代数的エフェクトとハンドラのための実装理論

理学専攻・情報科学コース　藤井 舞花

1　はじめに

限定継続 (ある時点における範囲が限定された残りの計算) を明示的に扱える言語機構のひとつである代数的エフェクトとハンドラ [1] を取り上げる。代数的エフェクトとハンドラは例外処理を一般化したものである。従来の例外処理は副作用が発生するとハンドラに事後処理を任せ、オペレーションが発生した時点での継続は捨てる。一方で代数的エフェクトとハンドラはオペレーションが発生した時点における限定継続も用いて処理を行うことができる。つまりユーザー自身が自由に定義できる大域ジャンプが可能となる。代数的エフェクトとハンドラのハンドラには深いもの (deep なハンドラ) と浅いもの (shallow なハンドラ) が存在する。この 2 種類のハンドラは限定継続の扱い方に違いがあるため、同じプログラムでも異なる場所に移動することが可能となる。

他にも限定継続を扱う演算子としての 4 種類の限定継続演算子なども存在する。この 4 つの限定継続演算子の中で $\mathrm{shift}_0/\mathrm{reset}_0$ は deep なハンドラと、$\mathrm{control}_0/\mathrm{prompt}_0$ は shallow なハンドラと限定継続の扱い方に関してとても近い関係性がある [4, 7]。しかし代数的エフェクトとハンドラと、4 種類の限定継続演算子は全く異なる定式化でそれぞれが議論されてきた。これまでの研究では 4 種類の限定継続演算子は、呼び出し文脈 (trail) [2] とメタ継続があれば各種の演算子のインタプリタが表現できる。 呼び出し文脈をリストではなく高階関数を用いて実装するため、呼び出し文脈の型が変化するようなプログラムの例でも型をつけることができる。さらに高階関数の形でインタプリタが定義されていることで、4 種類の限定継続演算子は正当性が保証された変換を施せば、インタプリタから抽象機械や仮想機械 [5] が導出できた。正当性が保証された変換を使うことで、別途証明を与えることなく抽象機械や仮想機械が正しいことが保証される。そして型システム [6] もインタプリタから機械的に導けており、停止性や健全性が示された。対して代数的エフェクトとハンドラはインタプリタや抽象機械、型システムが定義されているもののこれらは独立して定義される。

そのため本研究では代数的エフェクトとハンドラのためのインタプリタも、呼び出し文脈やメタ継続を用いて再定義することで実装理論を統一的にし、別途証明を与えずとも正当性が保証された代数的エフェクトとハンドラの直接実装を示す。そして代数的エフェクトとハンドラと、4 種類の限定継続演算子両者を同じ枠組みで議論するための第一歩とする。

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副作用を起こす演算子とそれを処理する演算子からなる。通常の例外処理とは異なり、オペレーションを起こした時点におけるハンドラで限定された継続を副作用時に扱うことができる。本研究では *try* ... *with* をハンドラとし、*call* とすることでオペレーション呼び出しをする。

具体的に $\underline{try}\, e_0\, \underline{with}\, (x; k).\, e_1$ と記述した式は、e_0 で $\underline{call}(a)$ とオペレーション呼び出しが行われると、a が x に、オペレーション呼び出しをした時点から *try* と *with* で限定された部分までの継続が k に代入され、e_1 を実行する。式 e_1 の中で x と k (限定継続) を用いることができる。代数的エフェクトとハンドラには deep なハンドラと shallow なハンドラの 2 種類のハンドラが存在し、k に代入する限定継続に違いがある。deep なハンドラは限定継続に元々ハンドルしていたハンドラが入り、shallow なハンドラには入らないという性質がある。

ここで本研究で扱うオペレーションは名前を付けることができない制約がある。4 種類の限定継続演算子のインタプリタを基にして定義したため、オペレーションが発生した際必ず直近の限定子内の限定継続しか扱うことができない。しかし本稿では述べないが構文に条件文と再帰関数を導入すれば名前が付いたオペレーションの処理を模倣することができる。

