Agda による CPS 変換の正当性の証明

石尾 千晶 (指導教員：浅井 健一)

1 はじめに

プログラミングにおける「継続」とは、計算中のある時点における残りの計算のことである。プログラム内で継続を明示的に扱う方法の一つに、プログラム全体を継続渡し形式 (Continuation-Passing Style; CPS) に変換するというものがある。CPS 変換をするとプログラム中の継続を管理できるため、コンパイラの中間言語に使われるなどの応用がある。

本研究は、より効率的な CPS 変換である selective CPS 変換 [1] の Agda [5] による定式化を試みる研究の一環であり、単純型付き λ 計算に限定継続命令 `shift/reset` を加えた体系における selective CPS 変換の正当性の証明の定式化を目指している。本稿では、単純型付き λ 計算を対象として、selective CPS 変換の基礎となる CPS 変換を Agda を用いて定式化し、その正当性を証明する [4]。定式化するにあたっては、変数束縛のある言語の抽象構文木を表現するための方法の一つである PHOAS (Parameterized Higher-Order Abstract Syntax) [2] を用いる。また、selective CPS 変換の正当性の証明に対する見通しを示す。

2 単純型付き λ 計算の定式化

本稿で対象とする言語は純粋な単純型付き λ 計算に自然数を加えたものである。その型と構文を図 1 に示す。型は自然数型か関数型のいずれかである。値は変数・自然数・λ 抽象の 3 つからなり、項は値か関数適用である。これを Agda で書くと次のようになる。値と項は相互再帰的に定義されており、図 1 の規則をそのまま埋め込めていることがわかる。

```
data typ : Set where
  Nat : typ -- 自然数型
  _⇒_ : typ → typ → typ -- 関数型
mutual
  data value[_]_ (var : typ → Set) : typ → Set where
    Var : {τ₁ : typ} → var τ₁ → value[ var ] τ₁ -- 変数
    Num : (n : ℕ) → value[ var ] Nat -- 自然数
    Fun : {τ₁ τ₂ : typ} → -- λ 抽象
      (e₁ : var τ₂ → term[ var ] τ₁) →
      value[ var ] (τ₂ ⇒ τ₁)
  data term[_]_ (var : typ → Set) : typ → Set where
    Val : {τ₁ : typ} →
      value[ var ] τ₁ → term[ var ] τ₁ -- 値
    App : {τ₁ τ₂ : typ} → -- 関数適用
      (e₁ : term[ var ] (τ₂ ⇒ τ₁)) →
      (e₂ : term[ var ] τ₂) → term[ var ] τ₁
```

Agda では、依存型を使って「型 τ を持つ項」を表現できるため、このように定義された構文のうち型が付くもののみを扱うこととなる。

また、図 2 に示す代入規則を定義する。$(\lambda y.\, e)[v] \mapsto e'$ と書くと、「項 e の中に出てくる y を値 v で置き換えると項 e' となる」と読む。さらに図 3 で項の評価規則のためのフレームと評価文脈を、図 4 で β 簡約規則を定義する。代入や簡約が行われる値と項は、図 1 のように相互再帰的に定義されているので、各規則もそれにしたがって相互再帰的な構造となる。

$$
\begin{array}{lrcl}
\text{型} & \tau & ::= & \mathsf{Nat} \mid \tau \to \tau \\
\text{値} & v & ::= & c \mid x \mid \lambda x.\, e \\
\text{項} & e & ::= & v \mid e_1\, e_2
\end{array}
$$

図 1: 単純型付き λ 計算

$$
(\lambda y.\, y)[v] \mapsto v \quad (\lambda y.\, x)[v] \mapsto x \quad (\lambda y.\, c)[v] \mapsto c
$$

$$
\frac{\forall x((\lambda y.\, (e\, y)\, x)[v] \mapsto e'\, x)}{(\lambda y.\, \lambda x.\, e\, y)[v] \mapsto \lambda x.\, e'}
$$

$$
\frac{(\lambda y.\, e_1\, y)[v] \mapsto e_1' \qquad (\lambda y.\, e_2\, y)[v] \mapsto e_2'}{(\lambda y.\, (e_1\, y)\, (e_2\, y))[v] \mapsto e_1'\, e_2'}
$$

図 2: 代入規則

3 CPS 変換の定式化

本稿では、Danvy/Filinski による one-pass の CPS 変換 [3] を扱う。型の変換を図 5 に、値と項の変換を図 6 に示す。CPS 変換は、値の変換 $[\![v]\!]_v$ と項の変換 $[\![e]\!]$, $[\![e]\!]'$ の 3 種類からなる。変換の結果は、アンダーラインのついた式とオーバーラインのついた式を使った 2 レベルの λ 計算で書かれている。アンダーラインのついた式は dynamic な式と呼ばれ、出力の構文を表す。一方、オーバーラインのついた式は static な式と呼ばれ、CPS 変換時にこれらの命令が実行されることを示す。static な命令は CPS 変換時に実行されるため、いわゆる administrative β-redex を生じることなく、コンパクトな CPS 変換結果を生成できる [3]。

