4種類の限定継続演算子のための型システム

石尾 千晶, 浅井 健一

お茶の水女子大学

ishio.chiaki@is.ocha.ac.jp, asai@is.ocha.ac.jp

概 要 限定継続演算子には多くの種類があり、それぞれ異なる特徴を持つ。中でも control0/prompt0 は、切り取った継続を呼び出す文脈や、現在の限定継続のひとつ外側の継続を操作できることが知られている。control0 を使うと、継続を切り取ることができる範囲が動的になる式を書けることから、shift のような演算子に比べて型システムが複雑になる。これまで、shift/reset, control/prompt, shift0/reset0 などの型システムを、CPS インタプリタから導出する研究がなされていたが、本研究ではこの方法を control0/prompt0 にも適用し型システムを導出する。この方法を拡張して、shift, control, shift0, control0 の 4 種類の演算子を含む λ 計算に対する一般的な型システムを導出し、定理証明系言語 Agda で実装する。また、既存研究の型システムとの関係を明らかにする。

1 はじめに

限定継続演算子とは、限定された範囲の継続を操作することができる演算子のことで、継続を捕捉する演算子と、継続の範囲を限定する演算子を組み合わせて継続を操作する。中でも、代表的な演算子である shift $(\mathcal{S}k.\,e)$ [9], control $(\mathcal{F}k.\,e)$ [10], shift0 $(\mathcal{S}_0k.\,e)$ [7], control0 $(\mathcal{F}_0k.\,e)$ [12] は、いずれも直近の限定子まで継続を切り取る演算子だが、これらは次のような動作の違いがある。

$$\begin{array}{rclcrcl} \langle(\mathcal{S}k.\,e)+3\rangle & = & \langle e[\lambda x.\,\langle x+3\rangle/k]\rangle & \qquad & \langle(\mathcal{F}k.\,e)+3\rangle & = & \langle e[\lambda x.\ x+3\ /k]\rangle \\ \langle(\mathcal{S}_0k.\,e)+3\rangle & = & e[\lambda x.\,\langle x+3\rangle/k] & & \langle(\mathcal{F}_0k.\,e)+3\rangle & = & e[\lambda x.\ x+3\ /k] \end{array}$$

各演算子を shift と比較してみると、control は、切り取った継続 k の body 部分に限定子 $\langle\,\rangle$ がつかず、shift0 は、継続を切り取った後、e の周りに限定子が残らない。また control0 ではその両方が起きている。

例えば $\langle\langle(\mathcal{S}_0k_1.\,\mathcal{S}_0k_2.\,e)+1\rangle+2\rangle$ のように shift0 がネストしている例を考えてみる。まず $\mathcal{S}_0k_1.\,\ldots$ が実行されると、$k_1=\lambda x.\,\langle x+1\rangle$ となって、直近の限定子までの継続を切り取るが、次に $\mathcal{S}_0k_2.\,\ldots$ が実行されると、直近の限定子よりもさらにひとつ外側の限定子に囲まれた継続までアクセスできるようになるため、$k_2=\lambda x.\,\langle x+2\rangle$ となる。この「直近の限定子よりも外側にある継続」を以下では「メタ継続」と呼ぶ。shift0 のようにメタ継続にアクセスできる演算子では、捕捉できる限定継続の範囲が動的に変化するため、型システムを考えるにはメタ継続を追跡する必要がある。

一方 control はどうだろうか。以下の式を例にとって考えてみる。

$$\begin{array}{rl} & \langle\langle(\lambda x.\,\mathcal{F}k_2.\,x)\,(\mathcal{F}k_1.\,(2+k_1\,1))\rangle\rangle \\ = & \langle\langle 2+k_1\,1\,[\lambda y.\,(\lambda x.\,\mathcal{F}k_2.\,x)\,y/k_1]\rangle\rangle \\ = & \langle\langle 2+\mathcal{F}k_2.\,1\rangle\rangle \\ = & \langle\langle 1\,[\lambda x.\,2+x/k_2]\rangle\rangle \end{array}$$

この例では、最初に切り取った継続 k_1 の中に、さらに `control` が入っている。`control` では切り取った継続の body 部分に限定子がつかないので、k_1 の中の `control` がどこまで継続を切り取るのかは、k_1 が呼び出されるまでわからない。実際に k_1 を呼び出すと、呼び出された文脈における直近の限定子まで継続を切り取るので、今回の例では $k_2 = \lambda x.\, 2 + x$ となっている。`control` を含む式の型付けをおこなうには「切り取った継続が呼び出される文脈」を追跡する必要があり、以下ではこれを「呼び出し文脈」と言うものとする。[1] また、`control0` では、呼び出し文脈だけでなく、メタ継続も追跡する必要がある。

このように、`control`, `shift0`, `control0` では、継続が切り取ることのできる範囲が動的に変化するため、`shift` に比べると型システムは複雑になる。これまで、型なしの状況下でこれらの限定継続演算子どうしの関係性が明らかにされてきた [14, 18] が、型付きの状況下では、`shift` や `shift0` などの CPS の階層の関係の研究 [16] などがあるものの、異なる限定継続演算子同士の関係性はまだ明らかになっていない。

本研究では、限定継続演算子の中でも代表的な `shift/reset`, `control/prompt`, `shift0/reset0`, `control0/prompt0` の 4 種類を取り上げる。これまでにまだ一般的な型システムが導出されていなかった `control0/prompt0` の型規則を導出し、`control0/prompt0` も含めた 4 種類の演算子のための統一的な型システムを構築する。本研究の型システムは、定理証明系言語 Agda [17] で実装されており、そのソースコードは `https://github.com/chiaki-i/type4d` で入手可能である。

本論文の構成は次の通りである。2 節で構文と CPS インタプリタを定義し、3 節でその型システムを導出する。4 節で先行研究の型システムとの比較をおこない、5 節では Agda による実装について触れる。また、6 節で関連研究を紹介して、最後に 7 節でまとめと今後の展望を述べる。

本論文での表記について　本論文では、数式中で色を使うため、カラーで印刷すると読みやすい。呼び出し文脈にかかわる箇所を赤字 (下線)で、メタ継続にかかわる箇所を青字 (波線) で示す。また、e^{τ} のように項の右肩に赤字で型を表すことがある。

2　構文と CPS インタプリタ

本節では、この論文で扱う言語 λ_D と λ_C を定義し、λ_D から λ_C に継続渡し形式 (continuation-passing style; CPS) に変換するインタプリタ (以下、CPS インタプリタ) も定義する。λ_D は、CPS インタプリタの入力言語で、限定継続演算子を含む λ 計算である。一方 λ_C は CPS インタプリタの出力言語で、継続渡し形式で書かれている。

2.1　λ_D の構文

図 1 に、λ_D の構文を示す。λ_D は、値呼びの単純型付き λ 計算に、4 種類の限定継続演算子を加えたものとなっている。項 e には、値と関数適用のほかに、継続を切り取る演算子として `shift`, `control`, `shift0`, `control0` と、限定子を加えた。この限定子は、継続を切り取る演算子に対応して `reset`, `prompt`, `reset0`, `prompt0` のように異なる名前で呼ばれることがあるが、中身はどれも同じふるまいをするものである [14] ため統一して表記する。

[1] この「呼び出し文脈」は英語では “a trail of invocation contexts” に相当するため、他の文献では「トレイル」と呼ぶことがある。

$$
\begin{array}{rrcl}
\text{値} & v & := & n \mid x \mid \lambda x.\, e \\
\text{項} & e & := & v \mid e_1\, e_2 \mid \mathcal{S}k.\, e \mid \mathcal{F}k.\, e \mid \mathcal{S}_0 k.\, e \mid \mathcal{F}_0 k.\, e \mid \langle e \rangle \\
\text{項の型} & \tau & := & \mathbb{N} \mid \tau_1 \to \tau_2 \,\langle \mu_\alpha,\, \sigma_\alpha \rangle\, \alpha \,\langle \mu_\beta,\, \sigma_\beta \rangle\, \beta \\
\text{呼び出し文脈の型} & \mu & := & \bullet_\mu \mid \tau_1 \to \langle \mu,\, \sigma \rangle\, \tau_2 \\
\text{メタ継続の型} & \sigma & := & \bullet_\sigma \mid ((\tau_1 \to \langle \mu_1,\, \sigma_1 \rangle\, \tau_2) \times \mu_2) :: \sigma_2
\end{array}
$$