3　インタプリタ

型なし λ 計算を代数的エフェクトとハンドラで拡張した構文を持つ言語を対象とし、インタプリタの項と値の定義は以下のようになる。

$$
\begin{array}{rcl}
\text{項 } e & ::= & x \mid \underline{\lambda} x.\, e \mid e_0\, \underline{@}\, e_1 \mid \underline{call}(e) \\
 & & \mid \underline{try}\, e_0\, \underline{with}\, (x; k).\, e_1 \\
\text{値 } v & = & v \to c \to t \to m \to v \\
\text{継続 } c & = & v \to t \to m \to v \\
\text{呼び出し文脈 } t & = & \bullet \;+\; (v \to t \to m \to v) \\
\text{ハンドラ } h & = & v \to v \to c \to t \to m \to v \\
\text{メタ継続 } m & = & (c \times t \times h)\ \mathit{list}
\end{array}
$$

呼び出し文脈とメタ継続を用いた代数的エフェクトとハンドラのための新たなインタプリタを図 1 に示す。λ 計算のインタプリタは呼び出し文脈 t とメタ継続 m を η-簡約すれば従来ものもと変わらないため割愛する。メタ継続 m によってハンドラの内側と外側の継続を分けて評価できる。つまりメタ継続に要素を追加することが、ハンドラで限定する範囲の区切りとなり、$\underline{try}\ \underline{with}$ を評価するたびにメタ継続の要素が増えていく。また呼び出し文脈 t を導入したことで shallow なハンドラの場合の評価において、限定継続を呼び出したときの継続を保持することができ限定継続 v_c にはハンドラが入らないことを表現する。

$$
\begin{array}{l}
\mathcal{E}\,[\![\underline{try}\, e_0\, \underline{with}\, (x; k).\, e_1]\!]\, \rho\, c \;=\; \lambda t.\, \lambda m.\, \mathcal{E}\,[\![e_0]\!]\, \rho\, id_c\, id_t\, ((c, t, h) :: m) \\
\quad \text{where } h = \lambda v.\, \lambda v_c.\, \lambda c'.\, \lambda t'.\, \lambda m'.\, \mathcal{E}\,[\![e_1]\!]\, \rho[v/x][v_c/k]\, c'\, t'\, m' \\
\mathcal{E}\,[\![\underline{call}(e)]\!]\, \rho\, c \;=\; \lambda t.\, \lambda m.\, \mathcal{E}\,[\![e]\!]\, \rho\, (\lambda v.\, \lambda t''.\, \lambda((c_0, t_0, h) :: m_0).\, h\, v\, v_c\, c_0\, t_0\, m_0)\, t\, m \\
\quad \textit{deep} \text{ なハンドラのとき where } v_c = \lambda v'.\, \lambda c'.\, \lambda t'.\, \lambda m'.\, c\, v'\, t''\, ((c', t', h) :: m') \\
\quad \textit{shallow} \text{ なハンドラのとき where } v_c = \lambda v'.\, \lambda c'.\, \lambda t'.\, \lambda m'.\, c\, v'\, (t'' \mathbin{@} c' :: t')\, m'
\end{array}
$$

図 1: 代数的エフェクトとハンドラのための CPS インタプリタ

$$
\begin{array}{rcl}
c & \Rightarrow & \langle c,\ VEnv([])\ ::\ [],\ [],\ []\rangle \\
\langle ICall\ ::\ c, v\ ::\ VContD(c', s', t', h)\ ::\ s, t, m\rangle & \Rightarrow & \langle c', v\ ::\ s', t', (c, s, t, h)\ ::\ m\rangle \\
\langle ICall\ ::\ c, v\ ::\ VContS(c', s', t')\ ::\ s, t, m\rangle & \Rightarrow & \langle c', v\ ::\ s', t'\ @\ (c, s)\ ::\ t, m\rangle \\
\langle ITrywith(c_0, c_1)\ ::\ c, VEnv(vs)\ ::\ s, t, m\rangle & \Rightarrow & \langle c_0, VEnv(vs)\ ::\ [],\ [],\ (c, s, t, (c_1, vs))\ ::\ m\rangle \\
\langle IMoveDH\ ::\ c, v\ ::\ s, t, (c_0, s_0, t_0, (c', vs))\ ::\ m_0\rangle & \Rightarrow & \langle c'\ @\ c_0, VEnv(VContD(c, s, t, (c', vs))\ ::\ v\ ::\ vs)\ ::\ s_0, t_0, m_0\rangle \\
\langle IMoveSH\ ::\ c, v\ ::\ s, t, (c_0, s_0, t_0, (c', vs))\ ::\ m_0\rangle & \Rightarrow & \langle c'\ @\ c_0, VEnv(VContS(c, s, t)\ ::\ v\ ::\ vs)\ ::\ s_0, t_0, m_0\rangle
\end{array}
$$

