型付き対称ラムダ計算の基礎理論

阪上紗里　(指導教員：浅井健一)

1　はじめに

継続を扱うための新しい言語体系を定義したので、ここに報告する。継続とは「残りの計算」を表すものであり、現在着目している式の次の計算からプログラムの終わりまでの計算のことを言う。本稿ではこの継続を扱うための基礎体系として、Filinski の体系 [2] に基づいた型付き対称ラムダ計算 (Symmetric λ Calculus, SLC) を定式化した。この SLC により、項と継続を対等に扱うことが出来るようになった。次に継続を扱うための $\mathcal{C}$ オペレータを含んだ Felleisen の Λ_C 計算 [1] を紹介し、今回定式化した SLC の中に Λ_C 計算の埋め込みが可能であること、及びそれらの間の対応関係について言及する。

2　Call-by-value 対称ラムダ計算 (SLC)

2.1　Syntax と型

本研究の SLC(ここでは簡約が決定性のある Call-by-value の Right-to-left 戦略に基づいたもの) を紹介する。まず Syntax と型について述べる。SLC の Syntax は (項) と (関数) と (継続) で構成されることが特徴的である。

$$
\begin{array}{rrcl}
\text{(値)} & v & ::= & x \mid \lceil f \rceil \mid n \mid [(y \Leftarrow c \downarrow \underline{y}) \uparrow v] \\
\text{(項)} & e & ::= & v \mid f \uparrow e \\
\text{(関数)} & f & ::= & x \Rightarrow e \mid y \Leftarrow c \mid \overline{e} \mid \underline{c} \\
\text{(継続)} & c & ::= & k \mid c \downarrow f \\
\text{(値継続)} & k & ::= & y \mid \lfloor f \rfloor \mid \bullet \\
\text{(型)} & T & ::= & T \to T' \mid T - T' \mid \mathsf{int} \\
\text{(項型)} & T_e & ::= & +T \\
\text{(関数型)} & T_f & ::= & \{^{+T \to +T'}_{\neg T' \to \neg T} \\
\text{(継続型)} & T_c & ::= & \neg T'
\end{array}
$$

(値) は変数 x と、値として表すために $\lceil\ \rceil$ でくくられた関数、および整数である n、そしてコンテキスト $[(y \Leftarrow c \downarrow \underline{y}) \uparrow v]$ で構成される。コンテキストについては後ほど説明する。最終的にプログラムは v のいずれかが返ってくる。(項) は値 v か、通常の関数呼び出し $f \uparrow e$ である。(関数) について説明する。まず $x \Rightarrow e$ は通常の関数であり、e の中に現れる x を、受け取った項に置き換えた e を返すものである。同じように $y \Leftarrow c$ は、継続側の関数ととらえることができ、c の中に現れる y を、受け取った継続に置き換えた c を返すものである。$\overline{e}$ は評価をすると関数になる項を表している。$\underline{c}$ は評価をすると関数になる継続を表している。(継続) は値継続 k と、 c を行う前に f を計算する $c \downarrow f$ である。(値継続) は継続側の値であり、変数 y と、値として表すために $\lfloor\ \rfloor$ でくくられた関数、及び初期継続 (空の継続) を表す $\bullet$ で構成される。

T は 中立の型。 T_e は項に対する型である。 T_c は継続に対する型で、$\neg T'$ とは T' 型を受け取る継続の型を表す。T_f は関数型で、後ほど型規則のところでその特徴を述べる。