図 1. λ_D の構文

項の型 τ は、自然数型と、矢印型からなる。矢印型 $\tau_1 \to \tau_2 \,\langle \mu_\alpha,\, \sigma_\alpha \rangle\, \alpha \,\langle \mu_\beta,\, \sigma_\beta \rangle\, \beta$ は、「$\tau_1 \to \tau_2$ 型の関数を、『μ_α 型の呼び出し文脈と σ_α 型のメタ継続のもとで実行した結果が α 型になるような継続』と、μ_β 型の呼び出し文脈、σ_β 型のメタ継続を渡して実行すると、その結果は β 型になる」ことをあらわしている。関数の body 部分に限定継続演算子が含まれていると、α と β が異なる型になる可能性がある。

ここで、「ある継続のもとで実行した結果の型」、つまり限定継続が返す型のことを、answer type [1] と呼ぶ。図 1 に定義されている矢印型では、α 型と β 型がそれにあたる。α 型は、項の実行前に、項全体が返すと予想される型 (initial answer type) をあらわし、β 型は、実際に式を実行したときに、最終的に項が返す型 (final answer type) をあらわす。例えば、`isZero` が `(int -> bool)` 型の関数だとすると、$\langle (\lambda x.\, \texttt{isZero}\, (x - 1))\, 1 \rangle$ のような式では、関数部分 $(\lambda x.\, \ldots)$ の initial answer type と final answer type の両方とも `bool` 型となる。一方で、関数の body 部分に `shift` が入った $\langle (\lambda x.\, \texttt{isZero}\, (\mathcal{S}k.\, x - 1))\, 1 \rangle$ のような式では、関数部分 $(\lambda x.\, \ldots)$ の initial answer type は `bool` 型、final answer type は `int` 型となり、answer type が変化する。

呼び出し文脈やメタ継続については、限定継続演算子の実行時に内部で操作がおこなわれるものなので、2.2, 2.3 節のように CPS になるまで明確な項はあらわれない。呼び出し文脈の型 μ は、空 $\bullet_\mu$ であるか、「τ_1 型の値、μ 型の呼び出し文脈、σ 型のメタ継続を受け取ると、τ_2 型の値を返す」継続であるかのいずれかである。メタ継続は、空 $\bullet_\sigma$ であるか、継続と呼び出し文脈のペアが入れ子状にメタ継続に積まれているかのどちらかである。

2.2 λ_C の構文

この節では、CPS インタプリタの出力言語である λ_C を定義する。λ_C は、値呼びの単純型付き λ 計算に、`unit` 型と直積型を加えた体系で、構文は図 2 の通りとなる。

継続を λ_C で書くと $\lambda v.\, \lambda t.\, \lambda m. \ldots$ のように表され、値 v、呼び出し文脈 t、メタ継続 m を受け取ると計算結果を返す関数である。呼び出し文脈の実装方法には、関数やリストなど複数あるが、今回は Shan [18] に従って関数 $\lambda v.\, \lambda t.\, \lambda m. \ldots$ として扱う[2] 。また、メタ継続は $((k,\, t),\, m)$ のように、そのときの継続 k と呼び出し文脈 t のペアを、入れ子にして表現する。呼び出し文脈やメタ継続が空の場合は、いずれも `unit` 型の値 () とする。

[2]呼び出し文脈の中に、$(\texttt{int} \to \mu_1 \to \sigma_1 \to \texttt{int})$, $(\texttt{string} \to \mu_2 \to \sigma_2 \to \texttt{int})$, ... のように、異なる型を持つ複数の継続が現れる場合に対応するためには、heterogeneous list もしくは関数を使って実装すればよい。詳しくは [6] を参照されたい。

$$
\begin{array}{lrcl}
\text{値} & v & := & n \mid () \mid x \mid \lambda x.\, e \\
\text{項} & e & := & v \mid e_1\, e_2 \mid (e_1,\, e_2) \\
\text{型} & \tau & := & \mathtt{int} \mid \mathtt{unit} \mid \tau_1 \to \tau_2 \mid \tau_1 \times \tau_2
\end{array}
$$

図 2. λ_C の構文

$$
\begin{array}{rcl}
\mathbb{N}^* & = & \mathtt{int} \\
(\tau_1 \to \tau_2 \langle \mu_\alpha,\, \sigma_\alpha \rangle\, \alpha \langle \mu_\beta,\, \sigma_\beta \rangle\, \beta)^* & = & \tau_1^* \to (\tau_2^* \to \mu_\alpha^* \to \sigma_\alpha^* \to \alpha^*) \to \mu_\beta^* \to \sigma_\beta^* \to \beta^* \\
\bullet_\mu^* & = & \mathtt{unit} \\
(\tau_1 \to \langle \mu,\, \sigma \rangle\, \tau_2)^* & = & \tau_1^* \to \mu^* \to \sigma^* \to {\tau_2}^* \\
\bullet_\sigma^* & = & \mathtt{unit} \\
(((\tau_1 \to \langle \mu_1,\, \sigma_1 \rangle\, \tau_2) \times \mu_2) :: \sigma_2)^* & = & (((\tau_1^* \to \mu_1^* \to \sigma_1^* \to {\tau_2}^*) \times {\mu_2}^*) \times {\sigma_2}^*
\end{array}
$$

図 3. λ_D の型の CPS 変換

2.3 λ_D から λ_C への CPS インタプリタ

この節では、λ_D から λ_C への型レベルの CPS 変換と、項の CPS インタプリタを定義する。図 3 では、型レベルの CPS 変換 $*$ を定義している。λ_D で継続の型は $(\tau_1 \to \langle \mu_1,\, \sigma_1 \rangle\, \tau_2)$ と書いたが、これを CPS にすると、値・呼び出し文脈・メタ継続を順番に引数として受け取って値を返す関数になっている。

次に、図 4 では、$\mathcal{E}[\![e]\!]$ は、実行時に環境 ρ、継続 κ、呼び出し文脈 t、メタ継続 m を受け取って処理をおこなう。このインタプリタ規則は、2CPS [3] のインタプリタ [7] に、呼び出し文脈に相当する引数 t を加えたものであり、Shan による 4 種類の限定継続演算子のための型なし CPS インタプリタ [18] を η-展開したものに等しい。図 4 では、呼び出し文脈にかかわる箇所を赤字 (下線) で、メタ継続にかかわる箇所を青字 (波線) で示している。

限定継続演算子の規則では、いずれも、切り取ってきた継続を $\lambda v.\, \lambda \kappa'.\, \lambda t'.\, \lambda m'.\, \ldots$ のように書き起こして、それを k に束縛した上で、再帰的に e の実行を進めている。

`shift` や `control` では、継続を切り取った後も、限定継続演算子を囲んでいた直近の限定子が残るので、初期継続 k_{id} と空の呼び出し文脈 $()$、そしてメタ継続は変わらず m のままで e が実行される。初期継続 k_{id} は、図 4 に定義が示されている通り、値・呼び出し文脈・メタ継続を受け取って、呼び出し文脈に積まれた継続を実行し、その後呼び出し文脈が空になったら、メタ継続に積まれた継続を順に実行する関数である。一方、`shift0` や `control0` では、限定継続演算子が継続を切り取ると、直近の限定子がなくなって、メタ継続にアクセスできるようになる。そのため、引数として受け取るメタ継続の形は $((\kappa_0,\, t_0),\, m_0)$ のように展開されて中身が使えるような形で書かれている。e を再帰的に実行するときには、そのときの継続 κ_0 と呼び出し文脈 t_0、e の実行時のメタ継続 m_0 が渡される。

では、切り取られる継続 k の中身は、各演算子ごとにどのような形をとるだろうか。`shift` や

[3] CPS で書かれた式をさらに CPS 変換にかけたもののことを 2CPS と呼ぶ。一般的な CPS では、そのときの継続を受け取って計算を進めるのに対して、2CPS は、そのときの「継続の継続」(メタ継続) と、そのときの継続、という両方を受け取って計算を進めていく。また、2CPS と区別するために、普通の CPS のことを 1CPS と呼ぶことがある。

$$
\begin{array}{rcl}
k_{id}\, v\, ()\, () & = & v \\
k_{id}\, v\, ()\, ((k_0,\, t_0),\, m_0) & = & k_0\, v\, t_0\, m_0 \\
k_{id}\, v\, t\, m & = & t\, v\, ()\, m
\end{array}
$$

$$
\begin{array}{rclcrcl}
() \mathbin{@} t & = & t & \qquad & k :: () & = & k \\
k \mathbin{@} t & = & k :: t & & k :: t & = & \lambda v.\, \lambda t'.\, \lambda m'.\, k\, v\, (t :: t')\, m'
\end{array}
$$

