shift0/reset0 入りの型付き言語における CPS 変換の正当性の証明

田村 優衣, 浅井 健一

お茶の水女子大学

{tamura.yui, asai}@is.ocha.ac.jp

概 要　直接形式のプログラムの中で継続を扱う言語機構として、限定継続演算子や代数的エフェクトとハンドラなどがある。これらを含むプログラムは、プログラムを継続渡し形式 (continuation-passing style; CPS) に変換することで実行でき、その変換の正当性は、直接形式の簡約と継続渡し形式の簡約が対応することにより示される。これまで、shift/reset を含む型付き言語における CPS 変換の正当性が証明されており、本研究ではこの方法を用いて shift0/reset0 に対する CPS 変換の正当性を証明する。まず、変換時に冗長な項が生成されないコロン変換をメタ継続を含むインタプリタから導出し、得られた変換の正当性を証明するとともに Agda 上で定式化した。本成果は Reflection の証明の土台となるものであり、将来的に代数的エフェクトとハンドラに対する証明への活用も期待される。

1　はじめに

限定継続とは、プログラム実行中のある時点における残りの計算のうち範囲が限定されたものを指す。直接形式のプログラムで限定継続を扱うには、限定継続演算子を用いる。限定継続演算子には、継続を捕捉する演算子と継続の範囲を限定する演算子があり、それらを組み合わせて継続を操作することができる。代表的な演算子として、shift/reset [7], control/prompt [9], shift0/reset0 [6], control0/prompt0 [11] の 4 種類がある。他にも明示的に継続が扱える言語機構として、代数的エフェクトとハンドラがある。ハンドラには、深いもの（deep handler）と浅いもの（shallow handler）が存在する。これらには、限定継続の扱い方に違いがあり、deep handler は shift0/reset0 と shallow handler は control0/prompt0 と近い関係があることが知られている [10, 15]。

限定継続演算子や代数的エフェクトとハンドラを含むプログラムは、継続渡し形式 (continuation-passing style; CPS) に変換することで実行できる。その変換の正当性は、ソース言語のプログラムの簡約とターゲット言語のプログラムの簡約が対応することにより示され、この証明は実行時の正しさを保証する上で重要である。また、より密接な関係である Reflection という関係が成り立つと、ソース言語とターゲット言語が 1 対 1 に対応することが明らかになる。限定継続演算子を含む体系における Reflection の証明は、shift/reset [2] や shift0/dollar [3] に対するものが既に存在しており、特に shift/reset は型が入った体系での証明と Agda での定式化もされている [19]。また、代数的エフェクトとハンドラに対するもの [5] も検討されている。

本研究では、shift0/reset0 を含む型付き言語における Reflection の証明に向けて、CPS 変換の正当性を証明し、定理証明支援系言語 Agda を用いて定式化した。shift0/reset0 は、近年盛んに研究されている代数的エフェクトとハンドラの deep handler と振る舞いが近いため、shift0/reset0 に対する証明が代数的エフェクトとハンドラに対する証明の土台になることが期待される。また、本研究は shift/reset に対する証明 [19] を元に進めている。演算子 shift0 に対する証明として shift0/dollar に対するもの [3] があるが、reset0 ではなく dollar を使っているのに加えて、独自の枠組みで証明しているため、これまでの shift/reset の reflection との関係も不明である。本研究では、shift/reset

に対する手法をそのまま応用する形でコロン変換を導き、そこから系統的に reflection の証明につなげている。shift0/reset0 は、Ishio らによって 4 種類の限定継続演算子のインタプリタと型システムが統一的に定義されている [13] ため、shift/reset との比較がしやすい。加えて、代数的エフェクトとハンドラに対しても同じ枠組みで定式化されており [1]、活用しやすいと考えられる。

本稿では、ソース言語は shift0/reset0（と shift）が入った単純型付き λ 計算、ターゲット言語は限定継続演算子が入っていない単純型付き λ 計算とする。前者は直接形式なので DS 言語、後者は継続渡し形式なので CPS 言語と呼び、DS 言語を $\lambda_{c\mathcal{S}_0}$、CPS 言語を $\lambda^*_{c\mathcal{S}_0}$ と記述する。また、DS 言語の項 M の CPS 言語への変換を CPS 変換と呼び M^* と記述する。この設定のもとで、一般的な CPS 変換の正当性は「任意の$\lambda_{c\mathcal{S}_0}$の項 M, N に対して、$M \twoheadrightarrow_{\lambda_{c\mathcal{S}_0}} N$ ならば $M^* =_{\lambda^*_{c\mathcal{S}_0}} N^*$」と表現できるが、本稿では、より強い性質である簡約の保存「任意の$\lambda_{c\mathcal{S}_0}$の項 M, N に対して、$M \twoheadrightarrow_{\lambda_{c\mathcal{S}_0}} N$ ならば $M^* \twoheadrightarrow_{\lambda^*_{c\mathcal{S}_0}} N^*$」が証明できている。

本稿の構成は以下のとおりである。まず、2 節で shift0/reset0 を説明し、3 節で DS 言語と CPS 言語を定式化する。4 節で CPS 変換の導出を紹介し、5 節で CPS 変換の簡約の保存を証明する。最後に、6 節で関連研究、7 節でまとめを述べる。

Agda による実装は、`http://pllab.is.ocha.ac.jp/~asai/jpapers/ppl/26/shift0/` で公開している。

2 shift0/reset0

本節では、shift0/reset0 の挙動を紹介する。以下の式は、shift/reset と shift0/reset0 の実行規則である。

$$\langle J[Sk.e]\rangle = \langle e[\lambda x.\langle J[x]\rangle/k]\rangle \qquad \langle J[S_0k.e]\rangle = e[\lambda x.\langle J[x]\rangle/k]$$

shift0 では、継続を切り取った後、e の周りに reset が残らないため、さらに外側の継続（メタ継続）を捕捉することができるのが特徴である。ここで、具体例を用いて詳しく見てみよう。

$$\begin{array}{ll}
 & \langle(\lambda x.\,1)\,@\,\langle(\lambda y.\,2)\,@\,Sk_1.\,Sk_2.\,(k_1\,@\,0)\rangle\rangle \\
= & \langle(\lambda x.\,1)\,@\,\langle Sk_2.\,(k_1\,@\,0)[\lambda z.\,\langle(\lambda y.\,2)\,@\,z\rangle/k_1]\rangle\rangle \\
= & \langle(\lambda x.\,1)\,@\,\langle Sk_2.\,\langle(\lambda y.\,2)\,@\,0\rangle\rangle\rangle \\
= & \langle(\lambda x.\,1)\,@\,\langle\langle(\lambda y.\,2)\,@\,0\rangle[\lambda z.\,\langle z\rangle/k_2]\rangle\rangle \\
= & \langle(\lambda x.\,1)\,@\,2\rangle \\
= & 1 \\
 & \langle(\lambda x.\,1)\,@\,\langle(\lambda y.\,2)\,@\,S_0k_1.\,S_0k_2.\,(k_1\,@\,0)\rangle\rangle \\
= & \langle(\lambda x.\,1)\,@\,(S_0k_2.\,(k_1\,@\,0)[\lambda z.\,\langle(\lambda y.\,2)\,@\,z\rangle/k_1])\rangle \\
= & \langle(\lambda x.\,1)\,@\,(S_0k_2.\,\langle(\lambda y.\,2)\,@\,0\rangle)\rangle \\
= & ((\lambda y.\,2)\,@\,0)[\lambda z.\,\langle(\lambda x.\,1)\,@\,z\rangle/k_2] \\
= & 2
\end{array}$$

上段が shift の例、下段が shift0 の例になっており、reset は中身が値になると外れる。4 行目の k_2 に着目すると、shift では $\lambda z.\,\langle z\rangle$ を束縛しているのに対し、shift0 では $\lambda z.\,\langle(\lambda x.\,1)\,@\,z\rangle$ を束縛している。つまり、より外側にあった「$\lambda x.1$ に値を渡す」という継続も捕捉できており、最終的な結果が異なる。これは、2 行目の引数部分で shift では reset が残っているのに対し、shift0 では reset が残っていないことに起因している。

このように、shift と shift0 の大きな違いは「より外側の継続（メタ継続）が捕捉できること」である。メタ継続を記録しておく必要があるため、本研究では、先行研究の証明をメタ継続が入った形に拡張していくことになる。

3　それぞれの言語の定式化

ここでは、本論文で扱う DS 言語 $\lambda_{c\mathcal{S}_0}$ と CPS 言語 $\lambda^*_{c\mathcal{S}_0}$ の構文と型規則について説明する。

3.1　DS 言語 $\lambda_{c\mathcal{S}_0}$

$$
\begin{array}{rlll}
\text{types} & \tau, \alpha, \beta & ::= & \mathbb{N} \mid \mathbb{B} \mid \tau_1 \to \tau_2 \, \langle \sigma_\alpha \rangle \, \alpha \, \langle \sigma_\beta \rangle \, \beta \\
\text{metacont types} & \sigma & ::= & \bullet_\sigma \mid (\tau_1 \triangleright \langle \sigma_1 \rangle \, \tau_2) :: \sigma_2 \\
\text{terms} & L, M, N & ::= & V \mid P \\
\text{values} & V, W & ::= & x \mid \underline{\lambda} x.\, M \mid \underline{\mathcal{S}} \mid \underline{\mathcal{S}_0} \\
\text{nonvalues} & P, Q & ::= & M \,\underline{@}\, N \mid \underline{\mathsf{let}}\ x = M\ \underline{\mathsf{in}}\ N \mid \underline{\langle} M \underline{\rangle} \\
\text{pure contexts} & J_\Delta, K_\Delta & ::= & [\,]_\Delta \mid K_\Delta[[\,] \,\underline{@}\, M] \mid K_\Delta[V \,\underline{@}\, [\,]] \mid K_\Delta[\underline{\mathsf{let}}\ x = [\,]\ \underline{\mathsf{in}}\ M]
\end{array}
$$

$$
\begin{array}{llcll}
(\beta.v) & (\underline{\lambda} x.\, M) \,\underline{@}\, V & \longrightarrow & M[V/x] & \\
(\eta.v) & \underline{\lambda} x.\, (V \,\underline{@}\, x) & \longrightarrow & V & \text{if } x \notin fv(V) \\
(\beta.let) & \underline{\mathsf{let}}\ x = V\ \underline{\mathsf{in}}\ M & \longrightarrow & M[V/x] & \\
(\eta.let) & \underline{\mathsf{let}}\ x = M\ \underline{\mathsf{in}}\ x & \longrightarrow & M & \\
(assoc) & \underline{\mathsf{let}}\ y = (\underline{\mathsf{let}}\ x = L\ \underline{\mathsf{in}}\ M)\ \underline{\mathsf{in}}\ N & \longrightarrow & \underline{\mathsf{let}}\ x = L\ \underline{\mathsf{in}}\ (\underline{\mathsf{let}}\ y = M\ \underline{\mathsf{in}}\ N) & \text{if } x \notin fv(N) \\
(let.1) & P \,\underline{@}\, N & \longrightarrow & \underline{\mathsf{let}}\ x = P\ \underline{\mathsf{in}}\ x \,\underline{@}\, N & \text{if } x \notin fv(N) \\
(let.2) & V \,\underline{@}\, Q & \longrightarrow & \underline{\mathsf{let}}\ y = Q\ \underline{\mathsf{in}}\ V \,\underline{@}\, y & \text{if } y \notin fv(V) \\
(\beta.\mathcal{S}) & \underline{\langle} J_\bullet[\underline{\mathcal{S}} \,\underline{@}\, W] \underline{\rangle} & \longrightarrow & \underline{\langle} W \,\underline{@}\, (\underline{\lambda} y.\, \underline{\langle} J_\bullet[y] \underline{\rangle}) \underline{\rangle} & \\
(\beta.\mathcal{S}_0) & \underline{\langle} J_\bullet[\underline{\mathcal{S}_0} \,\underline{@}\, W] \underline{\rangle} & \longrightarrow & W \,\underline{@}\, (\underline{\lambda} y.\, \underline{\langle} J_\bullet[y] \underline{\rangle}) & \\
(\beta.\mathcal{R}) & \underline{\langle} V \underline{\rangle} & \longrightarrow & V &
\end{array}
$$