$[\![e]\!]\,\overline{@}\,\kappa$ は、項 e を受け取ると、それを static な初期継続 κ のもとで CPS 変換した結果を返すことを意味する。このとき κ はメタ言語で書かれる関数であり、変換時に実行される。一方、$[\![e]\!]'\,\overline{@}\,k$ は、項 e を受け取ると、それを dynamic な初期継続 k のもとで CPS 変換した結果を返すことを意味する。このように 2 種類の変換を用意しているのは、CPS 変換時に継続がわかっている場合とそうでない場合があるためである。

4 CPS 変換の正当性の証明

この節では、3 節で示した CPS 変換が正しいこと、つまり CPS 変換をしても変換前の項と同等の β 簡約を行うことを示す。

定理 1 (CPS 変換の正当性) 任意の項 e, e' について $e \to e'$ が成り立つなら、任意の schematic な継続 κ について $[\![e]\!]\,\overline{@}\,\kappa \to^* [\![e']\!]\,\overline{@}\,\kappa$ が成り立つ。

これを Agda で書くと次のようになる。

```
correctEta :
  {var : typ → Set} → {τ : typ} →
  {e e' : term[ var ∘ cpsT ] τ} →
  (κ : value[ var ] cpsT τ → term[ var ] Nat) →
  Reduce e e' → schematic κ →
  Reduce* (cpsEta e κ) (cpsEta e' κ)
```

定理 1 は任意の κ について成り立つわけではない。static な継続 κ が「引数の変数の syntactic な構造を変更しない」という性質を持っている必要がある。この

$$\begin{array}{rrcl} \text{フレーム} & F & = & [\,]\,e \mid v\,[\,] \\ \text{評価文脈} & E & = & [\,] \mid F\,[E] \end{array}$$

図 3: 評価文脈

$$\frac{(\lambda x.\, e)[v] \mapsto e'}{(\lambda x.\, e)\, v \to e'} \qquad \frac{e \to e'}{F[e] \to F[e']}$$

図 4: 簡約規則

性質を schematic であるといい [3]、次のような式で示すことができる。

$$(\lambda y.\, \kappa \mathbin{\overline{@}} y)[v] \mapsto \kappa \mathbin{\overline{@}} v$$

つまり、κ に y を渡した結果の項の y の部分に v を入れたものは、最初から κ に v を渡したものと等しくなるということである。

定理 1 の証明の概略を示す。定理 1 は、$e \to e'$ の構造によって場合分けをして証明する。最初の場合分けは帰納法の base case に相当し、e が $(\underline{\lambda} x.\, e_1) \mathbin{\underline{@}} v$ の形のとき β 簡約をする。残りのケースはフレームの中が簡約される場合で、フレームの形によってさらに場合分けし、どちらも再帰する形で証明する。base case の証明は以下のようになる。

$$\begin{array}{lll} & [\![(\underline{\lambda} x.\, e_1) \mathbin{\underline{@}} v]\!] \mathbin{\overline{@}} \kappa & \\ = & ((\underline{\lambda} x.\, \underline{\lambda} k'.\, [\![e\, x]\!]' \mathbin{\overline{@}} k') \mathbin{\underline{@}} [\![v_2]\!]_v) \mathbin{\underline{@}} (\underline{\lambda} v.\, \kappa \mathbin{\overline{@}} v) & \\ \to & (\underline{\lambda} k'.\, [\![e']\!]' \mathbin{\overline{@}} k') \mathbin{\underline{@}} (\underline{\lambda} v.\, \kappa \mathbin{\overline{@}} v) & \text{補題 1} \\ \to & [\![e']\!]' \mathbin{\overline{@}} (\underline{\lambda} v.\, \kappa \mathbin{\overline{@}} v) & \text{補題 2} \\ \to^* & [\![e']\!] \mathbin{\overline{@}} \kappa & \text{補題 3} \end{array}$$

式中の β 簡約で以下の補題を利用することで、式をそのまま Agda のコードにすることができている。

補題 1 (CPS 変換と代入演算の可換性 1) 任意の項 e, e'、値 v、dynamic な継続 k について $(\lambda y.\, e\, y)[v] \mapsto e'$ が成り立つとき、$(\lambda y.\, [\![e\, y]\!]' \mathbin{\overline{@}} k)[[\![v]\!]_v] \mapsto [\![e']\!]' \mathbin{\overline{@}} k$ が成り立つ。