2.2　簡約規則

プログラムは途中 $\langle c \mid e \rangle$ 又は $\langle c \mid f \mid e \rangle$ という形で実行される。c はその時の継続、f は関数、e は項を表し、継続と項とが対称に現れていることがわかる。以下に簡約規則と簡約の挙動を簡単に説明する。

$$
\begin{array}{lrcl}
(\mathit{begin}) & e & \rightsquigarrow & \langle \bullet \mid e \rangle \\
(\overline{\mathit{pop}}) & \langle c \mid f \uparrow e \rangle & \rightsquigarrow & \langle c \mid f \mid e \rangle \\
(\mathit{push}_v) & \langle c \mid f \mid f' \uparrow e \rangle & \rightsquigarrow & \langle c \downarrow f \mid f' \uparrow e \rangle
\end{array}
$$

$$
\begin{array}{lrcl}
(\mathit{exch}_v) & \langle c \mid \overline{e} \mid v \rangle & \rightsquigarrow & \langle c \mid x \Rightarrow \overline{x} \uparrow v \mid e \rangle \\
(\beta_v) & \langle c \mid x \Rightarrow e \mid v \rangle & \rightsquigarrow & \langle c \mid e[v/x] \rangle \\
(\overline{\beta_v}) & \langle c \mid y \Leftarrow c' \mid v \rangle & \rightsquigarrow & \langle c'[c/y] \mid v \rangle \\
(\overline{\mathit{exch}_v}) & \langle c \mid \underline{c'} \mid v \rangle & \rightsquigarrow & \langle c' \mid [(y \Leftarrow c \downarrow \underline{y}) \uparrow v] \rangle
\end{array}
$$

$$
\begin{array}{lrcl}
(\overline{\mathit{context}}) & \langle \lfloor f \rfloor \mid [(y \Leftarrow c \downarrow \underline{y}) \uparrow v] \rangle & \rightsquigarrow & \langle c \mid f \mid v \rangle \\
(\mathit{pop}_v) & \langle c \downarrow f \mid v \rangle & \rightsquigarrow & \langle c \mid f \mid v \rangle \\
(\overline{\mathit{end}}) & \langle \bullet \mid v \rangle & \rightsquigarrow & v
\end{array}
$$

(begin) は計算したい項 e が与えられると、初期継続 $\bullet$ のもとで計算を行うことを表している。冒頭に述べた簡約戦略に基づき、($\overline{\mathit{pop}}$) を用いて 3 つ組にし、項部分が v になるまで (push_v) を使って関数を継続におしやる。3 つ組の項部分が v になると、関数部分によって簡約が進む。関数部分が $\overline{e}$ の場合 (exch_v) が適用され、まず e を先に計算したあとで v を適用させる (exch は $\mathit{exchange}$ の略)。関数部分が $x \Rightarrow e$ の場合通常の関数呼び出し (β 簡約) が (β_v) によって行われる。関数部分が $y \Leftarrow c$ の場合 ($\overline{\beta_v}$) が適用され、現在の継続を関数部分に適用させた新たな継続が返る。関数部分が $\underline{c'}$ の場合 ($\overline{\mathit{exch}_v}$) が適用され、後ほど適用したい c と v をパッケージ化した、Filinski がコンテキストと呼んでいるものに変換する。パッケージ化をせず $\langle c' \mid y \Leftarrow c \downarrow \underline{y} \mid v \rangle$ のように変換すると、($\overline{\beta_v}$) と (pop_v) が適用されて元に戻ってしまう。コンテキストはこれを防ぎ、 c' の継続部分を先に計算するようにしている。そして c' が $\lfloor f \rfloor$ に簡約されてはじめて、($\overline{\mathit{context}}$) により [ ] がはずれ c と v が適用される。実行中に $\langle c \downarrow f \mid v \rangle$ のように 2 つ組の項部分が v になった場合は、(pop_v) が適用され、継続から関数を関数部分に引き戻して簡約が進む。最終的にプログラムは $\bullet$ に値が渡されることで、評価が終わることを ($\overline{\mathit{end}}$) が表している。簡約中に $\overline{\lceil f \rceil}$ がでてきた場合、f としてよい。

2.3　型規則

型規則に関してはスペースの関係で省略するが、自然な型規則が与えられている。その中の関数型が特殊な形をしているので、ここで説明することにする。関数型 T_f は $\{^{+T \to +T'}_{\neg T' \to \neg T}$ で表される。これは $+T$ 型の項を受け取って $+T'$ 型の項を返す関数の型 $+T \to +T'$ と、$\neg T'$ 型の継続を受け取って $\neg T$ 型の継続を返す関数の型 $\neg T' \to \neg T$ を同時に表している。項側から見