図 1. DS 項の構文

図 1 に DS 言語の定義を示す。これは let 文を持つ単純型付き λ 計算に shift, shift0, reset が加わった言語となっている。値の中の $\underline{\mathcal{S}}$ は shift 、$\underline{\mathcal{S}_0}$ は shift0 を、値以外の項の中の $\underline{\langle} \cdot \underline{\rangle}$ は reset を表す。先行研究 [2, 19] と同じく shift や shift0 は定数として定義し、$\underline{\mathcal{S}} \,\underline{@}\, (\underline{\lambda} k.\, M)$ や $\underline{\mathcal{S}_0} \,\underline{@}\, (\underline{\lambda} k.\, M)$ の形で使う。これで、直近の reset までの継続を k に束縛して M を実行するという意味になる。

簡約規則も図 1 に示されている。最初の 4 つは普通の β 簡約と η 簡約である。$(\beta.\mathcal{R})$ は、reset の中が値になったら reset を外せることを示す。$(\beta.\mathcal{S})$ は、shift の簡約規則で、直近のコンテキスト $J_\bullet$ を取ってきて、$\underline{\lambda} y.\, \underline{\langle} J_\bullet[y] \underline{\rangle}$ という関数にし、それを W に渡すことで W の中で継続を使えるようにしている。ここで $J_\bullet$ は、項の中で reset に囲まれていない部分式を表すコンテキストである。コンテキストの定義に出てくる Δ は、コンテキストが reset に囲まれていることを示す $\bullet$ か、reset に囲まれていないことを示す k のどちらかの値をとる。$(\beta.\mathcal{S}_0)$ は、shift0 の簡約規則で、$(\beta.\mathcal{S})$ と同じように継続を扱えるようにしているが、簡約後の項全体に reset がつかない部分が大きな特徴である。これにより、より外側の継続（メタ継続）を捕捉できるようになる。

$(let.1)$ と $(let.2)$ は、部分式に名前を与えて A-正規形に変換するための規則である。これらを使って、値以外の項に名前を割り当てて、$(assoc)$ を使って入れ子になった let 文をフラットにしている。これら 3 つの規則を可能な限り適用した結果、得られる項が A-正規形となる。

図 2 に DS 項の型規則を示す。この型規則は、トレイルとメタ継続を用いた 4 種の限定継続演算子の型規則 [13] からトレイルを除いた形になっており、記法もそれに合わせている。型判断 $\Gamma \vdash M : \tau \, \langle \sigma_\alpha \rangle \, \alpha \, \langle \sigma_\beta \rangle \, \beta$ は「型環境 Γ のもとで式 M は τ 型を持ち、『 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、 σ_β 型のメタ継続のもとで実行すると、最終的に β 型になる。」と読む。

$$\frac{}{\Gamma \vdash n : \mathbb{N}\,\langle\sigma_\alpha\rangle\,\alpha\,\langle\sigma_\alpha\rangle\,\alpha}\;(\mathrm{T}\textsc{Num}) \qquad \frac{}{\Gamma \vdash b : \mathbb{B}\,\langle\sigma_\alpha\rangle\,\alpha\,\langle\sigma_\alpha\rangle\,\alpha}\;(\mathrm{T}\textsc{Bol})$$

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau\,\langle\sigma_\alpha\rangle\,\alpha\,\langle\sigma_\alpha\rangle\,\alpha}\;(\mathrm{T}\textsc{Var}) \qquad \frac{\Gamma, x : \tau_1 \vdash M : \tau_2\,\langle\sigma_\alpha\rangle\,\alpha\,\langle\sigma_\beta\rangle\,\beta}{\Gamma \vdash \underline{\lambda} x.\, M : (\tau_1 \to \tau_2\,\langle\sigma_\alpha\rangle\,\alpha\,\langle\sigma_\beta\rangle\,\beta)\,\langle\sigma_\gamma\rangle\,\gamma\,\langle\sigma_\gamma\rangle\,\gamma}\;(\mathrm{T}\textsc{Fun})$$

$$\frac{\textsf{id-cont-type}\,(\gamma, \sigma_{id}, \gamma')}{\begin{array}{l}\Gamma \vdash \underline{\mathcal{S}} : (((\tau \to \tau_1\,\langle\sigma_1\rangle\,\tau_2\,\langle\sigma_2\rangle\,\alpha) \to \gamma\,\langle\sigma_{id}\rangle\,\gamma'\,\langle\sigma_\beta\rangle\,\beta) \\ \qquad\qquad \to \tau\,\langle(\tau_1 \triangleright \langle\sigma_1\rangle\,\tau_2) :: \sigma_2\rangle\,\alpha\,\langle\sigma_\beta\rangle\,\beta)\langle\sigma_\gamma\rangle\gamma\langle\sigma_\gamma\rangle\gamma\end{array}}\;(\mathrm{T}\textsc{Shift})$$

$$\frac{}{\begin{array}{l}\Gamma \vdash \underline{\mathcal{S}_0} : (((\tau \to \tau_1\,\langle\sigma_1\rangle\,\tau_2\,\langle\sigma_2\rangle\,\alpha) \to \tau_0\,\langle\sigma_0\rangle\,\tau_0'\,\langle\sigma_0'\rangle\,\beta) \\ \qquad\qquad \to \tau\,\langle(\tau_1 \triangleright \langle\sigma_1\rangle\,\tau_2) :: \sigma_2\rangle\,\alpha\,\langle(\tau_0 \triangleright \langle\sigma_0\rangle\,\tau_0') :: \sigma_0'\rangle\,\beta)\langle\sigma_\gamma\rangle\gamma\langle\sigma_\gamma\rangle\gamma\end{array}}\;(\mathrm{T}\textsc{Shift0})$$

$$\frac{\Gamma \vdash M : (\tau_1 \to \tau_2\,\langle\sigma_\alpha\rangle\,\alpha\,\langle\sigma_\beta\rangle\,\beta)\,\langle\sigma_\gamma\rangle\,\gamma\,\langle\sigma_\delta\rangle\,\delta \quad \Gamma \vdash N : \tau_1\,\langle\sigma_\beta\rangle\,\beta\,\langle\sigma_\gamma\rangle\,\gamma}{\Gamma \vdash M \mathbin{\underline{@}} N : \tau_2\,\langle\sigma_\alpha\rangle\,\alpha\,\langle\sigma_\delta\rangle\,\delta}\;(\mathrm{T}\textsc{App})$$

$$\frac{\Gamma \vdash M : \tau_1\,\langle\sigma_\beta\rangle\,\beta\,\langle\sigma_\gamma\rangle\,\gamma \quad \Gamma, x : \tau_1 \vdash N : \tau_2\,\langle\sigma_\alpha\rangle\,\alpha\,\langle\sigma_\beta\rangle\,\beta}{\Gamma \vdash \underline{\textsf{let}}\ x = M\ \underline{\textsf{in}}\ N : \tau_2\,\langle\sigma_\alpha\rangle\,\alpha\,\langle\sigma_\gamma\rangle\,\gamma}\;(\mathrm{T}\textsc{Let})$$

$$\frac{\textsf{id-cont-type}\,(\gamma, \sigma_{id}, \gamma') \quad \Gamma \vdash M : \gamma\,\langle\sigma_{id}\rangle\,\gamma'\,\langle((\tau \triangleright \langle\sigma_\alpha\rangle\,\alpha) :: \sigma_\beta)\rangle\,\beta}{\Gamma \vdash \underline{\langle} M \underline{\rangle} : \tau\,\langle\sigma_\alpha\rangle\,\alpha\,\langle\sigma_\beta\rangle\,\beta}\;(\mathrm{T}\textsc{Reset})$$

$$\begin{array}{lcl}\textsf{id-cont-type}\,(\tau, \bullet_\sigma, \tau') & = & \tau \equiv \tau' \\ \textsf{id-cont-type}\,(\tau, (\tau_1 \triangleright \langle\sigma_1\rangle\,\tau_1') :: \sigma_2, \tau') & = & (\tau \equiv \tau_1) \land (\tau' \equiv \tau_1') \land (\sigma_1 \equiv \sigma_2)\end{array}$$

図 2. DS 項の型規則

3.2 CPS 言語 $\lambda^*_{c\mathcal{S}_0}$

次に、CPS 言語の定義を図 3 に示す。shift0/reset0 ではメタ継続を扱うため、CPS 言語の項は継続とメタ継続を引数に取る形になっている。項の中の、$K_\Delta \mathbin{\underline{@}} V \mathbin{\underline{@}} G_\Theta$ は、値 V と メタ継続 G_Θ を 継続 K_Δ に返すことを、$V \mathbin{\underline{@}} W \mathbin{\underline{@}} K_\Delta \mathbin{\underline{@}} G_\Theta$ は、CPS での関数 V の呼び出しを表す。値の中の S, S_0 は、それぞれ DS 項の $\underline{\mathcal{S}}, \underline{\mathcal{S}_0}$ を CPS 変換して得られる項を表している。

継続 K_Δ に出てくる Δ は、k または $\bullet$ のどちらかの値をとる。ここで k は継続を表す定数である。Δ を展開すると

$$\begin{array}{lrcl}\text{continuations} & J_k, K_k & ::= & k \mid \underline{\lambda} x.\, \underline{\lambda} g.\, M_{kg} \\ & J_\bullet, K_\bullet & ::= & k_{id} \mid \underline{\lambda} x.\, \underline{\lambda} g.\, M_{\bullet g}\end{array}$$

となる。K_k と $K_\bullet$ の区別は、それを使っている項 $M_{\Delta g}$ が CPS に変換される前に reset に囲まれていたかどうかを表す。例えば、$\underline{\lambda} x.\, x$ は CPS 変換すると、$\underline{\lambda} k.\, \underline{\lambda} g.\, k \mathbin{\underline{@}} (\underline{\lambda} x.\, \underline{\lambda} k'.\, \underline{\lambda} g'.\, k' \mathbin{\underline{@}} x \mathbin{\underline{@}} g') \mathbin{\underline{@}} g$ となるが、これは元の項が reset に囲まれていなかったため、継続部分 k が K_k の形になっている。一方で、$\underline{\langle} 1 \underline{\rangle}$ は CPS 変換すると、$\underline{\lambda} k.\, \underline{\lambda} g.\, k_{id} \mathbin{\underline{@}} 1 \mathbin{\underline{@}} (k \mathbin{\underline{::}} g)$ となり、継続部分 k_{id} が $K_\bullet$ の形になっており、reset に囲まれていたことを示している。

