階層化コントロールオペレータに対する型システムの構築

Construction of Type System for Family of Control Operator

鈴木 輝信 † 亀山 幸義 †

Terunobu Suzuki Yukiyoshi Kameyama

† 筑波大学システム情報工学研究科

Graduate School of Systems and Information Engineering, University of Tsukuba

コントロールオペレータは”残りの計算”を表す継続を扱うための機構であり、様々な制御構造を表現することができる。その一つである shift/reset は部分継続を扱うことができる点、さらに CPS 変換に基づいているため形式的に扱いやすいという点で有用である。また、階層化 shift/reset は、複数の shift/reset を区別したいときに必要であるが、これまで型システムやその性質については検討されてこなかった。そこで、本研究では階層化 shift/reset に対して型システムを与え、その健全性と型の一意性について検討する。

1 はじめに

コントロールオペレータは ”残りの計算 ”を表す継続を直接操作できるようにするための式である。このコントロールオペレータを用いることによってループなど様々な制御構造を表現できる [6]。

このコントロールオペレータの中の一つに shift/reset と呼ばれるものがある [1, 5]。この shift/reset は継続の一部分を表す部分継続を扱えるコントロールオペレータである。この部分継続を扱うコントロールオペレータとして control/prompt[2] と呼ばれるものもあるが、shift/reset のもう一つの特徴として CPS 変換によってその意味が与えられているため、形式的に取り扱いやすいという点がある。そのため、いままでに shift/reset に対する研究が行われている [1, 7]。

また、この shift/reset を様々な目的で用いる場合、それぞれ単独では意図したとおり動く場合でも、それらを組み合わせた場合にそれぞれの shift/reset が干渉しあって意図したとおりに動かない場合がある。この問題に対処するために階層化 shift/reset[3] が提案された。これは、それぞれの shift/reset を index をつけて区別できるようにして shift/reset の干渉を防げるようにしたものである。

しかし、この階層化 shift/reset にはまだ型システムは与えられていない。階層化 shift/reset によって制御構造を表現した場合、その動作が複雑であるため直感的にプログラムを理解することは難しい。そのため、プログラムを型によって抽象化することによってプログラムが分かりやすくなると考えられる。また、同じ理由で型の不整合が起こりやすいとも考えられるため、階層化 shift/reset に対する型システムはとても有用であると考えられる。

そこで、本論文では普通の (階層化されていない)shift/reset に対する型システム [4] を拡張して、階層化 shift/reset に対する型システムを提案し、その性質を示す。これにより、先に述べた問題が解決できるようになり、既存の強い型付けがあるプログラミング言語に対して階層化 shift/reset が導入できるようになると考えている。

本稿の以降の構成は次の通りである。まず、第 2 節で shift/reset と階層化 shift/reset の概要について述べ、第 3 節で階層化 shift/reset の形式的意味とこれに対する型システムについて述べる。第 4 節で型システムに対して成り立つ性質について述べ、最後にまとめを述べる。

2 コントロールオペレータ:shift/reset

2.1 コントロールオペレータ

コントロールオペレータとは継続を扱うための式のことである。継続は計算の残りを表すもので、対象となる式の計算後からプログラム終了までに行われる計算を表している。この継続を扱うために call/cc と呼ばれるコントロールオペレータがあり、これは scheme 等で実装されている。

これを発展させた部分継続を扱うコントロールオペレータが存在する。部分継続は、現在計算している式からみてプログラム終了までの計算の一部を表す。この部分継続を扱うための式のひとつが shift/reset

である。この部分継続を扱うためのコントロールオペレータとして control/prompt があるが、shift/reset の方が形式的に取り扱いやすいため、本研究では shift/reset を対象とする。

2.2 shift/reset

前節で述べたように、shift/reset は部分継続を扱うためのコントロールオペレータである。shift は $Sk.M$(k は変数、M は一般の式を表す。reset のときも同様) として表され、shift 自身の継続を変数 k に格納し、M を計算する式である。一方、reset は $\langle M \rangle$ として表現され、shift が継続として捕らえる範囲を限定した上で M を計算する式である。これらの式の意味を以下の例で考える。

$$\begin{aligned} &\langle 2 + (Sk.(3+4)) \rangle + 1 \\ =&(3+4)+1 = 8 \\ &\langle 2 + (Sk.3 + (k(k4))) \rangle + 1 \\ =&(3 + (2 + (2+4))) + 1 = 12 \end{aligned}$$