た場合は一般的な関数の型であることは理解できる。例えば $+T_1 \to +T_2$ 型をもつ関数 f は、$+T_1$ 型の項を $+T_2$ 型の項に変換し、その後に $\neg T_2$ 型を持つ継続に結果を返す。これを継続側からみた場合を考えよう。例えば下の $y \Leftarrow c$ のような $\neg T_2 \to \neg T_1$ 型を持つ関数 f は、継続側の関数適用である $c \downarrow f$ の型規則を見ると、$\neg T_2$ 型の継続 c を $\neg T_1$ 型の継続、すなわち T_1 を受け取るような継続に変換し、その後に $+T_1$ 型を持つ項を受け取る。以上のことから、関数 f は2つの型を併せ持つことがわかる。

$$\frac{\Gamma, y : \neg T_2 \vdash c : \neg T_1}{\Gamma \vdash y \Leftarrow c : \{^{+T_1 \to +T_2}_{\neg T_2 \to \neg T_1}} \qquad \frac{\Gamma \vdash f : \{^{+T_1 \to +T_2}_{\neg T_2 \to \neg T_1} \quad \Gamma \vdash c : \neg T_2}{\Gamma \vdash c \downarrow f : \neg T_1}$$

SLC はさらに型付き言語の基本的性質である Progress と Preservation が成り立ことがわかっている。このことから SLC は継続を議論する上での土台としてふさわしい言語と言える。

この節で定式化した SLC が継続計算を議論・表現するのに適していることを示すため、次の節で単純型付き λ 計算に継続を扱うコントロールオペレータ $\mathcal{C}$ を追加した Felleisen の Λ_C 計算を紹介し、その次の節では Λ_C 計算が SLC に自然に埋め込めていることを示す。

3 Call-by-value Λ_C 計算

3.1 Syntax

$$\begin{array}{rrcl} \text{(値)} & V & ::= & x \mid \lambda x.\, M \mid \mathcal{C} \\ \text{(項)} & M & ::= & V \mid M\, M' \\ \text{(フレーム)} & F & ::= & M\, [\,] \mid [\,]\, V \\ \text{(評価文脈)} & E & ::= & [\,] \mid E[F] \end{array}$$

3.2 簡約規則

$$\begin{array}{lrcl} (1) & E[(\lambda x.\, M)\, V] & \rightsquigarrow & E[M[V/x]] \\ (2) & E[\mathcal{C}\, V] & \rightsquigarrow & V\,(\lambda x.\, \mathcal{A}\,(E[x])) \end{array}$$

(1) の簡約規則は、通常の関数呼び出し (β 簡約) である。(2) がコントロールオペレータ $\mathcal{C}$ の挙動である。これは継続を関数に変換し、それを V に渡すように表されている。その継続が実際に適用されると、$E[x]$ が実行された後、終了する。ここで $\mathcal{A}$ とは *Abort* のことで、$\mathcal{A}\, M' \stackrel{\text{def}}{=} \mathcal{C}\,(\lambda \_.\, M')$ と定義され、それまでの継続を捨てて M' が返る。

これらに対して、自然な型規則が与えられているが、ここでは省略する。

4 SLC への変換

4.1 変換規則

Λ_C 計算でかかれたプログラムを SLC に変換するための変換規則 T を以下に定義する。

$$T[\![E[M]]\!] = \langle\, \bullet \mid T_e[\![E[M]]\!] \,\rangle$$

SLC のプログラム $E[M]$ は変換後、初期継続のもとで計算されるように上の様に定義される。これは下に示す補題 4.1 により $\langle\, \bullet \mid T_e[\![E[M]]\!] \,\rangle \approx \langle\, T_c[\![E]\!] \mid T_e[\![M]\!] \,\rangle$ の関係が成り立つ。$s_1 \approx s_2$ は s_1 と s_2 を実行するとどちらも同じ結果になることを表す。ここで Λ_C 計算における評価文脈である E が SLC の継続部分に対応し、Λ_C 計算の redex を含んだ項 M が SLC の項部分に対応していることがわかる。私たちが意図した継続と項が明確に分かれて現れている。変換後の計算は $\langle\, T_c[\![E]\!] \mid T_e[\![M]\!] \,\rangle$ から行うこととする。