メタ継続 G_Θ に出てくる Θ は、g または Δ のどちらかの値をとる。ここで g はメタ継続が空であることを表す。つまり、G_g と G_Δ はメタ継続の有無によって区別されている。ここで、先ほどの例でメタ継続部分を見てみる。$\underline{\lambda} x.\, x$ を CPS 変換した $\underline{\lambda} k.\, \underline{\lambda} g.\, k \mathbin{\underline{@}} (\underline{\lambda} x.\, \underline{\lambda} k'.\, \underline{\lambda} g'.\, k' \mathbin{\underline{@}} x \mathbin{\underline{@}} g') \mathbin{\underline{@}} g$ では、メタ継続部分 g が G_g の形になっており、メタ継続が空であることを示している。一方で、$\underline{\langle} 1 \underline{\rangle}$ を CPS 変換した $\underline{\lambda} k.\, \underline{\lambda} g.\, k_{id} \mathbin{\underline{@}} 1 \mathbin{\underline{@}} (k \mathbin{\underline{::}} g)$ では、メタ継続部分 $(k \mathbin{\underline{::}} g)$ が G_Δ の形になっている。継続に出てくる Δ とメタ継続に出てくる Θ の組み合わせは以下の 4 通りが考えられる。それぞれの場合を例を挙げて説明する。

$$
\begin{array}{rlll}
\text{types} & \tau, \alpha, \beta & ::= & \mathbb{N} \mid \mathbb{B} \mid \tau_1 \to (\tau_2 \to \sigma_\alpha \to \alpha) \to \sigma_\beta \to \beta \\
\text{cont types} & \delta & ::= & \tau_2 \to \sigma_\alpha \to \alpha \\
\text{metacont types} & \sigma & ::= & [\,]_\sigma \mid \delta :: \sigma \\
\\
\text{root} & R & ::= & \underline{\lambda} k.\, \underline{\lambda} g.\, M_{kg} \\
\text{terms} & M_{\Delta\Theta} & ::= & K_\Delta \mathbin{\underline{@}} V \mathbin{\underline{@}} G_\Theta \mid V \mathbin{\underline{@}} W \mathbin{\underline{@}} K_\Delta \mathbin{\underline{@}} G_\Theta \\
\text{values} & V, W & ::= & x \mid (\underline{\lambda} x.\, \underline{\lambda} k.\, \underline{\lambda} g.\, M_{kg}) \mid S \mid S_0 \\
\text{shift} & S & ::= & \underline{\lambda} w.\, \underline{\lambda} j.\, \underline{\lambda} g.\, w \mathbin{\underline{@}} (\underline{\lambda} y.\, \underline{\lambda} k.\, \underline{\lambda} g.\, j \mathbin{\underline{@}} y \mathbin{\underline{@}} (k \mathbin{\underline{::}} g)) \mathbin{\underline{@}} k_{id} \mathbin{\underline{@}} g \\
\text{shift0} & S_0 & ::= & \underline{\lambda} w.\, \underline{\lambda} j.\, \underline{\lambda} (k_0 \mathbin{\underline{::}} g).\, w \mathbin{\underline{@}} (\underline{\lambda} y.\, \underline{\lambda} k.\, \underline{\lambda} g.\, j \mathbin{\underline{@}} y \mathbin{\underline{@}} (k \mathbin{\underline{::}} g)) \mathbin{\underline{@}} k_0 \mathbin{\underline{@}} g \\
\text{continuations} & J_\Delta, K_\Delta & ::= & {}_{(\Delta = k)}\, k \mid_{(\Delta = \bullet)} k_{id} \mid \underline{\lambda} x.\, \underline{\lambda} g.\, M_{\Delta g} \\
\text{meta continuations} & G_\Theta & ::= & {}_{(\Theta = g)}\, g \mid_{(\Theta = \Delta)} G_\Delta \\
\text{non-empty metaconts} & G_\Delta & ::= & K_\Delta \mathbin{\underline{::}} g \mid K_\bullet \mathbin{\underline{::}} G_\Delta
\end{array}
$$

$$
\begin{array}{llcll}
(\beta.v) & (\underline{\lambda} x.\, \underline{\lambda} k.\, \underline{\lambda} g.\, M_{kg}) \mathbin{\underline{@}} V \mathbin{\underline{@}} K_\Delta \mathbin{\underline{@}} G_\Theta & \longrightarrow & M_{kg}[V/x, K_\Delta/k, G_\Theta/g] & \\
(\eta.v) & (\underline{\lambda} x.\, \underline{\lambda} k.\, \underline{\lambda} g.\, V \mathbin{\underline{@}} x \mathbin{\underline{@}} k \mathbin{\underline{@}} g) & \longrightarrow & V & \text{if } x, k, g \notin fv(V) \\
(\beta.let) & (\underline{\lambda} x.\, \underline{\lambda} g.\, M_{\Delta g}) \mathbin{\underline{@}} V \mathbin{\underline{@}} G_\Theta & \longrightarrow & M_{\Delta g}[V/x, G_\Theta/g] & \\
(\eta.let) & \underline{\lambda} x.\, \underline{\lambda} g.\, K_\Delta \mathbin{\underline{@}} x \mathbin{\underline{@}} g & \longrightarrow & K_\Delta & \text{if } x, g \notin fv(K) \\
(\beta.\mathcal{S}) & S \mathbin{\underline{@}} W \mathbin{\underline{@}} J_\bullet \mathbin{\underline{@}} G_\Theta & \longrightarrow & W \mathbin{\underline{@}} (\underline{\lambda} y.\, \underline{\lambda} k.\, \underline{\lambda} g.\, J_\bullet \mathbin{\underline{@}} y \mathbin{\underline{@}} (k \mathbin{\underline{::}} g)) \mathbin{\underline{@}} k_{id} \mathbin{\underline{@}} G_\Theta & \\
(\beta.\mathcal{S}_0) & S_0 \mathbin{\underline{@}} W \mathbin{\underline{@}} J_\bullet \mathbin{\underline{@}} (K_\Delta \mathbin{\underline{::}} G_\Theta) & \longrightarrow & W \mathbin{\underline{@}} (\underline{\lambda} y.\, \underline{\lambda} k.\, \underline{\lambda} g.\, J_\bullet \mathbin{\underline{@}} y \mathbin{\underline{@}} (k \mathbin{\underline{::}} g)) \mathbin{\underline{@}} K_\Delta \mathbin{\underline{@}} G_\Theta & \\
(\beta.\mathcal{R}) & k_{id} \mathbin{\underline{@}} V \mathbin{\underline{@}} (K_\Delta \mathbin{\underline{::}} G_\Theta) & \longrightarrow & K_\Delta \mathbin{\underline{@}} V \mathbin{\underline{@}} G_\Theta &
\end{array}
$$

図 3. CPS 項の構文

$\Delta = k, \Theta = g$ の場合

CPS 変換前の項が reset に囲まれておらず、メタ継続も空である場合に該当する。例えば、$\underline{\lambda} x.\, x$ を CPS 変換すると $\underline{\lambda} k.\, \underline{\lambda} g.\, k \mathbin{\underline{@}} (\underline{\lambda} x.\, \underline{\lambda} k'.\, \underline{\lambda} g'.\, k' \mathbin{\underline{@}} x \mathbin{\underline{@}} g') \mathbin{\underline{@}} g$ となり、$K_k \mathbin{\underline{@}} V \mathbin{\underline{@}} G_g$ の形になる。

$\Delta = \bullet, \Theta = g$ の場合

CPS 変換前の項が reset に囲まれており、メタ継続が空である場合に該当する。このような項は、継続の中に現れる。例えば、$\underline{\langle} \underline{\mathsf{let}}\ x = 2\ \underline{\mathsf{in}}\ x \underline{\rangle}$ の CPS 変換は $\underline{\lambda} k.\, \underline{\lambda} g.\, (\underline{\lambda} x.\, \underline{\lambda} g.\, k_{id} \mathbin{\underline{@}} x \mathbin{\underline{@}} g) \mathbin{\underline{@}} 2 \mathbin{\underline{@}} (k \mathbin{\underline{::}} g)$ となり、項は $K_\bullet \mathbin{\underline{@}} V \mathbin{\underline{@}} G_\Delta$ の形になる。このうち、継続部分 $K_\bullet$ である $\underline{\lambda} x.\, \underline{\lambda} g.\, M_{\Delta g}$ の $M_{\Delta g}$ に着目すると $k_{id} \mathbin{\underline{@}} x \mathbin{\underline{@}} g$ となっており、$K_\bullet \mathbin{\underline{@}} V \mathbin{\underline{@}} G_g$ の形になっていることがわかる。

$\Delta = \bullet, \Theta = \Delta$ の場合

CPS 変換前の項が reset に囲まれており、メタ継続が存在する場合に該当する。例えば、$\underline{\langle} 1 \underline{\rangle}$ を CPS 変換すると $\underline{\lambda} k.\, \underline{\lambda} g.\, k_{id} \mathbin{\underline{@}} 1 \mathbin{\underline{@}} (k \mathbin{\underline{::}} g)$ となり、$K_\bullet \mathbin{\underline{@}} V \mathbin{\underline{@}} G_\Delta$ の形になる。

$\Delta = k, \Theta = \Delta$ の場合

CPS 変換前の項が reset に囲まれておらず、メタ継続が存在する場合に該当するが、そのような項は存在しない。メタ継続は、現在の継続のより外側の継続、すなわち、直近の reset の外側かつその次の reset までの継続のことを指すので、CPS 変換前の項が reset に囲まれていない場合、メタ継続は空になる。

これを踏まえて、項 $M_{\Delta\Theta}$ の定義について Δ と Θ を展開すると、

$$
\begin{array}{rlll}
\text{terms} & M_{\Delta g} & ::= & K_\Delta \mathbin{\underline{@}} V \mathbin{\underline{@}} g \mid V \mathbin{\underline{@}} W \mathbin{\underline{@}} K_\Delta \mathbin{\underline{@}} g \\
& M_{\bullet\Delta} & ::= & K_\bullet \mathbin{\underline{@}} V \mathbin{\underline{@}} G_\Delta \mid V \mathbin{\underline{@}} W \mathbin{\underline{@}} K_\bullet \mathbin{\underline{@}} G_\Delta
\end{array}
$$

となる。ここで、空でないメタ継続の定義を見ると、$K_\Delta \mathbin{\underline{::}} g \mid K_\bullet \mathbin{\underline{::}} G_\Delta$ となっており、展開した項の定義と対応していることがわかる。

また、簡約規則も図 3 に示す。CPS 言語であるため $(\beta.v), (\beta.let)$ では、引数の代入だけでなく継続やメタ継続の代入も同時に行われるように定義されている。基本的にはDS 言語の簡約規則と対応しているが、$(let.1), (let.2), (assoc)$ の規則は含まれていない。これは、CPS 言語が既に部分式に名前を与えた形になっているためである。

