型付きの shift/reset を含む言語における Reflection にむけて

理学専攻　情報科学コース　2040659　山本 充子　(指導教員：浅井 健一)

1　はじめに

CPS とは、継続 (ある時点における残りの計算) を明示的に表すようにした表現形式のことである。また、CPS 変換前の表現形式を DS と呼ぶ。CPS のプログラムでは、全て tail-call(末尾呼び出し) になっているのに対して、DS のプログラムでは、関数呼び出しが末尾に限定されない。

CPS のプログラムに変換する CPS 変換は、コンパイラなどで使われおり [1]、コンパイラの正しさを保証するためには CPS 変換の正当性を言う必要がある。正当性として健全性と完全性を示す方法の一つが Galois Connection を示すことである。本研究は、`shift/reset` を含む言語に対して Galois Connection を示すとともに、Galois Connection の特別なケースである Reflection に関する証明を行なったものである。

プログラミング言語において型を導入することは、プログラムの安全性を保証できる。本研究では、型を導入することで、DS と CPS の世界で型がどのように対応していくのかをみていき、型がついた世界でも Reflection の証明ができるのかをみた。

型が付いた体系で証明を行おうとすると、複雑になり手で証明すると誤りが発生することがある。そこで、定理証明系の Agda を用いて型のついた形で定式化をして証明を行なった。本研究で示す証明は、DS のプログラムを let で正規化した体系の DSkernel 項と CPS 項の間の Reflection である。DSkernel の体系と CPS の体系の型システムと証明が完成しており、これらは DS と CPS の間における Reflection を示す第一歩となる。

2　Galois Connection と Reflection

これまで、CPS 変換の正当性の証明は様々に行われてきた。Sabry と Wadler は、Galois Connection を用いて call-by-value の λ 計算に対する CPS 変換の健全性と完全性を示すことで、CPS 変換の正当性を証明した [4]。彼らは、CPS 変換が Reflection であることも示した。Biernacki、Pyzik と Sieczkowski は、Sabry と Wadler の λ 式に `shift/reset`[3] を加えて拡張し、その λ 式においても Reflection を証明ができることを示した [2]。

Galois Connection の定義を Sabry と Wadler の論文より引用する [4]。以下の式における、$\to$ は、one step 簡約を意味し、$\twoheadrightarrow$ は、one step 以上の簡約を表す。また、$\lambda_c^{\mathcal{S}}$ は、DS 項を表し、 λ_{cps} は、CPS 項を表す。$\twoheadrightarrow_{\lambda_c^{\mathcal{S}}}$ は DS における簡約、$\twoheadrightarrow_{\lambda_{cps}}$ は CPS 項における簡約を表す。写像 $*: \lambda_c^{\mathcal{S}} \to \lambda_{cps}$ と $\#: \lambda_{cps} \to \lambda_c^{\mathcal{S}}$ はそれぞれ CPS 変換と DS 変換を表す。

定義 1 以下が成り立つとき、写像 $*$ と $\#$ は $\lambda_c^{\mathcal{S}}$ から λ_{cps} への *Galois Connection* であるという。$(M, M' \in \lambda_c^{\mathcal{S}},\ N, N' \in \lambda_{cps})$

$$M \twoheadrightarrow_{\lambda_c^{\mathcal{S}}} N^{\#} \textit{ if and only if } M^* \twoheadrightarrow_{\lambda_{cps}} N$$

定義 1 は以下のように書き換えることができる。

命題 2 以下の 4 つの条件が成り立つとき、写像 $*$ と $\#$ は $\lambda_c^{\mathcal{S}}$ から λ_{cps} への *Galois Connection* であるという。$(M, M' \in \lambda_c^{\mathcal{S}},\ N, N' \in \lambda_{cps})$

(1)　$M \twoheadrightarrow M^{*\#}$
(2)　$N^{\#*} \twoheadrightarrow N$
(3)　$M \twoheadrightarrow M'$ ならば $M^* \twoheadrightarrow M'^*$
(4)　$N \twoheadrightarrow N'$ ならば $N^{\#} \twoheadrightarrow N'^{\#}$

定義 3 写像 $*$ と $\#$ が *Galois Connection* であり、$N \equiv N^{\#*}$ のとき、$*$ と $\#$ は $\lambda_c^{\mathcal{S}}$ と λ_{cps} において、*Reflection* であるという。$(N \in \lambda_{cps})$

DS 変換 $\#$ をした結果は、すべての DS 項 $\lambda_c^{\mathcal{S}}$ に戻るとは限らない。よって、DS 変換して得られた項を DSkernel $\lambda_{c**}^{\mathcal{S}}$ と定義する。DSkernel の項は let で正規化された式になっている。Reflection が示されたとき、CPS 項 λ_{cps} と DSkernel 項 $\lambda_{c**}^{\mathcal{S}}$ は全てそれぞれ 1 対 1 対応しており、DSkernel 項 $\lambda_{c**}^{\mathcal{S}}$ から CPS 項 λ_{cps} への同型写像が存在する。本研究は、DSkernel と CPS の間で Reflection を証明したものである。