まず、最初の式では shift で捕らえた継続を利用しない場合について示している。shift ではまず shift の変数 k に対して shift の継続 (ただし、reset の内側のみ) が割り当てられる。この例では、shift から見た残っている計算は (reset を無視すれば)2 を足す、1 を足すという二つの計算だが、reset の外側にある計算は無視するため、k に割り当てられる計算は 2 を足す、という部分だけになる。そして、shift の内側の計算 (最初の例では 3+4) を行い、この部分の計算が終わると reset の継続 (1 を足す部分) に飛ぶ。結果として、2 を足す部分は計算されず、上記のようになる。

二つ目の式では、shift の内側の計算で k を用いた場合を示している。この変数 k に対して、値を適用しているので、k に割り当てられた計算が行われている。この場合、k は 2 を足すという計算を格納していたので (k を二回使っているので) 2 を足すという計算を二回行い、その後 3 を足す。この時点で shift の内側の式の計算が終わったので reset の外側に飛んで残りの計算を行うと、上記のようになる。

2.3 shift/reset の階層化

この shift/reset を拡張して、階層化した shift/reset が提案されている [3, 5]。これは、shift,reset に index を振り、shift が継続を捕らえる際に、どの reset までの継続を捕らえるかを表せるようにしたものである。例えば、2 つの shift/reset を導入した場合、以下のような例を書くことができる。

$$1 + \langle \langle (S_2 k.\ k(k1)) + 3 \rangle_1 + 2 \rangle_2$$

もし、この式の index が無いとすると、k で捕らえる継続は 3 を足す部分だけなので、$1+(((1+3)+3)+2) = 10$ になるが、この場合、shift の index が 2 なので、index が 1 の reset を無視して、外側の index が 2 の reset までの継続を捕らえることになり、結果は $1 + ((1+3+2)+3+2) = 12$ になる。この例では、index は 2 までしかないが、[3] では、もっと一般的に index を n まで振った場合について CPS 変換を用いて意味が定義されている。

3 shift/reset の意味と型システム

3.1 shift/reset と CPS 変換

まずは、この研究で対象とするプログラミング言語 λS_m を定義する。この言語は $1 \leq m$ となる m ごとに与えられる言語である。はじめに構文を定義する。

定義 1 (λS_m の構文)

$$\begin{aligned} M ::= &\ 0 \mid 1 \mid 2 \mid \ldots && (integer) \\ \mid &x && (variable) \\ \mid &\lambda x.M && (\lambda - abstraction) \\ \mid &M_1 M_2 && (application) \\ \mid &S_n k.M \quad (1 \leq n \leq m) && (shift) \\ \mid &\langle M \rangle_n \quad (1 \leq n \leq m) && (reset) \end{aligned}$$

この構文は、λ 計算に自然数と shift/reset を追加したものである。また、この言語での値は以下のように定義する。

定義 2 (λS_m の値)

$$\begin{aligned} V ::= &\ 0 \mid 1 \mid 2 \mid \ldots \\ \mid &x \\ \mid &\lambda x.M \end{aligned}$$

この言語の計算規則は文脈を用いて表現する。文脈と計算規則の定義を図 1 に示す。

shift/reset の CPS 変換 [3] を図 2 に示す。ただし、Var は変数の集合である。

E は evaluation-context,P_n は n レベルでの pure-context を表す。

$$
\begin{array}{ll}
E ::= [] & P_n ::= [] \\
\quad | EM \mid VE & \quad | P_n M \mid V P_n \\
\quad | \langle E \rangle_n & \quad | \langle P_n \rangle_i \qquad (i < n)
\end{array}
$$

計算規則は以下のように定義される。

$$
\begin{aligned}
E[(\lambda x.M)V] &\longrightarrow E[M\{x := V\}] \\
E[\langle P_n[S_n k.M] \rangle_j] &\longrightarrow E[\langle M\{k := \lambda x.\langle P_n[x] \rangle_n\} \rangle_j] \qquad (n \leq j) \\
E[\langle V \rangle_n] &\longrightarrow E[V]
\end{aligned}
$$

