shift/reset を含む体系における多相な型システムの拡張

朱 光漪　(指導教員：浅井 健一)

1　はじめに

限定継続はある時点での残りの計算の一部を表し、例外や非同期処理といった計算エフェクトを統一的に扱うことができる概念である。型システムとはプログラムを型で分類するための枠組みであり、主に式の型を表す型付け判断と、型付け判断を導出する型付け規則から成る。限定継続演算子の一種である shift/reset [2] を含む体系においては、let 多相と、捕捉された継続の answer type のみが多相となる型システム [1] が与えられている。本研究では、この多相な型システムを拡張し、捕捉した継続の型のうち型環境に現れない型変数を多相にする。

2　多相が必要な例

単純なプログラムにおいても、一つの変数が複数の場所で異なる型で使われる場合がある。単相な型システムでは、一つの変数に対して一つの型しか付けることができない。これに対し、多相な型システムでは、一つの変数に対して複数の型を付けることができるため、単相な型システムでは型が付かない式に対しても型を付けることができる。

多相が必要となる式の例として、以下の式について考える。

$$\langle \mathcal{S}k.\,\mathsf{if}\ (k\ true)\ \mathsf{then}\ (k\ 1)\ \mathsf{else}\ 0\rangle$$

$\mathcal{S}k.\,e$ は限定継続演算子 shift を用いた式であり、最も近い $\langle\rangle$ までの継続を k に束縛し、その後 e を実行する。例えば、上記の式では、$\mathcal{S}k.\,\mathsf{if}\cdots$ のすぐ外側に $\langle\rangle$ があるため、k は受け取った引数をそのまま返す恒等関数となる。この状態で、$\mathsf{if}\ (k\ true)\ \mathsf{then}\ (k\ 1)\ \mathsf{else}\ 0$ を実行する。ここで、k の型に注目すると、$k\ true$ では、k は $\texttt{bool}\to\texttt{bool}$ 型を持つ。一方で、$k\ 1$ では、k は $\texttt{int}\to\texttt{int}$ 型を持つ。このように、k は呼び出される場所によって異なる型を持ち、多相な型システムは単純なプログラミングにおいても欠かせないものである。

3　shift/reset の多相な型システム

本節では、shift/reset を含む体系の多相な型システムの定式化を行う。

3.1　Syntax

let 文は値のみを x に束縛することができる。

$$\begin{array}{rcll} v & ::= & n \mid x \mid \lambda x.\,e & \text{値}\\ e & ::= & v \mid e_1\ e_2 \mid \mathcal{S}k.\,e \mid \langle e\rangle \mid \mathsf{let}\ x = v_1\ \mathsf{in}\ e_2 & \text{式}\end{array}$$

3.2　コンテキスト

コンテキストとは、着目している式の周りの計算のことである。着目している式が [] (ホール) であり、その式を受け取る周りの計算を E で表す。E は一般的なコンテキスト、F はホールを囲む $\langle\rangle$ がないコンテキストである。

$$\begin{array}{rcl} E & ::= & [] \mid E\ e \mid v\ E \mid \langle E\rangle\\ F & ::= & [] \mid F\ e \mid v\ F\end{array}$$

3.3　簡約規則

簡約規則とは、式の計算を進めるための規則である。4 つ目の shift を含む規則では、ホールを囲む $\langle\rangle$ がないコンテキスト F が $\langle\rangle$ で囲まれている。よって、k に束縛される継続は $\lambda x.\,\langle F[x]\rangle$ となっている。

$$\begin{array}{rcl} E[(\lambda x.\,e)\ v] & \rightsquigarrow & E[e[v/x]]\\ E[\langle v\rangle] & \rightsquigarrow & E[v]\\ E[\mathsf{let}\ x = v\ \mathsf{in}\ e] & \rightsquigarrow & E[e[v/x]]\\ E[\langle F[\mathcal{S}k.\,e]\rangle] & \rightsquigarrow & E[\langle \mathsf{let}\ k = \lambda x.\,\langle F[x]\rangle\ \mathsf{in}\ e\rangle]\end{array}$$

3.4　型

式の型と、多相な型を扱うための型スキームを定式化する。

型スキームは、$\forall t.A$ という形で、A に含まれる型変数 t は任意の型に置き換えることができることを表す。例えば、一般的な型付き λ 計算における恒等関数の型スキームは $\forall t.t\to t$ であり、コンテキストによって $\texttt{int}\to\texttt{int}$ 型や $\texttt{bool}\to\texttt{bool}$ 型などに具体化される。

shift/reset を含む体系の型システムでは、型に answer type という情報が加わる。answer type とは $\langle\rangle$ までのコンテキストの返す型のことである。例えば、コンテキスト $\langle 3+[]\rangle$ について考えると、本来このコンテキストは $\texttt{int}\to\texttt{int}$ 型を持つ。しかし、ホールに $\mathcal{S}k.\,\mathsf{isZero}\ (k\ 1)$ を入れた場合、このコンテキストは $\texttt{int}\to\texttt{bool}$ 型を持つ。このように、コンテキストの返す型が変化することを answer type の変化という。変化前の answer type が α 、変化後の answer type が β とし、τ_1 型の引数を受け取り、τ_2 型の値を返す関数の型は $\tau_1\to\tau_2\ [\alpha,\beta]$ となる。$\tau_1,\tau_2,\alpha,\beta$ は $\forall$ を用いて多相にすることができる。