3　CPS 項

型	τ	::=	$\mathsf{Nat} \mid \mathsf{Bool} \mid \tau_2 \to [\tau_1 \to \tau_3] \to \tau_4$
項$_\Delta$	M, N	::=	$K_\Delta V \mid VWK_\Delta \mid K_\Delta M_\bullet$
値	V, W	::=	$n \mid x \mid \lambda xk.\, M_k \mid S$
`shift`	S	::=	$\lambda wj.\, w(\lambda yk.\, k(jy))(\lambda x.\, x)$
継続$_\Delta$	K	::=	${}_{(\Delta=k)}k \mid {}_{(\Delta=\bullet)}\lambda x.\, x \mid \lambda x.\, M_\Delta$

図 1: CPS 項

本研究は、DSkernel 項と CPS 項の間に Reflection を示すので、DS 項と CPS 変換の定義は省略する。図 1 を見ると、項と継続には Δ がついている。左辺に現れる Δ と右辺に現れる Δ には同じものが入り、Δ には変数 k または $\bullet$ のどちらかが入る。Δ をつけることで、継続が tail-recursive か returning call かを区別することができる。

4　DS 変換

$x^\natural$	=	x
$(\lambda xk.\, M_k)^\natural$	=	$\lambda x.\, \mathcal{S}(\lambda k.\, \langle M_k^\sharp \rangle)$
$(\lambda wj.\, w(\lambda yk.\, k(jy))(\lambda x.\, x))^\natural$	=	$\mathcal{S}$
$k^\flat$	=	$k[\ ]$
$(\lambda x.\, x)^\flat$	=	$[\ ]$
$(\lambda x.\, N_\Delta)^\flat$	=	$\mathsf{let}\ x = [\ ]\ \mathsf{in}\ N_\Delta^\sharp$
$(K_\Delta V)^\sharp$	=	$K_\Delta^\flat[V^\natural]$
$(VWK_\Delta)^\sharp$	=	$K_\Delta^\flat[V^\natural W^\natural]$
$(K_\Delta M_\bullet)^\sharp$	=	$K_\Delta^\flat[\langle M_\bullet^\sharp \rangle]$

図 2: DS 変換

$V^\natural$ は値の変換、$K^\flat$ は継続の変換、$M^\sharp$ は項の変換を表す。Biernacki らの定義では、関数抽象の変換において $(\lambda x k.\, M_k)^\natural = \lambda x.\, M_k^\sharp$ とし、継続 k を削除した。彼らの定義では、この k が後の計算でいずれ消えることが暗黙のうちにあるため、このような定義にした。しかし、これを Agda で表現しようとすると $M_k^\sharp$ 内の自由変数 k を削除するのは難しい。よって、k を明示するために図 2 のように定義する必要があった。

5 DSkernel 項

型	τ	::=	$\mathsf{Nat} \mid \mathsf{Bool} \mid \tau_1 \to \tau_2 @ \mathrm{cps}[\tau_3, \tau_4]$
項$_\Delta$	M, N	::=	$K_\Delta[V] \mid K_\Delta[P]$
値	V, W	::=	$n \mid x \mid \lambda x.\mathcal{S}(\lambda k.\langle M_k\rangle) \mid \mathcal{S}$
値でない	P, Q	::=	$VW \mid \langle M_\bullet \rangle$
コンテキスト$_\Delta$	K	::=	$_{(\Delta=k)}k[\ ] \mid {}_{(\Delta=\bullet)}[\ ] \mid \mathsf{let}\ x = [\ ]\ \mathsf{in}\ M_\Delta$

図 3: DSkernel 項

図 3 で示されているのは、DS 変換後の項、すなわち DSkernel 項の定義を表す。ここでは、`shift` を値として扱っている。通常、`shift` は、$\mathcal{S}k.\, e$ のように項として表すが、$\mathcal{S}k.\, e$ は $\mathcal{S}(\lambda k.\, e)$ と同義である。また、値の関数抽象で $\lambda x.\mathcal{S}(\lambda k.\langle M_k\rangle)$ となっているのは、4 節の DS 変換で扱った通りである。

6 CPS 同型写像

CPS 項と DS 項は 1 対 1 対応しているため、CPS 同型写像は DS 変換をひっくり返した定義となる。値の変換は $V^\dagger$、コンテキストの変換は $K^\ddagger$、項の変換は M° と表す。

