shift/reset を含む型付き言語における Reflection 証明

本田 華歩 [1], 山本充子 [2], 浅井 健一 [1]

[1] お茶の水女子大学

pon@pllab.is.ocha.ac.jp, asai@is.ocha.ac.jp

[2] NTT 社会情報研究所

juko.yamamoto.vp@hco.ntt.co.jp

概 要 直接形式のプログラムの中で継続を扱うには shift/reset などの限定継続演算子が使われる。限定継続演算子を使ったプログラムは、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る。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すが、両者のより密接な関係を示す手法として、先行研究で限定継続演算子 shift/reset が入った体系に対して reflection という性質が成り立つことを示している。しかし、この先行研究では対象言語に型がついていない。本研究では、型のついた言語に対する reflection を示すとともに、定理証明支援系言語 Agda においてその証明を定式化した。

1 はじめに

shift/reset [8] を始めとする限定継続演算子を使ったプログラムは、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る。この CPS 変換 [12] や類似する A-正規形変換 [9] などは、しばしばコンパイラの中でも使われ、その正当性を示すのは重要である。

CPS 変換の正当性は、変換前のプログラムの簡約と変換後のプログラムの簡約が対応することにより示される。これにより最低限、コンパイル後のコードがソースプログラムの実行に忠実であることが示される。さらに reflection [14] と呼ばれる関係が示されると、ソース言語とターゲット言語の間の 1 対 1 の関係が明らかになることに加え、CPS 変換後に行える最適化が CPS 変換前にも行えるようになることも意味している。すると、コンパイラ内部で CPS 変換することなく、各種の最適化が可能になる。そのため、近年、限定継続演算子を含む体系に対する reflection が示されるようになってきている。特に shift/reset に対するもの [3]、shift0/dollar に対するもの [4] などが示され、さらに代数的効果とハンドラに対するもの [6] も検討されている。

しかし、これらの仕事はいずれも型のない体系に対するものである。型はプログラムの安全性を担保する上でも重要だが、型の入った体系でも reflection の関係が成り立つかどうかはこれまで示されてこなかった。本稿では、shift/reset を含む単純型付きの λ 計算を対象として、reflection の関係が成り立つことを示す。さらに、それを定理証明支援系言語 Agda を使って定式化したので、その模様を報告する。

1.1 Reflection とは

Reflection [14] というのは、ソース言語とターゲット言語とその間の変換についての性質である。本稿では、ソース言語は shift/reset（と let 文）の入った単純型付き λ 計算、ターゲット言語は（shift/reset は入っていない）単純型付き λ 計算とする。後者は継続渡し形式で書かれたプログラムになるので CPS 言語、それに対して前者は直接形式で書かれているので DS 言語と呼ぶことにする。DS 言語の項 M を CPS 言語に変換するのは CPS 変換と呼ばれ M^* と記述する。一方、CPS 言語の項 N を DS 言語に逆変換するのは DS 変換と呼ばれ $N^\#$ と記述する。このような設定のもとで「 DS 言語と CPS 言語の間に reflection の関係が成り立つ」とは以下のように定義される。

定義 1 (Reflection) CPS 変換 $\cdot^*$ と DS 変換 $\cdot^\#$ が以下の 4 条件を満たすとき reflection をなすという。

1. 任意の DS 項 M について $M \longrightarrow^* M^{*\#}$ （この条件が $M = M^{*\#}$ と強まったものは Order Isomorphism と呼ばれる）
2. 任意の CPS 項 N について $N^{\#*} = N$ （この条件が $N^{\#*} \longrightarrow^* N$ と弱まったものは Galois Connection と呼ばれる）
3. 任意の DS 項 M, M' について $M \longrightarrow M'$ ならば $M^* \longrightarrow^* M'^*$
4. 任意の CPS 項 N, N' について $N \longrightarrow N'$ ならば $N^\# \longrightarrow^* N'^\#$

ここで $\longrightarrow$ は 1 ステップの簡約、$\longrightarrow^*$ は 0 ステップ以上の簡約を示す。

上記の性質のうち 3 が従来の CPS 変換の簡約の保存を示しており、4 はその逆である。さらに 1, 2 の条件が加わっている。1 の条件は CPS 変換した後 DS 変換すると、その結果は元の項を簡約したものになっていることを意味している。これの意味するところを理解するために、具体例を見てみよう。f, g, h, a という変数を使った DS 言語の項 $f \mathbin{\underline{@}} ((g \mathbin{\underline{@}} h) \mathbin{\underline{@}} a)$ を考えてみる。これを CPS 変換すると以下のような項が得られる。

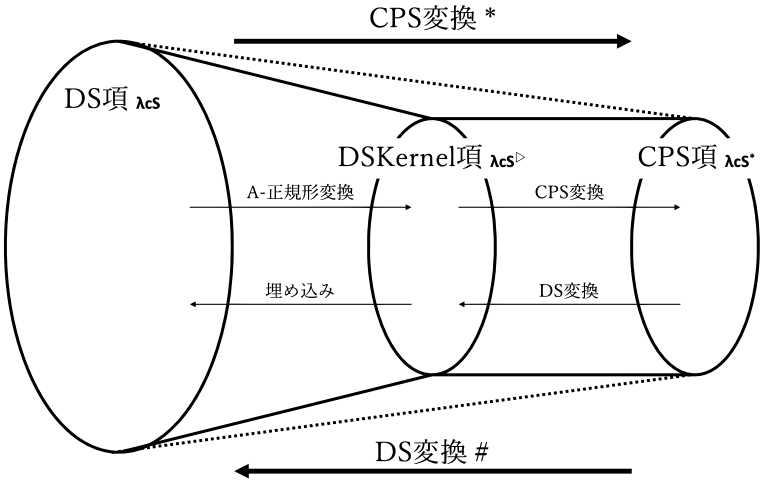
$$\underline{\lambda} k.\, g \mathbin{\underline{@}} h \mathbin{\underline{@}} (\underline{\lambda} x.\, x \mathbin{\underline{@}} a \mathbin{\underline{@}} (\underline{\lambda} y.\, f \mathbin{\underline{@}} y \mathbin{\underline{@}} k))$$

CPS 項の特徴は部分式に名前が与えられることである。この例では $g \mathbin{\underline{@}} h$ に x、$x \mathbin{\underline{@}} a$ に y という名前が与えられている。この項を DS 変換して得られる項は、部分式に名前を与えていることを反映して以下のようになる。

$$\underline{\mathsf{let}}\ x = g \mathbin{\underline{@}} h\ \underline{\mathsf{in}}\ \underline{\mathsf{let}}\ y = x \mathbin{\underline{@}} a\ \underline{\mathsf{in}}\ f \mathbin{\underline{@}} y$$

もともと入れ子を使って書かれていた $f \mathbin{\underline{@}} ((g \mathbin{\underline{@}} h) \mathbin{\underline{@}} a)$ という項は CPS 変換して DS 変換すると DS 言語の中で部分式に名前のついた項、つまり A-正規形になる。これが reflection の 1 の性質の意味である。そのように考えると reflection の 2 の性質は、A-正規形の言語（= CPS 言語を DS 変換して得られる言語）と CPS 言語が同型 (order isomorphic) であることを示している。A-正規形の言語は、reflection の世界では DS 言語の中のカーネルと呼ばれるため、本稿では DSKernel 言語と呼ぶことにする。

ここまでをまとめたのが以下の図である。本稿では Biernacki ら [3] に合わせて DS 言語を $\lambda_{c\mathcal{S}}$、DSKernel 言語を $\lambda^{\triangleright}_{c\mathcal{S}}$、CPS 言語を $\lambda^{*}_{c\mathcal{S}}$ と記述している。DSKernel 言語と CPS 言語の間の変換



は、同じ項を DS で書いているか CPS で書いているかの違いだけで自明な変換になる。つまり、従来の CPS 変換は、 A-正規形変換と、DSKernel 言語を CPS に変換する自明な変換との合成と表現される。逆に、従来の DS 変換は CPS 言語を DSKernel 言語に自明に変換した後、DS 言語に埋め込む形になる。

1.2 Agda による型付きの定式化

本研究では、DS 言語と DSKernel 言語の間の reflection と DSKernel 言語と CPS 言語の間の同型性を、定理証明支援系言語 Agda を使って定式化し、それらを合わせて DS 言語と CPS 言語の間の reflection の証明を定式化している。この手の定式化ではオブジェクト言語の束縛をどのように表現するかが難しいが、ここでは Parameterized Higher-Order Abstract Syntax (PHOAS) [5] を使用した。これは、オブジェクト言語の束縛をメタ言語（我々の場合は Agda）の束縛で表現するもので、多くの場合、変数名の付け替えのことを意識することなく定式化ができる。一方で、高階の手法を使っているため一部の性質を証明するのが難しくなる場合がある。本研究でも Chlipala [5] 同様、代入に関する自明な命題を公理として仮定する必要があった。加えて、関数の等価性を言うために関数の外延性の公理も仮定した。

定式化にあたっては、言語の定義に型規則を最初から組み込むことで、型の合う項しか定義できない形 [1] で行った。

本研究で行った定式化は以下の URL から得ることができる。

`http://pllab.is.ocha.ac.jp/~asai/jpapers/ppl/23/`

1.3 本稿の貢献と構成

本稿の貢献は以下のようにまとめられる。

- shift/reset を含む単純型付き λ 計算の reflection が成り立つことを証明した。そこでのポイントは、継続を表す変数 k を特別扱いすること、及び DSKernel 言語にも k に関する型情報を入れることである。
- 証明は、定理証明支援系言語 Agda で定式化し、関数の外延性の公理と代入に関する公理を仮定することで行った。

以下、2 節でそれぞれの言語を定式化し、3 節で 1 対 1 に対応している DSKernel 項と CPS 項間の reflection（同型性） を、4 節で DS 項と DSKernel 項間の reflection を証明する。5 節で、3 節と 4 節を合わせて DS 項と CPS 項間の reflection の証明を完成させる。また、6 節で関連研究、7 節でまとめと今後の課題を述べる。本文中には載せられなかった定式化の詳細は付録につけた。

2 それぞれの言語の定式化

この節では、DS 言語、DSKernel 言語、CPS 言語を順に示す。

2.1 shift/reset 入り DS 言語

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 と reset が加わった言語であり、基本的には Biernacki ら [3] の定義と同じである。項は、値か値以外の項からなる。値以外の項を独立して定義しているのは、$(let.1)$, $(let.2)$ などの簡約規則や後に示す CPS 変換などで、値の場合とそうでない場合で区別したくなるためである。値の中の $\underline{\mathcal{S}}$ は shift を、値以外の項の中の $\underline{\langle \cdot \rangle}$ は reset を表す。Biernacki らにならって shift は定数として定義し、$\underline{\mathcal{S}}\,\underline{@}\,(\underline{\lambda} k.\, M)$ 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。型とコンテキストについては後述する。

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

$$
\begin{array}{llll}
\text{types} & \tau_i & ::= & \mathsf{int} \mid \tau_1 \to \tau_2 @ [\tau_3, \tau_4] \\
\text{cont. types} & \delta & ::= & \tau_2 \to \tau_3 \\
\\
\text{terms} & L, M, N & ::= & V \mid P \\
\text{values} & V, W & ::= & x \mid \underline{\lambda} x.\, M \mid \underline{\mathcal{S}} \\
\text{nonvalues} & P, Q & ::= & M \mathbin{\underline{@}} N \mid \underline{\mathsf{let}}\ x = M\ \underline{\mathsf{in}}\ N \mid \underline{\langle} M \underline{\rangle} \\
\text{pure contexts} & J_\Delta, K_\Delta & ::= & [\,]_\Delta \mid K_\Delta[[\,] \mathbin{\underline{@}} M] \mid K_\Delta[V \mathbin{\underline{@}} [\,]] \mid K_\Delta[\underline{\mathsf{let}}\ x = [\,]\ \underline{\mathsf{in}}\ M]
\end{array}
$$

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

図 1. DS 項の構文

$J_\bullet$ 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 V[1] に渡すことで V の中で継続を使えるようにしている。ここで $J_\bullet$ は穴が reset に囲まれていないような pure なコンテキストを示す（図 1）。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 には囲まれていないことを示す k のどちらかの値を取る。Δ は、型のついた体系で CPS 言語との対応をとるときに重要な役目をはたすが、それについては後述する。

$(let.1)$ と $(let.2)$ は、簡約の方向が逆ではないかと思うかも知れないが、そうではない。これらは、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。$(let.1)$, $(let.2)$ を使って値でない項に名前を割り当て、$(assoc)$ 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。例えば、$f \mathbin{\underline{@}} ((g \mathbin{\underline{@}} h) \mathbin{\underline{@}} a)$ という項の部分式に名前を与えると次のようになる。

$$
\begin{array}{lll}
 & f \mathbin{\underline{@}} ((g \mathbin{\underline{@}} h) \mathbin{\underline{@}} a) & \\
\longrightarrow & \underline{\mathsf{let}}\ x = (g \mathbin{\underline{@}} h) \mathbin{\underline{@}} a\ \underline{\mathsf{in}}\ f \mathbin{\underline{@}} x & (let.2) \\
\longrightarrow & \underline{\mathsf{let}}\ x = \underline{\mathsf{let}}\ y = g \mathbin{\underline{@}} h\ \underline{\mathsf{in}}\ y \mathbin{\underline{@}} a\ \underline{\mathsf{in}}\ f \mathbin{\underline{@}} x & (let.1) \\
\longrightarrow & \underline{\mathsf{let}}\ y = g \mathbin{\underline{@}} h\ \underline{\mathsf{in}}\ \underline{\mathsf{let}}\ x = y \mathbin{\underline{@}} a\ \underline{\mathsf{in}}\ f \mathbin{\underline{@}} x & (assoc)
\end{array}
$$