$$
\begin{array}{rcl}
\mathcal{E}[\![x]\!]\, \rho\, \kappa & = & \lambda t.\, \lambda m.\, \kappa\, \rho(x)\, t\, m \\
\mathcal{E}[\![n]\!]\, \rho\, \kappa & = & \lambda t.\, \lambda m.\, \kappa\, n\, t\, m \\
\mathcal{E}[\![\lambda x.\, e]\!]\, \rho\, \kappa & = & \lambda t.\, \lambda m.\, \kappa\, (\lambda v.\, \mathcal{E}[\![e]\!]\, \rho[v/x])\, t\, m \\
\mathcal{E}[\![e_1\, e_2]\!]\, \rho\, \kappa & = & \lambda t.\, \lambda m.\, \mathcal{E}[\![e_1]\!]\, \rho\, (\lambda v_1.\, \lambda t_1.\, \lambda m_1.\, \mathcal{E}[\![e_2]\!]\, \rho\, (\lambda v_2.\, \lambda t_2.\, \lambda m_2.\, v_1\, v_2\, \kappa\, t_2\, m_2)\, t_1\, m_1)\, t\, m \\
\mathcal{E}[\![\mathcal{S} k.\, e]\!]\, \rho\, \kappa & = & \lambda t.\, \lambda m.\, \mathcal{E}[\![e]\!]\, \rho[\lambda v.\, \lambda \kappa'.\, \lambda t'.\, \lambda m'.\, \kappa\, v\, t\, ((\kappa',\, t'),\, m')/k]\, k_{id}\, ()\, m \\
\mathcal{E}[\![\mathcal{F} k.\, e]\!]\, \rho\, \kappa & = & \lambda t.\, \lambda m.\, \mathcal{E}[\![e]\!]\, \rho[\lambda v.\, \lambda \kappa'.\, \lambda t'.\, \lambda m'.\, \kappa\, v\, (t \mathbin{@} (\kappa' :: t'))\, m'/k]\, k_{id}\, ()\, m \\
\mathcal{E}[\![\mathcal{S}_0 k.\, e]\!]\, \rho\, \kappa & = & \lambda t.\, \lambda ((\kappa_0,\, t_0),\, m_0).\, \mathcal{E}[\![e]\!]\, \rho[\lambda v.\, \lambda \kappa'.\, \lambda t'.\, \lambda m'.\, \kappa\, v\, t\, ((\kappa',\, t'),\, m')/k]\ \kappa_0\, t_0\, m_0 \\
\mathcal{E}[\![\mathcal{F}_0 k.\, e]\!]\, \rho\, \kappa & = & \lambda t.\, \lambda ((\kappa_0,\, t_0),\, m_0).\, \mathcal{E}[\![e]\!]\, \rho[\lambda v.\, \lambda \kappa'.\, \lambda t'.\, \lambda m'.\, \kappa\, v\, (t \mathbin{@} (\kappa' :: t'))\, m'/k]\ \kappa_0\, t_0\, m_0 \\
\mathcal{E}[\![\langle e \rangle]\!]\, \rho\, \kappa & = & \lambda t.\, \lambda m.\, \mathcal{E}[\![e]\!]\, \rho\ k_{id}\, ()\, ((\kappa,\, t)\, ,\, m)
\end{array}
$$

図 4. λ_D の CPS インタプリタ

`shift0` では、k の呼び出し時の継続 κ' や呼び出し文脈 t' は、$((\kappa', t'), m')$ のように、現在のメタ継続とペアにする形で追加されていく。一方、`control` や `control0` のように呼び出し文脈が動的に定まる演算子では、κ' や t' は $(t \mathbin{@} (\kappa' :: t'))$ という形で呼び出し文脈に追加されている。ここで使われている演算子 $\_@\_$ は呼び出し文脈どうしを結合し、$\_ :: \_$ は継続を呼び出し文脈に追加している。本研究では、呼び出し文脈も継続と同様に、関数であらわされるので、$\_@\_$ も $\_ :: \_$ も内部的には関数合成をしている。[4]

最後に、$\langle e \rangle$ では、限定子の内側の項 e を再帰的に実行している。$\langle e \rangle$ の外側の継続は、e にとってのメタ継続に相当するので、現在の継続 κ と呼び出し文脈 t をメタ継続に積んだ上で計算を進めている。

このインタプリタは、3 節で述べる型規則のもとで、定理証明系言語 Agda で実装されている。そのため、CPS インタプリタが型を保存することや、λ_D の項に型がつけば CPS インタプリタが止まることが示せている。

3　型システムの導出

λ_D の型システムを図 5 に示す。型判断は $\Gamma \vdash e : \tau\ \langle \mu_\alpha,\, \sigma_\alpha \rangle\ \alpha\ \langle \mu_\beta,\, \sigma_\beta \rangle\ \beta$ と書いて、「型環境 Γ のもとで項 e は τ 型を持つ。『μ_α 型の呼び出し文脈と σ_α 型のメタ継続のもとで、answer type が α 型になるような継続』と、μ_β 型の呼び出し文脈、σ_β 型のメタ継続のもとで e を実行すると、answer type は β 型になる。」と読む。

本研究において、`control0/prompt0` の型規則が新たに導出した規則である。特に、限定子のための型規則は、`reset`, `prompt`, `reset0`, `prompt0` を包含する一般的な規則となっている。また、

[4] $\_@\_$ や $\_ :: \_$ は、Shan の論文 [18] における `compose` 関数に等しい。もし、呼び出し文脈を heterogeneous list で実装する場合は、それぞれリスト用の append や cons を用いて実装すればよい。

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

$$\frac{x \,:\, \tau \in \Gamma}{\Gamma \vdash x : \tau \,\langle \mu_\alpha,\, \sigma_\alpha \rangle\, \alpha \,\langle \mu_\alpha,\, \sigma_\alpha \rangle\, \alpha}\ (\textsc{TVar}) \qquad \frac{}{\Gamma \vdash n : \mathbb{N} \,\langle \mu_\alpha,\, \sigma_\alpha \rangle\, \alpha \,\langle \mu_\alpha,\, \sigma_\alpha \rangle\, \alpha}\ (\textsc{TNum})$$

$$\frac{\Gamma,\, x \,:\, \tau_1 \vdash e : \tau_2 \,\langle \mu_\alpha,\, \sigma_\alpha \rangle\, \alpha \,\langle \mu_\beta,\, \sigma_\beta \rangle\, \beta}{\Gamma \vdash (\lambda\, x.\, e) : (\tau_1 \to \tau_2 \,\langle \mu_\alpha,\, \sigma_\alpha \rangle\, \alpha \,\langle \mu_\beta,\, \sigma_\beta \rangle\, \beta) \,\langle \mu_\gamma,\, \sigma_\gamma \rangle\, \gamma \,\langle \mu_\gamma,\, \sigma_\gamma \rangle\, \gamma}\ (\textsc{TLam})$$

$$\frac{\Gamma \vdash e_1 : (\tau_1 \to \tau_2 \,\langle \mu_\alpha,\, \sigma_\alpha \rangle\, \alpha \,\langle \mu_\beta,\, \sigma_\beta \rangle\, \beta) \,\langle \mu_\gamma,\, \sigma_\gamma \rangle\, \gamma \,\langle \mu_\delta,\, \sigma_\delta \rangle\, \delta \quad \Gamma \vdash e_2 : \tau_1 \,\langle \mu_\beta,\, \sigma_\beta \rangle\, \beta \,\langle \mu_\gamma,\, \sigma_\gamma \rangle\, \gamma}{\Gamma \vdash (e_1\, e_2) : \tau_2 \,\langle \mu_\alpha,\, \sigma_\alpha \rangle\, \alpha \,\langle \mu_\delta,\, \sigma_\delta \rangle\, \delta}\ (\textsc{TApp})$$

$$\frac{\begin{array}{c}\mathsf{id\text{-}cont\text{-}type}(\gamma,\, \mu_{id},\, \sigma_{id},\, \gamma') \\ \Gamma,\, k \,:\, \tau \to \tau_1 \,\langle \mu_1,\, \sigma_1 \rangle\, \tau_2 \,\langle \mu_2,\, \sigma_2 \rangle\, \alpha \vdash e : \gamma \,\langle \mu_{id},\, \sigma_{id} \rangle\, \gamma' \,\langle \bullet_\mu,\, \sigma_\beta \rangle\, \beta\end{array}}{\Gamma \vdash \mathcal{S}k.\, e : \tau \,\langle \mu_\beta,\, (((\tau_1 \to \langle \mu_1,\, \sigma_1 \rangle\, \tau_2) \times \mu_2) :: \sigma_2) \rangle\, \alpha \,\langle \mu_\beta,\, \sigma_\beta \rangle\, \beta}\ (\textsc{TShift})$$