図 2: 代数的エフェクトとハンドラのための仮想機械

4　仮想機械

3 節で示したインタプリタからコンパイラと仮想機械を導出した。導出方法として 4 種類の限定継続演算子のインタプリタに施した手法 [5] を用いた。前半でインタプリタにスタックを導入する変換、後半にインタプリタをコンパイラと仮想機械に分割する変換を行った。得られた仮想機械の状態遷移規則を図 2 に示す。状態は継続 c、スタック s、呼び出し文脈 t、メタ継続 m の 4 つ組 $(c,\ s,\ t,\ m)$ で、継続 c の先頭の命令に応じて遷移する。この規則は戻り番地や値の環境の保存と復元などの標準の呼び出しの動作だけでなく、限定継続の切り取りや呼び出しでのデータスタックコピーの動作をモデル化し、代数的エフェクトとハンドラのための機械語実装の指針を示す。

5　型システム

3 節で提唱したインタプリタは継続 c やハンドラ h など全てを高階関数として定義した。高階関数で定義すると、特定のデータ構造を要求せずとも挙動を規定しており汎用性の高いものになる。そのため、高階関数で定義されたインタプリタから自然に型システムを導くことが可能となる。具体的な手法としては Cong ら [3] の方針に従い、インタプリタに型を割り振ることで導出する。この型付き言語を定義し、それに対するインタプリタは定理証明支援言語 Agda で実装できており、停止性や健全性を証明した。

型システムの定義を図 3 に示す。型判定は

$$\Gamma \vdash e : \tau\,\langle \mu_\alpha, \sigma_\alpha\rangle\,\alpha\,\langle \mu_\beta, \sigma_\beta\rangle\,\beta$$

という形になる。answer type に呼び出し文脈とメタ継続の型が付随する。これで「型環境 Γ のもとで式 e は τ 型を持ち、e を「τ 型の値と μ_α 型の呼び出し文脈、σ_α 型のメタ継続のもとで answer type が α になる継続」と μ_β 型の呼び出し文脈、σ_β 型のメタ継続のもとで実行すると answer type が β 型になる」と読む。

6　まとめ

本研究では呼び出し文脈やメタ継続を用いた新たな代数的エフェクトとハンドラのための CPS インタプリタを提唱した。本稿では述べないが名前付きオペレーションに対応するため条件文や再帰関数を追加して模倣し、従来のインタプリタと同等の挙動であることを確認した。

そして提唱したインタプリタから正当性が保証されたプログラム変換を用いて、仮想機械と抽象機械を機械的に導出した。また型システムもインタプリタから半機械的に導出した。いずれも 4 種類の限定継続演算子のインタプリタに対して施した手法と同様の手順で導くことができた。

本研究で示したインタプリタ、仮想機械、型システムにより、今回定義した新たなインタプリタを基に正当性が保証された代数的エフェクトとハンドラのための直接実装の基盤を提供する。さらに代数的エフェクトとハンドラと、4 種類の限定継続演算子を呼び出し文脈やメタ継続を用いた同じ枠組みの環境で定義できたため、両者を統一的に議論できる土台も提供する。

$$\frac{\begin{array}{l}\eta = \tau' \to \tau_{vc} \to \tau_3\,\langle \mu_3, \sigma_3\rangle\,\tau_4\,\langle \mu_4, \sigma_4\rangle\,\tau_5 \\ \mathsf{id\text{-}cont\text{-}type}(\gamma,\ \mu_{id},\ \sigma_{id},\ \gamma') \\ \Gamma \vdash e_0 : \gamma\,\langle \mu_{id}, \sigma_{id}\rangle\,\gamma'\,\langle \bullet, \tau \to \langle \mu_\alpha, \sigma_\alpha\rangle\,\alpha \times \mu_\beta \times \eta :: \sigma_\beta\rangle\,\beta \\ \Gamma,\ x : \tau',\ k : \tau_{vc} \vdash e_1 : \tau_3\,\langle \mu_3, \sigma_3\rangle\,\tau_4\,\langle \mu_4, \sigma_4\rangle\,\tau_5\end{array}}{\Gamma \vdash \underline{try}\ e_0\ \underline{with}\ (x, k).\ e_1 : \tau\,\langle \mu_\alpha, \sigma_\alpha\rangle\,\alpha\,\langle \mu_\beta, \sigma_\beta\rangle\,\beta}\ (Handler)$$