図 1: 計算規則

$$
\begin{aligned}
C_m &: Term \to Env \to Cont_1 \to \ldots \to Cont_{m+1} \to Ans \\
Env &= Var \to Val \\
\kappa_i \in Cont_i &= Val \to Cont_{i+1} \to \ldots \to Cont_{m+1} \to Ans \qquad (1 \leq i \leq m) \\
\kappa_{m+1} \in Cont_{m+1} &= Val \to Ans \\
Val &::= n \mid Val \to Cont_1 \to \ldots \to Cont_{m+1} \to Ans \qquad (n \text{ は自然数})
\end{aligned}
$$

$$
\begin{aligned}
C_m[\![n]\!]\rho\kappa_1 \ldots \kappa_{m+1} &= \kappa_1 n \kappa_2 \ldots \kappa_{m+1} \\
C_m[\![x]\!]\rho\kappa_1 \ldots \kappa_{m+1} &= \kappa_1 \rho(x) \kappa_2 \ldots \kappa_{m+1} \qquad (x \text{ は変数}) \\
C_m[\![\lambda x.E]\!]\rho\kappa_1 \ldots \kappa_{m+1} &= \kappa_1 (\lambda v \kappa_1^{'} \ldots \kappa_{m+1}^{'}.C_m[\![E]\!]\rho[x \mapsto v]\kappa_1^{'} \ldots \kappa_{m+1}^{'}) \kappa_2 \ldots \kappa_{m+1} \\
C_m[\![E_1 E_2]\!]\rho\kappa_1 \ldots \kappa_{m+1} &= C_m[\![E_1]\!]\rho(\lambda f \kappa_2^{'} \ldots \kappa_{m+1}^{'}. \\
&\qquad C_m[\![E_2]\!]\rho(\lambda a \kappa_2^{''} \ldots \kappa_{m+1}^{''}. \\
&\qquad\quad f\ a \kappa_1 \kappa_2^{''} \ldots \kappa_{m+1}^{''}) \kappa_2^{'} \ldots \kappa_{m+1}^{'}) \kappa_2 \ldots \kappa_{m+1} \\
C_m[\![\langle E \rangle_n]\!]\rho\kappa_1 \ldots \kappa_{m+1} &= C_m[\![E]\!]\rho\theta_1 \ldots \theta_n \\
&\qquad (\lambda v \kappa_{n+2}^{'} \ldots \kappa_{m+1}^{'}.\theta_0 v \kappa_1 \ldots \kappa_{n+1} \kappa_{n+2}^{'} \ldots \kappa_{m+1}^{'}) \kappa_{n+2} \ldots \kappa_{m+1} \\
C_m[\![S_n k.E]\!]\rho\kappa_1 \ldots \kappa_{m+1} &= C_m[\![E]\!]\rho[k \mapsto p]\theta_1 \ldots \theta_n \kappa_{n+1} \ldots \kappa_{m+1}
\end{aligned}
$$

ただし、

$$
\begin{aligned}
p &= \lambda v \kappa_1^{'} \ldots \kappa_{m+1}^{'}.\theta_0 v \kappa_1 \ldots \kappa_n (\lambda w \kappa_{n+2}^{''} \ldots \kappa_{m+1}^{''}.\theta_0 w \kappa_1^{'} \ldots \kappa_{n+1}^{'} \kappa_{n+2}^{''} \ldots \kappa_{m+1}^{''}) \kappa_{n+2}^{'} \ldots \kappa_{m+1}^{'} \\
\theta_i &= \lambda v \kappa_{i+1} \ldots \kappa_{m+1}.\kappa_{i+1} v \kappa_{i+2} \ldots \kappa_{m+1} \qquad (0 \leq i \leq m) \\
\theta_{m+1} &= \lambda v.v
\end{aligned}
$$

図 2: shift/reset の CPS 変換の定義

ここで定義している CPS 変換は評価関数 C_m を用いて再帰的に定義している。この C_m は環境と m 個の継続を受け取って計算結果を返す関数である。

まず、shift/reset 以外の計算では、通常の CPS 変換を行い、2 個目以降のそれぞれの継続を η 変換によって付け加えたものと等しくなる。これは、これらの計算では 2 個目以降の継続は直接操作することが無いためである。

reset の計算では、n 番目までの継続を $n+1$ 番目の継続にまとめ、そして n 番目までの継続は初期化して reset の内部の式の評価を行う。これによって、n 以下の index を持つ shift は reset によってまとめられた継続は見えなくなるということになる。