$$\frac{\begin{array}{c}\mathsf{id\text{-}cont\text{-}type}(\gamma,\, \mu_{id},\, \sigma_{id},\, \gamma') \\ \mathsf{compatible}((\tau_1 \to \langle \mu_1,\, \sigma_1 \rangle\, \tau_2),\, \mu_2,\, \mu_\gamma) \quad \mathsf{compatible}(\mu_\beta,\, \mu_\gamma,\, \mu_\alpha) \\ \Gamma,\, k \,:\, \tau \to \tau_1 \,\langle \mu_1,\, \sigma_1 \rangle\, \tau_2 \,\langle \mu_2,\, \sigma_\alpha \rangle\, \alpha \vdash e : \gamma \,\langle \mu_{id},\, \sigma_{id} \rangle\, \gamma' \,\langle \bullet_\mu,\, \sigma_\beta \rangle\, \beta\end{array}}{\Gamma \vdash \mathcal{F}k.\, e : \tau \,\langle \mu_\alpha,\, \sigma_\alpha \rangle\, \alpha \,\langle \mu_\beta,\, \sigma_\beta \rangle\, \beta}\ (\textsc{TControl})$$

$$\frac{\Gamma,\, k \,:\, \tau \to \tau_1 \,\langle \mu_1,\, \sigma_1 \rangle\, \tau_2 \,\langle \mu_2,\, \sigma_2 \rangle\, \alpha \vdash e : \tau_0 \,\langle \mu_0,\, \sigma_0 \rangle\, \tau_0' \,\langle \mu_0',\, \sigma_0' \rangle\, \beta}{\Gamma \vdash \mathcal{S}_0 k.\, e : \tau \,\langle \mu_\beta,\, (((\tau_1 \to \langle \mu_1,\, \sigma_1 \rangle\, \tau_2) \times \mu_2) :: \sigma_2) \rangle\, \alpha \,\langle \mu_\beta,\, (((\tau_0 \to \langle \mu_0,\, \sigma_0 \rangle\, \tau_0') \times \mu_0') :: \sigma_0') \rangle\, \beta}\ (\textsc{TShift0})$$

$$\frac{\begin{array}{c}\mathsf{compatible}((\tau_1 \to \langle \mu_1,\, \sigma_1 \rangle\, \tau_2),\, \mu_2,\, \mu_\gamma) \quad \mathsf{compatible}(\mu_\beta,\, \mu_\gamma,\, \mu_\alpha) \\ \Gamma,\, k \,:\, \tau \to \tau_1 \,\langle \mu_1,\, \sigma_1 \rangle\, \tau_2 \,\langle \mu_2,\, \sigma_\alpha \rangle\, \alpha \vdash e : \tau_0 \,\langle \mu_0,\, \sigma_0 \rangle\, \tau_0' \,\langle \mu_0',\, \sigma_0' \rangle\, \beta\end{array}}{\Gamma \vdash \mathcal{F}_0 k.\, e : \tau \,\langle \mu_\alpha,\, \sigma_\alpha \rangle\, \alpha \,\langle \mu_\beta,\, (((\tau_0 \to \langle \mu_0,\, \sigma_0 \rangle\, \tau_0') \times \mu_0') :: \sigma_0') \rangle\, \beta}\ (\textsc{TControl0})$$

$$\frac{\begin{array}{c}\mathsf{id\text{-}cont\text{-}type}(\gamma,\, \mu_{id},\, \sigma_{id},\, \gamma') \\ \Gamma \vdash e : \gamma \,\langle \mu_{id},\, \sigma_{id} \rangle\, \gamma' \,\langle \bullet_\mu,\, (((\tau \to \langle \mu_\alpha,\, \sigma_\alpha \rangle\, \alpha) \times \mu_\beta) :: \sigma_\beta) \rangle\, \beta\end{array}}{\Gamma \vdash \langle e \rangle : \tau \,\langle \mu_\alpha,\, \sigma_\alpha \rangle\, \alpha \,\langle \mu_\beta,\, \sigma_\beta \rangle\, \beta}\ (\textsc{TPrompt0})$$

$$\begin{array}{rcl}
\mathsf{id\text{-}cont\text{-}type}(\tau,\, \bullet_\mu,\, \bullet_\sigma,\, \tau') & = & \tau \equiv \tau' \\
\mathsf{id\text{-}cont\text{-}type}(\tau,\, \bullet_\mu, & = & (\tau \equiv \tau_1) \wedge (\tau' \equiv \tau_1') \wedge (\mu_1 \equiv \mu_2) \wedge (\sigma_1 \equiv \sigma_2) \\
\quad (((\tau_1 \to \langle \mu_1,\, \sigma_1 \rangle\, \tau_1') \times \mu_2) :: \sigma_2),\, \tau') & & \\
\mathsf{id\text{-}cont\text{-}type}(\tau,\, (\tau_1 \to \langle \mu_1,\, \sigma_1 \rangle\, \tau_1'),\, \sigma_2,\, \tau') & = & (\tau \equiv \tau_1) \wedge (\tau' \equiv \tau_1') \wedge (\mu_1 \equiv \bullet_\mu) \wedge (\sigma_1 \equiv \sigma_2) \\
 & & \\
\mathsf{compatible}(\bullet_\mu,\, \mu_2,\, \mu_3) & = & \mu_2 \equiv \mu_3 \\
\mathsf{compatible}((\tau_1 \to \langle \mu_1,\, \sigma_1 \rangle\, \tau_1'),\, \bullet_\mu,\, \mu_3) & = & (\tau_1 \to \langle \mu_1,\, \sigma_1 \rangle\, \tau_1') \equiv \mu_3 \\
\mathsf{compatible}((\tau_1 \to \langle \mu_1,\, \sigma_1 \rangle\, \tau_1'), & = & \bot \\
\quad (\tau_2 \to \langle \mu_2,\, \sigma_2 \rangle\, \tau_2'),\, \bullet_\mu) & & \\
\mathsf{compatible}((\tau_1 \to \langle \mu_1,\, \sigma_1 \rangle\, \tau_1'), & = & (\tau_1 \equiv \tau_3) \wedge (\tau_1' \equiv \tau_3') \wedge (\sigma_1 \equiv \sigma_3) \\
\quad (\tau_2 \to \langle \mu_2,\, \sigma_2 \rangle\, \tau_2'),\, (\tau_3 \to \langle \mu_3,\, \sigma_3 \rangle\, \tau_3')) & & \quad \wedge (\mathsf{compatible}((\tau_2 \to \langle \mu_2,\, \sigma_2 \rangle\, \tau_2'),\, \mu_3,\, \mu_1))
\end{array}$$

図 5. λ_D の型システム

そのほかの `shift`, `control`, `shift0` のための型規則も、CPS インタプリタで呼び出し文脈やメタ継続に対する操作をおこなうのにあわせて拡張している。

3.1 `control0` の型規則の導出

図 5 の型規則はすべて図 4 から導出されているため、型規則の中身をひとつひとつ CPS インタプリタと対応付けるとわかりやすい。そこで、(TControl0) を例にとって導出過程を追ってみることにする。以下に、`control0` のインタプリタの各項に対して型を割り当てる順番を示したものと、型を割り当て終えたものを示す。

$$\mathcal{E}[\![\mathcal{F}_0 k.\, e]\!]\,\rho\,\kappa \;=\; \lambda t^{\boxed{1}}.\,\lambda((\kappa_0^{\boxed{2}},\, t_0^{\boxed{3}}),\, m_0^{\boxed{4}}).\,\mathcal{E}[\![e]\!]\,\rho[\lambda v^{\boxed{7}}.\,\lambda\kappa'^{\boxed{7}}.\,\lambda t'^{\boxed{7}}.\,\lambda m'^{\boxed{7}}.\,(\kappa^{\boxed{9}}\,v^{\boxed{8}}\,(t^{\boxed{5}}\,@\,(\kappa'^{\boxed{8}} :: t'^{\boxed{8}})^{\boxed{11}})^{\boxed{12}}\,m'^{\boxed{8}})^{\boxed{10}}\,/\,k]\quad \kappa_0^{\boxed{6}}\,t_0^{\boxed{6}}\,m_0^{\boxed{6}}$$

$$\mathcal{E}[\![\mathcal{F}_0 k.\, e]\!]\,\rho\,\kappa \;=\; \lambda t^{\mu^*_\beta}.\,\lambda((\kappa_0^{(\tau_0 \to \langle \mu_0,\, \sigma_0\rangle\, \tau_0')^*},\, t_0^{\mu_0'^*}),\, m_0^{\sigma_0'^*}).\,\mathcal{E}[\![e]\!]\,\rho[\lambda v^{\tau^*}.\,\lambda\kappa'^{(\tau_1 \to \langle \mu_1,\, \sigma_1\rangle\, \tau_2)^*}.\,\lambda t'^{\mu^*_2}.\,\lambda m'^{\sigma^*_2}.\,(\kappa^{(\tau^* \to \mu^*_\alpha \to \sigma^*_\alpha \to \alpha^*)}\,v^{\tau^*}\,(t^{\mu^*_\beta}\,@\,(\kappa'^{(\tau_1 \to \langle \mu_1,\, \sigma_1\rangle\, \tau_2)^*} :: t'^{\mu^*_2})^{\mu^*_\gamma})^{\mu^*_\alpha}\,m'^{\sigma^*_\alpha})^{\alpha^*}\,/\,k]\quad \kappa_0^{(\tau_0 \to \langle \mu_0,\, \sigma_0\rangle\, \tau_0')^*}\,t_0^{\mu_0'^*}\,m_0^{\sigma_0'^*}$$