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash_v x : \tau}\ (\textsc{TVar}) \qquad \frac{}{\Gamma \vdash_v n : \mathbb{N}}\ (\textsc{TNum}) \qquad \frac{}{\Gamma \vdash_v b : \mathbb{B}}\ (\textsc{TBol})$$

$$\frac{\Gamma, x : \tau_1\, [k : \tau_2 \to \sigma_\alpha \to \alpha, g : \sigma_\beta] \vdash M : \beta}{\Gamma \vdash_v \underline{\lambda} x.\, \underline{\lambda} k.\, \underline{\lambda} g.\, M_{kg} : \tau_1 \to (\tau_2 \to \sigma_\alpha \to \alpha) \to \sigma_\beta \to \beta}\ (\textsc{TFun})$$

$$\frac{\mathsf{id\text{-}cont\text{-}type}\,(\gamma \to \sigma_{id} \to \gamma')}{\begin{array}{l}\Gamma \vdash_v S : ((\tau \to (\tau_1 \to \sigma_1 \to \tau_2) \to \sigma_2 \to \alpha) \to (\gamma \to \sigma_{id} \to \gamma') \to \sigma_\beta \to \beta) \\ \qquad \to (\tau \to ((\tau_1 \to \sigma_2 \to \tau_2) :: \sigma_2) \to \alpha) \to \sigma_\beta \to \beta\end{array}}\ (\textsc{TShift})$$

$$\frac{}{\begin{array}{l}\Gamma \vdash_v S_0 : ((\tau \to (\tau_1 \to \sigma_1 \to \tau_2) \to \sigma_2 \to \alpha) \to (\tau_0 \to \sigma_0 \to \tau'_0) \to \sigma'_0 \to \beta) \\ \qquad \to (\tau \to ((\tau_1 \to \sigma_2 \to \tau_2) :: \sigma_2) \to \alpha) \to (\tau_0 \to \sigma_0 \to \tau'_0) :: \sigma'_0 \to \beta\end{array}}\ (\textsc{TShift0})$$

$$\frac{\Gamma[\Delta : \delta] \vdash_k K_\Delta : \tau \to \sigma \to \alpha \quad \Gamma \vdash_v V : \tau \quad \Gamma[\Theta : \theta, g : \sigma'] \vdash_g G_\Theta : \sigma}{\Gamma[(\Delta +\!\!+ \Theta) : (\delta +\!\!+ \theta), g : \sigma'] \vdash K_\Delta \,\underline{@}\, V \,\underline{@}\, G_\Theta : \alpha}\ (\textsc{TVal})$$

$$\frac{\begin{array}{ll}\Gamma \vdash_v V : \tau_1 \to (\tau_2 \to \sigma_\alpha \to \alpha) \to \sigma_\beta \to \beta & \Gamma \vdash_v W : \tau_1 \\ \Gamma[\Delta : \delta] \vdash_k K_\Delta : \tau_2 \to \sigma_\alpha \to \alpha & \Gamma[\Theta : \theta, g : \sigma] \vdash_g G_\Theta : \sigma_\beta\end{array}}{\Gamma[(\Delta +\!\!+ \Theta) : (\delta +\!\!+ \theta), g : \sigma] \vdash V \,\underline{@}\, W \,\underline{@}\, K_\Delta \,\underline{@}\, G_\Theta : \beta}\ (\textsc{TApp})$$

$$\frac{}{\Gamma[k : \tau_1 \to \sigma \to \tau_2] \vdash_k k : \tau_1 \to \sigma \to \tau_2}\ (\textsc{TKVar}) \quad \frac{\mathsf{id\text{-}cont\text{-}type}\,(\gamma \to \sigma_{id} \to \gamma')}{\Gamma[\bullet : \gamma \to \sigma_{id} \to \gamma'] \vdash_k k_{id} : \gamma \to \sigma_{id} \to \gamma'}\ (\textsc{TKId})$$

$$\frac{\Gamma, x : \tau\, [\Delta : \delta, g : \sigma_\alpha] \vdash M_{\Delta g} : \alpha}{\Gamma[\Delta : \delta] \vdash_k \underline{\lambda} x.\, \underline{\lambda} g.\, M_{\Delta g} : \tau \to \sigma_\alpha \to \alpha}\ (\textsc{TKLet})$$

$$\frac{}{\Gamma[g : \sigma, g : \sigma] \vdash_g g : \sigma}\ (\textsc{TGVar}) \quad \frac{\Gamma[\Delta : \delta] \vdash_k K_\Delta : \delta' \quad \Gamma[\Theta : \theta, g : \sigma] \vdash_g G_\Theta : \sigma'}{\Gamma[(\Delta +\!\!+ \Theta) : (\delta +\!\!+ \theta), g : \sigma] \vdash_g (K_\Delta \,\underline{::}\, G_\Theta) : (\delta' :: \sigma')}\ (\textsc{TGCons})$$

$$\begin{array}{lcl}\mathsf{id\text{-}cont\text{-}type}\,(\tau \to [\,] \to \tau') & = & \tau \equiv \tau' \\ \mathsf{id\text{-}cont\text{-}type}\,(\tau \to ((\tau_1 \to \sigma_1 \to \tau'_1) :: \sigma_2) \to \tau') & = & (\tau \equiv \tau_1) \wedge (\tau' \equiv \tau'_1) \wedge (\sigma_1 \equiv \sigma_2)\end{array}$$

$$\begin{array}{cccc}\Delta & +\!\!+ & g & = \Delta \\ \bullet & +\!\!+ & \Delta & = \Delta\end{array}$$

図 4. CPS 項の型規則

図 4 に CPS 項の型規則を示す。型判断は、値用、項用、継続用、メタ継続用の 4 つに分かれている。そのうち、値用以外の型判断では型環境と別に、継続 Δ の型やメタ継続 Θ の型、初期のメタ継続 g の型を管理しており [...] で囲んで表記している。例えば、値用の規則 (TFun) の前提部では、k や g を特別な変数として環境に入れずに管理するため、k と g の型を [...] で囲んで表記している。これは、Agda 上で定式化する際、Parameterized Higher-Order Abstract Syntax (PHOAS) [4] を用いて束縛情報を表現しているが、そこに k や g を含めないことを意味している。[...] に入るものは、項用の型判断 $\Gamma[\Delta : \delta, g : \sigma] \vdash M : \tau$ で継続 Δ と初期のメタ継続 g、継続用の型判断 $\Gamma[\Delta : \delta] \vdash_k K : \delta$ で継続 Δ 、メタ継続用の型判断 $\Gamma[\Theta : \theta, g : \sigma] \vdash_g G : \sigma'$ でメタ継続 Θ と初期のメタ継続 g である。

[...] の中に入った Δ は最終的に継続の型規則で参照される。その際、Δ が $\bullet$ の場合 (TKId) では、初期継続のための型の制約 id-cont-type がついている。$\gamma \to \sigma_{id} \to \gamma'$ 型を持つ継続が初期継続であるためには、$\mathsf{id\text{-}cont\text{-}type}\,(\gamma \to \sigma_{id} \to \gamma')$ を満たせば良いことになる。id-cont-type は、(TShift)

の前提部にも使われている。

同様に、[...] の中に入った Θ はメタ継続の型規則で参照される。その際、Θ が g の場合 (TGVar) では g の型を [...] に追加し、そのまま引き回す。つまり、型規則 (TGVar) の $[g:\sigma, g:\sigma]$ は、1 つ目がメタ継続 Θ の型、2 つ目が初期のメタ継続 g の型であり、これらの型は一致するようになっている。もう一方のメタ継続の型規則 (TGCons) は、Θ が Δ の場合である。結論部では Δ と Θ の合成 (Δ++Θ) があり、Θ が g の場合は Δ を使い、Θ が Δ の場合は Δ が $\bullet$ になるため Δ を使う。Δ と Θ の合成は、項の型規則 (TVal), (TApp) でも行われ、ここでは継続 Δ の型として [...] に入る。

4 CPS 変換

本節では、DS 言語 $\lambda_{c\mathcal{S}_0}$ の項を CPS 言語 $\lambda^*_{c\mathcal{S}_0}$ の項に変換する CPS 変換を定義する。本研究では、冗長な項が生成されず扱いやすいコロン変換（図 10）を定義した。Danvy ら [8] の手法に沿って、単純な CPS 変換から導出した過程を説明する。

4.1 単純な 2CPS 変換

図 5 に単純な CPS 変換を示す。4 種の限定継続演算子に対するトレイルとメタ継続を用いたインタプリタ [13] からトレイルを除去し、メタ継続のみを扱うインタプリタに変形したものを参考にしている。shift0/reset0 に対応するために、継続とメタ継続の両方を引数に取る形になっている。

$$
\begin{array}{rcl}
\llbracket x \rrbracket & = & \underline{\lambda} k.\, \underline{\lambda} g.\, k \mathbin{\underline{@}} x \mathbin{\underline{@}} g \\
\llbracket \underline{\lambda} x.\, M \rrbracket & = & \underline{\lambda} k.\, \underline{\lambda} g.\, k \mathbin{\underline{@}} (\lambda x.\, \underline{\lambda} k'.\, \underline{\lambda} g'.\, \llbracket M \rrbracket \mathbin{\underline{@}} k' \mathbin{\underline{@}} g') \mathbin{\underline{@}} g \\
\llbracket M_1 \mathbin{\underline{@}} M_2 \rrbracket & = & \underline{\lambda} k.\, \underline{\lambda} g.\, \llbracket M_1 \rrbracket \mathbin{\underline{@}} (\underline{\lambda} v_1.\, \underline{\lambda} g_1.\, \llbracket M_2 \rrbracket \mathbin{\underline{@}} (\underline{\lambda} v_2.\, \underline{\lambda} g_2.\, v_1 \mathbin{\underline{@}} v_2 \mathbin{\underline{@}} k \mathbin{\underline{@}} g_2) \mathbin{\underline{@}} g_1) \mathbin{\underline{@}} g \\
\llbracket \underline{\mathsf{let}}\ x = M\ \underline{\mathsf{in}}\ N \rrbracket & = & \underline{\lambda} k.\, \underline{\lambda} g.\, \llbracket M \rrbracket \mathbin{\underline{@}} (\underline{\lambda} x.\, \underline{\lambda} g'.\, \llbracket N \rrbracket \mathbin{\underline{@}} k \mathbin{\underline{@}} g') \mathbin{\underline{@}} g \\
\llbracket \underline{\mathcal{S}} \rrbracket & = & \underline{\lambda} k.\, \underline{\lambda} g.\, k \mathbin{\underline{@}} (\lambda w.\, \underline{\lambda} k'.\, \underline{\lambda} g'.\, w \mathbin{\underline{@}} (\lambda y.\, \underline{\lambda} k''.\, \underline{\lambda} g''.\, k' \mathbin{\underline{@}} y \mathbin{\underline{@}} (k'' \mathbin{\underline{::}} g'')) \mathbin{\underline{@}} k_{id} \mathbin{\underline{@}} g') \mathbin{\underline{@}} g \\
\llbracket \underline{\mathcal{S}_0} \rrbracket & = & \underline{\lambda} k.\, \underline{\lambda} g.\, k \mathbin{\underline{@}} (\lambda w.\, \underline{\lambda} k'.\, \underline{\lambda} (k_0 \mathbin{\underline{::}} g').\, w \mathbin{\underline{@}} (\lambda y.\, \underline{\lambda} k''.\, \underline{\lambda} g''.\, k' \mathbin{\underline{@}} y \mathbin{\underline{@}} (k'' \mathbin{\underline{::}} g'')) \mathbin{\underline{@}} k_0 \mathbin{\underline{@}} g') \mathbin{\underline{@}} g \\
\llbracket \underline{\langle} M \underline{\rangle} \rrbracket & = & \underline{\lambda} k.\, \underline{\lambda} g.\, \llbracket M \rrbracket \mathbin{\underline{@}} k_{id} \mathbin{\underline{@}} (k \mathbin{\underline{::}} g)
\end{array}
$$

図 5. 単純な 2CPS 変換 (defunctionalized)

ここでアンダーラインが引かれているものは、入力プログラムの構文木を表している。この変換で出力される項は非常に複雑になってしまうため、次節以降でよりコンパクトな結果を出力する CPS 変換を導出する。