また、shift の計算は n 番目までの継続をまとめて変数 k に束縛させ、n 番目までの継続を初期化する。これにより、shift の内部の式で shift の継続を利用することができるようになり、この shift の働きを制限する reset までの継続は捨てられることになる。

3.2 型システムの設計方針

型システムの設計方針は、原則として図 2 の CPS 変換の式を型付 λ 計算とみなして、それぞれの項を型推論し、その型情報を今回の型システム上で表現することによって型システムを構築していく。これにより、基本的な型システムは生成できるが、CPS 変換の Answer type に相当する部分は型情報を保持しなくても、その Answer type が同じ型を持つことさえ分かればよい事を用いると型の判断に必要な型情報を減らすことができることがわかる。このような考え方の元で型システムを構築する。

ただし今回は、一般の m を扱うと型システムが煩雑になるため $m = 2$ の場合について述べる。

3.3 階層化 shift/reset に対する型システム

階層化 shift/reset の型を考える時、shift/reset が継続を操作していることを考慮すると型付 λ 計算のように対象としている項の型だけを考えるだけでは不十分であり、それぞれの継続の型を考える必要がある。そのため、型の判断を以下のように拡張する。

定義 3 (型判断) 階層化 shift/reset に対する型判断を以下のように定義する。

$$\Gamma; \alpha_1, \alpha_2, \alpha_3 \vdash M : \tau, \beta_1, \beta_2, \beta_3$$

Γ は型環境、M は Term であり、τ,α_1,α_2,α_3,β_1,β_2,β_3 はそれぞれ型を表す。これは以下のように M を CPS 変換する際の型を表している。ただし、ρ は Γ に対応する環境を表すとする。

$$\begin{aligned} C_2[\![M]\!]\rho : \quad & (\tau \to (\alpha_1 \to (\alpha_2 \to *) \to *) \\ & \to (\alpha_3 \to *) \to *) \\ & \to (\beta_1 \to (\beta_2 \to *) \to *) \\ & \to (\beta_3 \to *) \to * \end{aligned}$$

ただし、$*$ は Answer type を表すものとする。

また、これ以降では簡単のために型の列 $\alpha_1, \alpha_2, \alpha_3$ を $\overline{\alpha}$ と表記する。

これに付随して関数型も拡張する必要がある。関数に引数を与えた時、関数本体の状況 (その時の継続の型など) を関数型に含める必要があるためである。

定義 4 (関数型) 階層化 shift/reset に対する型システムでの関数型を以下のように定義する。

$$\phi/\overline{\alpha} \to \tau/\overline{\beta}$$

これは、この関数に ϕ 型の値を適用した時、関数本体を表す式を M とすると関数本体の判断が

$$\Gamma; \overline{\alpha} \vdash M : \tau, \overline{\beta}$$

となることを表す。

このように拡張した上で shift/reset の型付け規則を表現したものが図 3 である。

はじめに、shift/reset 以外の式の型付けについて述べる。自然数の場合、CPS 変換で継続 κ_1 にその値と他の継続が適用されているため、左右の型の列は同じものになる。λ 抽象の場合、これも値なのでそれぞれの継続の結果がそのまま次の継続に渡される。また、関数本体の型は先に述べたように関数型の情報を元に型付けを行うようにすればよい。関数適用の場合、図 2 の意味に従うと、この計算は引数部分の計算を関数部分の継続として扱い、関数本体の計算を引数部分の継続としていることから、図のようになる。

次に、$reset_n$ の場合、n 番目までの継続が $n+1$ 番目の継続にまとめられるという作用が内部式の判断で表現されており、継続が初期化されているという作用は γ_i によって表されている部分の型が同じであることによって表現されている。

$$\frac{n \in N}{\Gamma; \overline{\alpha} \vdash n : int, \overline{\alpha}}\ int$$

$$\frac{\rho(x) = \tau}{\Gamma; \overline{\alpha} \vdash x : \tau, \overline{\alpha}}\ var$$

$$\frac{\Gamma[x \mapsto \phi]; \overline{\gamma} \vdash E : \tau, \overline{\delta}}{\Gamma; \overline{\alpha} \vdash \lambda x.E : (\phi/\overline{\gamma} \to \tau/\overline{\delta}), \overline{\alpha}}\ lambda$$