型規則を導出するには、まず λ 抽象で新たに導入された変数に対して新しい型変数名を割り当て、その後、関数適用などの条件を加味して型を決めていくという方法をとる。

`control0` のインタプリタ規則の冒頭で、呼び出し文脈 t と、メタ継続 $((\kappa_0, t_0), m_0)$ が λ 抽象で導入されている。`control0` はメタ継続の中身にアクセスできることから、初めからメタ継続は展開された形で表記されている。これらの変数のうち、t, t_0, m_0 については新しい型変数をおく (1 ∼ 4)。その中でも、κ_0 は継続であるため、型は $(? \to \langle ?,\, ?\rangle\, ?)^*$ の形をとるはずで、それに応じて型変数をおく (2)。この導出過程全体を通して注意したいのが、CPS インタプリタの各項に型をつけるという作業をしているので、これまで定めた型も、正確には、型レベルの CPS 変換 (図 3) がかかっているという点である。そのため型変数の右肩に CPS 変換を示す $*$ がついていることがわかる。例えば κ_0 の型も、CPS 変換されると次のように展開できる。これ以降の説明では、変換前後の 2 つの表記を区別せずに扱う。

$$(\tau_0 \to \langle \mu_0,\, \sigma_0\rangle\, \tau_0')^* = (\tau_0^* \to \mu_0^* \to \sigma_0^* \to \tau_0'^*)$$

同じ変数には同じ型が割り振られるので、インタプリタ内に現れる t, κ_0, t_0, m_0 の型も定まる (5, 6)。`control0` の body 部分である e に対して再帰的に CPS インタプリタを実行する際、切り取った継続は k に束縛され、それが環境 ρ に登録される。切り取った継続 k の中身にも λ 抽象が登場するので、上述と同様の方法で型変数をおき (7)、同じ変数名のところにも書き写す (8)。ここまで、λ 抽象で導入された変数に対しては、全て新たに異なる型変数をおいている。

次に、インタプリタの実行時に与えられた継続 κ の型について考える。κ は、値 v、呼び出し文脈 $(t\,@\,\kappa' :: t')$、メタ継続 m' を受け取る継続であるため、κ の型は、これまでに 8 の手順で定めた v, m' の型を用いると $(\tau^* \to ?^* \to \sigma^*_\alpha \to ?^*)$ という形をとるとわかる (9)。継続 κ が返す型については、新たに α^* とおく (10)。また、呼び出し文脈 $(t\,@\,(\kappa' :: t'))$ については、全体の型を一旦 μ^*_α とし (12)、内側の $(\kappa' :: t')$ の型を μ^*_γ とする (11)。ここまでで得られた型変数名を型規則の形に書き直すと、(TControl0) になる。ただし、呼び出し文脈については、3 つの呼び出し文脈 t, κ', t' の型についての制約を、別途示す必要がある。これについては 3.2 節で述べる。

3.2 呼び出し文脈の合成

今回は、呼び出し文脈どうしの合成 $\_@\_$ や継続の追加 $\_ :: \_$ は、2.3 節で述べた通り、いずれも関数合成で実装されている。そのため図 4 の定義に従って型の制約をくくり出すことにする。呼び出し文脈の型の制約は compatible(μ_1, μ_2, μ_3) と書くと、$(\mu_1$ 型の $t_1) \,@\, (\mu_2$ 型の $t_2) = (\mu_3$ 型の $t_3)$ もしくは $(\mu_1$ 型の $k_1) :: (\mu_2$ 型の $t_2) = (\mu_3$ 型の $t_3)$ に対応している。具体的には、compatible の 1 つ目の規則は $() \,@\, t = t$ に、2 つ目の規則は $k :: () = k$ に対応している。3 つ目の規則は、空でない継続や呼び出し文脈を合成したときの結果は、空の呼び出し文脈にはなりえないことを意味する。4 つ目の規則は一見難しいようだが、$k :: t = \ldots$ の式と、次のように対応している。

$$
\begin{aligned}
& k^{(\tau_1 \to \langle \mu_1,\, \sigma_1 \rangle\, \tau_1')^*} :: t^{(\tau_2 \to \langle \mu_2,\, \sigma_2 \rangle\, \tau_2')^*} \\
= \;& \lambda v^{\tau_3^*}.\, \lambda t'^{\mu_3^*}.\, \lambda m'^{\sigma_3^*}.\, (k^{(\tau_1^* \to \mu_1^* \to \sigma_1^* \to \tau_1'^*)}\, v^{\tau_3^*}\, (t^{(\tau_2 \to \langle \mu_2,\, \sigma_2 \rangle\, \tau_2')^*} :: t'^{\mu_3^*})\, m'^{\sigma_3^*})^{\tau_3'^*}
\end{aligned}
$$

式の 2 行目において、継続 k が、値 v を受け取るためには $\tau_1^* \equiv \tau_3^*$ になればよい。また、k が呼び出し文脈 $(t :: t')$ を受け取るためには、compatible を再帰的に適用させて、$(\tau_2 \to \langle \mu_2,\, \sigma_2 \rangle\, \tau_2')^*$ 型と μ_3^* 型を持つ呼び出し文脈を合成した結果が、μ_1^* 型の呼び出し文脈に一致すればよい。k が受け取るメタ継続や、k が返す型についても同様に対応付けすることで、compatible を導出できる。

3.3 初期継続の型の性質

図 4 の CPS インタプリタでは、`shift`, `control`, `prompt0` において初期継続を使う場面がある。これに対応する形で、図 5 の型システムでも、初期継続のための型の制約 id-cont-type が設けられている。$(\gamma \to \langle \mu_{id},\, \sigma_{id} \rangle\, \gamma')$ 型を持つ継続が初期継続であるためには、id-cont-type$(\gamma, \mu_{id}, \sigma_{id}, \gamma')$ を満たせばよい。

id-cont-type も、これまでの規則の導出と同様の方法で、対応する k_{id} の定義から素直に導出できる。例えば、id-cont-type の 2 つ目の規則では、k_{id} の規則と以下のように対応づけられる。

$$
\begin{aligned}
& (k_{id}\;\; v^{\tau^*}\;\; ()^{\bullet^*_{\mu}}\, ((k_0,\, t_0),\, m_0)^{(((\tau_1 \to \langle \mu_1,\, \sigma_1 \rangle\, \tau_1') \times \mu_2) :: \sigma_2)^*})^{\tau'^*} \\
= \;& k_0^{(\tau_1^* \to \mu_1^* \to \sigma_1^* \to \tau_1'^*)}\, v^{\tau^*}\, t_0^{\mu_2^*}\, m_0^{\sigma_2^*}
\end{aligned}
$$

式の 2 行目で、k_0 が引数 v, t_0, m_0 を受け取るためには型が一致する必要がある。また、初期継続 k_{id} はもともと τ'^* 型を返すことになっていたが、それはメタ継続に記録されていた継続 k_0 の返す型と一致するはずなので、$\tau'^* \equiv \tau_1'^*$ となることがわかる。

4 他の型システムとの比較

この節では、本論文で導出した、4 種類の限定継続演算子のための型システム (以下、*4D* と表記する) と、先行研究で提案されている各限定継続演算子のための型システムを比較する。

4.1 `shift/reset`

3 節では、Danvy と Filinski による `shift/reset` の型システム導出の方法 [7] に倣って、CPS インタプリタから型システムを導出をおこなった。ここでは、その型システムと、*4D* を比較する。以下の式は、Danvy と Filinski の型システム (以下、*DF* と表記する) の `shift` の型付け規則を、