最終的に得られた項は、すべての部分式について let 文で名前が与えられている。これは後の節で述べる DSKernel 言語の項になっている。

図 2 に DS 項の型規則を示す。本稿では shift/reset を加えた体系を扱うことにより、shift で切り取られる継続が返す型を考慮する必要がある。ここに示す型規則は従来の（単相の）型規則 [7, 2] と同一だが、後に出てくる別の言語の型規則との対応をとるため記法を少し変更している。型判定 $\Gamma \vdash M_k : [\tau_1, \tau_2]\, \tau_3$ は「型環境 Γ の元で式 M_k は τ_1 型を持ち、M_k を $\tau_1 \to \tau_2$ という継続の元で実行すると最終的に τ_3 型の値を返す」と読む。また、$\Gamma \vdash_v V : \tau$ は「型環境 Γ の元で値 V は τ 型を持つ」と読む。関数の型は $\tau_1 \to \tau_2 @ [\tau_3, \tau_4]$ と表し（図 1）、これで「τ_1 型の引数を $\tau_2 \to \tau_3$ 型の継続の元で受け取ると、最終的には τ_4 型の値を返す」関数を表す。

継続の型 δ はコンテキストの型を表す。コンテキストに対する型規則は後述する。また本研究では、型システムに基づいたインタプリタを Agda に載せているので、型健全性は成り立っている。

[1] Biernacki ら [3] はここを値 V ではなく任意の項 M としている。本稿では call by value の言語でより自然と思われる V を採用している。

$$\frac{x:\tau\in\Gamma}{\Gamma\vdash_v x:\tau}\ (\textsc{TVar}) \qquad \frac{\Gamma,x:\tau_1\vdash M:[\tau_2,\tau_3]\,\tau_4}{\Gamma\vdash_v \underline{\lambda}x.\,M:\tau_1\to\tau_2@[\tau_3,\tau_4]}\ (\textsc{TFun})$$

$$\frac{}{\Gamma\vdash_v \underline{\mathcal{S}}:((\tau_1\to\tau_2@[\tau_3,\tau_3])\to\tau_4@[\tau_4,\tau_5])\to\tau_1@[\tau_2,\tau_5]}\ (\textsc{TShift})$$

$$\frac{\Gamma\vdash_v V:\tau_1}{\Gamma\vdash V:[\tau_1,\tau_2]\,\tau_2}\ (\textsc{TVal}) \qquad \frac{\Gamma\vdash M:[\tau_1\to\tau_2@[\tau_3,\tau_4],\tau_5]\,\tau_6 \quad \Gamma\vdash N:[\tau_1,\tau_4]\,\tau_5}{\Gamma\vdash M\mathbin{\underline{@}}N:[\tau_2,\tau_3]\,\tau_6}\ (\textsc{TApp})$$

$$\frac{\Gamma\vdash M:[\tau_1,\tau_4]\,\tau_5 \quad \Gamma,x:\tau_1\vdash N:[\tau_2,\tau_3]\,\tau_4}{\Gamma\vdash \underline{\mathsf{let}}\ x=M\ \underline{\mathsf{in}}\ N:[\tau_2,\tau_3]\,\tau_5}\ (\textsc{TLet}) \qquad \frac{\Gamma\vdash M:[\tau_1,\tau_1]\,\tau_2}{\Gamma\vdash \underline{\langle}M\underline{\rangle}:[\tau_2,\tau_3]\,\tau_3}\ (\textsc{TReset})$$

図 2. DS 項の型規則

types	τ_i	::=	$\mathsf{int} \mid \tau_1\to(\tau_2\to\tau_3)\to\tau_4$
cont. types	δ	::=	$\tau_2 \xrightarrow{k} \tau_3 \mid \tau_2 \xrightarrow{\bullet} \tau_2$
terms	M_Δ	::=	$K_\Delta \mathbin{\underline{@}} V \mid V \mathbin{\underline{@}} W \mathbin{\underline{@}} K_\Delta \mid K_\Delta \mathbin{\underline{@}} M_\bullet$
values	V,W	::=	$x \mid \underline{\lambda}x.\,\underline{\lambda}k.\,M_k \mid S$
shift	S	::=	$\underline{\lambda}w.\,\underline{\lambda}j.\,w \mathbin{\underline{@}} (\underline{\lambda}y.\,\underline{\lambda}k.\,k \mathbin{\underline{@}} (j \mathbin{\underline{@}} y)) \mathbin{\underline{@}} (\underline{\lambda}x.\,x)$
continuations	J_Δ,K_Δ	::=	$_{(\Delta=k)}\ k \mid {}_{(\Delta=\bullet)}\ (\underline{\lambda}x.\,x) \mid \underline{\lambda}x.\,M_\Delta$

$(\beta.v)$	$(\underline{\lambda}x.\,\underline{\lambda}k.\,M_k) \mathbin{\underline{@}} V \mathbin{\underline{@}} K_\Delta$	$\longrightarrow$	$M_k[V/x,K_\Delta/k]$	
$(\eta.v)$	$\underline{\lambda}x.\,\underline{\lambda}k.\,V \mathbin{\underline{@}} x \mathbin{\underline{@}} k$	$\longrightarrow$	V	if $x,k\notin fv(V)$
$(\beta.let)$	$(\underline{\lambda}x.\,M_\Delta) \mathbin{\underline{@}} V$	$\longrightarrow$	$M_\Delta[V/x]$	
$(\eta.let)$	$\underline{\lambda}x.\,K_\Delta \mathbin{\underline{@}} x$	$\longrightarrow$	K_Δ	if $x\notin fv(K_\Delta)$
$(\beta.\mathcal{S})$	$K_\Delta \mathbin{\underline{@}} (S \mathbin{\underline{@}} W \mathbin{\underline{@}} J_\bullet)$	$\longrightarrow$	$K_\Delta \mathbin{\underline{@}} (W \mathbin{\underline{@}} (\underline{\lambda}y.\,\underline{\lambda}k.\,k \mathbin{\underline{@}} (J_\bullet \mathbin{\underline{@}} y)) \mathbin{\underline{@}} (\underline{\lambda}x.\,x))$	if $y,k\notin fv(J_\bullet)$
$(\beta.\mathcal{R})$	$K_\Delta \mathbin{\underline{@}} ((\underline{\lambda}x.\,x) \mathbin{\underline{@}} V)$	$\longrightarrow$	$K_\Delta \mathbin{\underline{@}} V$	

図 3. CPS 項の構文

2.2 CPS 言語

次に CPS 言語の定義を図 3 に示す。$K_\Delta \mathbin{\underline{@}} V$ は値 V を継続 K_Δ に返すことを、$V \mathbin{\underline{@}} W \mathbin{\underline{@}} K_\Delta$ は CPS での関数 V の呼び出しを示す。すべての部分式に名前がつく形にならなくてはいけないので、V や W は値でなくてはならない。$K_\Delta \mathbin{\underline{@}} M_\bullet$ は reset を CPS 変換して得られる項である。$M_\bullet$ は CPS で書かれた式だが、その結果が直接形式で K_Δ に渡されているため、ここで継続が区切られていることになる。値のところに出てくる S は DS 項の $\underline{\mathcal{S}}$ 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 か $\bullet$ のどちらかの値を取る。ここで k は継続を表す定数である。純粋な CPS 言語では、継続を表す変数同士はお互いに干渉しないため、いつも決まった k という名前の変数を使えることが知られている [14]。継続 K_Δ の定義は 1 行で書かれているが、Δ を展開すると

$$\begin{array}{rrcl} \text{continuations} & J_k,K_k & ::= & k \mid \underline{\lambda}x.\,M_k \\ & J_\bullet,K_\bullet & ::= & \underline{\lambda}x.\,x \mid \underline{\lambda}x.\,M_\bullet \end{array}$$

と書いているのと同じことである。K_k と $K_\bullet$ の区別は、それを使っている M_Δ が CPS に変換される前に reset で囲まれていたかどうかを示す。例えば $\underline{\lambda}x.\,x$ は CPS 変換すると $\underline{\lambda}x.\,\underline{\lambda}k.\,k \mathbin{\underline{@}} x$ となるが、この $k \mathbin{\underline{@}} x$ の k は K_k の形をしており、これで $k \mathbin{\underline{@}} x$ が reset には囲まれていなかったことを示す。一方、$\underline{\langle}1\underline{\rangle}$ は CPS 変換すると $\underline{\lambda}k.\,k \mathbin{\underline{@}} ((\underline{\lambda}x.\,x) \mathbin{\underline{@}} 1)$ となるが、この $((\underline{\lambda}x.\,x) \mathbin{\underline{@}} 1)$ の $\underline{\lambda}x.\,x$ は $K_\bullet$ の形をしており、これで $((\underline{\lambda}x.\,x) \mathbin{\underline{@}} 1)$ が reset に囲まれていたことを示す。この区別は CPS 言語を正確に表現するだけでなく、型をつける上でも重要な役割を果たす。

$$\frac{x:\tau\in\Gamma}{\Gamma\vdash_v x:\tau}\ (\textsc{TVar}) \qquad \frac{\Gamma, x:\tau_1\,[k:\tau_2 \xrightarrow{k} \tau_3]\vdash M_k:\tau_4}{\Gamma\vdash_v \underline{\lambda}x.\,\underline{\lambda}k.\,M_k:\tau_1\to(\tau_2\to\tau_3)\to\tau_4}\ (\textsc{TFun})$$

$$\frac{}{\Gamma\vdash_v S:((\tau_1\to(\tau_2\to\tau_3)\to\tau_3)\to(\tau_4\to\tau_4)\to\tau_5)\to(\tau_1\to\tau_2)\to\tau_5}\ (\textsc{TShift})$$

$$\frac{\Gamma\,[\Delta:\delta]\vdash_k K_\Delta:\tau_1\to\tau_2 \quad \Gamma\vdash_v V:\tau_1}{\Gamma\,[\Delta:\delta]\vdash K_\Delta\,\underline{@}\,V:\tau_2}\ (\textsc{TVal})$$

$$\frac{\Gamma\vdash_v V:\tau_1\to(\tau_2\to\tau_3)\to\tau_4 \quad \Gamma\vdash_v W:\tau_1 \quad \Gamma\,[\Delta:\delta]\vdash_k K_\Delta:\tau_2\to\tau_3}{\Gamma\,[\Delta:\delta]\vdash V\,\underline{@}\,W\,\underline{@}\,K_\Delta:\tau_4}\ (\textsc{TApp})$$

$$\frac{\Gamma\,[\Delta:\delta]\vdash_k K_\Delta:\tau_2\to\tau_3 \quad \Gamma\,[\bullet:\tau_1 \xrightarrow{\bullet} \tau_1]\vdash M_\bullet:\tau_2}{\Gamma\,[\Delta:\delta]\vdash K_\Delta\,\underline{@}\,M_\bullet:\tau_3}\ (\textsc{TReset})$$

$$\frac{}{\Gamma\,[k:\tau_2\xrightarrow{k}\tau_3]\vdash_k k:\tau_2\to\tau_3}\ (\textsc{TKVar}) \qquad \frac{}{\Gamma\,[\bullet:\tau_2\xrightarrow{\bullet}\tau_2]\vdash_k \underline{\lambda}x.\,x:\tau_2\to\tau_2}\ (\textsc{TKId})$$

$$\frac{\Gamma, x:\tau_2\,[\Delta:\delta]\vdash M_\Delta:\tau_3}{\Gamma\,[\Delta:\delta]\vdash_k \underline{\lambda}x.\,M_\Delta:\tau_2\to\tau_3}\ (\textsc{TKLet})$$

図 4. CPS 項の型規則

CPS 言語の簡約規則は、単純型付き λ 計算の簡約規則として自然なものになっている。CPS 言語であるため $(\beta.v)$ では引数の代入と継続の代入が同時に行われるように定義されている。この簡約規則の定義は、DS 言語の簡約規則と対応しているが、ここには $(let.1)$, $(let.2)$, $(assoc)$ の規則は現れない。これは CPS 言語ではもともと全ての部分式に名前がついており、名前付け用の規則が不要になっているためである。

図 4 に CPS 項の型規則を示す。型規則は、値用、項用、継続用の 3 つに分かれている。(TFun) は、前提部で k の型が [...] に囲まれていることを除けば普通である。k の型を [...] で囲んでいるのは、k は変数ではなく継続を表す特別な定数として、k の型を環境には入れず、別途、管理しているためである。特に Agda で定式化する際には、束縛情報を Parameterized Higher-Order Abstract Syntax (PHOAS) [5] を使って表現しているが、そこには k は含めないことを意味している。一方、(TReset) では [...] の中に入るのは $\bullet$ の型となる。これで、$M_\bullet$ の周りに来ているのは初期継続 $\underline{\lambda}x.\,x$ であることを示している。[...] の中に入った Δ は最終的には継続の型規則で参照される。その際、Δ が $\bullet$ だった場合 (TKId)、その型は恒等関数の型でなくてはならない。