補題 2 (CPS 変換と代入演算の可換性 2) 任意の項 e、値 v、dynamic な継続 k について $(\lambda k.\, [\![e\, y]\!]' \mathbin{\overline{@}} k)[v] \mapsto [\![e\, y]\!]' \mathbin{\overline{@}} v$ が成り立つ。

base case の最後に現れる $\to^*$ では、e' の構造によって簡約のステップ数が異なるため以下に示す補題 3 で別途証明をしている。e' が値のときは 1 ステップの簡約が必要で、そうでないとき簡約は必要ない。以上より、定理 1 で $[\![e]\!] \mathbin{\overline{@}} \kappa$ を 2 ステップもしくは 3 ステップ簡約することで $[\![e']\!] \mathbin{\overline{@}} \kappa$ に到達することがわかる。

補題 3 (0 または 1 ステップの簡約) 任意の項 e、schematic な継続 κ について $[\![e]\!]' \mathbin{\overline{@}} (\underline{\lambda} v.\, \kappa \mathbin{\overline{@}} v) \to^* [\![e]\!] \mathbin{\overline{@}} \kappa$ が成り立つ。

5 selective CPS 変換の定式化に向けて

selective CPS 変換では、限定継続命令 `shift/reset` を導入する。`shift` はその時点の継続を切り取ってくる命令で、`reset` は切り取られる継続の範囲を限定する命令である。`shift` を使った項を impure な項と呼び、現在のコンテキストの型をあらわす answer type が変化する可能性があることを意味する。部分項が impure である場合は、項全体が impure であるとみなす。それ以外の項と値は pure であるといい、answer type は変化しない。このように全ての項と値を pure または impure に分類し、注釈をつけて定式化する。3 節の CPS 変換では無条件に全ての値と項を変換の対象としたが、selective CPS 変換は impure な項のみを対象として変換を行うため、より効率的である [1]。

`shift/reset` を導入した体系で定式化を行うには、answer type についての各種の証明が必要となる。また、与えられた項が pure か impure かによっても場合分けが必要である。

$$\begin{array}{rcl} \mathsf{Nat}^* & = & \mathsf{Nat} \\ (\tau_1 \Rightarrow \tau_2)^* & = & {\tau_1}^* \Rightarrow ({\tau_2}^* \Rightarrow \mathsf{Nat}) \Rightarrow \mathsf{Nat} \end{array}$$

図 5: 型の CPS 変換

$$\begin{array}{rcl} [\![c]\!]_v & = & c \\ [\![x]\!]_v & = & x \\ [\![\lambda x.\, e]\!]_v & = & \underline{\lambda} x.\, \underline{\lambda} k.\, [\![e]\!]' \mathbin{\overline{@}} k \\ [\![v]\!] & = & \overline{\lambda} \kappa.\, \kappa \mathbin{\overline{@}} [\![v]\!]_v \\ [\![e_1 \mathbin{@} e_2]\!] & = & \overline{\lambda} \kappa.\, [\![e_1]\!] \mathbin{\overline{@}} (\overline{\lambda} m.\, [\![e_2]\!] \mathbin{\overline{@}} \\ & & \qquad (\overline{\lambda} n.\, (m \mathbin{\underline{@}} n) \mathbin{\underline{@}} (\underline{\lambda} a.\, \kappa \mathbin{\overline{@}} a))) \\ [\![v]\!]' & = & \overline{\lambda} k.\, k \mathbin{\underline{@}} [\![v]\!]_v \\ [\![e_1 \mathbin{@} e_2]\!]' & = & \overline{\lambda} k.\, [\![e_1]\!] \mathbin{\overline{@}} \\ & & \qquad (\overline{\lambda} m.\, [\![e_2]\!] \mathbin{\overline{@}} (\overline{\lambda} n.\, (m \mathbin{\underline{@}} n) \mathbin{\underline{@}} k)) \end{array}$$

図 6: 単純型付き λ 計算の CPS 変換

6 まとめと今後の課題

単純型付き λ 計算における Danvy/Filinski の one-pass の CPS 変換を定式化し、その正当性を Agda で示した。定式化の方法に PHOAS を採用することで変数束縛による証明の複雑化を避け、手で書いたものとほぼ対応した証明を書くことができた。また、`shift/reset` の入った体系での selective CPS 変換の定式化への見通しを示した。今後、selective CPS 変換を定式化し証明を完成させたい。

参考文献

[1] K. Asai and C. Uehara. Selective CPS Transformation for Shift and Reset. *Proceedings of the ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation (PEPM'18)*, pp. 40–52, 2018.

[2] A. Chlipala. Parametric Higher-Order Abstract Syntax for Mechanized Semantics. *Proceedings of the ACM SIGPLAN International Conference on Functional Programming (ICFP'08)*, pp. 143–156, 2008.

[3] O. Danvy and A. Filinski. Representing Control: a Study of the CPS Transformation. *Mathematical Structures in Computer Science*, Vol. 2, No. 4, pp. 361–391, 1992.

[4] 石尾千晶, 山田麗, 浅井健一. Agda による PHOAS を用いた CPS 変換の正当性の証明. 第 20 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論文集掲載予定, 2018.

[5] A. Stump. *Verified Functional Programming in Agda*. Morgan & Claypool, 2016.