4D の表記に寄せて書いたもの (DF-Shift) である。型判断は $\Gamma \vdash e : \tau,\ \alpha,\ \beta$ と書いて「項 e は型環境 Γ のもとで τ 型を持つ。$\tau \to \alpha$ 型の継続のもとで実行すると最終的に β 型になる」と読む。

$$\frac{\Gamma,\ k : \tau \to \alpha,\ \delta,\ \delta \vdash e : \epsilon,\ \epsilon,\ \beta}{\Gamma \vdash \mathcal{S}k.\, e : \tau,\ \alpha,\ \beta}\ \text{(DF-Shift)}$$

この (DF-Shift) は 1CPS で書かれたインタプリタ [8] をもとに導出されており、呼び出し文脈やメタ継続に相当する引数は含まれないため上のように表記されている。

この型規則を図 5 の (TShift) と比較した時、異なる点は 2 つある。1 つが、`shift` の body 部分 e の answer type、もう 1 つが、切り取った継続 k の型判断である。

まず、(DF-Shift) の e の型判断を見てみよう。(DF-Shift) では answer type はいずれも ϵ だが、(TShift) では γ, γ' となっている。図 5 中の id-cont-type の定義を見ると、必ず $\gamma \equiv \gamma'$ となるのは、呼び出し文脈とメタ継続の両方とも空の場合 (id-cont-type の 1 つ目の規則) のみである。e に限定継続演算子のうちいずれかが現れると、呼び出し文脈やメタ継続に継続が積まれるため γ, γ' は異なるものになる可能性がある。

一方、切り取った継続 k の型はどうだろうか。(DF-Shift) では k の矢印型の α が $\mathcal{S}k.\, e$ の initial answer type になるが、(TShift) では k は、メタ継続のもとで実行して得られる final answer type である α が $\mathcal{S}k.\, e$ の initial answer type になる。このような違いが生じたのは、(DF-Shift) の元となるインタプリタが 1CPS で書かれていたのに対して、(TShift) では 2CPS のインタプリタで書かれているからである。`shift` のための 2CPS インタプリタに基づいて型システムを導出すると、(TShift) から呼び出し文脈を除いたものに等しくなる。

4.2 `control/prompt`

これまでに、著者らが構築した `control/prompt` のための型システム [6] は、本論文と同様に、4.1 節で取り上げた *DF* の導出方法に則って作られている。`control` のための型規則は、*4D* の (TControl) のメタ継続部分を除いたものに一致する。`prompt` のための型規則を比較するため、以下に [6] から抜粋したものを (TPrompt′) として示す。*4D* の (TPrompt0) とは、型規則の結論部の $\langle e \rangle$ の型判断が異なる。

$$\frac{\mathsf{id\text{-}cont\text{-}type}(\gamma,\ \mu_{id},\ \gamma') \quad \Gamma \vdash e : \gamma\, \langle \mu_{id} \rangle\, \gamma'\, \langle \bullet_{\mu} \rangle\, \tau}{\Gamma \vdash \langle e \rangle : \tau\, \langle \mu_{\alpha} \rangle\, \alpha\, \langle \mu_{\alpha} \rangle\, \alpha}\ \text{(TPrompt}')$$

これも、4.1 節と同様の理由だが、元となるインタプリタの `prompt` のみが 1CPS で書かれているためこのような違いが現れている。

4.3 `shift0/reset0`

ここでは、`shift0/reset0` の型システムの先行研究として、Materzok と Biernacki による型システム [15] (以下、*MB* と表記する) と比較する。まずは、比較対象となる構文 λ_{S_0} (図 6) と型システム (図 7) について説明する。

4.3.1 λ_{S_0} の構文

λ_{S_0} は、値呼びの単純型付き λ 計算に `shift0/reset0` を加えたものである。その型は自然数型 $\mathbb{N}$ か、矢印型からなる。また、型注釈は、限定継続演算子によって継続が直接操作される可能性がない (pure である) ことを示す ϵ か、そうでない (impure である) かのいずれかである。型や型注釈は、単体ではわかりにくいが、型判断の中で一緒に考えると理解しやすい。型判断は

$$
\begin{array}{llll}
\text{値} & v & := & n \mid x \mid \lambda x.\, e \\
\text{項} & e & := & v \mid e_1\, e_2 \mid \mathcal{S}_0 k.\, e \mid \langle e \rangle
\end{array}
\qquad
\begin{array}{llll}
\text{型} & \tau & := & \mathbb{N} \mid \tau \xrightarrow{\sigma} \tau' \\
\text{型注釈} & \sigma & := & \epsilon \mid [\tau\, \sigma]\, \tau'\, \sigma'
\end{array}
$$

図 6. λ_{S_0} の構文

$$\frac{}{\Gamma,\, x : \tau_x \vdash x : \tau_x\, [\tau\, \sigma]\, \tau\, \sigma}\ (\text{MB-Var}) \qquad \frac{\Gamma,\, x : \tau_1 \vdash e : \tau_2\, \sigma}{\Gamma \vdash \lambda x.\, e : \tau_1 \xrightarrow{\sigma} \tau_2\, [\tau\, \sigma']\, \tau\, \sigma'}\ (\text{MB-Abs})$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \xrightarrow{[\tau_1'\, \sigma_1]\, \tau_3'\, \sigma_3} \tau_2\, [\tau_4'\, \sigma_4]\, \tau_2'\, \sigma_2 \quad \Gamma \vdash e_2 : \tau_1\, [\tau_3'\, \sigma_3]\, \tau_4'\, \sigma_4}{\Gamma \vdash e_1\, e_2 : \tau_2\, [\tau_1'\, \sigma_1]\, \tau_2'\, \sigma_2}\ (\text{MB-App})$$

$$\frac{\Gamma,\, k : \tau_1 \xrightarrow{\sigma_1} \tau_2 \vdash e : \tau_3\, \sigma_2}{\Gamma \vdash \mathcal{S}_0 k.\, e : \tau_1\, [\tau_2\, \sigma_1]\, \tau_3\, \sigma_2}\ (\text{MB-Shift0}) \qquad \frac{\Gamma \vdash e : \tau'\, [\tau''\, \sigma'']\, \tau''\, \sigma''}{\Gamma \vdash \langle e \rangle : \tau\, \sigma}\ (\text{MB-Reset0})$$

$$\frac{\Gamma,\, x : \tau_1 \vdash e : \tau_2\ [\tau_3\, \sigma_3]\, \tau_4\, \sigma_4}{\Gamma \vdash \lambda x.\, e : \tau_1 \xrightarrow{[\tau_3\, \sigma_3]\, \tau_4\, \sigma_4} \tau_2\, [\tau\, \sigma']\, \tau\, \sigma'}\ (\text{MB-Abs-Ext})$$

$$\frac{\Gamma,\, k : \tau_1 \xrightarrow{[\tau_4\, \sigma_4]\, \tau_5\, \sigma_5} \tau_2 \vdash e : \tau_3\ [\tau_6\, \sigma_6]\, \tau_7\, \sigma_7}{\Gamma \vdash \mathcal{S}_0 k.\, e : \tau_1\, [\tau_2\ [\tau_4\, \sigma_4]\, \tau_5\, \sigma_5]\, \tau_3\ [\tau_6\, \sigma_6]\, \tau_7\, \sigma_7}\ (\text{MB-Shift0-Ext})$$

図 7. Materzok と Biernacki による `shift0/reset0` のための型システム [15]

$\Gamma \vdash e : \tau_1'\, [\tau_1\, \sigma_1]\, \tau_2' \ldots [\tau_n\, \sigma_n]\, \tau$ と書いて、[5] 「型環境 Γ のもとで *MB* の項 e を、ある文脈のもとで実行すると最終的に τ 型になる。その文脈は、継続が連なったものであり、それらの型は $\tau_1' \xrightarrow{\sigma_1} \tau_1, \ldots, \tau_n' \xrightarrow{\sigma_n} \tau_n$ となっている」と読める。

MB の型注釈の表現の幅を理解するために、簡単な型判断 $\Gamma \vdash e : \tau_1\, [\tau_2\, \sigma_2]\, \tau_3\, \sigma_3$ を例にとって説明する。この型判断の最後の型注釈 σ_3 が impure と仮定すると、$\Gamma \vdash e : \tau_1\, [\tau_2\, \sigma_2]\, \tau_3\ [\tau_4\, \sigma_4]\, \tau_5\, \sigma_5$ のように展開でき、先ほどのように、$\tau_1 \xrightarrow{\sigma_2} \tau_2$, $\tau_3 \xrightarrow{\sigma_4} \tau_4$ という型を持つ継続が連なっていると言える。では、$\Gamma \vdash e : \tau_1\, [\tau_2\, \sigma_2]\, \tau_3\, \sigma_3$ の型注釈 σ_2 が impure の場合はどうなるか。σ_2 を展開すると、$\Gamma \vdash e : \tau_1\, [\tau_2\ [\tau_6\, \sigma_6]\, \tau_7\, \sigma_7]\, \tau_3\, \sigma_3$ と書いて、「$\tau_1 \xrightarrow{[\tau_6\, \sigma_6]\, \tau_7\, \sigma_7} \tau_2$ 型の継続のもとで e を実行すると最終結果が τ_3 型になる」と読める。仮に、残る型注釈 σ_6, σ_7 が ϵ だとすると、継続の継続、つまりメタ継続の型は、直感的には $\tau_1 \to (\tau_2 \to \tau_6) \to \tau_7$ と言える。