Δ の型をこのように引き回すのは、その型情報 (特に k の型情報) が項の型を決めるのに必要なためである。それを見るため、例として DS 項 $f\,\underline{@}\,((g\,\underline{@}\,h)\,\underline{@}\,a)$ を考えてみよう。この項は次のような型を持つ。(変数の上には (TVal) と (TVar) の規則が続くが、紙面の都合上、それらは省略している。)

$$\dfrac{\Gamma\vdash f:[\tau_1\to\tau_2@[\tau_3,\tau_4],\tau_5]\,\tau_5 \quad \dfrac{\dfrac{\Gamma\vdash g:[\tau_8\to(\tau_6\to\tau_1@[\tau_4,\tau_7])@[\tau_7,\tau_5],\tau_5]\,\tau_5 \quad \Gamma\vdash h:[\tau_8,\tau_5]\,\tau_5}{\Gamma\vdash g\,\underline{@}\,h:[\tau_6\to\tau_1@[\tau_4,\tau_7],\tau_7]\,\tau_5}\ (\textsc{TApp}) \quad \Gamma\vdash a:[\tau_6,\tau_7]\,\tau_7}{\Gamma\vdash (g\,\underline{@}\,h)\,\underline{@}\,a:[\tau_1,\tau_4]\,\tau_5}\ (\textsc{TApp})}{\Gamma\vdash f\,\underline{@}\,((g\,\underline{@}\,h)\,\underline{@}\,a):[\tau_2,\tau_3]\,\tau_5}\ (\textsc{TApp})$$

$$\Gamma = f:\tau_1\to\tau_2@[\tau_3,\tau_4], g:\tau_8\to(\tau_6\to\tau_1@[\tau_4,\tau_7])@[\tau_7,\tau_5], h:\tau_8, a:\tau_6$$

ここで、この式が $\underline{\lambda}z.$ の下に現れる場合はこのまま

$$\frac{\Gamma, z:\tau_z\vdash f\,\underline{@}\,((g\,\underline{@}\,h)\,\underline{@}\,a):[\tau_2,\tau_3]\,\tau_5}{\Gamma\vdash_v \underline{\lambda}z.\,f\,\underline{@}\,((g\,\underline{@}\,h)\,\underline{@}\,a):\tau_z\to\tau_2@[\tau_3,\tau_5]}\ (\textsc{TFun})$$

となるが、$\underline{\langle}\cdot\underline{\rangle}$ の中に現れる場合は (TReset) の規則により次のように $\tau_2=\tau_3$ となる。

$$\frac{\Gamma\vdash f\,\underline{@}\,((g\,\underline{@}\,h)\,\underline{@}\,a):[\tau_2,\tau_2]\,\tau_5}{\Gamma\vdash \underline{\langle} f\,\underline{@}\,((g\,\underline{@}\,h)\,\underline{@}\,a)\underline{\rangle}:[\tau_5,\tau_9]\,\tau_9}\ (\textsc{TReset})$$

このように、DS 項では最後に (TFun) を使うのか (TReset) を使うかによって型が決まる。(上の例では $\tau_2 = \tau_3$ となるかどうかが決まる。)

一方、CPS 項の場合は事情が異なる。これらふたつの項をそれぞれ CPS 変換すると、

$$\underline{\lambda} z.\, \underline{\lambda} k.\, g \mathbin{\underline{@}} h \mathbin{\underline{@}} (\underline{\lambda} x.\, x \mathbin{\underline{@}} a \mathbin{\underline{@}} (\underline{\lambda} y.\, f \mathbin{\underline{@}} y \mathbin{\underline{@}} k))$$
$$k \mathbin{\underline{@}} (g \mathbin{\underline{@}} h \mathbin{\underline{@}} (\underline{\lambda} x.\, x \mathbin{\underline{@}} a \mathbin{\underline{@}} (\underline{\lambda} y.\, f \mathbin{\underline{@}} y \mathbin{\underline{@}} (\underline{\lambda} x.\, x))))$$

になる。これらに型をつけると以下のようになる。

$$\dfrac{\Gamma \vdash_v g : \tau_8 \to (\tau_x \to \tau_7) \to \tau_5 \quad \Gamma \vdash_v h : \tau_8 \quad \dfrac{\Gamma \vdash_v a : \tau_6 \quad \dfrac{\Gamma \vdash_v x : \tau_x \quad \dfrac{\Gamma \vdash_v a : \tau_6 \quad \dfrac{\Gamma \vdash_v x : \tau_x \quad \dfrac{\Gamma \vdash_v f : \tau_1 \to (\tau_2 \to \tau_3) \to \tau_4 \quad \Gamma \vdash_v y : \tau_1 \quad \Gamma [k : \tau_2 \xrightarrow{k} \tau_3] \vdash_k k : \tau_2 \to \tau_3}{\Gamma [k : \tau_2 \xrightarrow{k} \tau_3] \vdash f \mathbin{\underline{@}} y \mathbin{\underline{@}} k : \tau_4}\,(\textsc{TApp})}{\Gamma [k : \tau_2 \xrightarrow{k} \tau_3] \vdash x \mathbin{\underline{@}} a \mathbin{\underline{@}} (\underline{\lambda} y.\, f \mathbin{\underline{@}} y \mathbin{\underline{@}} k) : \tau_7}\,(\textsc{TKLet})}{\Gamma [k : \tau_2 \xrightarrow{k} \tau_3] \vdash_k \underline{\lambda} x.\, x \mathbin{\underline{@}} a \mathbin{\underline{@}} (\underline{\lambda} y.\, f \mathbin{\underline{@}} y \mathbin{\underline{@}} k) : \tau_x \to \tau_7}\,(\textsc{TApp})}{\Gamma [k : \tau_2 \xrightarrow{k} \tau_3] \vdash g \mathbin{\underline{@}} h \mathbin{\underline{@}} (\underline{\lambda} x.\, x \mathbin{\underline{@}} a \mathbin{\underline{@}} (\underline{\lambda} y.\, f \mathbin{\underline{@}} y \mathbin{\underline{@}} k)) : \tau_5}\,(\textsc{TKLet})}{\Gamma \vdash_v \underline{\lambda} z.\, \underline{\lambda} k.\, g \mathbin{\underline{@}} h \mathbin{\underline{@}} (\underline{\lambda} x.\, x \mathbin{\underline{@}} a \mathbin{\underline{@}} (\underline{\lambda} y.\, f \mathbin{\underline{@}} y \mathbin{\underline{@}} k)) : \tau_z \to (\tau_2 \to \tau_3) \to \tau_5}\,(\textsc{TFun})$$

$$\Gamma = f : \tau_1 \to (\tau_2 \to \tau_3) \to \tau_4, g : \tau_8 \to (\tau_x \to \tau_7) \to \tau_5, h : \tau_8, a : \tau_6$$
$$\Gamma' = \Gamma, z : \tau_z \qquad \Gamma'' = \Gamma', x : \tau_x \qquad \Gamma''' = \Gamma'', y : \tau_1 \qquad \tau_x = \tau_6 \to (\tau_1 \to \tau_4) \to \tau_7$$

$$\dfrac{\Gamma \vdash_v g : \tau_8 \to (\tau_x \to \tau_7) \to \tau_5 \quad \Gamma \vdash_v h : \tau_8 \quad \dfrac{\Gamma \vdash_v a : \tau_6 \quad \dfrac{\Gamma \vdash_v x : \tau_x \quad \dfrac{\Gamma \vdash_v a : \tau_6 \quad \dfrac{\Gamma \vdash_v x : \tau_x \quad \dfrac{\Gamma \vdash_v f : \tau_1 \to (\tau_2 \to \tau_2) \to \tau_4 \quad \Gamma \vdash_v y : \tau_1 \quad \Gamma [\bullet : \tau_2 \xrightarrow{\bullet} \tau_2] \vdash_k \underline{\lambda} x.\, x : \tau_2 \to \tau_2}{\Gamma [\bullet : \tau_2 \xrightarrow{\bullet} \tau_2] \vdash f \mathbin{\underline{@}} y \mathbin{\underline{@}} (\underline{\lambda} x.\, x) : \tau_4}\,(\textsc{TApp})}{\Gamma [\bullet : \tau_2 \xrightarrow{\bullet} \tau_2] \vdash x \mathbin{\underline{@}} a \mathbin{\underline{@}} (\underline{\lambda} y.\, f \mathbin{\underline{@}} y \mathbin{\underline{@}} (\underline{\lambda} x.\, x)) : \tau_7}\,(\textsc{TKLet})}{\Gamma [\bullet : \tau_2 \xrightarrow{\bullet} \tau_2] \vdash_k \underline{\lambda} x.\, x \mathbin{\underline{@}} a \mathbin{\underline{@}} (\underline{\lambda} y.\, f \mathbin{\underline{@}} y \mathbin{\underline{@}} (\underline{\lambda} x.\, x)) : \tau_x \to \tau_7}\,(\textsc{TApp})}{\Gamma [\bullet : \tau_2 \xrightarrow{\bullet} \tau_2] \vdash g \mathbin{\underline{@}} h \mathbin{\underline{@}} (\underline{\lambda} x.\, x \mathbin{\underline{@}} a \mathbin{\underline{@}} (\underline{\lambda} y.\, f \mathbin{\underline{@}} y \mathbin{\underline{@}} (\underline{\lambda} x.\, x))) : \tau_5}\,(\textsc{TKLet})}{\Gamma [k : \tau_5 \xrightarrow{k} \tau] \vdash k \mathbin{\underline{@}} (g \mathbin{\underline{@}} h \mathbin{\underline{@}} (\underline{\lambda} x.\, x \mathbin{\underline{@}} a \mathbin{\underline{@}} (\underline{\lambda} y.\, f \mathbin{\underline{@}} y \mathbin{\underline{@}} (\underline{\lambda} x.\, x)))) : \tau}\,(\textsc{TReset})$$

$$\Gamma = f : \tau_1 \to (\tau_2 \to \tau_2) \to \tau_4, g : \tau_8 \to (\tau_x \to \tau_7) \to \tau_5, h : \tau_8, a : \tau_6$$
$$\Gamma' = \Gamma \qquad \Gamma'' = \Gamma', x : \tau_x \qquad \Gamma''' = \Gamma'', y : \tau_1 \qquad \tau_x = \tau_6 \to (\tau_1 \to \tau_4) \to \tau_7$$

両者は、最後の継続の位置（証明木の一番右上の前提）に k がくるのか $\underline{\lambda} x.\, x$ がくるのかで異なっている。前者では (TFun) で必要な k の型が [...] の中に入って伝わっている。一方、後者では k は出てこないが、(TReset) を使っているので最後の継続が $\underline{\lambda} x.\, x$ であり $\tau_2 = \tau_3$ となることが必要である。この情報が [...] を使って伝わっている。[2]

このように CPS 項では、証明木の一番上で $\tau_2 = \tau_3$ となるかどうかが証明木一番下で (TFun) を使うのか (TReset) を使うのかによって決まる。この間の情報伝達をしているのが Δ である。

2.3 shift/reset 入りカーネル言語

図 5 に DSKernel 言語の定義を示す。この言語は、見た目上は直接形式の項になっているが、CPS 言語と一対一に対応した言語となっている。例えば、CPS 言語の $V \mathbin{\underline{@}} W \mathbin{\underline{@}} K_\Delta$ に対応して DSKernel 言語には $K_\Delta[V \mathbin{\underline{@}} W]$ がある。この K_Δ が $\underline{\mathsf{let}}\ x = [\,]\ \underline{\mathsf{in}}\ M_\Delta$ の形なら $\underline{\mathsf{let}}\ x = V \mathbin{\underline{@}} W\ \underline{\mathsf{in}}\ M_\Delta$ となり、$V \mathbin{\underline{@}} W$ に x という名前がついていることがわかる。

DSKernel 言語と CPS 言語の対応は、簡約規則においても保たれている。いずれの規則も CPS 言語の簡約規則と一対一に対応している。Sabry ら [14] の示したカーネル言語では、$(\beta.v)$ の規則において K_Δ は maximal という条件がついている。これは、例えば $\underline{\mathsf{let}}\ x = (\underline{\lambda} y.\, M_k) \mathbin{\underline{@}} 1\ \underline{\mathsf{in}}\ x$ のよう

[2] CPS 項の場合は最後の継続が $\underline{\lambda} x.\, x$ であることが分かっているので、この情報は必ずしも必要ではないが、後に述べる DSKernel 項では必要になる。

$$
\begin{array}{llll}
\text{types} & \tau_i & ::= & \mathsf{int} \mid \tau_1 \to \tau_2 @ [\tau_3, \tau_4] \\
\text{cont. types} & \delta & ::= & \tau_2 \xrightarrow{k} \tau_3 \mid \tau_2 \xrightarrow{\bullet} \tau_2 \\
\text{terms} & M_\Delta & ::= & K_\Delta[V] \mid K_\Delta[V \mathbin{\underline{@}} W] \mid K_\Delta[\underline{\langle} M_\bullet \underline{\rangle}] \\
\text{values} & V, W & ::= & x \mid \underline{\lambda} x.\, M_k \mid \underline{\mathcal{S}} \\
\text{contexts} & J_\Delta, K_\Delta & ::= & {}_{(\Delta = k)}\ [\,]_k \mid {}_{(\Delta = \bullet)}\ [\,]_\bullet \mid \underline{\mathsf{let}}\ x = [\,]\ \underline{\mathsf{in}}\ M_\Delta
\end{array}
$$

$$
\begin{array}{llll}
(\beta.v) & K_\Delta[(\underline{\lambda} x.\, M_k) \mathbin{\underline{@}} V] & \longrightarrow & M_k[V/x][K_\Delta/k] \\
(\eta.v) & \underline{\lambda} x.\, (V \mathbin{\underline{@}} x) & \longrightarrow & V \qquad \text{if } x \notin fv(V) \\
(\beta.let) & \underline{\mathsf{let}}\ x = V\ \underline{\mathsf{in}}\ M_\Delta & \longrightarrow & M_\Delta[V/x] \\
(\eta.let) & \underline{\mathsf{let}}\ x = [\,]\ \underline{\mathsf{in}}\ K_\Delta[x] & \longrightarrow & K_\Delta \qquad \text{if } x \notin fv(K_\Delta) \\
(\beta.\mathcal{S}) & K_\Delta[\underline{\langle} J_\bullet[\underline{\mathcal{S}} \mathbin{\underline{@}} W] \underline{\rangle}] & \longrightarrow & K_\Delta[\underline{\langle} W \mathbin{\underline{@}} (\underline{\lambda} y.\, \underline{\langle} J_\bullet[y] \underline{\rangle}) \underline{\rangle}] \quad \text{if } y \notin fv(J_\bullet) \\
(\beta.\mathcal{R}) & K_\Delta[\underline{\langle} V \underline{\rangle}] & \longrightarrow & K_\Delta[V]
\end{array}
$$