7 正当性の証明

Reflection を示すため、以下の定理を証明する必要がある。

定理 4 († と ‡ 、 ∘ の左逆元・右逆元) 次のことが成り立つ:

1. $M \equiv M^{\circ\sharp}$
2. $V \equiv V^{\dagger\natural}$
3. $K \equiv K^{\ddagger\flat}$

1. $M_\Delta^{\sharp\circ} \equiv M_\Delta$
2. $V^{\natural\dagger} \equiv V$
3. $K_\Delta^{\flat\ddagger} \equiv K_\Delta$

左側にある 3 つの式は、左逆元の定理、右側の 3 つの式は右逆元の定理である。左逆元について、$\equiv$ になっているのは、DSkernel $\lambda_{c**}^{\mathcal{S}}$ と CPS λ_{cps} は全ての項に対して 1 対 1 対応しているためである。それぞれの定理の証明ではまず、1 において項に関する帰納法で解く。さらに、1 では 2, 3 の定理を必要とする。2 は値に関する帰納法、3 は継続 または コンテキストに関する帰納法で解き、1, 2, 3 を相互再帰して証明を行う。

定理 5 (CPS 変換の 1step 簡約の保存証明) 次のことが成り立つ：

1. $M \to M'$ ならば $M^\circ \to M'^\circ$
2. $\langle M\rangle \to \langle M'\rangle$ ならば $M^\circ \to M'^\circ$
3. $V \to V'$ ならば $V^\dagger \to V'^\dagger$
4. $K \to K'$ ならば $K^\ddagger \to K'^\ddagger$

1 は $(\beta.v)$、2 は、$(\beta.\mathcal{S})$、3 は $(\eta.v)$ を、4 は $(\eta.let)$ の簡約の保存証明をそれぞれ行なった。

定理 6 (DS 変換の 1step 簡約の保存証明) 次のことが成り立つ：

1. $M_k \to M_k'$ ならば $M_k^\sharp \to M'^\sharp_k$
2. $M_\bullet \to M'_\bullet$ ならば $\langle M_\bullet^\sharp\rangle \to \langle M'^\sharp_\bullet\rangle$
3. $V \to V'$ ならば $V^\natural \to V'^\natural$
4. $K \to K'$ ならば $K_\Delta^\flat \to K'^\flat_\Delta$

定理 5 と同様、1 は $(\beta.v)$、2 は、$(\beta.\mathcal{S})$、3 は $(\eta.v)$ を、4 は $(\eta.let)$ の簡約の保存証明をそれぞれ行なった。

定理 5 では、次の代入補題を必要とする。

補題 7 (CPS 変換における代入補題) 次のことが成り立つ：

1. $V_1{}^\dagger[x := V^\dagger] = (V_1[x := V])^\dagger$
2. $K_1{}^\ddagger[x := V^\dagger] = (K_1[x := V])^\ddagger$
3. $M^\circ[x := V^\dagger] = (M[x := V])^\circ$
4. $K_1{}^\ddagger[x := V^\dagger][k' := K^\ddagger] = (K_1[x := V][k' := K])^\ddagger$
5. $M^\circ[x := V^\dagger][k' := K^\ddagger] = (M[x := V][k' := K])^\circ$

1 は値、2, 4 は継続、3, 5 は項に関する帰納法でそれぞれ解く。

同様に、定理 6 でも以下のような代入補題を必要とする。

補題 8 (DS 変換における代入補題) 次のことが成り立つ：

1. $V_1{}^\natural[x := V^\natural] = (V_1[x := V])^\natural$
2. $K_1{}^\flat[x := V^\natural] = (K_1[x := V])^\flat$
3. $M^\sharp[x := V^\natural] = (M[x := V])^\sharp$
4. $K_1{}^\flat[x := V^\natural][k' := K^\flat] = (K_1[x := V][k' := K])^\flat$
5. $M^\sharp[x := V^\natural][k' := K^\flat] = (M[x := V][k' := K])^\sharp$

補題 7 と同様、値、コンテキスト、項に関する帰納法でそれぞれ解く。

8 まとめ

本研究では、Biernacki らの `shift/reset` 入りのラムダ計算に型をつけ、DSkernel と CPS の間の Reflection を証明した。また、それを Agda にのせた。DSkernel 項の定義を一般の DS 項に拡張し、DS 項と CPS 項の間における Reflection を証明しようとしているが、定義においてまだ課題が残っている。

参考文献

[1] Andrew W Appel. *Compiling with continuations*. Cambridge university press, 2007.

[2] Dariusz Biernacki, Mateusz Pyzik, and Filip Sieczkowski. A Reflection on Continuation-Composing Style. In Zena M. Ariola, editor, *5th International Conference on Formal Structures for Computation and Deduction (FSCD 2020)*, Vol. 167 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pp. 18:1–18:17, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

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

[4] 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.