$$\frac{\Gamma; \overline{\gamma} \vdash E_1 : (\phi/\overline{\alpha} \to \tau/\overline{\delta}), \overline{\beta} \qquad \Gamma; \overline{\delta} \vdash E_2 : \phi, \overline{\gamma}}{\Gamma; \overline{\alpha} \vdash E_1 E_2 : \tau, \overline{\beta}}\ app$$

$$\frac{\Gamma; \gamma_1, \gamma_2, \gamma_2 \vdash E : \gamma_1, \tau, \alpha_3, \beta}{\Gamma; \overline{\alpha} \vdash \langle E \rangle_1 : \tau, \alpha_1, \alpha_2, \beta}\ reset_1$$

$$\frac{\Gamma; \gamma_1, \gamma_2, \gamma_2 \vdash E : \gamma_1, \gamma_3, \gamma_3, \tau}{\Gamma; \overline{\alpha} \vdash \langle E \rangle_2 : \tau, \overline{\alpha}}\ reset_2$$

$$\frac{\Gamma^{'}; \gamma_1, \gamma_2, \gamma_2 \vdash E : \gamma_1, \overline{\beta}}{\Gamma; \overline{\alpha} \vdash S_1 k.E : \tau, \overline{\beta}}\ shift_1$$

$$\frac{\Gamma^{''}; \gamma_1, \gamma_2, \gamma_2 \vdash E : \gamma_1, \gamma_3, \gamma_3, \beta}{\Gamma; \overline{\alpha} \vdash S_2 k.E : \tau, \alpha_1, \alpha_2, \beta}\ shift_2$$

$$\Gamma^{'} = \Gamma[k \mapsto (\tau/\delta_1, \delta_2, \alpha_2 \to \alpha_1/\delta_1, \delta_2, \alpha_3)]$$
$$\Gamma^{''} = \Gamma[k \mapsto (\tau/\overline{\delta} \to \alpha_3/\overline{\delta})]$$

図 3: $m = 2$ の場合に対する階層化 shift/reset の型付け規則

また、shift の場合、k に継続が束縛されていることは k の引数の型が shift の一番目の継続の引数の型と同じであること、関数本体の型がその継続の型と同じになっていることによって表現されている。また、継続が初期化されていることは reset と同じように表現されている。

なお、この型システムに関して、

1. $shift_2$ 規則と $reset_2$ 規則を除去する
2. 判断を変換

$$\Gamma; \overline{\alpha} \vdash E : \tau, \overline{\beta}$$

を以下のように変換する。

$$\Gamma'; \alpha_1 \vdash E : \tau, \beta_1$$

ただし、Γ' は Γ に含まれる型変数にマップされている型に対して 3 の操作を行ったものである。

3. 関数型を変換

$$\phi/\overline{\alpha} \to \tau/\overline{\beta}$$

を以下のように変換する。

$$\phi/\alpha_1 \to \tau/\beta_1$$

これにより、この型システムは文献 [4] で述べられている $m = 1$ の場合の型システムに一致させることができる。

4 型システムの性質

この節では前節で定義した型システムに対して成り立つ性質について述べる。このうち、健全性と principal types に対する定理は一般の m について述べているが、まだ $m = 2$ の場合しか証明しておらず、一般の m に対しては証明していない。

4.1 型付 λ 計算との対応

まずは、型システムの設計方針に即しているかどうかを述べるため、型付 λ 計算との対応を形式的に述べる。

その前の準備として、まず型に対する CPS 変換を定義する。

定義 5 (型の CPS 変換) 型 A を CPS 変換した型 A^* を以下のように定義する。

$$\begin{aligned}
&A^* = A \qquad (A \text{ は basic type}) \\
&(\phi/\overline{\gamma} \to \tau/\overline{\delta})^* \\
=&\phi^* \to \\
&(\tau^* \to (\gamma_1^* \to (\gamma_2^* \to X) \to X) \\
&\qquad \to (\gamma_3^* \to X) \to X) \\
\to&(\delta_1^* \to (\delta_2^* \to X) \to X) \\
\to&(\delta_3^* \to X) \to X
\end{aligned}$$