図 5. DSKernel 項の構文

$$
\frac{x : \tau \in \Gamma}{\Gamma \vdash_v x : \tau}\ (\textsc{TVar}) \qquad \frac{\Gamma, x : \tau_1\, [k : \tau_2 \xrightarrow{k} \tau_3] \vdash M_k : \tau_4}{\Gamma \vdash_v \underline{\lambda} x.\, M_k : \tau_1 \to \tau_2 @ [\tau_3, \tau_4]}\ (\textsc{TFun})
$$

$$
\frac{}{\Gamma \vdash_v \underline{\mathcal{S}} : ((\tau_1 \to \tau_2 @ [\tau_3, \tau_3]) \to \tau_4 @ [\tau_4, \tau_5]) \to \tau_1 @ [\tau_2, \tau_5]}\ (\textsc{TShift})
$$

$$
\frac{\Gamma\, [\Delta : \delta] \vdash_k K_\Delta : \tau_1 \to \tau_2 \quad \Gamma \vdash_v V : \tau_1}{\Gamma\, [\Delta : \delta] \vdash K_\Delta[V] : \tau_2}\ (\textsc{TVal})
$$

$$
\frac{\Gamma \vdash_v V : \tau_1 \to \tau_2 @ [\tau_3, \tau_4] \quad \Gamma \vdash_v W : \tau_1 \quad \Gamma\, [\Delta : \delta] \vdash_k K_\Delta : \tau_2 \to \tau_3}{\Gamma\, [\Delta : \delta] \vdash K_\Delta[V \mathbin{\underline{@}} W] : \tau_4}\ (\textsc{TApp})
$$

$$
\frac{\Gamma\, [\Delta : \delta] \vdash_k K_\Delta : \tau_2 \to \tau_3 \quad \Gamma\, [\bullet : \tau_1 \xrightarrow{\bullet} \tau_1] \vdash M_\bullet : \tau_2}{\Gamma\, [\Delta : \delta] \vdash K_\Delta[\underline{\langle} M_\bullet \underline{\rangle}] : \tau_3}\ (\textsc{TReset})
$$

$$
\frac{}{\Gamma\, [k : \tau_2 \xrightarrow{k} \tau_3] \vdash_k [\,]_k : \tau_2 \to \tau_3}\ (\textsc{TKVar}) \qquad \frac{}{\Gamma\, [\bullet : \tau_2 \xrightarrow{\bullet} \tau_2] \vdash_k [\,]_\bullet : \tau_2 \to \tau_2}\ (\textsc{TKId})
$$

$$
\frac{\Gamma, x : \tau_2\, [\Delta : \delta] \vdash M_\Delta : \tau_3}{\Gamma\, [\Delta : \delta] \vdash_k \underline{\mathsf{let}}\ x = [\,]\ \underline{\mathsf{in}}\ M_\Delta : \tau_2 \to \tau_3}\ (\textsc{TKLet})
$$

図 6. DSKernel 項の型規則

な項を考えたとき、$(\underline{\lambda} y.\, M_k) \mathbin{\underline{@}} 1$ の部分だけを切り出し、$K_\Delta = [\,]_\Delta$ だと思って $(\beta.v)$ を使ってはいけないという意味である。この状況は本稿でも同じだが、$\underline{\mathsf{let}}\ x = (\underline{\lambda} y.\, M_k) \mathbin{\underline{@}} 1\ \underline{\mathsf{in}}\ x$ の $(\underline{\lambda} y.\, M_k) \mathbin{\underline{@}} 1$ の部分を $K_\Delta[(\underline{\lambda} y.\, M_k) \mathbin{\underline{@}} 1]$ と解釈すると外側が構文的に成り立たなくなるので、入力を構文木だと解釈すれば K_Δ maximal の条件は不要になる。さらに、Sabry らはコロン変換を使って K_Δ を let 文の in 以下に移動している。これは β-簡約後の項が DSKernel 言語の構文には入らない（名前のつかない部分式が現れる可能性がある）ためである。ここでは、コンテキストの穴を（CPS 言語の k と $\underline{\lambda} x.\, x$ に対応して）ふたつ（$[\,]_k$ と $[\,]_\bullet$）に分け、コロン変換の代わりに対応する $[\,]_k$ に代入する（対応する $[\,]_k$ を K_Δ で置き換える）ようにしている。こちらの方が CPS 言語とより直接的に対応していると思われる。

図 6 に DSKernel 言語の型規則を示す。これは、CPS 言語の型規則の構文を DSKernel 言語に置き換えただけである。そういう意味では自然だが、一方で (TFun) を見ると M_k には項としては出てこない k の型が [...] に入って保持されている。また (TReset) でも [...] が使われている。これらが必要な理由は CPS 言語の場合とほぼ同じで、reset に囲まれているかどうかで型が変わって

しまうためである。

再び、同じ例を考えてみよう。前節で出てきた CPS 項

$$\underline{\lambda} z.\, \underline{\lambda} k.\, g \mathbin{\underline{@}} h \mathbin{\underline{@}} (\underline{\lambda} x.\, x \mathbin{\underline{@}} a \mathbin{\underline{@}} (\underline{\lambda} y.\, f \mathbin{\underline{@}} y \mathbin{\underline{@}} k))$$
$$k \mathbin{\underline{@}} (g \mathbin{\underline{@}} h \mathbin{\underline{@}} (\underline{\lambda} x.\, x \mathbin{\underline{@}} a \mathbin{\underline{@}} (\underline{\lambda} y.\, f \mathbin{\underline{@}} y \mathbin{\underline{@}} (\underline{\lambda} x.\, x))))$$

をそれぞれ DSKernel 項に変換させると、

$$\underline{\lambda} z.\, \underline{\mathsf{let}}\ x = [g \mathbin{\underline{@}} h]_k\ \underline{\mathsf{in}}\ \underline{\mathsf{let}}\ y = [x \mathbin{\underline{@}} a]_k\ \underline{\mathsf{in}}\ [f \mathbin{\underline{@}} y]_k$$
$$[\underline{\langle} \underline{\mathsf{let}}\ x = [g \mathbin{\underline{@}} h]_\bullet\ \underline{\mathsf{in}}\ \underline{\mathsf{let}}\ y = [x \mathbin{\underline{@}} a]_\bullet\ \underline{\mathsf{in}}\ [f \mathbin{\underline{@}} y]_\bullet \underline{\rangle}]_k$$

になる。これらに型をつけると以下のようになる。

$$\dfrac{\dfrac{\Gamma' \vdash_v g : \tau_8 \to \tau_x @ [\tau_7, \tau_5] \quad \dfrac{\dfrac{\dfrac{\Gamma''' \vdash_v f : \tau_1 \to \tau_2 @ [\tau_3, \tau_4] \quad \Gamma''' \vdash_v y : \tau_1 \quad \Gamma''' [k : \tau_2 \xrightarrow{k} \tau_3] \vdash_k []_k : \tau_2 \to \tau_3}{\Gamma''' [k : \tau_2 \xrightarrow{k} \tau_3] \vdash [f \mathbin{\underline{@}} y]_k : \tau_4}\ (\textsc{TKLet}) \quad \Gamma'' \vdash_v a : \tau_6}{\Gamma'' [k : \tau_2 \xrightarrow{k} \tau_3] \vdash_k \underline{\mathsf{let}}\ y = []\ \underline{\mathsf{in}}\ [f \mathbin{\underline{@}} y]_k : \tau_1 \to \tau_4}\ (\textsc{TApp}) \quad \Gamma'' [k : \tau_2 \xrightarrow{k} \tau_3] \vdash \underline{\mathsf{let}}\ y = [x \mathbin{\underline{@}} a]_k\ \underline{\mathsf{in}}\ [f \mathbin{\underline{@}} y]_k : \tau_7}\ (\textsc{TKLet}) \quad \Gamma' \vdash_v h : \tau_8}{\Gamma' [k : \tau_2 \xrightarrow{k} \tau_3] \vdash_k \underline{\mathsf{let}}\ x = []\ \underline{\mathsf{in}}\ \underline{\mathsf{let}}\ y = [x \mathbin{\underline{@}} a]_k\ \underline{\mathsf{in}}\ [f \mathbin{\underline{@}} y]_k : \tau_x \to \tau_7}\ (\textsc{TApp})}{\Gamma' [k : \tau_2 \xrightarrow{k} \tau_3] \vdash \underline{\mathsf{let}}\ x = [g \mathbin{\underline{@}} h]_k\ \underline{\mathsf{in}}\ \underline{\mathsf{let}}\ y = [x \mathbin{\underline{@}} a]_k\ \underline{\mathsf{in}}\ [f \mathbin{\underline{@}} y]_k : \tau_5}\ (\textsc{TKLet})}{\Gamma \vdash_v \underline{\lambda} z.\, \underline{\mathsf{let}}\ x = [g \mathbin{\underline{@}} h]_k\ \underline{\mathsf{in}}\ \underline{\mathsf{let}}\ y = [x \mathbin{\underline{@}} a]_k\ \underline{\mathsf{in}}\ [f \mathbin{\underline{@}} y]_k : \tau_z \to \tau_2 @ [\tau_3, \tau_5]}\ (\textsc{TFun})$$

$$\Gamma = f : \tau_1 \to \tau_2 @ [\tau_3, \tau_4], g : \tau_8 \to \tau_x @ [\tau_7, \tau_5], h : \tau_8, a : \tau_6$$
$$\Gamma' = \Gamma, z : \tau_z \qquad \Gamma'' = \Gamma', x : \tau_x \qquad \Gamma''' = \Gamma'', y : \tau_1 \qquad \tau_x = \tau_6 \to \tau_1 @ [\tau_4, \tau_7]$$

$$\dfrac{\dfrac{\Gamma \vdash_v g : \tau_8 \to \tau_x @ [\tau_7, \tau_5] \quad \dfrac{\dfrac{\dfrac{\Gamma''' \vdash_v f : \tau_1 \to \tau_2 @ [\tau_2, \tau_4] \quad \Gamma''' \vdash_v y : \tau_1 \quad \Gamma''' [\bullet : \tau_2 \xrightarrow{\bullet} \tau_2] \vdash_k []_\bullet : \tau_2 \to \tau_2}{\Gamma''' [\bullet : \tau_2 \xrightarrow{\bullet} \tau_2] \vdash [f \mathbin{\underline{@}} y]_\bullet : \tau_4}\ (\textsc{TKLet}) \quad \Gamma'' \vdash_v a : \tau_6}{\Gamma'' [\bullet : \tau_2 \xrightarrow{\bullet} \tau_2] \vdash_k \underline{\mathsf{let}}\ y = []\ \underline{\mathsf{in}}\ [f \mathbin{\underline{@}} y]_\bullet : \tau_1 \to \tau_4}\ (\textsc{TApp}) \quad \Gamma'' [\bullet : \tau_2 \xrightarrow{\bullet} \tau_2] \vdash \underline{\mathsf{let}}\ y = [x \mathbin{\underline{@}} a]_\bullet\ \underline{\mathsf{in}}\ [f \mathbin{\underline{@}} y]_\bullet : \tau_7}\ (\textsc{TKLet}) \quad \Gamma' \vdash_v h : \tau_8}{\Gamma' [\bullet : \tau_2 \xrightarrow{\bullet} \tau_2] \vdash_k \underline{\mathsf{let}}\ x = []\ \underline{\mathsf{in}}\ \underline{\mathsf{let}}\ y = [x \mathbin{\underline{@}} a]_\bullet\ \underline{\mathsf{in}}\ [f \mathbin{\underline{@}} y]_\bullet : \tau_x \to \tau_7}\ (\textsc{TApp})}{\Gamma [k : \tau_5 \xrightarrow{k} \tau] \vdash_k []_k : \tau_5 \to \tau \qquad \Gamma' [\bullet : \tau_2 \xrightarrow{\bullet} \tau_2] \vdash \underline{\mathsf{let}}\ x = [g \mathbin{\underline{@}} h]_\bullet\ \underline{\mathsf{in}}\ \underline{\mathsf{let}}\ y = [x \mathbin{\underline{@}} a]_\bullet\ \underline{\mathsf{in}}\ [f \mathbin{\underline{@}} y]_\bullet : \tau_5}\ (\textsc{TReset})$$
$$\Gamma [k : \tau_5 \xrightarrow{k} \tau] \vdash [\underline{\langle} \underline{\mathsf{let}}\ x = [g \mathbin{\underline{@}} h]_\bullet\ \underline{\mathsf{in}}\ \underline{\mathsf{let}}\ y = [x \mathbin{\underline{@}} a]_\bullet\ \underline{\mathsf{in}}\ [f \mathbin{\underline{@}} y]_\bullet \underline{\rangle}]_k : \tau$$

$$\Gamma = f : \tau_1 \to \tau_2 @ [\tau_2, \tau_4], g : \tau_8 \to \tau_x @ [\tau_7, \tau_5], h : \tau_8, a : \tau_6$$
$$\Gamma' = \Gamma \qquad \Gamma'' = \Gamma', x : \tau_x \qquad \Gamma''' = \Gamma'', y : \tau_1 \qquad \tau_x = \tau_6 \to \tau_1 @ [\tau_4, \tau_7]$$