4.2 administrative η-redex を生成しない CPS 変換

前節の変換では、「CPS 変換をするために導入された式」が出力に含まれてしまうため、複雑になっていた。そこで、CPS 変換の過程で計算を進めることで、不要な式を出力に含めないような変換がある。これが Danvy and Filinski の 1-pass CPS 変換 [7] である。しかし、この変換を用いても簡約は保存されない。administrative η-redex と呼ばれる冗長な式が出力に含まれてしまうためである。そこで、administrative η-redex を生成しないように CPS 変換を定義する。図 6 にその変換を示す。

この定義では、図 5 に出てくる $\underline{@}$ のうちいくつかを $\overline{@}$ にし、CPS 変換中に簡約を行う。ここで、オーバーラインが引かれているものは、実行可能な（CPS 変換中に実行される）プログラムである。以後、オーバーラインが引かれているものを static 、アンダーラインが引かれているものを dynamic と表現する。

$$
\begin{array}{rcl}
k_{id} \,\underline{@}\, v \,\underline{@}\, [\,] & = & v \\
k_{id} \,\underline{@}\, v \,\underline{@}\, (\kappa \;\underline{::}\; g) & = & \kappa \,\underline{@}\, v \,\underline{@}\, g
\end{array}
$$

$$
\begin{array}{rcl}
\llbracket x \rrbracket & = & \overline{\lambda} k.\, \overline{\lambda} g.\, k \,\overline{@}\, x \,\overline{@}\, g \\
\llbracket \underline{\lambda} x.\, M \rrbracket & = & \overline{\lambda} k.\, \overline{\lambda} g.\, k \,\overline{@}\, (\underline{\lambda} x.\, \underline{\lambda} k.\, \underline{\lambda} g.\, \llbracket M \rrbracket' \,\overline{@}\, k \,\overline{@}\, g) \,\overline{@}\, g \\
\llbracket M_1 \,\underline{@}\, M_2 \rrbracket & = & \overline{\lambda} k.\, \overline{\lambda} g.\, \llbracket M_1 \rrbracket \,\overline{@}\, (\overline{\lambda} v_1.\, \overline{\lambda} g_1.\, \llbracket M_2 \rrbracket \,\overline{@}\, (\overline{\lambda} v_2.\, \overline{\lambda} g_2.\, v_1 \,\underline{@}\, v_2 \,\underline{@}\, (\underline{\lambda} a.\, \underline{\lambda} g'.\, k \,\overline{@}\, a \,\overline{@}\, g') \,\underline{@}\, g_2) \,\overline{@}\, g_1) \,\overline{@}\, g \\
\llbracket \underline{\mathsf{let}}\ x = M\ \underline{\mathsf{in}}\ N \rrbracket & = & \overline{\lambda} k.\, \overline{\lambda} g.\, \llbracket M \rrbracket' \,\overline{@}\, (\underline{\lambda} x.\, \underline{\lambda} g'.\, \llbracket N \rrbracket' \,\overline{@}\, (\underline{\lambda} a.\, \underline{\lambda} g''.\, k \,\overline{@}\, a \,\overline{@}\, g'') \,\overline{@}\, g') \,\overline{@}\, g \\
\llbracket \underline{\mathcal{S}} \rrbracket & = & \overline{\lambda} k.\, \overline{\lambda} g.\, k \,\overline{@}\, (\underline{\lambda} w.\, \underline{\lambda} k'.\, \underline{\lambda} g'.\, w \,\underline{@}\, (\underline{\lambda} y.\, \underline{\lambda} k''.\, \underline{\lambda} g''.\, k' \,\underline{@}\, y \,\underline{@}\, (k'' \;\underline{::}\; g'')) \,\underline{@}\, k_{id} \,\underline{@}\, g') \,\overline{@}\, g \\
\llbracket \underline{\mathcal{S}_0} \rrbracket & = & \overline{\lambda} k.\, \overline{\lambda} g.\, k \,\overline{@}\, (\underline{\lambda} w.\, \underline{\lambda} k'.\, \underline{\lambda} (k_0 \;\underline{::}\; g').\, w \,\underline{@}\, (\underline{\lambda} y.\, \underline{\lambda} k''.\, \underline{\lambda} g''.\, k' \,\underline{@}\, y \,\underline{@}\, (k'' \;\underline{::}\; g'')) \,\underline{@}\, k_0 \,\underline{@}\, g') \,\overline{@}\, g \\
\llbracket \underline{\langle} M \underline{\rangle} \rrbracket & = & \overline{\lambda} k.\, \overline{\lambda} g.\, \llbracket M \rrbracket' \,\overline{@}\, k_{id} \,\overline{@}\, ((\underline{\lambda} a.\, \underline{\lambda} g'.\, k \,\overline{@}\, a \,\overline{@}\, g') \;\underline{::}\; g)
\end{array}
$$

$$
\begin{array}{rcl}
\llbracket x \rrbracket' & = & \overline{\lambda} k.\, \overline{\lambda} g.\, k \,\underline{@}\, x \,\underline{@}\, g \\
\llbracket \underline{\lambda} x.\, M \rrbracket' & = & \overline{\lambda} k.\, \overline{\lambda} g.\, k \,\underline{@}\, (\underline{\lambda} x.\, \underline{\lambda} k.\, \underline{\lambda} g.\, \llbracket M \rrbracket' \,\overline{@}\, k \,\overline{@}\, g) \,\underline{@}\, g \\
\llbracket M_1 \,\underline{@}\, M_2 \rrbracket' & = & \overline{\lambda} k.\, \overline{\lambda} g.\, \llbracket M_1 \rrbracket \,\overline{@}\, (\overline{\lambda} v_1.\, \overline{\lambda} g_1.\, \llbracket M_2 \rrbracket \,\overline{@}\, (\overline{\lambda} v_2.\, \overline{\lambda} g_2.\, v_1 \,\underline{@}\, v_2 \,\underline{@}\, k \,\underline{@}\, g_2) \,\overline{@}\, g_1) \,\overline{@}\, g \\
\llbracket \underline{\mathsf{let}}\ x = M\ \underline{\mathsf{in}}\ N \rrbracket' & = & \overline{\lambda} k.\, \overline{\lambda} g.\, \llbracket M \rrbracket' \,\overline{@}\, (\underline{\lambda} x.\, \underline{\lambda} g'.\, \llbracket N \rrbracket' \,\overline{@}\, k \,\overline{@}\, g') \,\overline{@}\, g \\
\llbracket \underline{\mathcal{S}} \rrbracket' & = & \overline{\lambda} k.\, \overline{\lambda} g.\, k \,\underline{@}\, (\underline{\lambda} w.\, \underline{\lambda} k'.\, \underline{\lambda} g'.\, w \,\underline{@}\, (\underline{\lambda} y.\, \underline{\lambda} k''.\, \underline{\lambda} g''.\, k' \,\underline{@}\, y \,\underline{@}\, (k'' \;\underline{::}\; g'')) \,\underline{@}\, k_{id} \,\underline{@}\, g') \,\underline{@}\, g \\
\llbracket \underline{\mathcal{S}_0} \rrbracket' & = & \overline{\lambda} k.\, \overline{\lambda} g.\, k \,\underline{@}\, (\underline{\lambda} w.\, \underline{\lambda} k'.\, \underline{\lambda} (k_0 \;\underline{::}\; g').\, w \,\underline{@}\, (\underline{\lambda} y.\, \underline{\lambda} k''.\, \underline{\lambda} g''.\, k' \,\underline{@}\, y \,\underline{@}\, (k'' \;\underline{::}\; g'')) \,\underline{@}\, k_0 \,\underline{@}\, g') \,\underline{@}\, g \\
\llbracket \underline{\langle} M \underline{\rangle} \rrbracket' & = & \overline{\lambda} k.\, \overline{\lambda} g.\, \llbracket M \rrbracket' \,\overline{@}\, k_{id} \,\overline{@}\, (k \;\underline{::}\; g)
\end{array}
$$

図 6. administrative η-redex を生成しない 2CPS 変換 (defunctionalized)

4.3　コロン変換

前節で得られた CPS 変換は、

- CPS インタプリタと似た形をしていて対応を取りやすい。
- 2-level にしている部分もインタプリタとコンパイラの違いと捉えると理解しやすい。

などわかりやすいものになっているが、一方で

- 変換と static な実行という二つのものを同時に考える必要がある。
- 2 つの変換の相互再帰で定義されていて、扱いにくい

などの問題がある。これらの問題を解決する手段の一つとして、コロン変換 [16] がある。コロン変換は、図 6 の CPS 変換に対して以下の変換を行うことで得られる [8]。

- 値（trivial term）に対する変換と値以外（serious term）に対する変換に分ける
- $\llbracket\cdot\rrbracket$ は使わず、$\llbracket\cdot\rrbracket'$ のみを使う。関数呼び出しで $\llbracket\cdot\rrbracket$ を使っているが、関数部分と引数部分が値かどうかによって場合分けして展開することで実現できる。

ここからは、上記の流れに沿って具体的にコロン変換を導出する。

まずは、変換する項が値か値以外かで分ける。すると、図 7 のようになる。ここでは、$\llbracket\cdot\rrbracket_v$ と $\llbracket\cdot\rrbracket'_v$ の定義が全く同じになっていることに注意してほしい。

次に、$\llbracket\cdot\rrbracket$ を廃止する。関数呼び出しの部分で $\llbracket\cdot\rrbracket$ が使われているので、M_1 と M_2 が値かどうかで場合分けして展開する。図 8 に、その様子を示す。値を V, W 、値以外を P, Q と表記している。例えば M_1, M_2 がどちらも値であった場合、展開と static な実行を進めると、$\llbracket V\rrbracket_v \,\underline{@}\, \llbracket W\rrbracket_v \,\underline{@}\, k \,\underline{@}\, g$ という式が得られる。ここで、$\llbracket\cdot\rrbracket_v$ と $\llbracket\cdot\rrbracket'_v$ の定義は同じであるため $\llbracket V\rrbracket'_v \,\underline{@}\, \llbracket W\rrbracket'_v \,\underline{@}\, k \,\underline{@}\, g$ とすることができ、最終的に $\llbracket\cdot\rrbracket$ が現れない式が導出できる。他の場合も同様に展開していくと、$\llbracket\cdot\rrbracket_s$ への呼び出しが残ってしまう。しかし、$\llbracket\cdot\rrbracket_s$ と $\llbracket\cdot\rrbracket'_s$ の定義の違いは、継続部分が $(\underline{\lambda} a.\, \underline{\lambda} g'.\, k \,\overline{@}\, a \,\overline{@}\, g')$ か k かのみであるため、現在 static になっている継続を dynamic にしておいた上で $\llbracket\cdot\rrbracket_s$ を呼ぶ代わりに $\llbracket\cdot\rrbracket'_s$ を呼べば良いことがわかる。その結果をまとめると、図 9 のようになる。