また、図 7 では、λ_{S_0} のための型規則を示した。(MB-Var) から (MB-Reset0) までは、文献 [15] からそのまま引用した規則である。(MB-Abs-Ext) は、(MB-Abs) 中の型注釈 σ を展開して $[\tau_3\, \sigma_3]\, \tau_4\, \sigma_4$ としたものである。また、(MB-Shift0-Ext) は (MB-Shift0) 中の型注釈 σ_1, σ_2 を展開してそれぞれ $[\tau_4\, \sigma_4]\, \tau_5\, \sigma_5$, $[\tau_6\, \sigma_6]\, \tau_7\, \sigma_7$ としたものである。これは、λ_{S_0} の項において、関数や `shift0` の中で仮に pure な部分項があらわれたとき、それを impure であるとみなして、型注釈を冗長に展開したということを意味している。[6] このとき、(MB-Abs-Ext) や (MB-Shift0-Ext) を使って型付けできる項の範囲と、(MB-Abs) や (MB-Shift0) を使って型付けできる項の範囲は同じである。このことは、図 7 の型システムの中で、型注釈が ϵ であることを要求

[5] この型判断は本来 $\Gamma \vdash e : \tau_1'\, [\tau_1\, \sigma_1]\, \tau_2' \ldots [\tau_n\, \sigma_n]\, \tau\, \epsilon$ で、最後の型注釈は ϵ だが、冗長なので省かれている。これ以降の説明や型規則中でも、型注釈が ϵ のときは省略されることが多い。

[6] 著者らは、関数や `shift0` の中には pure な部分項は現れないだろうと考えているが、これについては現在確認中である。

$$
\begin{aligned}
\mathbb{N}^{\rightharpoonup^{\tau}_{m}} &= \mathbb{N} \\
(\tau_1 \to \tau_2 \langle\sigma_\alpha\rangle\, \alpha \langle\sigma_\beta\rangle\, \beta)^{\rightharpoonup^{\tau}_{m}} &= {\tau_1}^{\rightharpoonup^{\tau}_{m}} \xrightarrow{[\tau_3\,\sigma_3]\,\tau_4\,\sigma_4} {\tau_2}^{\rightharpoonup^{\tau}_{m}} \\
&\quad \textit{where} \quad (\tau_3,\, \sigma_3) = (\sigma_\alpha,\, \alpha)^{\rightharpoonup^{\sigma}_{m}}, \quad (\tau_4,\, \sigma_4) = (\sigma_\beta,\, \beta)^{\rightharpoonup^{\sigma}_{m}}
\end{aligned}
$$

$$
\begin{aligned}
(\bullet_\sigma,\, \tau)^{\rightharpoonup^{\sigma}_{m}} &= (\tau^{\rightharpoonup^{\tau}_{m}},\, \epsilon) \\
(((\tau_1 \to \langle\sigma_1\rangle\, \tau_2) :: \sigma_2),\, \tau)^{\rightharpoonup^{\sigma}_{m}} &= ({\tau_1}^{\rightharpoonup^{\tau}_{m}},\, [\tau_3\,\sigma_3]\,\tau_4\,\sigma_4) \\
&\quad \textit{where} \quad (\tau_3,\, \sigma_3) = (\sigma_1,\, \tau_2)^{\rightharpoonup^{\sigma}_{m}}, \quad (\tau_4,\, \sigma_4) = (\sigma_2,\, \tau)^{\rightharpoonup^{\sigma}_{m}}
\end{aligned}
$$

$$
\begin{aligned}
\mathbb{N}^{\rightharpoonup^{\tau}_{d}} &= \mathbb{N} \\
\left(\tau_1 \xrightarrow{[\tau_3\,\sigma_3]\,\tau_4\,\sigma_4} \tau_2\right)^{\rightharpoonup^{\tau}_{d}} &= ({\tau_1}^{\rightharpoonup^{\tau}_{d}}) \to ({\tau_2}^{\rightharpoonup^{\tau}_{d}}) \langle\sigma_\alpha\rangle\, \alpha \langle\sigma_\beta\rangle\, \beta \\
&\quad \textit{where} \quad (\sigma_\alpha,\, \alpha) = (\tau_3,\, \sigma_3)^{\rightharpoonup^{\sigma}_{d}}, \quad (\sigma_\beta,\, \beta) = (\tau_4,\, \sigma_4)^{\rightharpoonup^{\sigma}_{d}} \\
(\tau,\, \epsilon)^{\rightharpoonup^{\sigma}_{d}} &= (\bullet_\sigma,\, \tau^{\rightharpoonup^{\tau}_{d}}) \\
(\tau,\, (\,[\tau_1\,\sigma_1]\,\tau_2\,\sigma_2))^{\rightharpoonup^{\sigma}_{d}} &= (((\tau^{\rightharpoonup^{\tau}_{d}} \to \langle\sigma_3\rangle\, \tau_3) :: \sigma_4),\, \tau_4) \\
&\quad \textit{where} \quad (\sigma_3,\, \tau_3) = (\tau_1,\, \sigma_1)^{\rightharpoonup^{\sigma}_{d}}, \quad (\sigma_4,\, \tau_4) = (\tau_2,\, \sigma_2)^{\rightharpoonup^{\sigma}_{d}}
\end{aligned}
$$

図 8. *4D* と *MB* の型レベルの変換規則

する型規則がないこと、つまり「必ず pure でなければいけない部分項」を持つ型規則がないことからも明らかだろう。本節において、これ以降、*MB* と言うとき、(MB-Var), (MB-Abs-Ext), (MB-App), (MB-Shift0-Ext), (MB-Reset0) からなる型システムのことを指すものとする。

ここまでで λ_{S_0} の構文やその型規則を眺めてきたが、λ_D では answer type やメタ継続を使って `shift0` の動作を定式化しているのに対して、λ_{S_0} では型と型注釈を使って継続やメタ継続をすべて書き下すことで `shift0` の動作を表現している。このように、継続やメタ継続の表記方法が異なるので *4D* と *MB* の型規則の中に現れる型変数どうしを逐一対応付けることは難しいが、*4D* の項と *MB* の項の持つ型を互いに変換することはできそうだ。[7]

4.3.2 *4D* と *MB* の関係

図 8 に、*4D* と *MB* の型レベルの変換規則を定義した。$\rightharpoonup^{\tau}_{m}, \rightharpoonup^{\sigma}_{m}$ は、*4D* から *MB* への変換で、それぞれ型と型注釈へ変換をおこなう。特に、型注釈への変換 $\rightharpoonup^{\sigma}_{m}$ は、*4D* のメタ継続と、メタ継続を適用した後の answer type の 2 つを与えると、対応する *MB* の型と型注釈を返すような関数である。また、$\rightharpoonup^{\tau}_{d}, \rightharpoonup^{\sigma}_{d}$ は、上述の変換の逆変換となっている。ここまでで定めた 4 つの変換規則のあいだには、次のような性質が成り立つ。

補題 4.1 λ_{S_0} の型 τ_1 について、$({\tau_1}^{\rightharpoonup^{\tau}_{d}})^{\rightharpoonup^{\tau}_{m}} \equiv \tau_1$ が成り立つ。また、λ_{S_0} の型注釈 $[\tau_2\,\sigma_2]\,\tau_3\,\sigma_3$ について、$(((\tau_2,\, \sigma_2)^{\rightharpoonup^{\sigma}_{d}})^{\rightharpoonup^{\sigma}_{m}},\, ((\tau_3,\, \sigma_3)^{\rightharpoonup^{\sigma}_{d}})^{\rightharpoonup^{\sigma}_{m}}) \equiv ((\tau_2,\, \sigma_2),\, (\tau_3,\, \sigma_3))$ が成り立つ。 □

補題 4.2 λ_D の型 τ について、$(\tau^{\rightharpoonup^{\tau}_{m}})^{\rightharpoonup^{\tau}_{d}} \equiv \tau$ が成り立つ。また、λ_D のメタ継続 σ_α と、メタ継続の answer type である α に対して、$((\sigma_\alpha,\, \alpha)^{\rightharpoonup^{\sigma}_{m}})^{\rightharpoonup^{\sigma}_{d}} \equiv (\sigma_\alpha,\, \alpha)$ が成り立つ。 □

