tau_1$ を渡して発生させたとき、継続の hole が、エフェクト σ が発生しうる τ_2 型になることを示す。 T-With は、 $\tau!\Delta$ 型の式をハンドルし、 $\tau'!\Delta'$ を返す。このとき e の型は適宜部分型により小さくなる。

3.3　意味論

意味論は代数的効果を持つ値呼びの体系として、素直に与えられていると仮定する。 CEK マシンに基づいた抽象機械の状態遷移による小ステップ意味論の一部を Figure 6に示す。 $K\,/\!/\,\sigma$ はスタックフレーム K

$$\frac{\sigma<:\sigma' \qquad K\,/\!/\,\sigma=\left(K',(\mathtt{with}\ w_h\ \mathtt{handle}\ \square)^{\sigma'},K''\right) \qquad \text{where } w_h=\mathtt{closh}\left(\mathtt{handler}\ \sigma'\ (\mathtt{val}\ x\to e)\ ((y,k)\to e'),E'\right)}{\langle w;\,E;\,(\mathtt{perform}\ \sigma\ \square)::K\rangle\longrightarrow\left\langle\begin{array}{c}e';\\(y=w)::(k=K'*E)::E';\\(\mathtt{with}\ w_h\ \mathtt{handle}\ \square)^{\sigma'}::K''\end{array}\right\rangle}\ \text{(E-Handle}\,\sigma\text{)}$$

図 6　$\lambda_{\sigma_{<:}}$ の意味論 (一部)

からエフェクト σ に対応するハンドラでハンドルするフレームと、その前後のスタックフレームから成る

3 つ組を返す補助関数である。 $\texttt{closh}\,(h, E)$ は環境 E を close したハンドラ h のランタイム表現である。 $(\texttt{with}\ w\ \texttt{handle}\ \square)^{\sigma}$ はエフェクト σ に対応するハンドラ w で式をハンドルするフレームである。

E-Handleσ は発生したエフェクト σ に対し、 σ の基本型となる σ' のハンドラによりハンドルされる。 E-Handleσ 規則中の $\sigma <: \sigma'$ は補助関数 (//) の結果に対するアサーションである。補助関数 (//) の定義を Figure 7に示す。

$$
\begin{aligned}
((\texttt{with}\ w\ \texttt{handle}\ \square)^{\sigma} :: K)\ //\ \sigma' &= ([]\,, (\texttt{with}\ w\ \texttt{handle}\ \square)^{\sigma}\,, K) \quad \text{iff } \sigma' <: \sigma \\
(F :: K)\ //\ \sigma &= (F :: K', F', K'') \\
&\quad \text{where } F\ \neq (\texttt{with}\ w\ \texttt{handle}\ \square)^{\sigma} \\
&\quad \text{and } (K', F', K'') = K\ //\ \sigma
\end{aligned}
$$

図 7　補助関数 (//)

スタックフレーム K のトップを先頭から走査し、 σ' の基本型となる σ に対応するハンドラを用いたハンドル式を取得する。ここで、 S-Refl 規則より σ' に対応するハンドラも取得されうることを改めて述べておく。

3.4　健全性とか

読者の課題とするで

4　実装

Figure 6に基づき、 Ruby 言語上の代数的効果ライブラリ *Ruff* に対し、エフェクトに関する部分型を実装した。 $\lambda_{\sigma <:}$ は 1 つのハンドラは 1 つのエフェクトしかハンドルできなかったが、実装ではその制約は取り除かれている。 1 つのハンドラが部分型関係にある複数のエフェクトハンドラは、ハンドルできるエフェクトが発生した場合、ハンドラに登録されているもののうち**最小**のエフェクトとそれに対応するハンドラが選択される。 section 2の Exception の階層を用いて例を示す。 ZeroDivisionExceiptionと Exceptionをハンドルする partacc_handlerを定義する。

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

このハンドラのもとで ZeroDivisionExceiptionを発生させた場合、 partacc_handlerは ZeroDivisonException の基本型である ZeroDivisionException(∵ S-Refl) もしくは Exception(∵S-Trans) によってハンド

ルできる。この 2 つのうち**最小**のエフェクトとは、部分型における親子関係のターミノロジーを用いれば、**最も若い世代**のエフェクトである。したがって、 `ZeroDivisionException`のハンドラが使われ、`ZeroDivisonException`を出力する。

5　おわりに

本記事では、エフェクトに部分型のある代数的効果を持つ言語 $\lambda_{\sigma_{<:}}$ を定義した。また、 $\lambda_{\sigma_{<:}}$ に基づき、Ruby 言語上の代数的効果ライブラリに部分型を追加した。エフェクトに部分型を導入することで、例外のような階層のある計算エフェクトを代数的効果で自然にエンコードすることができる。代数的効果がさらに実用的な言語機能として発展していくことを望む。

謝辞

エフェクトに部分型が星井〜というアイデアは `@ryotakameoka` 氏によるものです。面白い題材をくださってありがとうございます。

References

[1] Pretnar, Matija and Saleh, Amr Hany and Faes, Axel and Schrijvers, Tom. “Efficient compilation of algebraic effects and handlers”. In: *CW Reports, volume CW708* 32 (2017).