この変換は通常、$\llbracket M\rrbracket'_s \,\overline{@}\, K \,\overline{@}\, G$ を $M : K : G$ 、$\llbracket V\rrbracket'_v$ を $V^\dagger$ と表現して、図 10 のように書き、コロン変換と呼ぶ。

$$
\begin{array}{rcl}
\llbracket v \rrbracket & = & \overline{\lambda}k.\,\overline{\lambda}g.\,k \;\overline{@}\; \llbracket v \rrbracket_v \;\overline{@}\; g \\
\llbracket s \rrbracket & = & \overline{\lambda}k.\,\overline{\lambda}g.\,\llbracket s \rrbracket_s \;\overline{@}\; k \;\overline{@}\; g \\[1ex]
\llbracket x \rrbracket_v & = & x \\
\llbracket \underline{\lambda}x.\,M \rrbracket_v & = & \underline{\lambda}x.\,\underline{\lambda}k.\,\underline{\lambda}g.\,\llbracket M \rrbracket' \;\overline{@}\; k \;\overline{@}\; g \\
\llbracket M_1 \;\underline{@}\; M_2 \rrbracket_s & = & \overline{\lambda}k.\,\overline{\lambda}g.\,\llbracket M_1 \rrbracket \;\overline{@}\; (\overline{\lambda}v_1.\,\overline{\lambda}g_1.\,\llbracket M_2 \rrbracket \;\overline{@}\; (\overline{\lambda}v_2.\,\overline{\lambda}g_2.\,v_1 \;\underline{@}\; v_2 \;\underline{@}\; (\underline{\lambda}a.\,\underline{\lambda}g'.\,k \;\overline{@}\; a \;\overline{@}\; g') \;\underline{@}\; g_2) \;\overline{@}\; g_1) \;\overline{@}\; g \\
\llbracket \underline{\mathsf{let}}\ x = M\ \underline{\mathsf{in}}\ N \rrbracket_s & = & \overline{\lambda}k.\,\overline{\lambda}g.\,\llbracket M \rrbracket' \;\overline{@}\; (\underline{\lambda}x.\,\underline{\lambda}g'.\,\llbracket N \rrbracket' \;\overline{@}\; (\underline{\lambda}a.\,\underline{\lambda}g''.\,k \;\overline{@}\; a \;\overline{@}\; g'') \;\overline{@}\; g') \;\overline{@}\; g \\
\llbracket \underline{\mathcal{S}} \rrbracket_v & = & \underline{\lambda}w.\,\underline{\lambda}k'.\,\underline{\lambda}g'.\,w \;\underline{@}\; (\underline{\lambda}y.\,\underline{\lambda}k''.\,\underline{\lambda}g''.\,k' \;\underline{@}\; y \;\underline{@}\; (k'' \;\underline{::}\; g'')) \;\underline{@}\; k_{id} \;\underline{@}\; g' \\
\llbracket \underline{\mathcal{S}_0} \rrbracket_v & = & \underline{\lambda}w.\,\underline{\lambda}k'.\,\underline{\lambda}(k_0 \;\underline{::}\; g').\,w \;\underline{@}\; (\underline{\lambda}y.\,\underline{\lambda}k''.\,\underline{\lambda}g''.\,k' \;\underline{@}\; y \;\underline{@}\; (k'' \;\underline{::}\; g'')) \;\underline{@}\; k_0 \;\underline{@}\; g' \\
\llbracket \underline{\langle} M \underline{\rangle} \rrbracket_s & = & \overline{\lambda}k.\,\overline{\lambda}g.\,\llbracket M \rrbracket' \;\overline{@}\; k_{id} \;\overline{@}\; ((\underline{\lambda}a.\,\underline{\lambda}g'.\,k \;\overline{@}\; a \;\overline{@}\; g') \;\underline{::}\; g) \\[1ex]
\llbracket v \rrbracket' & = & \overline{\lambda}k.\,\overline{\lambda}g.\,k \;\underline{@}\; \llbracket v \rrbracket'_v \;\underline{@}\; g \\
\llbracket s \rrbracket' & = & \overline{\lambda}k.\,\overline{\lambda}g.\,\llbracket s \rrbracket'_s \;\overline{@}\; k \;\overline{@}\; g \\[1ex]
\llbracket x \rrbracket'_v & = & x \\
\llbracket \underline{\lambda}x.\,M \rrbracket'_v & = & \underline{\lambda}x.\,\underline{\lambda}k.\,\underline{\lambda}g.\,\llbracket M \rrbracket' \;\overline{@}\; k \;\overline{@}\; g \\
\llbracket M_1 \;\underline{@}\; M_2 \rrbracket'_s & = & \overline{\lambda}k.\,\overline{\lambda}g.\,\llbracket M_1 \rrbracket \;\overline{@}\; (\overline{\lambda}v_1.\,\overline{\lambda}g_1.\,\llbracket M_2 \rrbracket \;\overline{@}\; (\overline{\lambda}v_2.\,\overline{\lambda}g_2.\,v_1 \;\underline{@}\; v_2 \;\underline{@}\; k \;\underline{@}\; g_2) \;\overline{@}\; g_1) \;\overline{@}\; g \\
\llbracket \underline{\mathsf{let}}\ x = M\ \underline{\mathsf{in}}\ N \rrbracket'_s & = & \overline{\lambda}k.\,\overline{\lambda}g.\,\llbracket M \rrbracket' \;\overline{@}\; (\underline{\lambda}x.\,\underline{\lambda}g'.\,\llbracket N \rrbracket' \;\overline{@}\; k \;\overline{@}\; g') \;\overline{@}\; g \\
\llbracket \underline{\mathcal{S}} \rrbracket'_v & = & \underline{\lambda}w.\,\underline{\lambda}k'.\,\underline{\lambda}g'.\,w \;\underline{@}\; (\underline{\lambda}y.\,\underline{\lambda}k''.\,\underline{\lambda}g''.\,k' \;\underline{@}\; y \;\underline{@}\; (k'' \;\underline{::}\; g'')) \;\underline{@}\; k_{id} \;\underline{@}\; g' \\
\llbracket \underline{\mathcal{S}_0} \rrbracket'_v & = & \underline{\lambda}w.\,\underline{\lambda}k'.\,\underline{\lambda}(k_0 \;\underline{::}\; g').\,w \;\underline{@}\; (\underline{\lambda}y.\,\underline{\lambda}k''.\,\underline{\lambda}g''.\,k' \;\underline{@}\; y \;\underline{@}\; (k'' \;\underline{::}\; g'')) \;\underline{@}\; k_0 \;\underline{@}\; g' \\
\llbracket \underline{\langle} M \underline{\rangle} \rrbracket'_s & = & \overline{\lambda}k.\,\overline{\lambda}g.\,\llbracket M \rrbracket' \;\overline{@}\; k_{id} \;\overline{@}\; (k \;\underline{::}\; g)
\end{array}
$$

図 7. 値と値以外に対する変換に分けた CPS 変換

$$
\begin{array}{rcl}
\llbracket V \;\underline{@}\; W \rrbracket'_s \;\overline{@}\; k \;\overline{@}\; g & = & \llbracket V \rrbracket \;\overline{@}\; (\overline{\lambda}v_1.\,\overline{\lambda}g_1.\,\llbracket W \rrbracket \;\overline{@}\; (\overline{\lambda}v_2.\,\overline{\lambda}g_2.\,v_1 \;\underline{@}\; v_2 \;\underline{@}\; k \;\underline{@}\; g_2) \;\overline{@}\; g_1) \;\overline{@}\; g \\
 & = & (\overline{\lambda}k.\,\overline{\lambda}g.\,k \;\overline{@}\; \llbracket V \rrbracket_v \;\overline{@}\; g) \;\overline{@}\; (\overline{\lambda}v_1.\,\overline{\lambda}g_1.\,\llbracket W \rrbracket \;\overline{@}\; (\overline{\lambda}v_2.\,\overline{\lambda}g_2.\,v_1 \;\underline{@}\; v_2 \;\underline{@}\; k \;\underline{@}\; g_2) \;\overline{@}\; g_1) \;\overline{@}\; g \\
 & \longrightarrow & (\overline{\lambda}v_1.\,\overline{\lambda}g_1.\,\llbracket W \rrbracket \;\overline{@}\; (\overline{\lambda}v_2.\,\overline{\lambda}g_2.\,v_1 \;\underline{@}\; v_2 \;\underline{@}\; k \;\underline{@}\; g_2) \;\overline{@}\; g_1) \;\overline{@}\; \llbracket V \rrbracket_v \;\overline{@}\; g \\
 & \longrightarrow & \llbracket W \rrbracket \;\overline{@}\; (\overline{\lambda}v_2.\,\overline{\lambda}g_2.\,\llbracket V \rrbracket_v \;\underline{@}\; v_2 \;\underline{@}\; k \;\underline{@}\; g_2) \;\overline{@}\; g \\
 & = & (\overline{\lambda}k.\,\overline{\lambda}g.\,k \;\overline{@}\; \llbracket W \rrbracket_v \;\overline{@}\; g) \;\overline{@}\; (\overline{\lambda}v_2.\,\overline{\lambda}g_2.\,\llbracket V \rrbracket_v \;\underline{@}\; v_2 \;\underline{@}\; k \;\underline{@}\; g_2) \;\overline{@}\; g \\
 & \longrightarrow & (\overline{\lambda}v_2.\,\overline{\lambda}g_2.\,\llbracket V \rrbracket_v \;\underline{@}\; v_2 \;\underline{@}\; k \;\underline{@}\; g_2) \;\overline{@}\; \llbracket W \rrbracket_v \;\overline{@}\; g \\
 & \longrightarrow & \llbracket V \rrbracket_v \;\underline{@}\; \llbracket W \rrbracket_v \;\underline{@}\; k \;\underline{@}\; g \\
 & = & \llbracket V \rrbracket'_v \;\underline{@}\; \llbracket W \rrbracket'_v \;\underline{@}\; k \;\underline{@}\; g \\
\llbracket V \;\underline{@}\; Q \rrbracket'_s \;\overline{@}\; k \;\overline{@}\; g & = & \llbracket Q \rrbracket_s \;\overline{@}\; (\overline{\lambda}v_2.\,\overline{\lambda}g_2.\,\llbracket V \rrbracket'_v \;\underline{@}\; v_2 \;\underline{@}\; k \;\underline{@}\; g_2) \;\overline{@}\; g \\
\llbracket P \;\underline{@}\; W \rrbracket'_s \;\overline{@}\; k \;\overline{@}\; g & = & \llbracket P \rrbracket_s \;\overline{@}\; (\overline{\lambda}v_1.\,\overline{\lambda}g_1.\,v_1 \;\underline{@}\; \llbracket W \rrbracket'_v \;\underline{@}\; k \;\underline{@}\; g_1) \;\overline{@}\; g \\
\llbracket P \;\underline{@}\; Q \rrbracket'_s \;\overline{@}\; k \;\overline{@}\; g & = & \llbracket P \rrbracket_s \;\overline{@}\; (\overline{\lambda}v_1.\,\overline{\lambda}g_1.\,\llbracket Q \rrbracket_s \;\overline{@}\; (\overline{\lambda}v_2.\,\overline{\lambda}g_2.\,v_1 \;\underline{@}\; v_2 \;\underline{@}\; k \;\underline{@}\; g_2) \;\overline{@}\; g_1) \;\overline{@}\; g
\end{array}
$$