[7] λ_{S_0} との変換を考えるのは、λ_D のうち、「λ 計算と `shift0/reset0` のみからなる部分」である。そのため、これ以降で、この節にあらわれる *4D* の型では、呼び出し文脈は省略する。

これらの補題 4.1, 4.2 が成り立つので、*MB* で型が付くなら *4D* でも型が付くこと、またその逆が成り立つことが言える。その定理を次に示す。また、図 6, 7, 8 やこれらの補題・定理は、いずれも Agda 上で定式化されている。

定理 4.3 λ_{S_0} の項 e_m が *MB* で $\Gamma \vdash e_m : \tau_1\,[\tau_2\,\sigma_2]\,\tau_3\,\sigma_3$ であるとする。ここで、${\tau_1}^{\rightarrow^{\tau}_{d}} = \tau$ かつ $((\tau_2,\,\sigma_2)^{\rightarrow^{\sigma}_{d}},\,(\tau_3,\,\sigma_3)^{\rightarrow^{\sigma}_{d}}) = ((\sigma_\alpha,\,\alpha),\,(\sigma_\beta,\,\beta))$ のとき、e_m は、 *4D* で $\Gamma \vdash e : \tau\,\langle\sigma_\alpha\rangle\,\alpha\,\langle\sigma_\beta\rangle\,\beta$ であるような λ_D の項 e_d に変換できる。

λ_D の項のうち λ 計算と `shift0/reset0` のみからなる項 e'_d が *4D* で $\Gamma' \vdash e'_d : \tau'\,\langle\sigma'_\alpha\rangle\,\alpha'\,\langle\sigma'_\beta\rangle\,\beta'$ であるとする。ここで、${\tau'}^{\rightarrow^{\tau}_{m}} = \tau'_1$ かつ $((\sigma'_\alpha,\,\alpha')^{\rightarrow^{\sigma}_{m}},\,(\sigma'_\beta,\,\beta')^{\rightarrow^{\sigma}_{m}}) = ((\tau'_2,\,\sigma'_2),\,(\tau'_3,\,\sigma'_3))$ のとき、e'_d は、 *MB* で $\Gamma' \vdash e'_m : \tau'_1\,[\tau'_2\,\sigma'_2]\,\tau'_3\,\sigma'_3$ であるような λ_{S_0} の項 e'_m に変換できる。 □

5 Agda による実装

本論文で定義したもののうち、λ_D や λ_C の構文 (図 1, 2)、λ_D から λ_C への CPS インタプリタ (図 3, 4) と、型システム *4D* (図 5)、そして λ_{S_0} の構文と型システム *MB* (図 6, 7)、*4D* と *MB* の変換 (図 8) と、それらに関連する性質 (補題 4.1, 補題 4.2, 定理 4.3) については、Agda で実装されている。

4D や *MB* の構文を実装するにあたり、変数束縛には PHOAS [5] を使用している。例えば、図 4 の λ_D の CPS インタプリタでは、環境 ρ は実装中には明示的には現れず、PHOAS を使うことで Agda の変数束縛の機能を使って実装されている。

6 関連研究

本論文の 3 節では、著者らの構築した `control/prompt` の型システム [6] と同様に、Shan の CPS インタプリタ [18] を η-展開したものをもとに型システムの導出をおこなった。また、このインタプリタは、Biernacki らによる CPS インタプリタ [3] に等しい。[3] では、`control/prompt` を含む λ 計算のための抽象機械を定義し、それを非関数化することで CPS インタプリタを導出している。

本研究で構築した型システムは、answer type が単相のもののみに対応している。`shift/reset` の型システムについては、Danvy と Filinski による型システム [7] の answer type が単相であるため、これを let 多相で拡張し、多相の answer type に対応した `shift/reset` のための型システムが作られた [1]。また、これをもとに、多相の answer type に対応した `control/prompt` のための型システムも構築されている [13]。Biernacki と Materzok は、λ_{S_0} の型システムとして answer type が単相の型システム (4.3 節の *MB*) を構築し、それに部分型付けを加えることで answer type が多相の型システムも構築した [15]。

また、近年、代数的エフェクトハンドラ [2] と、継続演算子との関係性を明らかにする試みが多くなされている。中でも、deep handler と `shift0`、shallow handler と `control0` に深い関係があることが知られている [4, 11]。

7　まとめと今後の展望

本論文では、`shift/reset`, `control/prompt`, `shift0/reset0`, `control0/prompt0` の 4 種類の限定継続演算子のための統一的な型システムを構築し、これを Agda で実装した。また、先行研究での型システムとの比較をおこない、特に `shift0/reset0` の型システムとの対応を証明した。

本研究の型システムは、answer type が単相のものに対応しているため、今後は多相の answer type にも対応できるように拡張したい。また、この型システムをもとに、代数的エフェクトハンドラとの型レベルの関係性を明らかにしたい。

謝辞

有益なコメントをくださった査読者の皆様に感謝申し上げます。また、本研究は一部 JSPS 科研費 18H03218 の助成を受けたものです。

参考文献

[1] K. Asai and Y. Kameyama. Polymorphic Delimited Continuations. *Proceedings of the 5th Asian conference on Programming languages and systems (APLAS'07)*, pages 239–254, 2007.

[2] A. Bauer and M. Pretnar. Programming with algebraic effects and handlers. *Journal of logical and algebraic methods in programming*, 84(1):108–123, 2015.

[3] D. Biernacki, O. Danvy, and K. Millikin. A Dynamic Continuation-Passing Style for Dynamic Delimited Continuations. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 38(1):1–25, 2015.

[4] D. Biernacki, M. Piróg, P. Polesiuk, and F. Sieczkowski. Handle with Care: Relational Interpretation of Algebraic Effects and Handlers. *Proc. ACM Program. Lang.*, 2(POPL), dec 2017.

[5] A. Chlipala. Parametric higher-order abstract syntax for mechanized semantics. In *Proceedings of the ACM SIGPLAN International Conference on Functional Programming*, ICFP 2008, pages 143–156, 2008.

[6] Y. Cong, C. Ishio, K. Honda, and K. Asai. A functional abstraction of typed invocation contexts. In *6th International Conference on Formal Structures for Computation and Deduction (FSCD 2021)*, pages 12:1–12:18, 2021.

[7] O. Danvy and A. Filinski. A functional abstraction of typed contexts. *BRICS 89/12*, 1989.

[8] O. Danvy and A. Filinski. Abstracting control. *Proceedings of the ACM conference on LISP and Functional Programming (LFP'90)*, pages 151–160, 1990.

[9] O. Danvy and A. Filinski. Representing control: a study of the CPS transformation. *Mathematical Structures in Computer Science*, 2(4):361–391, 1992.

[10] M. Felleisen. The theory and practice of first-class prompts. In *Proceedings of the 15th ACM Symposium on Principles of Programming Languages (POPL '88)*, pages 180–190, 1988.

[11] Y. Forster, O. Kammar, S. Lindley, and M. Pretnar. On the Expressive Power of User-Defined Effects: Effect Handlers, Monadic Reflection, Delimited Control. *Proc. ACM Program. Lang.*, 1(ICFP), aug 2017.

[12] C. A. Gunter, D. Rémy, and J. G. Riecke. A generalization of exceptions and control in ML-like languages. In *Proceedings of the 7th International Conference on Functional Programming Languages and Computer Architecture*, pages 12–23, 1995.

[13] Y. Kameyama and T. Yonezawa. Typed dynamic control operators for delimited continuations. In *International Symposium on Functional and Logic Programming*, pages 239–254. Springer, 2008.

[14] O. Kiselyov. How to remove a dynamic prompt: static and dynamic delimited continuation operators. *Technical Report 611. Indiana University*, 2005.

[15] M. Materzok and D. Biernacki. Subtyping Delimited Continuations. In *Proceedings of the ACM SIGPLAN International Conference on Functional Programming (ICFP'11)*, pages 81–93, 2011.

[16] M. Materzok and D. Biernacki. A Dynamic Interpretation of the CPS Hierarchy. In *Asian Symposium on Programming languages and systems (APLAS '12)*, pages 296–311. Springer, 2012.

[17] U. Norell. *Towards a practical programming language based on dependent type theory.* PhD thesis, Chalmers University of Technology, SE-412 96 Göteborg, Sweden, September 2007.

[18] C. Shan. A Static Simulation of Dynamic Delimited Control. *Higher-Order and Symbolic Computation*, 20(4):371–401, 2007.