両者は、最後の継続の位置（証明木の一番右上の前提）に $[]_k$ がくるのか $[]_\bullet$ がくるのかで異なっている。CPS 項のときは、最後の継続が k か $\underline{\lambda} x.\, x$ で別の形をしていたが、ここではどちらも空のコンテキストになっている。これらは項としては同じだが、型が異なっている。この型の情報が [...] を使って伝わっている。

3 DSKernel 項と CPS 項間の Reflection

ここでは、まず DSKernel 項と CPS 項間の Reflection について考える。DSKernel 項と CPS 項は、項や型、簡約規則から型規則まで定義が一対一に対応しているため、両者の間には Reflection より強い Order Isomorphism の関係が成り立つ。（つまり、両者は見た目が違うだけで同型である。）DSKernel 項 M の CPS 変換を M°、CPS 項 N の DS 変換を $N^\sharp$ と書く（定義はいずれも自明な形になる。付録を参照）ことにすると、以下の定理が成り立つ。

定理 2 (DSKernel 項と CPS 項の間の Order Isomorphism)

1. 任意の DSKernel 項 M_Δ について $M_\Delta = M_\Delta^{\circ\sharp}$

2. 任意の CPS 項 N_Δ について $N_\Delta^{\sharp\circ} = N_\Delta$
3. 任意の DSKernel 項 M_Δ, M'_Δ について $M_\Delta \longrightarrow M'_\Delta$ ならば $M_\Delta^\circ \longrightarrow M'^\circ_\Delta$
4. 任意の CPS 項 N_Δ, N'_Δ について $N_\Delta \longrightarrow N'_\Delta$ ならば $N_\Delta^\sharp \longrightarrow N'^\sharp_\Delta$

項が、値と継続との相互再帰で定義されているため、正確にはこれら 3 つについての文言になる（付録を参照）。証明もこれらの間の相互帰納法で行う。

ふたつの言語が一対一に対応しているので、この定理の証明は素直に行うことができる。しかし、これを Agda で定式化するのは必ずしも簡単ではない。というのは DSKernel 項の $\underline{\lambda}x.\, M_k$ の M_k には x のみが束縛変数として現れているのに対して、CPS 項の $\underline{\lambda}x.\, \underline{\lambda}k.\, M_k$ の M_k には x と k が束縛変数として現れているためである。本稿の定式化では、変数名の付け替えを自然に扱うために PHOAS を使っているが、そうすると異なる束縛変数を持つふたつの M_k を関係づけることができなくて破綻する。ここでは CPS 項の k を束縛変数ではなく特別な定数として扱うことで、両者の束縛変数を合わせた。さらに DSKernel 項の M_k の型に（この M_k の中には項としては k は現れていないにも関わらず）k の型情報を入れることで、両者の間の厳密な関係を維持することができるようになり、証明を完成させることができた。

上記に加えて、2. の性質の証明には関数の外延性の公理が必要だった。これは、λ 抽象についても、CPS 変換して DS 変換した結果が不変であることを言う必要があるためである。PHOAS を使った定式化では、関数はメタ言語 (Agda) の関数として表現されるが、それの等価性を直接示すことはできず、外延性の公理、つまり「すべての引数 x について $f(x) = g(x)$ なら $f = g$」という等価性を使う必要があった。

4 DS 項と DSKernel 項間の Reflection

この節では、DS 項と DSKernel 項との間の Reflection 関係を示す。まず最初に DS 項と DSKernel 項の間の変換規則を示し、その後で Reflection の関係について述べる。いずれも Agda で型付きで定式化しているため、変換前に型が付いていると変換後にも型が付くことが保証されている。

図 7 に DS 項から DSKernel 項への変換規則を示す。これは A-正規形変換であり、値ではないすべての部分式に名前を与える変換になっている。この変換はコロン変換 [12] を使って値でない項を分解するとともに、継続を let 文の in 以下に移動することでフラットな let 文にしている。返す項を DSKernel 項ではなく、それと同型の CPS 項にすれば従来の CPS 変換になる。DS 言語の項 M を DSKernel 言語のコンテキスト K_Δ のもとで A-正規形変換するには $M :: K_\Delta$ とする。

図 7 の変換自体はごく普通の定義だが、DSKernel 項には直接は現れない Δ の情報を引き回しているところが型付きで定式化する際のポイントである。Δ を引き回しても、構文的には最終的にどちらも [] になるので引き回さなくても同じ結果を得ることができる。しかし、Δ の情報を引き回していないと、例えば $K_\Delta = []_\Delta$ のときに $K_\Delta[V^{\dagger\dagger}]$ に型をつけようとした場合、全体の型が $V^{\dagger\dagger}$ の型と等しくなくてはいけないのか（$\Delta = \bullet$ の場合）、等しくなくてもよいのか（$\Delta = k$ の場合）を判断できなくなってしまう。

一方、図 8 は DSKernel 項から DS 項への埋め込みを表す変換 $M^\oplus$ である。DSKernel 言語は DS 言語の一部なので、その埋め込みは自明と思われるかも知れないが、必ずしもそうではない。DSKernel 言語は項の定義にコンテキストを使っているが DS 言語では項の定義にコンテキストは使っていない。ここでのポイントは、図 8 のように自然に埋め込みを定義できるように DS 言語のコンテキストを定義することにある。

再び $f \mathbin{\underline{@}} ((g \mathbin{\underline{@}} h) \mathbin{\underline{@}} a)$ を例に考えてみよう。この項は A-正規形変換すると以下のようになる。

$$\underline{\mathsf{let}}\ x = [g \mathbin{\underline{@}} h]_\Delta\ \underline{\mathsf{in}}\ \underline{\mathsf{let}}\ y = [x \mathbin{\underline{@}} a]_\Delta\ \underline{\mathsf{in}}\ [f \mathbin{\underline{@}} y]_\Delta$$

$$
\begin{array}{rcl}
\mathsf{int}^{\triangleright} & = & \mathsf{int} \\
(\tau_1 \to \tau_2 @ [\tau_3, \tau_4])^{\triangleright} & = & \tau_1^{\triangleright} \to (\tau_2^{\triangleright} \to \tau_3^{\triangleright}) \to \tau_4^{\triangleright} \\
(\tau_2 \to \tau_3)^{\triangleright} & = & \tau_2^{\triangleright} \xrightarrow{k} \tau_3^{\triangleright} \\
V :: K_\Delta & = & K_\Delta[V^{\dagger\!\dagger}] \\
(P \mathbin{\underline{@}} Q) :: K_\Delta & = & P :: (\underline{\mathsf{let}}\ x = [\,]\ \underline{\mathsf{in}}\ (Q :: (\underline{\mathsf{let}}\ y = [\,]\ \underline{\mathsf{in}}\ K_\Delta[x \mathbin{\underline{@}} y]))) \\
(P \mathbin{\underline{@}} W) :: K_\Delta & = & P :: (\underline{\mathsf{let}}\ x = [\,]\ \underline{\mathsf{in}}\ K_\Delta[x \mathbin{\underline{@}} W^{\dagger\!\dagger}]) \\
(V \mathbin{\underline{@}} Q) :: K_\Delta & = & Q :: (\underline{\mathsf{let}}\ y = [\,]\ \underline{\mathsf{in}}\ K_\Delta[V^{\dagger\!\dagger} \mathbin{\underline{@}} y]) \\
(V \mathbin{\underline{@}} W) :: K_\Delta & = & K_\Delta[V^{\dagger\!\dagger} \mathbin{\underline{@}} W^{\dagger\!\dagger}] \\
(\underline{\mathsf{let}}\ x = M\ \underline{\mathsf{in}}\ N) :: K_\Delta & = & M :: (\underline{\mathsf{let}}\ x = [\,]\ \underline{\mathsf{in}}\ (N :: K_\Delta)) \\
\underline{\langle} M \underline{\rangle} :: K_\Delta & = & K_\Delta[\underline{\langle} M :: [\,]_\bullet \underline{\rangle}] \\
x^{\dagger\!\dagger} & = & x \\
(\underline{\lambda} x.\, M)^{\dagger\!\dagger} & = & \underline{\lambda} x.\, (M :: [\,]_k) \\
\underline{\mathcal{S}}^{\dagger\!\dagger} & = & \underline{\mathcal{S}}
\end{array}
$$

図 7. $\lambda_{c\mathcal{S}}$ の A-正規形変換

この項は「コンテキスト [項]」の形で書き直すと以下のようになる。

$$K_\Delta^1[g \mathbin{\underline{@}} h],\quad K_\Delta^1 = \underline{\mathsf{let}}\ x = [\,]_\Delta\ \underline{\mathsf{in}}\ K_\Delta^2[x \mathbin{\underline{@}} a],\quad K_\Delta^2 = \underline{\mathsf{let}}\ y = [\,]_\Delta\ \underline{\mathsf{in}}\ K_\Delta^3[f \mathbin{\underline{@}} y],\quad K_\Delta^3 = [\,]_\Delta$$

一方、もとの DS 項はコンテキストを使って次のように書き直せる。

$$K^1[g \mathbin{\underline{@}} h],\quad K^1 = K^2[[\,] \mathbin{\underline{@}} a],\quad K^2 = K^3[f \mathbin{\underline{@}} [\,]],\quad K^3 = [\,]$$

このように書くと DSKernel 側の $K_\Delta[V \mathbin{\underline{@}} W]$（これは DSKernel の項）と DS 側の $K_\Delta[V \mathbin{\underline{@}} W]$（これは DS 項 $V \mathbin{\underline{@}} W$ を DS のコンテキスト K_Δ に plug したもの）の順番が合い、対応が取れることがわかる。

ここで、DS 言語のコンテキスト（図 1）が inside-out の格好で定義されていることに注意したい。より多く用いられるのは outside-in のコンテキスト

$$K \quad ::= \quad [\,] \mid K \mathbin{\underline{@}} M \mid V \mathbin{\underline{@}} K \mid \underline{\mathsf{let}}\ x = K\ \underline{\mathsf{in}}\ M$$

だが、この定義を使ってしまうと、先の例は

$$K^1[g \mathbin{\underline{@}} h],\quad K^1 = f \mathbin{\underline{@}} K^2,\quad K^2 = K^3 \mathbin{\underline{@}} a,\quad K^3 = [\,]$$

となり、これだとコンテキストの順番が逆になり、DSKernel 項との対応がわからなくなってしまう。[3] 以上の考察を経ると DS 言語のコンテキストの型規則を作ることができる（図 9）。これは、DSKernel 言語のコンテキストと対応がつく形になっている。

以上のセットアップをすると、Reflection の証明を行うことができる。

定理 3 (DS 項と DSKernel 項の間の Reflection)

1. 任意の DS 項 M と DSKernel 言語の継続 K_Δ について $K_\Delta^{\ominus}[M] \longrightarrow^* (M :: K_\Delta)^{\oplus}$
2. 任意の DSKernel 項 N_Δ について $N_\Delta^{\oplus} : [\,]_\Delta = N_\Delta$
3. 任意の DS 項 M, M' と DSKernel 言語のコンテキスト K_Δ について $M \longrightarrow M'$ ならば $M :: K_\Delta \longrightarrow^* M' :: K_\Delta$
4. 任意の DSKernel 項 N_Δ, N'_Δ について $N_\Delta \longrightarrow N'_\Delta$ ならば $N_\Delta^{\oplus} \longrightarrow^* N_\Delta'^{\oplus}$

[3] Biernacki ら [3] は outside-in のコンテキストを論文では書いているが、実際には inside-out のコンテキストを使っているのではないかと想像される。

$$
\begin{array}{rcl}
\mathsf{int}^{\oplus} & = & \mathsf{int} \\
(\tau_1 \to \tau_2 @ [\tau_3, \tau_4])^{\oplus} & = & \tau_1^{\oplus} \to \tau_2^{\oplus} @ [\tau_3^{\oplus}, \tau_4^{\oplus}] \\
(\tau_2 \xrightarrow{k} \tau_3)^{\ominus} & = & \tau_2^{\oplus} \to \tau_3^{\oplus} \\
(\tau_2 \xrightarrow{\bullet} \tau_2)^{\ominus} & = & \tau_2^{\oplus} \to \tau_2^{\oplus} \\
(K_\Delta[V])^{\oplus} & = & K_\Delta^{\ominus}[V^{\odot}] \\
(K_\Delta[V \mathbin{\underline{@}} W])^{\oplus} & = & K_\Delta^{\ominus}[V^{\odot} \mathbin{\underline{@}} W^{\odot}] \\
(K_\Delta[\underline{\lfloor} M_\bullet \underline{\rfloor}])^{\oplus} & = & K_\Delta^{\ominus}[\underline{\lfloor} M_\bullet^{\oplus} \underline{\rfloor}] \\
x^{\odot} & = & x \\
(\underline{\lambda} x.\, M_k)^{\odot} & = & \underline{\lambda} x.\, M_k^{\oplus} \\
\underline{\mathcal{S}}^{\odot} & = & \underline{\mathcal{S}} \\
[]_k^{\ominus} & = & []_k \\
[]_\bullet^{\ominus} & = & []_\bullet \\
(\underline{\mathsf{let}}\ x = [\,]\ \underline{\mathsf{in}}\ N_\Delta)^{\ominus} & = & \underline{\mathsf{let}}\ x = [\,]\ \underline{\mathsf{in}}\ N_\Delta^{\oplus}
\end{array}
$$