図 8. 関数呼び出しを場合分け

$$
\begin{array}{rcl}
\llbracket v \rrbracket' & = & \overline{\lambda}k.\,\overline{\lambda}g.\,k \;\underline{@}\; \llbracket v \rrbracket'_v \;\underline{@}\; g \\
\llbracket s \rrbracket' & = & \overline{\lambda}k.\,\overline{\lambda}g.\,\llbracket s \rrbracket'_s \;\overline{@}\; k \;\overline{@}\; g \\[1ex]
\llbracket x \rrbracket'_v & = & x \\
\llbracket \underline{\lambda}x.\,M \rrbracket'_v & = & \underline{\lambda}x.\,\underline{\lambda}k.\,\underline{\lambda}g.\,\llbracket M \rrbracket' \;\overline{@}\; k \;\overline{@}\; g \\
\llbracket \underline{\mathcal{S}} \rrbracket'_v & = & \underline{\lambda}w.\,\underline{\lambda}k'.\,\underline{\lambda}g'.\,w \;\underline{@}\; (\underline{\lambda}y.\,\underline{\lambda}k''.\,\underline{\lambda}g''.\,k' \;\underline{@}\; y \;\underline{@}\; (k'' \;\underline{::}\; g'')) \;\underline{@}\; k_{id} \;\underline{@}\; g' \\
\llbracket \underline{\mathcal{S}_0} \rrbracket'_v & = & \underline{\lambda}w.\,\underline{\lambda}k'.\,\underline{\lambda}(k_0 \;\underline{::}\; g').\,w \;\underline{@}\; (\underline{\lambda}y.\,\underline{\lambda}k''.\,\underline{\lambda}g''.\,k' \;\underline{@}\; y \;\underline{@}\; (k'' \;\underline{::}\; g'')) \;\underline{@}\; k_0 \;\underline{@}\; g' \\[1ex]
\llbracket V \;\underline{@}\; W \rrbracket'_s & = & \overline{\lambda}k.\,\overline{\lambda}g.\,\llbracket V \rrbracket'_v \;\underline{@}\; \llbracket W \rrbracket'_v \;\underline{@}\; k \;\underline{@}\; g \\
\llbracket V \;\underline{@}\; Q \rrbracket'_s & = & \overline{\lambda}k.\,\overline{\lambda}g.\,\llbracket Q \rrbracket'_s \;\overline{@}\; (\underline{\lambda}v_2.\,\underline{\lambda}g_2.\,\llbracket V \rrbracket'_v \;\underline{@}\; v_2 \;\underline{@}\; k \;\underline{@}\; g_2) \;\overline{@}\; g \\
\llbracket P \;\underline{@}\; W \rrbracket'_s & = & \overline{\lambda}k.\,\overline{\lambda}g.\,\llbracket P \rrbracket'_s \;\overline{@}\; (\underline{\lambda}v_1.\,\underline{\lambda}g_1.\,v_1 \;\underline{@}\; \llbracket W \rrbracket'_v \;\underline{@}\; k \;\underline{@}\; g_1) \;\overline{@}\; g \\
\llbracket P \;\underline{@}\; Q \rrbracket'_s & = & \overline{\lambda}k.\,\overline{\lambda}g.\,\llbracket P \rrbracket'_s \;\overline{@}\; (\underline{\lambda}v_1.\,\underline{\lambda}g_1.\,\llbracket Q \rrbracket'_s \;\overline{@}\; (\underline{\lambda}v_2.\,\underline{\lambda}g_2.\,v_1 \;\underline{@}\; v_2 \;\underline{@}\; k \;\underline{@}\; g_2) \;\overline{@}\; g_1) \;\overline{@}\; g \\
\llbracket \underline{\mathsf{let}}\ x = M\ \underline{\mathsf{in}}\ N \rrbracket'_s & = & \overline{\lambda}k.\,\overline{\lambda}g.\,\llbracket M \rrbracket' \;\overline{@}\; (\underline{\lambda}v.\,\underline{\lambda}g'.\,\llbracket N \rrbracket' \;\overline{@}\; k \;\overline{@}\; g') \;\overline{@}\; g \\
\llbracket \underline{\langle} M \underline{\rangle} \rrbracket'_s & = & \overline{\lambda}k.\,\overline{\lambda}g.\,\llbracket M \rrbracket' \;\overline{@}\; k_{id} \;\overline{@}\; (k \;\underline{::}\; g)
\end{array}
$$

図 9. 得られた CPS 変換

$$
\begin{array}{rcl}
\mathbb{N}^* & = & \mathbb{N} \\
\mathbb{B}^* & = & \mathbb{B} \\
(\tau_1 \rightarrow \tau_2 \, \langle \sigma_\alpha \rangle \, \alpha \, \langle \sigma_\beta \rangle \, \beta)^* & = & \tau_1^* \rightarrow (\tau_2^* \rightarrow \sigma_\alpha^* \rightarrow \alpha^*) \rightarrow \sigma_\beta^* \rightarrow \beta^* \\
(\bullet_\sigma)^* & = & [\,]_\sigma \\
((\tau_1 \triangleright \langle \sigma_1 \rangle \, \tau_2) :: \sigma_2)^* & = & (\tau_1^* \rightarrow \sigma_1^* \rightarrow \tau_2^*) :: \sigma_2^*
\end{array}
$$

$$
M^* \;\; = \;\; \underline{\lambda} k.\, \underline{\lambda} g.\, (M : k : g) \quad (k, g \notin fv(M))
$$

$$
\begin{array}{rcl}
x^\dagger & = & x \\
(\underline{\lambda} x.\, M)^\dagger & = & \underline{\lambda} x.\, M^* \\
\underline{\mathcal{S}}^\dagger & = & \underline{\lambda} w.\, \underline{\lambda} j.\, \underline{\lambda} g.\, w \,\underline{@}\, (\underline{\lambda} y.\, \underline{\lambda} k.\, \underline{\lambda} g.\, j \,\underline{@}\, y \,\underline{@}\, (k \,\underline{::}\, g)) \,\underline{@}\, k_{id} \,\underline{@}\, g \\
\underline{\mathcal{S}_0}^\dagger & = & \underline{\lambda} w.\, \underline{\lambda} j.\, \underline{\lambda} (k_0 \,\underline{::}\, g).\, w \,\underline{@}\, (\underline{\lambda} y.\, \underline{\lambda} k.\, \underline{\lambda} g.\, j \,\underline{@}\, y \,\underline{@}\, (k \,\underline{::}\, g)) \,\underline{@}\, k_0 \,\underline{@}\, g
\end{array}
$$

$$
\begin{array}{rcl}
V : K : G & = & K \,\underline{@}\, V^\dagger \,\underline{@}\, G \\
(V \,\underline{@}\, W) : K : G & = & V^\dagger \,\underline{@}\, W^\dagger \,\underline{@}\, K \,\underline{@}\, G \\
(V \,\underline{@}\, Q) : K : G & = & Q : (\underline{\lambda} y.\, \underline{\lambda} g.\, V^\dagger \,\underline{@}\, y \,\underline{@}\, K \,\underline{@}\, g) : G \\
(P \,\underline{@}\, W) : K : G & = & P : (\underline{\lambda} x.\, \underline{\lambda} g.\, x \,\underline{@}\, W^\dagger \,\underline{@}\, K \,\underline{@}\, g) : G \\
(P \,\underline{@}\, Q) : K : G & = & P : (\underline{\lambda} x.\, \underline{\lambda} g.\, Q : (\underline{\lambda} y.\, \underline{\lambda} g.\, x \,\underline{@}\, y \,\underline{@}\, K \,\underline{@}\, g) : g) : G \\
(\underline{\mathsf{let}}\; x = M \;\underline{\mathsf{in}}\; N) : K : G & = & M : (\underline{\lambda} x.\, \underline{\lambda} g.\, (N : K : g)) : G \\
\underline{\langle} M \underline{\rangle} : K : G & = & M : k_{id} : (K \,\underline{::}\, G)
\end{array}
$$

図 10. CPS 変換（コロン変換）

5 CPS 変換の正当性の証明

ここでは、CPS 変換の正当性より強い性質である簡約の保存を示す。任意の DS 項とそれを簡約した結果をそれぞれ CPS 変換すると、変換後の 2 つの項も簡約関係にあること

任意の$\lambda_{c\mathcal{S}_0}$の項 M, N に対して、$M \twoheadrightarrow_{\lambda_{c\mathcal{S}_0}} N$ ならば $M^* \twoheadrightarrow_{\lambda^*_{c\mathcal{S}_0}} N^*$

を証明する。証明はすべて Agda で定式化されており、全体の流れは shift/reset に対する証明に沿う形である。まず、代入補題を 3 つ示す。

補題 1 (値の代入補題, `Reflect3.agda` の `lemma-substV, lemma-subst`).

(1) 任意の $\lambda_{c\mathcal{S}_0}$ の値 V, W について $(W[V/x])^\dagger = W^\dagger[V^\dagger/x]$

(2) 任意の $\lambda_{c\mathcal{S}_0}$ の項 M と値 V, $\lambda^*_{c\mathcal{S}_0}$ の継続 K, メタ継続 G について
$M[V/x] : K_\Delta[V^\dagger/x] : G_\Theta[V^\dagger/x] = (M : K_\Delta : G_\Theta)[V^\dagger/x]$

証明. W, M の相互帰納法。 □

この補題は、任意の DS 項に対して、値を代入してから CPS 変換した項と、CPS 変換してから値を代入した項が等しいことを示している。

補題 2 (継続の代入補題, `Reflect3.agda` の `lemma-csubst`). 任意の $\lambda_{c\mathcal{S}_0}$ の項 M と値 V, $\lambda^*_{c\mathcal{S}_0}$ の継続 K_Δ, J_Δ, メタ継続 G_Θ について、以下が成り立つ。

- $\Delta = k, \Theta = g$ ならば $M : (K_k[J_\Delta/k]) : G_g = (M : K_k : G_g)[J_\Delta/k]$
- $\Delta = \bullet, \Theta = k$ ならば $M : K_\bullet : (G_k[J_\Delta/k]) = (M : K_\bullet : G_k)[J_\Delta/k]$

証明. M に関する帰納法。 □

継続の代入では、Δ と Θ によって場合分けが必要である。これは、メタ継続 G_Θ にも継続 k が現れる可能性があるためであり、継続 k を代入するためには、継続 K_Δ とメタ継続 G_Θ のどちらかに k が現れなければならない。そのため、有効な Δ と Θ の組み合わせのうち Δ++$\Theta = k$ となる、$\Delta = k, \Theta = g$ と $\Delta = \bullet, \Theta = k$ で場合分けをしている。$\Delta = \bullet, \Theta = g$ の場合は、継続 k がどちらにも現れないため、代入ができず意味を成さない。補題の1つ目は、継続 k が継続 K_k にある場合で、代入後の継続でCPS変換した項と、CPS変換してから継続を代入した項が等しいことを示しており、2つ目は、継続 k がメタ継続 G_k にある場合で、代入後のメタ継続でCPS変換した項と、CPS変換してから継続を代入した項が等しいことを示している。

補題 3 (メタ継続の代入補題, `Reflect3.agda` の `lemma-msubst`)**.** 任意の $\lambda_{c\mathcal{S}_0}$ の項 M と値 V, $\lambda^*_{c\mathcal{S}_0}$ の継続 K_Δ, メタ継続 $G_\Theta, G'_{\Theta'}$ について $M : K_\Delta : (G_\Theta[G'_{\Theta'}/g]) = (M : K_\Delta : G_\Theta)[G'_{\Theta'}/g]$

証明. M に関する帰納法。 □

この補題は、代入後のメタ継続でCPS変換した項と、CPS変換してからメタ継続を代入した項が等しいことを示している。

以下で、その他の補題を示す。

補題 4 (`Reflect3.agda` の `correctCM`)**.** 任意の $\lambda_{c\mathcal{S}_0}$ の項 M について、以下が成り立つ。

- 任意の $\lambda^*_{c\mathcal{S}_0}$ の継続 K_Δ, K'_Δ とメタ継続 G_Θ について $K_\Delta \twoheadrightarrow_{\lambda^*_{c\mathcal{S}_0}} K'_\Delta$ ならば $M : K_\Delta : G_\Theta \twoheadrightarrow_{\lambda^*_{c\mathcal{S}_0}} M : K'_\Delta : G_\Theta$
- 任意の $\lambda^*_{c\mathcal{S}_0}$ の継続 K_Δ とメタ継続 G_Θ, G'_Θ について $G_\Theta \twoheadrightarrow_{\lambda^*_{c\mathcal{S}_0}} G'_\Theta$ ならば $M : K_\Delta : G_\Theta \twoheadrightarrow_{\lambda^*_{c\mathcal{S}_0}} M : K_\Delta : G'_\Theta$