$$\frac{\begin{array}{l}\eta = \tau' \to \tau_{vc} \to \tau_0\,\langle \mu_0, \sigma_0\rangle\,\tau_0'\,\langle \mu_0', \sigma_0'\rangle\,\gamma \\ \tau_{vc} = \tau \to \tau_1\,\langle \mu_1, \sigma_1\rangle\,\tau_2\,\langle \mu_2, \sigma_2\rangle\,\alpha \\ \Gamma \vdash e : \tau'\,\langle \mu_\alpha, \tau_0 \to \langle \mu_0, \sigma_0\rangle\,\tau_0' \times \mu_0' \times \eta :: \sigma_0'\rangle\,\gamma\,\langle \mu_\beta, \sigma_\beta\rangle\,\beta\end{array}}{\Gamma \vdash \underline{call_d}(e) : \tau\,\langle \mu_\alpha, \tau_1 \to \langle \mu_1, \sigma_1\rangle\,\tau_2 \times \mu_2 \times \eta :: \sigma_2\rangle\,\alpha\,\langle \mu_\beta, \sigma_\beta\rangle\,\beta}\ (Operation\,Call-Deep)$$

$$\frac{\begin{array}{l}\eta = \tau' \to \tau_{vc} \to \tau_0\,\langle \mu_0, \sigma_0\rangle\,\tau_0'\,\langle \mu_0', \sigma_0'\rangle\,\gamma \\ \tau_{vc} = \tau \to \tau_1\,\langle \mu_1, \sigma_1\rangle\,\tau_2\,\langle \mu_2, \sigma_\alpha\rangle\,\alpha \\ \mathsf{compatible}((\tau_1 \to \langle \mu_1, \sigma_1\rangle\,\tau_2),\ \mu_2,\ \mu_2') \\ \mathsf{compatible}(\mu', \mu_2', \mu_\alpha) \\ \Gamma \vdash e : \tau'\,\langle \mu', (\tau_0 \to \langle \mu_0, \sigma_0\rangle\,\tau_0') \times \mu_0' \times \eta :: \sigma_0'\rangle\,\gamma\,\langle \mu_\beta, \sigma_\beta\rangle\,\beta\end{array}}{\Gamma \vdash \underline{call_s}(e) : \tau\,\langle \mu_\alpha, \sigma_\alpha\rangle\,\alpha\,\langle \mu_\beta, \sigma_\beta\rangle\,\beta}\ (Operation\,Call-Shallow)$$

図 3: 代数的エフェクトとハンドラのための型システム

今後の課題としては本稿で定義した型システムとこれまでに提唱された型システムとの等価性を証明することや、名前付きオペレーションの模倣が正しいことを示すフォーマルな証明をすることが考えられる。

参考文献

[1] Andrej Bauer and Matija Pretnar. Programming with algebraic effects and handlers. *J. Log. Algebraic Methods Program.*, Vol. 84, No. 1, pp. 108–123, 2015.

[2] Dariusz Biernacki, Olivier Danvy, and Kevin Millikin. A dynamic continuation-passing style for dynamic delimited continuations. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, Vol. 38, No. 1, pp. 1–25, 2015.

[3] Youyou Cong, Chiaki Ishio, Kaho Honda, and Kenichi Asai. A functional abstraction of typed invocation contexts. In *6th International Conference on Formal Structures for Computation and Deduction (FSCD 2021)*, pp. 12:1–12:18, 2021.

[4] Yannick Forster, Ohad Kammar, Sam Lindley, and Matija Pretnar. On the expressive power of user-defined effects: Effect handlers, monadic reflection, delimited control. *Proc. ACM Program. Lang.*, Vol. 1, No. ICFP, pp. 13:1–13:29, August 2017.

[5] Maika Fujii and Kenichi Asai. Derivation of a virtual machine for four variants of delimited-control operators. In *6th International Conference on Formal Structures for Computation and Deduction (FSCD 2021)*, pp. 16:1–16:19, 2021.

[6] Chiaki Ishio and Kenichi Asai. Type system for four delimited control operators. In *Proceedings of the 21st ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences*, pp. 45–58, 2022.

[7] Maciej Piróg, Piotr Polesiuk, and Filip Sieczkowski. Typed equivalence of effect handlers and delimited control. In *4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.