図 8. $\lambda_{c\mathcal{S}}^{\triangleright}$ の $\lambda_{c\mathcal{S}}$ への埋め込み

$$
\frac{}{\Gamma\,[\tau_2 \to \tau_3] \vdash_k []_k : \tau_2 \to \tau_3}\ (\textsc{TKVar}) \qquad \frac{}{\Gamma\,[\tau_2 \to \tau_2] \vdash_k []_\bullet : \tau_2 \to \tau_2}\ (\textsc{TKId})
$$

$$
\frac{\Gamma\,[\Delta] \vdash_k K_\Delta : \tau_2 \to \tau_3 \quad \Gamma \vdash M : [\tau_1, \tau_4]\,\tau_5}{\Gamma\,[\Delta] \vdash_k K_\Delta[[\,] \mathbin{\underline{@}} M] : (\tau_1 \to \tau_2 @ [\tau_3, \tau_4]) \to \tau_5}\ (\textsc{TKApp1})
$$

$$
\frac{\Gamma\,[\Delta] \vdash_k K_\Delta : \tau_2 \to \tau_3 \quad \Gamma \vdash_v V : \tau_1 \to \tau_2 @ [\tau_3, \tau_4]}{\Gamma\,[\Delta] \vdash_k K_\Delta[V \mathbin{\underline{@}} [\,]] : \tau_1 \to \tau_4}\ (\textsc{TKApp2})
$$

$$
\frac{\Gamma\,[\Delta] \vdash_k K_\Delta : \tau_2 \to \tau_3 \quad \Gamma, x : \tau_1 \vdash M : [\tau_2, \tau_3]\,\tau_4}{\Gamma\,[\Delta] \vdash_k K_\Delta[\underline{\mathsf{let}}\ x = [\,]\ \underline{\mathsf{in}}\ M] : \tau_1 \to \tau_4}\ (\textsc{TKLet})
$$

図 9. DS 項の pure contexts の型規則

帰納法を回すために一部、一般化して DSKernel 言語のコンテキスト K_Δ を使った文言になっている。ここで、$M :: K_\Delta$ が M の（K_Δ のもとでの）A-正規形変換、$N_\Delta^{\oplus}$ が N_Δ の DS 項への埋め込みを表している。$K_\Delta = []_\Delta$ とすれば 1. は $M \longrightarrow^* (M :: []_\Delta)^{\oplus}$、3. は $M :: []_\Delta \longrightarrow^* M' :: []_\Delta$ という本来、示したかった文言になる。$K_\Delta^{\ominus}$ は K_Δ を DS 言語のコンテキストに変換する写像である。定義については付録を参照。

証明はすべて Agda で定式化されているが、全体の流れはほぼ Biernacki らの証明に沿う形である。shift の挙動をとらえる主要な補題は以下になる。

補題 4 任意の DS 言語のコンテキスト J と DSKernel 言語の継続 K_Δ について、以下のふたつを満たすような DSKernel 言語の継続 $\hat{J}_\Delta$ が存在する。

- 任意の DS 言語の値でない項 M について $J[M] :: K_\Delta = M :: \hat{J}_\Delta$
- 任意の DS 言語の値 V について $V :: \hat{J}_\Delta \longrightarrow^* J[V] :: K_\Delta$

この補題の $J[M]$ は M が shift で J がその周りのコンテキストである場合に使用される。前半は J を K_Δ の方に押しやることができること、後半は M を実行した結果の V でその後の実行を続けた結果と、もとのコンテキスト 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。この補題は J に関する帰納法で証明できるが、Biernacki らの論文では後半の $\longrightarrow^*$ が $=$ になっている。これは単純なミスだと思われる。

最後に、PHOAS に起因する問題点について述べる。3 節と同様に関数の外延性の公理がここでも必要になる。さらに、Reflection の 4. の証明を行うにあたり、次の命題を公理として仮定する必要があった。

仮定 5 任意の DSKernel 言語の項 M_Δ について $M_\Delta[x/x] = M_\Delta$

これは、M_Δ の中の変数 x を x 自身で置き換えたら M_Δ になるという意味で、自明に成り立つはずである。ところが PHOAS を使っていると、上の命題は x を自由変数に持つ M_Δ について $\underline{\lambda}x.\, M_\Delta$ と x と M_Δ の間の 3 項間関係として記述される。通常、手でこの命題を証明するには最初の M_Δ についての場合分けを行う。ところが PHOAS を使っていると最初の M_Δ は $\underline{\lambda}x.\, M_\Delta$ のように λ 抽象の下に隠れている。$\underline{\lambda}x.\, M_\Delta$ は全体としては関数なので場合分けを行うことはできず、ここで証明が行き詰まってしまうことになる。

この問題は PHOAS を提案した Chlipala 自身も述べている [5]。PHOAS を使わずに、例えば de Bruijn index を使って定式化し直せばこのような問題は生じないが、そのためには変数名の付け替えに関する膨大な定式化を行わなくてはならなくなる。また、M_Δ が具体的に与えられれば、それに対して上の命題を Agda で示すことは簡単にできる。したがって、どのような入力プログラムが与えられたとしても、その後で上の命題が成り立つことを示せば、その入力プログラムが reflection の関係を満たしていることは保証される。これらのことを考えると、上の命題を公理として認めるのは妥当なことでないかと考えられる。

5 DS 項と CPS 項間の Reflection

3 節で示した DSKernel 項と CPS 項間の Reflection と 4 節で示した DS 項と DSKernel 項間の Reflection を合わせると、DS 項と CPS 項の間の Reflection の証明が完成する。

系 6 (DS 項と CPS 項の間の Reflection)

1. 任意の DS 項 M と CPS 言語の継続 K_Δ について $K_\Delta^{\flat\ominus}[M] \longrightarrow^* (M :: K_\Delta^\flat)^{\circ\sharp\oplus}$
2. 任意の CPS 項 N_Δ について $(N_\Delta^{\sharp\oplus} :: [\,]_\Delta)^\circ = N_\Delta$
3. 任意の DS 項 M_Δ, M'_Δ と CPS 言語の継続 K_Δ について $M_\Delta \longrightarrow M'_\Delta$ ならば $(M_\Delta :: K_\Delta^\flat)^\circ \longrightarrow^* (M'_\Delta :: K_\Delta^\flat)^\circ$
4. 任意の CPS 項 N_Δ, N'_Δ について $N_\Delta \longrightarrow N'_\Delta$ ならば $N_\Delta^{\sharp\oplus} \longrightarrow^* N_\Delta^{\prime\sharp\oplus}$

ここで $(M :: K_\Delta^\flat)^\circ$ が M の（K_Δ のもとでの）CPS 変換、$N_\Delta^{\sharp\oplus}$ が N_Δ の DS 変換である。それぞれ DSKernel 言語を経由したふたつの関数の合成になっている。

6 関連研究

λ 計算に対する reflection の証明は Sabry と Wadler [14] が行なっている。これを拡張して shift/reset の入った体系の reflection を示したのが Biernacki ら [3] である。本稿は Biernacki らの証明を型のある体系で行い、さらに Agda で定式化したものである。その過程で若干、不正確と思われるところを発見したが、概して Biernacki らの証明はとても正確であったということができる。これまでに reflection の証明を定理証明支援系で行なった研究は、筆者らの知るところでは存在しない。

Biernacki らは shift0/dollar に対する reflection も証明している [4]。shift0 に対する型システムは、Materzok と Biernacki [11] が部分型を許す体系を提示している。また、我々のグループでも

単相の型システムを提案している [10]。これらの型が入っても reflection が成り立つかどうかは未解決である。さらに叢ら [6] は代数的効果とハンドラ [13] に対する reflection を示そうと試みている。これも型のない体系を対象にしている。

これらの体系に型を入れて reflection を証明できるか、また定理証明支援系言語にのせることができるかは未知数である。shift/reset に対する型付きの reflection は、結果的には Biernacki らの証明に沿って、ほぼそのまま型を入れた形になった。しかし、そこに至る過程には本稿で示したような問題があり、すぐにうまくいく定義を見つけるのは容易ではなかった。型のない体系で証明できたからといって、すぐに型の入った体系で証明できるとは考えない方がよいと思われる。

7 まとめと今後の課題

本稿では、型のついた shift/reset 入り λ 計算の体系に対する reflection を示すとともに、定理証明支援系言語 Agda においてその証明を定式化した。型を入れるにあたっては、継続の変数 k を特別扱いすること、その型情報をカーネル言語にも入れることの 2 点が重要だった。前者は、PHOAS を使って Agda で定式化する際に必要だった Agda 特有の難しさであり、後者は、型付き言語で reflection を証明する上で本質的な点である。

今後は、まず shift を定数ではなく $\underline{\mathcal{S}}k.\,M$ の形の special form とした場合でも同様に成り立つかを確認したい。reflection の証明を手動で変更するのは大変だが、Agda で定式化しているためこういった変更を加えるのは容易になっていると期待される。より長期的には、shift0 の入った体系における型付き証明や、代数的効果とハンドラの定式化などが課題となる。

参考文献

[1] T. Altenkirch and B. Reus. Monadic presentations of lambda terms using generalized induc tive types. In *International Workshop on Computer Science Logic*, pp. 453–468. Springer, 1999.

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

[3] 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, 2020.

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

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

[6] Youyou Cong and Kenichi Asai. Towards a reflection for effect handlers. In *2023 ACM SIGPLAN International Workshop on Partial Evaluation and Program Manipulation (PEPM 2023)*, 2023. 11 pages, to appear.

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

[8] O. Danvy and A. Filinski. Abstracting Control. In *Proc. 1990 ACM Conference on Lisp and Functional Programming*, pp. 151–160, 1990.

[9] Cormac Flanagan, Amr Sabry, and Bruce F. Duba. The essence of compiling with continuations. In *Proceedings of the ACM SIGPLAN 1993 conference on Programming language design and implementation (PLDI 1993)*, pp. 237–247. ACM New York, NY, USA, 1993.

[10] 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 (GPCE 2022)*, pp. 45–58, 2022.

[11] Marek Materzok and Dariusz Biernacki. Subtyping Delimited Continuations. In *Proceedings of the 16th ACM SIGPLAN International Conference on Functional Programming*, ICFP '11, pp. 81–93, New York, NY, USA, 2011. ACM.

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

[13] Gordon Plotkin and Matija Pretnar. Handlers of algebraic effects. In Giuseppe Castagna, editor, *Proceedings of the 18th European Symposium on Programming Languages and Systems (ESOP '09)*, pp. 80–94, Berlin, Heidelberg, 2009. Springer-Verlag.

[14] 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.

付録

付録には、本文中に示した定理の正確な文言を示す。いずれも Agda での定式化に沿ったものとなっている。Order Isomorphism や reflection は 4 つの性質に分かれているので、それに合わせて節を分けている。

また、各定理には、言語名を使った一言の要約と定式化されているファイル名が付してある。さらに、どの言語の簡約・等号なのかも下付き文字で明示している。付録では 1 ステップの簡約と 0 ステップ以上の簡約は区別しておらず、どちらも $\longrightarrow$ と記している。本研究では、変換規則を型付きで Agda に載せているため、変換前に型がついたら変換後にも型がつくことが示されている。

A DSKernel 項と CPS 項の間の Order Isomorphism

まず、定理を示す前に、本文中では省略した DSKernel 項と CPS 項の間の（自明な）変換規則を図 10 と図 11 に示す。

$$
\begin{aligned}
\mathsf{int}^\sharp &= \mathsf{int} \\
(\tau_1 \to (\tau_2 \to \tau_3) \to \tau_4)^\sharp &= \tau_1^\sharp \to \tau_2^\sharp @ [\tau_3^\sharp, \tau_4^\sharp] \\
(\tau_2 \xrightarrow{k} \tau_3)^\flat &= \tau_2^\flat \xrightarrow{k} \tau_3^\flat \\
(\tau_2 \xrightarrow{\bullet} \tau_2)^\flat &= \tau_2^\flat \xrightarrow{\bullet} \tau_2^\flat \\
(K_\Delta \mathbin{\underline{@}} V)^\sharp &= K_\Delta^\flat [V^\natural] \\
(V \mathbin{\underline{@}} W \mathbin{\underline{@}} K_\Delta)^\sharp &= K_\Delta^\flat [V^\natural \mathbin{\underline{@}} W^\natural] \\
(K_\Delta \mathbin{\underline{@}} M_\bullet)^\sharp &= K_\Delta^\flat [\underline{\langle} M_\bullet^\sharp \underline{\rangle}] \\
x^\natural &= x \\
(\underline{\lambda} x.\, \underline{\lambda} k.\, M_k)^\natural &= \underline{\lambda} x.\, M_k^\sharp \\
(\underline{\lambda} w.\, \underline{\lambda} j.\, w \mathbin{\underline{@}} (\underline{\lambda} y.\, \underline{\lambda} k.\, k \mathbin{\underline{@}} (j \mathbin{\underline{@}} y)) \mathbin{\underline{@}} (\underline{\lambda} x.\, x))^\natural &= \underline{\mathcal{S}} \\
k_k^\flat &= [\,]_k \\
(\underline{\lambda} x.\, x)^\flat &= [\,]_\bullet \\
(\underline{\lambda} x.\, N_\Delta)^\flat &= \underline{\mathsf{let}}\ x = [\,]\ \underline{\mathsf{in}}\ N_\Delta^\sharp
\end{aligned}
$$