その他の変換規則は以下である。

$$\begin{array}{rcl} T_e[\![x]\!] & = & x \\ T_e[\![\lambda x.\, M]\!] & = & \lceil x \Rightarrow T_e[\![M]\!] \rceil \\ T_e[\![M\, N]\!] & = & T_f[\![M]\!] \uparrow T_e[\![N]\!] \\ T_e[\![\mathcal{C}]\!] & = & \lceil y \Leftarrow (\bullet \downarrow (x \Rightarrow \overline{x} \uparrow (\lceil \_ \Leftarrow y \rceil))) \rceil \\ T_f[\![M]\!] & = & \overline{T_e[\![M]\!]} \\ T_c[\![[\,]]\!] & = & \bullet \\ T_c[\![E[F]]\!] & = & T_c[\![E]\!] \downarrow T_F[\![F]\!] \\ T_F[\![M\ [\,]]\!] & = & T_f[\![M]\!] \\ T_F[\![[\,]\ V]\!] & = & x \Rightarrow (\overline{x} \uparrow T_e[\![V]\!]) \qquad (x : \text{fresh}) \end{array}$$

コントロールオペレータ $\mathcal{C}$ の変換、$T_e[\![\mathcal{C}]\!]$ について述べる。コントロールオペレータ $\mathcal{C}$ はそのときの継続を渡すものだが、$T_e[\![\mathcal{C}]\!]$ を変換すると、まずその時の継続が $\bullet$ によりリセットされ、その時の継続を y に渡すようにして、呼び出された際に関数に渡すように変換されている。このようにしてコントロールオペレータ $\mathcal{C}$ の挙動も変換できる。

これらの定義から、以下が成り立つことを示した。

補題 4.1 $\langle\, \bullet \mid T_e[\![E[M]]\!] \,\rangle \approx \langle\, T_c[\![E]\!] \mid T_e[\![M]\!] \,\rangle$

これは E の構造に関する帰納法により証明できる。

定理 4.2 Λ_C 計算の式 M の、一回簡約を $M \rightsquigarrow N$ と表すと、$T[\![M]\!] \approx T[\![N]\!]$ が成り立つ。

これは以下の代入補題を用いて証明できる。

補題 4.3 $T_e[\![M]\!][T_e[\![V]\!]/x] = T_e[\![M[V/x]]\!]$

代入補題は、M の構造に関する帰納法より証明できる。

以上により SLC に Λ_C 計算を埋め込むことができる。

5 まとめ

本稿では継続を直感的に扱うための SLC の定義、及び SLC への Λ_C の埋め込みと、両者の対応関係について紹介した。紙面の都合で省略したが、本研究では非決定性の SLC を定式化し、その型付き言語としての基本的性質も証明している。さらに Call-by-value Left-to-right Λ_C 計算の埋め込み、及び Call-by-name $\lambda\mu$ 計算の埋め込みを実現し、それぞれの対応関係も同様に示すことができている。今後は dual 計算との関連から論理学的観点からの見解を深めたい。

参考文献

[1] Felleisen, M., and R. Hieb "The Revised Report on the Syntactic Theories of Sequential Control and State," *Theoretical Computer Science*, Vol. 103, No. 2, pp. 235–271 (September 1992).

[2] Filinski, A. "Declarative Continuations and Categorical Duality," Master's thesis, DIKU Report 89/11, University of Copenhagen (August 1989).