ただし、X は Answer Type を表す型変数である。さらに、型環境に含まれる型を CPS 変換できるようにこの $*$ を以下のように拡張する。

$$\begin{aligned}
[\,]^* &= [\,] \\
\Gamma[x \mapsto T]^* &= \Gamma^*[x \mapsto T^*]
\end{aligned}$$

この CPS 変換を用いて、型付 λ 計算との対応を示す。

定理 1 (型付 λ 計算との対応) 階層化 shift/reset に対する型システム (m=2) によって

$$\Gamma; \overline{\alpha} \vdash E : \tau, \overline{\beta}$$

が得られたとすると、

$$\begin{aligned}
&\Gamma^* \vdash C_2[\![E]\!] : \\
&\quad (\tau^* \to (\alpha_1^* \to (\alpha_2^* \to X) \to X) \\
&\qquad\qquad \to (\alpha_3^* \to X) \to X) \\
&\qquad \to (\beta_1^* \to (\beta_2^* \to X) \to X) \\
&\qquad\quad \to (\beta_3^* \to X) \to X
\end{aligned}$$

が型付 λ 計算で導出できる。

これにより、型付 λ 計算との対応を示すことができる。

4.2 健全性

次に型システムの健全性について示す。最初に述べたようにこれ以降の定理は $m = 2$ の場合のみ証明している。ここでは、一般の m に対応した形で表記する。ただし、$l = 2^m - 1$ とし、$\alpha_{(i,j)}$ は型の列 $\alpha_i, \ldots, \alpha_j$ を表すものとする。

健全性は型付 λ 計算と同じように、progress と preservation によって証明される。これらの定理について述べる前にこの言語での計算の 1 ステップを定義しておく。

定義 6 この言語での計算の 1 ステップとは、図 1 の左の式を右に変換することをさす。

これを用いて progress 定理と preservation 定理を記述する。

定理 2 (progress) 式 $\langle E\rangle_m$ に対して

$$[\,];\overline{\alpha}_{(1,l)} \vdash \langle E\rangle_m : \tau, \overline{\beta}_{(1,l)}$$

となる $\overline{\alpha}_{(1,l)}, \tau, \overline{\beta}_{(1,l)}$ が存在するならば、$\langle E\rangle_m$ は値であるか、計算を 1 ステップ以上行うことができる。

ここで、対象とする式が $\langle E\rangle_m$ となっているのは、計算規則から reset のない shift を含む式 (例えば、$S_1 k.k$) は値ではなく、計算も進められない式となってしまうためである。そこで、与えられた式 E に対して reset を最初にかけることによって shift を含む式の計算ができるようにしている。

次は preservation 定理である。

定理 3 (preservation) 式 E に対して

$$\Gamma;\overline{\alpha}_{(1,l)} \vdash E : \tau, \overline{\beta}_{(1,l)}$$

となる $\alpha_1, \ldots, \alpha_l, \tau, \beta_1, \ldots, \beta_l$ が存在し、かつ $E \longrightarrow E^{'}$ となるならば、

$$\Gamma;\overline{\alpha}_{(1,l)} \vdash E^{'} : \tau, \overline{\beta}_{(1,l)}$$

となる。

この二つの定理により、型システムが計算規則と対応していることが示される。

4.3 principal types の存在

principal type を表現するために、まず型に対する置換を定義する。

定義 7 置換は型から型へのマップを表す。置換 S は以下のように定義される。

$$[\,](x) = x \qquad (\text{if } S = [\,])$$
$$S^{'}[x \mapsto T](x) = T \qquad (\text{if } S = S^{'}[x \mapsto T])$$
$$S[x \mapsto T](y) = S \quad (\text{if } S = S^{'}[x \mapsto T] \wedge x \neq y)$$

$$S(\tau/\alpha_1, \ldots, \alpha_l \to \sigma/\beta_1, \ldots, \beta_l) =$$
$$S(\tau)/S(\alpha_1), \ldots, S(\alpha_l) \to S(\sigma)/S(\beta_1), \ldots, S(\beta_l)$$

この置換は右にあるものから適用することとする。

これを用いて、型の間の関係を定義する。

定義 8 型 T_1, T_2 に対し、$T_1 \leq T_2$ と $T_1 \equiv T_2$ を以下のように定義する。

$$T_1 \leq T_2 \Leftrightarrow \exists S.S(T_1) = T_2 \qquad (S \text{ は置換})$$
$$T_1 \equiv T_2 \Leftrightarrow T_1 \leq T_2 \wedge T_2 \leq T_1$$