図 10. $\lambda_{c\mathcal{S}}^*$ の DS 変換

$$
\begin{array}{rcl}
\mathsf{int}^{\circ} & = & \mathsf{int} \\
(\tau_1 \to \tau_2 @ [\tau_3, \tau_4])^{\circ} & = & \tau_1^{\circ} \to (\tau_2^{\circ} \to \tau_3^{\circ}) \to \tau_4^{\circ} \\
(\tau_2 \xrightarrow{k} \tau_3)^{\ddagger} & = & \tau_2^{\ddagger} \xrightarrow{k} \tau_3^{\ddagger} \\
(\tau_2 \xrightarrow{\bullet} \tau_2)^{\ddagger} & = & \tau_2^{\ddagger} \xrightarrow{\bullet} \tau_2^{\ddagger} \\
(K_\Delta[V])^{\circ} & = & K_\Delta^{\ddagger} \mathbin{\underline{@}} V^{\dagger'} \\
(K_\Delta[V \mathbin{\underline{@}} W])^{\circ} & = & V^{\dagger'} \mathbin{\underline{@}} W^{\dagger'} \mathbin{\underline{@}} K_\Delta^{\ddagger} \\
(K_\Delta[\underline{\langle} M_\bullet \underline{\rangle}])^{\circ} & = & K_\Delta^{\ddagger} \mathbin{\underline{@}} M_\bullet^{\circ} \\
x^{\dagger'} & = & x \\
(\underline{\lambda} x.\, M_k)^{\dagger'} & = & \underline{\lambda} x.\, \underline{\lambda} k.\, M_k^{\circ} \\
\underline{\mathcal{S}}^{\dagger'} & = & \underline{\lambda} w.\, \underline{\lambda} j.\, w \mathbin{\underline{@}} (\underline{\lambda} y.\, \underline{\lambda} k.\, k \mathbin{\underline{@}} (j \mathbin{\underline{@}} y)) \mathbin{\underline{@}} (\underline{\lambda} x.\, x) \\
[\,]_k^{\ddagger} & = & k_k \\
[\,]_\bullet^{\ddagger} & = & (\underline{\lambda} x.\, x) \\
(\underline{\mathsf{let}}\ x = [\,]\ \underline{\mathsf{in}}\ N_\Delta)^{\ddagger} & = & \underline{\lambda} x.\, N_\Delta^{\circ}
\end{array}
$$

図 11. $\lambda^{\triangleright}_{c\mathcal{S}}$ の CPS 変換

A.1 Order Isomorphism (1)

Order Isomorphism の (1) は DSKernel 項を CPS 変換して DS 変換をすると、元の項と等しくなることを示す。DSKernel 項と CPS 項間では、$M \longrightarrow M^{*\#}$ ではなく $M \equiv M^{*\#}$ を示すことができる。

定理 7 ($\lambda^{\triangleright}_{c\mathcal{S}} =_{\lambda^{\triangleright}_{c\mathcal{S}}} (\lambda^{\triangleright}_{c\mathcal{S}} \overset{\mathbf{cps}}{\Rightarrow} \lambda^{*}_{c\mathcal{S}} \overset{\mathbf{ds}}{\Rightarrow} \lambda^{\triangleright}_{c\mathcal{S}})$**,** `Reflect1b.agda`**)**

1. 任意の $\lambda^{\triangleright}_{c\mathcal{S}}$ の値 V について $V =_{\lambda^{\triangleright}_{c\mathcal{S}}} V^{\dagger'\natural}$
2. 任意の $\lambda^{\triangleright}_{c\mathcal{S}}$ の項 M_Δ について $M_\Delta =_{\lambda^{\triangleright}_{c\mathcal{S}}} M_\Delta^{\circ\sharp}$
3. 任意の $\lambda^{\triangleright}_{c\mathcal{S}}$ の継続 K_Δ について $K_\Delta =_{\lambda^{\triangleright}_{c\mathcal{S}}} K_\Delta^{\ddagger\flat}$

証明 V, M_Δ, K_Δ の相互帰納法。 □

A.2 Order Isomorphism (2)

CPS 項を DSKernel 項へと DS 変換してから CPS 変換をすると元の項と等しくなることを示す。

定理 8 ($(\lambda^{*}_{c\mathcal{S}} \overset{\mathbf{ds}}{\Rightarrow} \lambda^{\triangleright}_{c\mathcal{S}} \overset{\mathbf{cps}}{\Rightarrow} \lambda^{*}_{c\mathcal{S}}) =_{\lambda^{*}_{c\mathcal{S}}} \lambda^{*}_{c\mathcal{S}}$**,** `Reflect2a.agda`**)**

1. 任意の $\lambda^{*}_{c\mathcal{S}}$ の値 V について $V^{\flat\dagger'} =_{\lambda^{*}_{c\mathcal{S}}} V$
2. 任意の $\lambda^{*}_{c\mathcal{S}}$ の項 M_Δ について $M_\Delta^{\sharp\circ} =_{\lambda^{*}_{c\mathcal{S}}} M_\Delta$
3. 任意の $\lambda^{*}_{c\mathcal{S}}$ の継続 K_Δ について $K_\Delta^{\flat\ddagger} =_{\lambda^{*}_{c\mathcal{S}}} K_\Delta$

証明 V, M_Δ, K_Δ の相互帰納法。 □

A.3 Order Isomorphism (3)

任意の DSKernel 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を示す。これを証明するために、まず代入補題を示す。

補題 9 (値の代入補題, `Reflect3b.agda`**)**

1. 任意の $\lambda^{\triangleright}_{c\mathcal{S}}$ の値 W, V について $(W[V/x])^{\dagger'} = W^{\dagger'}[V^{\dagger'}/x]$
2. 任意の $\lambda^{\triangleright}_{c\mathcal{S}}$ の項 M、値 V について $(M[V/x])^{\circ} = M^{\circ}[V^{\dagger'}/x]$
3. 任意の $\lambda^{\triangleright}_{c\mathcal{S}}$ の継続 K_Δ、値 V について $(K_\Delta[V/x])^{\ddagger} = K^{\ddagger}_\Delta[V^{\dagger'}/x]$

証明 $W[V/x]$, $M[V/x]$, $K_\Delta[V/x]$ の相互帰納法。 □

継続の代入規則では、値は変化しない（λ 抽象に出てくる k が代入したい k を隠すため）ので、項とコンテキストだけの間の相互再帰になる。

補題 10 (継続の代入補題, `Reflect3b.agda`**)**
1. 任意の $\lambda^{\triangleright}_{c\mathcal{S}}$ の項 M_k と継続 J_Δ について $(M_k[J_\Delta/k])^{\circ} = M^{\circ}_k[J^{\ddagger}_\Delta/k]$
2. 任意の $\lambda^{\triangleright}_{c\mathcal{S}}$ の継続 K_k, J_Δ について $(K_k[J_\Delta/k])^{\ddagger} = K^{\ddagger}_k[J^{\ddagger}_\Delta/k]$

証明 $M_k[J_\Delta/k]$, $K_k[J_\Delta/k]$ の相互再帰。 □

定理 11 ($\longrightarrow_{\lambda^{\triangleright}_{c\mathcal{S}}} \overset{\mathbf{cps}}{\Rightarrow} \longrightarrow_{\lambda^{*}_{c\mathcal{S}}}$**,** `Reflect3b.agda`**)**
1. 任意の $\lambda^{\triangleright}_{c\mathcal{S}}$ の値 V, W について $V \longrightarrow_{\lambda^{\triangleright}_{c\mathcal{S}}} W$ ならば $V^{\dagger'} \longrightarrow_{\lambda^{*}_{c\mathcal{S}}} W^{\dagger'}$
2. 任意の $\lambda^{\triangleright}_{c\mathcal{S}}$ の項 M, N について $M \longrightarrow_{\lambda^{\triangleright}_{c\mathcal{S}}} N$ ならば $M^{\circ} \longrightarrow_{\lambda^{*}_{c\mathcal{S}}} N^{\circ}$
3. 任意の $\lambda^{\triangleright}_{c\mathcal{S}}$ の継続 J_Δ, K_Δ について $J_\Delta \longrightarrow_{\lambda^{\triangleright}_{c\mathcal{S}}} K_\Delta$ ならば $J^{\ddagger}_\Delta \longrightarrow_{\lambda^{*}_{c\mathcal{S}}} K^{\ddagger}_\Delta$

証明 簡約 $V \longrightarrow_{\lambda^{\triangleright}_{c\mathcal{S}}} W$ と $M \longrightarrow_{\lambda^{\triangleright}_{c\mathcal{S}}} N$ と $J_\Delta \longrightarrow_{\lambda^{\triangleright}_{c\mathcal{S}}} K_\Delta$ の相互帰納法。 □

A.4 Order Isomorphism (4)

任意の CPS 項とそれを簡約した結果をそれぞれ DSKernel 項に変換すると、変換後の 2 つの項も簡約関係にあることを示す。これを証明するために、ここでもまず代入補題を示す。

補題 12 (値の代入補題, `Reflect4a.agda`**)**
1. 任意の $\lambda^{*}_{c\mathcal{S}}$ の値 W, V について $(W[V/x])^{\natural} = W^{\natural}[V^{\natural}/x]$
2. 任意の $\lambda^{*}_{c\mathcal{S}}$ の項 M、値 V について $(M[V/x])^{\sharp} = M^{\sharp}[V^{\natural}/x]$
3. 任意の $\lambda^{*}_{c\mathcal{S}}$ の継続 K_Δ、値 V について $(K_\Delta[V/x])^{\flat} = K^{\flat}_\Delta[V^{\natural}/x]$

証明 $W[V/x]$, $M[V/x]$, $K_\Delta[V/x]$ の相互帰納法。 □

補題 13 (継続の代入補題, `Reflect4a.agda`**)**
1. 任意の $\lambda^{*}_{c\mathcal{S}}$ の項 M_k と継続 J_Δ について $(M_k[J_\Delta/k])^{\sharp} = M^{\sharp}_k[J^{\flat}_\Delta/k]$
2. 任意の $\lambda^{*}_{c\mathcal{S}}$ の継続 K_k, J_Δ について $(K_k[J_\Delta/k])^{\flat} = K^{\flat}_k[J^{\flat}_\Delta/k]$

証明 $M_k[J_\Delta/k]$, $K_k[J_\Delta/k]$ の相互再帰。 □

定理 14 ($\longrightarrow_{\lambda^{*}_{c\mathcal{S}}} \overset{\mathbf{ds}}{\Rightarrow} \longrightarrow_{\lambda^{\triangleright}_{c\mathcal{S}}}$**,** `Reflect4a.agda`**)**
1. 任意の $\lambda^{*}_{c\mathcal{S}}$ の値 V, W について $V \longrightarrow_{\lambda^{*}_{c\mathcal{S}}} W$ ならば $V^{\natural} \longrightarrow_{\lambda^{\triangleright}_{c\mathcal{S}}} W^{\natural}$
2. 任意の $\lambda^{*}_{c\mathcal{S}}$ の項 M_Δ, N_Δ について $M_\Delta \longrightarrow_{\lambda^{*}_{c\mathcal{S}}} N_\Delta$ ならば $M^{\sharp}_\Delta \longrightarrow_{\lambda^{\triangleright}_{c\mathcal{S}}} N^{\sharp}_\Delta$
3. 任意の $\lambda^{*}_{c\mathcal{S}}$ の継続 J_Δ, K_Δ について $J_\Delta \longrightarrow_{\lambda^{*}_{c\mathcal{S}}} K_\Delta$ ならば $J^{\flat}_\Delta \longrightarrow_{\lambda^{\triangleright}_{c\mathcal{S}}} K^{\flat}_\Delta$

証明 簡約 $V \longrightarrow_{\lambda^{*}_{c\mathcal{S}}} W$ と $M_\Delta \longrightarrow_{\lambda^{*}_{c\mathcal{S}}} N_\Delta$ と $J_\Delta \longrightarrow_{\lambda^{*}_{c\mathcal{S}}} K_\Delta$ の相互帰納法。 □

B DS 項と DSKernel 項間の Reflection

B.1 Reflection(1)

DS 項を A-正規形変換して DS 変換をすると元の項と等しくなることを示す。

補題 15 (Generalized associativity, `Reflect1a.agda`**)** 任意の $\lambda_{c\mathcal{S}}$ の項 e_1, e_2 と任意の $\lambda^{\triangleright}_{c\mathcal{S}}$ の継続 K_Δ について $K^{\ominus}_\Delta[\underline{\mathsf{let}}\ x = e_1\ \underline{\mathsf{in}}\ e_2] \longrightarrow_{\lambda_{c\mathcal{S}}} \underline{\mathsf{let}}\ x = e_1\ \underline{\mathsf{in}}\ K^{\ominus}_\Delta[e_2]$

証明 K_Δ に関する場合分け。 □

この補題は Biernacki ら [3] が示したものと同一である。この補題に出てくる $K^{\ominus}_\Delta$ は任意の $\lambda_{c\mathcal{S}}$ のコンテキストでは成り立たず、$\lambda^{\triangleright}_{c\mathcal{S}}$ のコンテキスト K_Δ を戻した $K^{\ominus}_\Delta$ の形をしていなくてはならない。

定理 16 ($\lambda_{c\mathcal{S}} \longrightarrow_{\lambda_{c\mathcal{S}}} (\lambda_{c\mathcal{S}} \overset{\mathbf{anormal}}{\Rightarrow} \lambda^{\triangleright}_{c\mathcal{S}} \overset{\mathbf{embed}}{\Rightarrow} \lambda_{c\mathcal{S}})$**,** `Reflect1a.agda`**)**

1. 任意の $\lambda_{c\mathcal{S}}$ の値 V について $V \longrightarrow_{\lambda_{c\mathcal{S}}} V^{\dagger'\odot}$
2. 任意の $\lambda_{c\mathcal{S}}$ の項 M と $\lambda^{\triangleright}_{c\mathcal{S}}$ の継続 K_Δ について $K^{\ominus}_\Delta[M] \longrightarrow_{\lambda_{c\mathcal{S}}} (M :: K_\Delta)^{\oplus}$

証明 V, M に関する相互帰納法。 □

B.2 Reflection(2)

DSKernel 項を DS 項へと変換してから A-正規形変換をすると元の項と等しくなることを示す。

定理 17 ($(\lambda^{\triangleright}_{c\mathcal{S}} \overset{\mathbf{embed}}{\Rightarrow} \lambda_{c\mathcal{S}} \overset{\mathbf{anormal}}{\Rightarrow} \lambda^{\triangleright}_{c\mathcal{S}}) =_{\lambda^{\triangleright}_{c\mathcal{S}}} \lambda^{\triangleright}_{c\mathcal{S}}$**,** `Reflect2b.agda`**)**