証明. M に関する帰納法。 □

この補題は、補題2と同様に場合分けがある。1つ目は、任意のDS項を、任意のCPSの継続とそれを簡約した継続を用いてそれぞれCPS変換すると、変換後の2つの項も簡約関係にあることを表しており、2つ目は、任意のDS項を、任意のCPSのメタ継続とそれを簡約したメタ継続を用いてそれぞれCPS変換すると、変換後の2つの項も簡約関係にあることを表している。これらは独立して示せるように見えるが、1つ目を M に関する帰納法で証明すると、$\underline{\langle} M \underline{\rangle}$ の場合で帰納法の仮定が成り立たない。これは、$\underline{\langle} M \underline{\rangle}$ に対するコロン変換（図 10）で、継続がメタ継続の先頭に追加されるためである。そこで、2つ目の命題も同じ補題に含めることで証明できる。

補題 5 (`Reflect3.agda` の `contExist`)**.** 任意の $\lambda_{c\mathcal{S}_0}$ のコンテキスト J と $\lambda^*_{c\mathcal{S}_0}$ の継続 K_Δ について、以下の2つを満たすような $\lambda^*_{c\mathcal{S}_0}$ の継続 J'_Δ が存在する。

- 任意の $\lambda_{c\mathcal{S}_0}$ の値でない項 P と $\lambda^*_{c\mathcal{S}_0}$ のメタ継続 G_Θ について、$J[P] : K_\Delta : G_\Theta = P : J'_\Delta : G_\Theta$
- 任意の $\lambda_{c\mathcal{S}_0}$ の値 V と $\lambda^*_{c\mathcal{S}_0}$ のメタ継続 G_Θ について、$V : J'_\Delta : G_\Theta \twoheadrightarrow_{\lambda^*_{c\mathcal{S}_0}} J[V] : K_\Delta : G_\Theta$

証明. J に関する帰納法。 □

shift と shift0 の挙動を捉える補題が、補題5である。この補題は、先行研究のものをメタ継続が入った形に拡張した。$J[P]$ は、P が shift か shift0 で J がその周りのコンテキストである。1つ目は J を K_Δ に押しやることができること、2つ目は P を実行した結果の V でその後の実行を続けた結果と、元のコンテキスト J のもとで実行を続けた結果が同じになることを示している。その際、メタ継続は全く関与しないので、任意のメタ継続を用いて示すことができる。

これらの補題を用いて、定理6を証明することで、CPS変換の簡約の保存を示すことができる。この定理の証明は、$V \twoheadrightarrow_{\lambda_{c\mathcal{S}_0}} W$ と $M \twoheadrightarrow_{\lambda_{c\mathcal{S}_0}} N$ に関する相互帰納法で証明できる。

定理 6 (`Reflect3.agda` の `correctV`, `correctE`).

(1) 任意の $\lambda_{c\mathcal{S}_0}$ の値 V, W に対して、$V \twoheadrightarrow_{\lambda_{c\mathcal{S}_0}} W$ ならば $V^\dagger \twoheadrightarrow_{\lambda^*_{c\mathcal{S}_0}} W^\dagger$

(2) 任意の $\lambda_{c\mathcal{S}_0}$ の項 M, N と $\lambda^*_{c\mathcal{S}_0}$ の継続 K_Δ, メタ継続 G_Θ について
$M \twoheadrightarrow_{\lambda_{c\mathcal{S}_0}} N$ ならば $M : K_\Delta : G_\Theta \twoheadrightarrow_{\lambda^*_{c\mathcal{S}_0}} N : K_\Delta : G_\Theta$

証明. $V \twoheadrightarrow_{\lambda_{c\mathcal{S}_0}} W$ と $M \twoheadrightarrow_{\lambda_{c\mathcal{S}_0}} N$ の相互帰納法。 □

定理 6 の証明で着目すべきは、shift の簡約規則 $(\beta.\mathcal{S})$ と shift0 の簡約規則 $(\beta.\mathcal{S}_0)$ の場合である。これらの規則では、コンテキスト J が関与してくるため、補題 5 を使う。

6 関連研究

λ 計算に対する reflection の証明は Sabry と Wadler [17] が行っており、これを拡張して shift/reset 入りの体系で reflection を示したのが Biernacki ら [2] である。本田ら [19] は、Biernacki らの証明を型のある体系で行い、さらに Agda での定式化も行なっている。Biernacki らは shift0/dollar に対する reflection も証明している [3]。これを参考に、叢らは代数的エフェクトとハンドラの deep handler に対する reflection 証明 [5] に取り組んだが、完全な証明には至っていない。叢らはその原因として、採用した Hillerström ら [12] の CPS 変換では、純粋な評価文脈とハンドラの return 節の区別が失われてしまう点を挙げている。本稿は、本田らの手法を用いた shift0/reset0 を含む型付き言語の reflection の証明の一環である。

4 節で導出したコロン変換は、Plotkin によって提案された変換であり [16], Danvy と Filinski らは、この変換が評価文脈の操作から体系的に導出できることを示している [7, 8]。本稿での貢献は、この導出手法が限定継続演算子 shift0 を含む体系、すなわちメタ継続を必要とする複雑な制御構造に対しても有効であることを示した点にある。Materzok と Biernacki [14] は、shift0 を含む体系に対してメタ継続を明示的に扱う CPS 変換を定義し、それに基づいてサブタイピング付きの型システムを構築した。彼らの CPS 変換は意味論的に正しいものであるが、その定義は Shan ら [18] が提案した CPS 変換を拡張して与えられたものであり、導出過程については深く議論されていない。これに対し本稿では、Danvy らの手法 [8] に基づき、インタプリタからコロン変換を系統的に導出できることを示した。これにより、メタ継続を伴う複雑な変換規則が、評価文脈の構造から自然に導かれることが明らかになった。

この貢献は、代数的エフェクトとハンドラの deep handler に対する reflection の証明にも応用できると考えている。Asai と Fujii は、メタ継続を用いた代数的エフェクトとハンドラのインタプリタを提案している [1] 。これに対して、本稿で示した手法を用いてコロン変換を導出すれば、Hillerström らの CPS 変換 [12] の問題点を克服し、reflection の証明に適した CPS 変換が導出できると考えられる。これにより、未解決になっている deep handler に対する reflection の証明に取り組むことが可能になると期待される。

7 まとめと今後の展望

本稿では、shift0/reset0 を含む型付き言語に対する reflection の証明に向けて、CPS 変換の正当性よりも強い性質である簡約の保存を示すとともに、定理証明支援系言語 Agda にて証明を定式化した。メタ継続の入った CPS 言語には、先行研究にならって継続の種類を表す Δ を導入するだけでなく、メタ継続の種類を表す Θ も導入し、継続とメタ継続の組み合わせによって項の形が変わることを捉えた。CPS 変換の定義ではインタプリタからコロン変換を導出する手法を用い、メタ継続

が入った複雑な制御構造に対しても有効であることを示した。簡約の保存の証明では、先行研究と同じ流れで証明を進めたが、メタ継続の追加に伴いいくつかの補題で注目すべき拡張があった。

今後は、shift0/reset0 に対する reflection の証明を完成させることを目指す。本稿では CPS 変換の簡約の保存を示したが、reflection の証明にはさらに 3 つの性質の証明が必要である。その際、CPS 変換後の項から CPS 変換前の項を復元する逆変換が必要になる。まずはその変換を定義し、reflection の証明を進めていきたい。さらに、shift0/reset0 を含む体系に対する reflection の証明を基に、代数的エフェクトとハンドラを含む体系に対する reflection の証明にも取り組みたい。

参考文献

[1] Kenichi Asai and Maika Fujii. Defining algebraic effects and handlers via trails and metacontinuations. In *Proceedings of the Workshop Dedicated to Olivier Danvy on the Occasion of His 64th Birthday*, pp. 26–39, 2025.

[2] Dariusz Biernacki, Mateusz Pyzik, and Filip Sieczkowski. A Reflection on Continuation-Composing Style. In *5th International Conference on Formal Structures for Computation and Deduction (FSCD 2020)*, pp. 18:1–18:17. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020.

[3] Dariusz Biernacki, Mateusz Pyzik, and Filip Sieczkowski. Reflecting stacked continuations in a fine-grained direct-style reduction theory. In *Proceedings of the 23rd International Symposium on Principles and Practice of Declarative Programming*, pp. 1–13, 2021.

[4] Adam Chlipala. Parametric higher-order abstract syntax for mechanized semantics. In *Proceedings of the 13th ACM SIGPLAN International Conference on Functional programming*, pp. 143–156, 2008.

[5] Youyou Cong and Kenichi Asai. Towards a reflection for effect handlers. In *Proceedings of the 2023 ACM SIGPLAN International Workshop on Partial Evaluation and Program Manipulation*, pp. 55–65, 2023.

[6] Olivier Danvy and Andrzej Filinski. Abstracting control. In *Proceedings of the 1990 ACM Conference on LISP and Functional Programming*, pp. 151–160, 1990.

[7] Olivier Danvy and Andrzej Filinski. Representing control: A study of the cps transformation. *Mathematical Structures in Computer Science*, Vol. 2, No. 4, pp. 361–391, 1992.

[8] Olivier Danvy and Lasse R Nielsen. A first-order one-pass cps transformation. *Theoretical computer science*, Vol. 308, No. 1, pp. 239–257, 2003.

[9] Mattias Felleisen. The theory and practice of first-class prompts. In *Proceedings of the 15th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pp. 180–190, 1988.

[10] Yannick Forster, Ohad Kammar, Sam Lindley, and Matija Pretnar. On the expressive power of user-defined effects: Effect handlers, monadic reflection, delimited control. *Proceedings of the ACM on Programming Languages*, Vol. 1, No. ICFP, pp. 1–29, 2017.

[11] Carl A Gunter, Didier Rémy, and Jon G Riecke. A generalization of exceptions and control in ML-like languages. In *Proceedings of the seventh international conference on functional programming languages and computer architecture*, pp. 12–23, 1995.

[12] Daniel Hillerström, Sam Lindley, and Robert Atkey. Effect handlers via generalised continuations. *Journal of Functional Programming*, Vol. 30, No. e5, pp. 1–69, 2020.

[13] 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.

[14] Marek Materzok and Dariusz Biernacki. Subtyping delimited continuations. In *Proceedings of the 16th ACM SIGPLAN International Conference on Functional programming*, pp. 81–93, 2011.

[15] 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)*, pp. 30:1–30:16. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2019.

[16] Gordon D. Plotkin. Call-by-name, call-by-value and the λ-calculus. *Theoretical computer science*, Vol. 1, No. 2, pp. 125–159, 1975.

[17] Amr Sabry and Philip Wadler. A reflection on call-by-value. *ACM transactions on programming languages and systems (TOPLAS)*, Vol. 19, No. 6, pp. 916–941, 1997.

[18] Chung-chieh Shan. A static simulation of dynamic delimited control. *Higher-order and symbolic computation*, Vol. 20, No. 4, pp. 371–401, 2007.

[19] 本田華歩, 山本充子, 浅井健一. shift/reset を含む型付き言語における reflection 証明. 第 25 回プログラミングおよびプログラミング言語ワークショップ, pp. 1–20, 2023.