さらに、この関係を型の列に対して適用できるように拡張する。

$$(\overline{\alpha}_{(1,n)}) \leq (\overline{\beta}_{(1,n)}) \Leftrightarrow \forall i.\alpha_i \leq \beta_i$$
$$(\overline{\alpha}_{(1,n)}) \equiv (\overline{\beta}_{(1,n)}) \Leftrightarrow \forall i.\alpha_i \equiv \beta_i$$

次に、式に対する principal types の存在に関して、以下の定理が証明できる。

定理 4 (principal types) 式 E と型環境 Γ に対して、

$$\Gamma;\overline{\alpha}_{(1,l)} \vdash E : \tau, \overline{\beta}_{(1,l)}$$

が成り立ち、

$$\Gamma;\overline{\gamma}_{(1,l)} \vdash E : \sigma, \overline{\delta}_{(1,l)}$$

を満たす全ての $(\overline{\gamma}_{(1,l)}, \sigma, \overline{\delta}_{(1,l)})$ に対して

$$(\overline{\alpha}_{(1,l)}, \tau, \overline{\beta}_{(1,l)}) \leq (\overline{\gamma}_{(1,l)}, \sigma, \overline{\delta}_{(1,l)})$$

を満たすような $(\overline{\alpha}_{(1,l)}, \tau, \overline{\beta}_{(1,l)})$ が存在する。この時の $(\overline{\alpha}_{(1,l)}, \tau, \overline{\beta}_{(1,l)})$ を principal types と呼ぶ。

また、この principal types に対して一意であることが以下の定理によって示される。

定理 5 (principal types の一意性) 式 E と型環境 ρ に対して ($\overline{\alpha}_{(1,l)}$) と ($\overline{\beta}_{(1,l)}$) がともに *principal types* であるならば、($\overline{\alpha}_{(1,l)}$) ≡ ($\overline{\beta}_{(1,l)}$) が成立する。

これにより、最も一般的な型を表す principal types が型変数の名前の違いを除いて一意に得られることが分かる。これにより、型推論を考えられるようになる。

5 まとめ

継続を扱うためのコントロールオペレータの一つに、shift/reset と呼ばれるものがある。これは、部分継続を扱える上、形式的に取り扱いやすいという特徴があるため大変有用である。これを同時に複数

用いると互いに干渉して考えたとおりに動かなくなる場合がある。そのために階層化 shift/reset が提案された。

しかし、この階層化 shift/reset には今まで型システムが与えられていなかったため本研究では、階層化 shift/reset に対する型システムを構築し、それに関する性質や型推論について検討した。これにより、プログラムが分かりやすくなる、実行時の型エラーを実行前に防ぐことができるなどの効果を得ることができる。

今後は、型システム及び型推論アルゴリズムの実装や言語の拡張の検討などを行っていく予定である。

参考文献

[1] Malgorzata Biernacka, Dariusz Biernacki, and Olivier Danvy. An operational foundation for delimited continuations in the CPS hierarchy. *Logical Methods in Computer Science*, 1(2), 2005.

[2] Dariusz Biernacki, Olivier Danvy, and Chung chieh Shan. On the static and dynamic extents of delimited continuations. Technical Report RS-05-36, DAIMI,the Universities of Aarhus in Denmark, 2005.

[3] Olivier Danvy and Andrej Filinski. Abstracting control. In *Proceedings of the 1990 ACM Conference on LISP and Functional Programming, Nice*, pages 151–160, New York, NY, 1990. ACM.

[4] Olivier Danvy and Andrzej Filinski. A functional abstraction of typed contexts. Technical Report 89/12, DIKU, University of Copenhagen, 1989.

[5] Yukiyoshi Kameyama and Masahito Hasegawa. A sound and complete axiomatization of delimited continuations. In *Proc. International Conference on Functional Programming*, pages 177–188, 2003.

[6] Christopher Strachey and Christopher P. Wadsworth. Continuations: A mathematical semantics for handling full jumps. *Higher-Order and Symbolic Computation*, 13(1/2):135–152, 2000.

[7] Peter Thiemann. Cogen in six lines. In *Proc. International Conference on Functional Programming*, pages 180–189, 1996.