1. 任意の $\lambda^{\triangleright}_{c\mathcal{S}}$ の値 V について $V^{\odot\dagger\dagger} =_{\lambda^{\triangleright}_{c\mathcal{S}}} V$
2. 任意の $\lambda^{\triangleright}_{c\mathcal{S}}$ の項 M_Δ について $M^{\oplus}_\Delta : [\,]_\Delta =_{\lambda^{\triangleright}_{c\mathcal{S}}} M_\Delta$
3. 任意の $\lambda^{\triangleright}_{c\mathcal{S}}$ の継続 K_Δ と $\lambda_{c\mathcal{S}}$ の項 M について $(K^{\ominus}_\Delta[M])^{\oplus} : [\,]_\Delta =_{\lambda^{\triangleright}_{c\mathcal{S}}} M : K_\Delta$

証明 V, M_Δ, K_Δ の相互再帰。 □

B.3 Reflection(3)

任意の DS 項とそれを簡約した結果をそれぞれ A-正規形変換すると、変換後の 2 つの項も簡約関係にあることを示す。これを証明するために、まず代入補題を示す。

補題 18 (値の代入補題, `Reflect3a.agda`**)**

1. 任意の $\lambda_{c\mathcal{S}}$ の値 W, V について $(W[V/x])^{\dagger\dagger} = W^{\dagger\dagger}[V^{\dagger\dagger}/x]$
2. 任意の $\lambda_{c\mathcal{S}}$ の項 M、値 V と $\lambda^{\triangleright}_{c\mathcal{S}}$ の継続 K_Δ について
   $M[V/x] : K_\Delta[V^{\dagger\dagger}/x] = (M : K_\Delta)[V^{\dagger\dagger}/x]$

証明 $W[V/x], M[V/x]$ の相互帰納法。 □

補題 19 (継続の代入補題, `Reflect3a.agda`**)**

任意の $\lambda_{c\mathcal{S}}$ の項 M と $\lambda^{\triangleright}_{c\mathcal{S}}$ の継続 K_k, J_Δ について $M :: (K_k[J_\Delta/k]) = (M :: K_k)[J_\Delta/k]$

証明 M に関する帰納法。 □

補題 20 (`Reflect3a.agda`) 任意の $\lambda_{c\mathcal{S}}$ の項 M と $\lambda^{\triangleright}_{c\mathcal{S}}$ の継続 K_Δ, K'_Δ について $K_\Delta \longrightarrow_{\lambda^{\triangleright}_{c\mathcal{S}}} K'_\Delta$ ならば $M :: K_\Delta \longrightarrow_{\lambda^{\triangleright}_{c\mathcal{S}}} M :: K'_\Delta$

証明 M に関する帰納法。 □

補題 21 (`Reflect3a.agda`) 任意の $\lambda_{c\mathcal{S}}$ のコンテキスト J と $\lambda^{\triangleright}_{c\mathcal{S}}$ の継続 K_Δ について、以下のふたつを満たすような $\lambda^{\triangleright}_{c\mathcal{S}}$ の継続 $\hat{J}_\Delta$ が存在する。

- 任意の $\lambda_{c\mathcal{S}}$ の値でない項 M について $J[M] :: K_\Delta = M :: \hat{J}_\Delta$
- 任意の $\lambda_{c\mathcal{S}}$ の値 V について $V :: \hat{J}_\Delta \longrightarrow_{\lambda^{\triangleright}_{c\mathcal{S}}} J[V] :: K_\Delta$

証明 J に関する帰納法。 □

定理 22 ($\longrightarrow_{\lambda_{c\mathcal{S}}} \overset{\mathbf{anormal}}{\Rightarrow} \longrightarrow_{\lambda^{\triangleright}_{c\mathcal{S}}}$, `Reflect3a.agda`)

1. 任意の $\lambda_{c\mathcal{S}}$ の値 V, W について $V \longrightarrow_{\lambda_{c\mathcal{S}}} W$ ならば $V^{\dagger\dagger} \longrightarrow_{\lambda^{\triangleright}_{c\mathcal{S}}} W^{\dagger\dagger}$
2. 任意の $\lambda_{c\mathcal{S}}$ の項 M, N と $\lambda^{\triangleright}_{c\mathcal{S}}$ の継続 K_Δ について $M \longrightarrow_{\lambda_{c\mathcal{S}}} N$ ならば $M :: K_\Delta \longrightarrow_{\lambda^{\triangleright}_{c\mathcal{S}}} N :: K_\Delta$

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

B.4 Reflection(4)

任意の DSKernel 項とそれを簡約した結果をそれぞれ DS 項に変換すると、変換後の 2 つの項も簡約関係にあることを示す。これを証明するために、まず代入補題を示す。

仮定 23 (`Reflect4b.agda`) 任意の $\lambda^{\triangleright}_{c\mathcal{S}}$ の項 M_Δ について $M_\Delta[x/x] = M_\Delta$

補題 24 (`DSterm.agda`) 任意の $\lambda_{c\mathcal{S}}$ の項 M、値 V、コンテキスト K_Δ について $(K_\Delta[M])[V/x] = (K_\Delta[V/x])[M[V/x]]$

証明 $K_\Delta[V/x]$ に関する帰納法。 □

補題 25 (値の代入補題, `Reflect4b.agda`)

1. 任意の $\lambda^{\triangleright}_{c\mathcal{S}}$ の値 W, V について $(W[V/x])^{\odot} = W^{\odot}[V^{\odot}/x]$
2. 任意の $\lambda^{\triangleright}_{c\mathcal{S}}$ の項 M、値 V について $(M[V/x])^{\oplus} = M^{\oplus}[V^{\odot}/x]$
3. 任意の $\lambda^{\triangleright}_{c\mathcal{S}}$ の継続 K_Δ、値 V について $(K_\Delta[V/x])^{\ominus} = K^{\ominus}_\Delta[V^{\odot}/x]$

証明 $W[V/x]$, $M[V/x]$, $K_\Delta[V/x]$ の相互帰納法。 □

補題 26 (`Reflect4b.agda`)

任意の $\lambda^{\triangleright}_{c\mathcal{S}}$ の項 M_k、継続 K_Δ について $K^{\ominus}_\Delta[M^{\oplus}_k] \longrightarrow_{\lambda_{c\mathcal{S}}} (M_k[K_\Delta/k])^{\oplus}$

証明 K_Δ についての帰納法をかけた上で、$M_k[K_\Delta/k]$ についての帰納法。 □

定理 27 ($\longrightarrow_{\lambda^{\triangleright}_{c\mathcal{S}}} \overset{\mathbf{embed}}{\Rightarrow} \longrightarrow_{\lambda_{c\mathcal{S}}}$, `Reflect4b.agda`)

1. 任意の $\lambda^{\triangleright}_{c\mathcal{S}}$ の値 V, W について $V \longrightarrow_{\lambda^{\triangleright}_{c\mathcal{S}}} W$ ならば $V^{\odot} \longrightarrow_{\lambda_{c\mathcal{S}}} W^{\odot}$
2. 任意の $\lambda^{\triangleright}_{c\mathcal{S}}$ の項 M_Δ, N_Δ について $M_\Delta \longrightarrow_{\lambda^{\triangleright}_{c\mathcal{S}}} N_\Delta$ ならば $M^{\oplus}_\Delta \longrightarrow_{\lambda_{c\mathcal{S}}} N^{\oplus}_\Delta$
3. 任意の $\lambda^{\triangleright}_{c\mathcal{S}}$ の継続 J_Δ, K_Δ について $J_\Delta \longrightarrow_{\lambda^{\triangleright}_{c\mathcal{S}}} K_\Delta$ ならば任意の $\lambda_{c\mathcal{S}}$ の項 M について $J^{\ominus}_\Delta[M] \longrightarrow_{\lambda_{c\mathcal{S}}} K^{\ominus}_\Delta[M]$

証明 簡約 $V \longrightarrow_{\lambda^{\triangleright}_{c\mathcal{S}}} W$ と $M_\Delta \longrightarrow_{\lambda^{\triangleright}_{c\mathcal{S}}} N_\Delta$ と $J_\Delta \longrightarrow_{\lambda^{\triangleright}_{c\mathcal{S}}} K_\Delta$ の相互帰納法。 □

C　DS 項と CPS 項間の Reflection

DS 項と DSKernel 項間の Reflection と DSKernel 項と CPS 項間の Reflection を合わせると Reflection の証明が完成する。

系 28 ($\lambda_{c\mathcal{S}} \longrightarrow_{\lambda_{c\mathcal{S}}} (\lambda_{c\mathcal{S}} \overset{\mathbf{anormal}}{\Rightarrow} \lambda^{\triangleright}_{c\mathcal{S}} \overset{\mathbf{cps}}{\Rightarrow} \lambda^{*}_{c\mathcal{S}} \overset{\mathbf{ds}}{\Rightarrow} \lambda^{\triangleright}_{c\mathcal{S}} \overset{\mathbf{embed}}{\Rightarrow} \lambda_{c\mathcal{S}})$, `Reflect1.agda`)

1. 任意の $\lambda_{c\mathcal{S}}$ の値 V について $V \longrightarrow_{\lambda_{c\mathcal{S}}} V^{\dagger\dagger'\natural\odot}$
2. 任意の $\lambda_{c\mathcal{S}}$ の項 M と $\lambda^{*}_{c\mathcal{S}}$ の継続 K_Δ について $K^{\flat\ominus}_\Delta[M] \longrightarrow_{\lambda_{c\mathcal{S}}} (M :: K^{\flat}_\Delta)^{\circ\sharp\oplus}$

証明　定理 16 と定理 7 から。　□

系 29 ($(\lambda^{*}_{c\mathcal{S}} \overset{\mathbf{ds}}{\Rightarrow} \lambda^{\triangleright}_{c\mathcal{S}} \overset{\mathbf{embed}}{\Rightarrow} \lambda_{c\mathcal{S}} \overset{\mathbf{anormal}}{\Rightarrow} \lambda^{\triangleright}_{c\mathcal{S}} \overset{\mathbf{cps}}{\Rightarrow} \lambda^{*}_{c\mathcal{S}}) =_{\lambda^{*}_{c\mathcal{S}}} \lambda^{*}_{c\mathcal{S}}$, `Reflect2.agda`)

1. 任意の $\lambda^{*}_{c\mathcal{S}}$ の値 V について $V^{\natural\odot\dagger\dagger'} =_{\lambda^{*}_{c\mathcal{S}}} V$
2. 任意の $\lambda^{*}_{c\mathcal{S}}$ の項 M_Δ について $(M^{\sharp\oplus}_\Delta :: [\,]_\Delta)^{\circ} =_{\lambda^{*}_{c\mathcal{S}}} M_\Delta$
3. 任意の $\lambda^{*}_{c\mathcal{S}}$ の継続 K_Δ と $\lambda_{c\mathcal{S}}$ の項 M について $(K^{\flat\ominus}_\Delta[M] : [\,]_\Delta)^{\circ} =_{\lambda^{*}_{c\mathcal{S}}} M : K_\Delta$

証明　定理 8 と定理 17 から。　□

系 30 ($\longrightarrow_{\lambda_{c\mathcal{S}}} \overset{\mathbf{cps}\circ\mathbf{anormal}}{\Rightarrow} \longrightarrow_{\lambda^{*}_{c\mathcal{S}}}$, `Reflect3.agda`)

1. 任意の $\lambda_{c\mathcal{S}}$ の値 V, W について $V \longrightarrow_{\lambda_{c\mathcal{S}}} W$ ならば $V^{\dagger\dagger'} \longrightarrow_{\lambda^{*}_{c\mathcal{S}}} W^{\dagger\dagger'}$
2. 任意の $\lambda_{c\mathcal{S}}$ の項 M_Δ, N_Δ と $\lambda^{*}_{c\mathcal{S}}$ の継続 K_Δ について $M_\Delta \longrightarrow_{\lambda_{c\mathcal{S}}} N_\Delta$ ならば $(M_\Delta :: K^{\flat}_\Delta)^{\circ} \longrightarrow_{\lambda^{*}_{c\mathcal{S}}} (N_\Delta :: K^{\flat}_\Delta)^{\circ}$

証明　定理 22 と定理 11 から。　□

系 31 ($\longrightarrow_{\lambda^{*}_{c\mathcal{S}}} \overset{\mathbf{embed}\circ\mathbf{ds}}{\Rightarrow} \longrightarrow_{\lambda_{c\mathcal{S}}}$, `Reflect4.agda`)

1. 任意の $\lambda^{*}_{c\mathcal{S}}$ の値 V, W について $V \longrightarrow_{\lambda^{*}_{c\mathcal{S}}} W$ ならば $V^{\natural\odot} \longrightarrow_{\lambda_{c\mathcal{S}}} W^{\natural\odot}$
2. 任意の $\lambda^{*}_{c\mathcal{S}}$ の項 M_Δ, N_Δ について $M_\Delta \longrightarrow_{\lambda^{*}_{c\mathcal{S}}} N_\Delta$ ならば $M^{\sharp\oplus}_\Delta \longrightarrow_{\lambda_{c\mathcal{S}}} N^{\sharp\oplus}_\Delta$
3. 任意の $\lambda^{*}_{c\mathcal{S}}$ の継続 J_Δ, K_Δ について $J_\Delta \longrightarrow_{\lambda^{*}_{c\mathcal{S}}} K_\Delta$ ならば任意の $\lambda_{c\mathcal{S}}$ の項 M について $J^{\flat\ominus}_\Delta[M] \longrightarrow_{\lambda_{c\mathcal{S}}} K^{\flat\ominus}_\Delta[M]$

証明　定理 14 と定理 27 から。　□