5M-03

代数的エフェクトとハンドラにおける Higher-Order Effects を扱える言語の実装の試み

今村　洸陽 †　　中才　恵太朗 †
大阪公立大学工業高等専門学校　総合工学システム学科 †

1　はじめに

代数的エフェクトとハンドラ [1] は副作用を含むプログラムに対する新しい手法として近年注目されている．代数的エフェクトとハンドラが持つ利点の一つが，副作用を発生させるプログラムの記述を変えることなく副作用の動作を変更することが可能であるということである．しかし，エフェクトを発生させる計算自体を引数とする Higher-Order Effects を直接扱うことはできない．本研究では，Higher-Order Effects を直接扱い代数的エフェクトと同程度のモジュール性を持ったプログラムを記述できるような言語の実装を試みる．

2　代数的エフェクトとハンドラ

代数的エフェクトとハンドラは，プログラムの実行中に発生する副作用を型レベルで扱うための言語機能である．例として以下に疑似プログラムとその型を示す．[2]

```
hello: Unit!Output
hello = out "hello"; out "world"
```

`out` は `Output` エフェクトに属する `String -> Unit!Output` 型のオペレーションであり，文字列の出力を表す．`hello` では`"hello"`と`"world"`を引数として `out` オペレーションを呼び出すことで `Output` エフェクトを発生させている．これは全体の型である `Unit!Output` にも表れている．

実際に `hello` を利用するにはエフェクトに意味を与えるハンドラが必要である．

```
hOut = f -> handler {
    x => (x, ""),
    (out s, k) => let (x, s') = k unit in
        (x, (f s) + " " + s'),
}
```

```
hOut1: T!(Output, E) => (T, String)!E
hOut1 = hOut (s -> s)
```

`hOut1` の型より，このハンドラは `Output` エフェクトをハンドルし，任意の `T` 型の値を`(T, String)`型に変換するハンドラだとわかる．

最後に，`hOut1` を用いて `hello` の `Output` エフェクトがハンドルすることにより，単なる`(Unit, String)`型の値を得ることができる．

```
handle hello with hOut1
    --> (unit, "hello world")
```

また，代数的エフェクトとハンドラの重要な利点として，`hello` の記述を変更することなくその振る舞いを変えることが可能だというものがある．

```
hOut2 = hOut (s ->
    if s = "hello" then "goodbye" else s)

handle hello with hOut2
    --> (unit, "goodbye world")
```

しかし，代数的エフェクトとハンドラでは直接実装できないようなエフェクトが存在する．例えば，`(String -> String) -> T!(Output, E) -> T!(Output, E)`型の `censor` をオペレーションとしてもつ `Censor` エフェクトと，`String -> String` 型の値 `f` を用いて書かれた以下のプログラムを考える．

```
censor f hello: Unit!Output
```

`censor f hello` は `f` を用いて `hello` 中の `out s` を `out (f s)`に変換したものを意味する．しかし，単純な代数的エフェクトとハンドラではこのような高階エフェクトは実装できない．本研究では，このような高階エフェクトを直接扱うことのできる言語の作成を試みる．

3　$F_!$言語

言語の名前を$F_!$とし，変数のメタ変数をxとする．項tを 図 1，型Tを 図 2，カインドKを 図 3 で定義する．

An Attempt to Implement a Language Capable of Treating Higher-Order Effects in Algebraic Effects and Handlers
† Imamura Hiroya, Department of Technological Systems, Osaka Metropolitan University College of Technology
† Nakasai Keitaro, Department of Technological Systems, Osaka Metropolitan University College of Technology

$$
\begin{aligned}
t ::= \; & x \\
& \lambda x : T.t \\
& t\ t \\
& \lambda x : K.t \\
& t\ T \\
& T\ t \\
& T\ t \ggg t \\
& t \rhd \{* \to t, T \to t, ..., T \to t\}
\end{aligned}
$$

図 1　項の定義

$$
\begin{aligned}
T ::= \; & x \\
& \lambda x : K.T \\
& T\ T \\
& T \to T!T \\
& x : K \to T!T \\
& T \rightsquigarrow T \\
& \{T, ..., T\} \\
& T \cup T \\
& T \setminus T
\end{aligned}
$$

図 2　型の定義

$$
\begin{aligned}
K ::= \; & * \\
& ! \\
& \{K\} \\
& K \to K
\end{aligned}
$$

図 3　カインドの定義

エフェクトとオペレーションは同一視され，そのカインドは!である．また，$\{T, ..., T\}$は型の集合を表し，$T \cup T$で和集合，$T \setminus T$で差集合を得られる．

$$\frac{\Gamma \vdash T_1 : * \quad \Gamma \vdash T_2 : *}{\Gamma \vdash T_1 \rightsquigarrow T_2 : !}$$

$$\frac{\forall i. \Gamma \vdash T_i : K}{\Gamma \vdash \{T_1, ..., T_n\} : \{K\}}$$

型付け関係は$\Gamma \perp t : T_1!T_2$のように表され，項tは文脈Γの下で型T_1を持ちエフェクトT_2を発生させることを表す．エフェクトとオペレーションを同一視し，$T\ t$はエフェクトTを引数tで発生させることを表す．

$$\frac{\Gamma \vdash T : ! \quad \Gamma \vdash t : T_1!T_0 \quad T \equiv T_1 \rightsquigarrow T_2}{\Gamma \vdash T\ t : T_2!(T_0 \cup \{T\})}$$

4　プログラム例

$F_!$での `censor` の実装例を以下に示す．ただし，基本的な値や型はすでに定義されているものとする．

```
let censor = λT : ∗ .λE : {!}.
  (String → String !{}, Unit → T!E)
   ⇝ (Unit → T!E) in
let hCensor =
  λT1 : ∗ .λE1 : {!}.
  λT2 : ∗ .λE2 : {!}.λm : Unit → T2!E2.
  m unit ▷ {
    ∗ →
      λr : X : ∗ → T2!(E2 \ {catch T1 E1}).r Unit
    catch T1 E1 →
      λp : (String → String, Unit → T1!E1).
      λk : (Unit → T1!E1)
         → T2!(E2 \ {catch T1 E1}).
        k (λ_ : Unit.
           let (f, m) = p in
           let (x, s) = hOut T1 E1 m in
             out (f s); x)
  } in unit
```

5　おわりに

本研究では，代数的エフェクトとハンドラにおける高階エフェクトを直接扱えるような言語の作成を試みた．今後の課題としては，型付けやカインド付けが正しく機能しているかといった安全性を証明する必要がある．また，現在の仕様ではエフェクトの弱化が出来ないため，部分型付けに対応する必要もある．

参考文献

[1] Gordon Plotkin and Matija Pretnar, “Handlers of algebraic effects”, *European Symposium on Programming*, pp. 80–94, 2009.

[2] Casper Bach Poulsen and Cas van der Rest, “Hefty algebras: Modular elaboration of higher-order algebraic effects”, *Proceedings of the ACM on Programming Languages*, vol. 7, pp. 1801–1831, 2023.