以上を踏まえて、式の型と型スキームは以下のようになる。

$$\begin{array}{rcll} \alpha,\beta,\tau & ::= & t \mid \texttt{int} \mid \tau_1\to\tau_2\ [\alpha,\beta] & \text{式の型}\\ A & ::= & \alpha \mid \forall t.A & \text{型スキーム}\end{array}$$

3.5　型環境

型環境 Γ は、変数とその型または型スキームの対応を表す。

$$\Gamma ::= \bullet \mid \Gamma, x : A$$

3.6　型付け規則

要点となる (TLet), (TShift) の型付け規則と、先行研究である浅井・亀山 [1] の (TShift) の型付け規則の比較を以下に示す。その他の規則は先行研究と同様である。

本研究：

$$\frac{\Gamma, k : \mathsf{Gen}(\tau\to\alpha\ [t,t], \Gamma) \vdash e : \gamma\ [\gamma,\beta]}{\Gamma \vdash \mathcal{S}k.\,e : \tau\ [\alpha,\beta]}\ (\mathsf{TShift})$$

浅井・亀山 [1]：

$$\frac{\Gamma, k : \forall t.\tau\to\alpha\ [t,t] \vdash e : \gamma\ [\gamma,\beta]}{\Gamma \vdash \mathcal{S}k.\,e : \tau\ [\alpha,\beta]}\ (\mathsf{TShift})$$

(TLet) の型付け規則：

$$\frac{\Gamma \vdash_p v_1 : \tau_1 \quad \Gamma, x : \mathsf{Gen}(\tau_1, \Gamma) \vdash e_2 : \tau\ [\alpha, \beta]}{\Gamma \vdash \mathsf{let}\ x = v_1\ \mathsf{in}\ e_2 : \tau\ [\alpha, \beta]}\ (\mathsf{TLet})$$

$\mathsf{Gen}(\rho, \Gamma)$ は、ρ の型変数のうち、型環境 Γ に現れない型変数を多相にする。これは次のように定式化される。$\{t_1, \cdots, t_n\} = \mathtt{FTV}(\rho) - \mathtt{FTV}(\Gamma)$ について、$\mathsf{Gen}(\rho, \Gamma) \equiv \forall t_1. \cdots \forall t_n. \rho$

$\Gamma \vdash_p e : \tau$ は、任意の α について、$\Gamma \vdash e : \tau\ [\alpha, \alpha]$ が導出できることを表す。

先行研究の型システムでは捕捉した継続 k の answer type のみが多相であったのに対し、本研究では捕捉した継続の型のうち、型環境に現れない型変数を多相にする形で拡張した。2 節で挙げた例は、先行研究の型システムでは型付けできないが、本研究の型システムでは型付け可能である。Let 多相においては値を持つ x を多相にしており、k も値であるため、同様に多相にできる。

4 型安全性

型安全性とは、型付け可能なプログラムは型エラーを発生せずに実行できる性質のことである。この性質により、型システムはプログラムの信頼性を高めることができる。

型安全性は型保存性と進行性から成り立つ。本研究では、提案した型システムにおいて、型保存性が成り立つことを示した。

証明は、浅井・亀山 [1] の型保存性の証明と同様の手順を用いる。

定理 1 (型保存性) $\Gamma \vdash R : \tau$ かつ $R \rightsquigarrow^* e$ ならば、$\Gamma \vdash e : \tau$ が成り立つ。

型保存性は、型が付いた式は簡約されても型が保存されることを表す。証明には以下の 3 つの補題を用いる。補題はいずれも式の構造に関する帰納法で証明でき、変数名の衝突が生じる際に新しい変数名に α 変換を行う点だけ注意する必要がある。

補題 2 (型環境の弱化) $\Gamma_1 \subset \Gamma_2$ とする。$\Gamma_1 \vdash e : \tau\ [\alpha, \beta]$ ならば $\Gamma_2 \vdash e : \tau\ [\alpha, \beta]$ が成り立つ。

補題 3 (単相の変数の代入) $\Gamma_1 \subset \Gamma_2$, $\Gamma_1 \vdash_p v : \gamma$ とする。$\Gamma_2, x : \gamma \vdash e : \tau\ [\alpha, \beta]$ ならば $\Gamma_2 \vdash e[v/x] : \tau\ [\alpha, \beta]$ が成り立つ。

補題 4 (多相の変数の代入) $\Gamma_1 \subset \Gamma_2$, $\Gamma_1 \vdash_p v : \gamma$ とする。$\Gamma_2, x : \mathsf{Gen}(\gamma, \Gamma_1) \vdash e : \tau\ [\alpha, \beta]$ ならば $\Gamma_2 \vdash e[v/x] : \tau\ [\alpha, \beta]$ が成り立つ。

簡約規則の上から 3 つの規則は、

$E[\langle v \rangle] \rightsquigarrow E[v]$：型付け規則より明らか

$E[(\lambda x.\, e)\, v] \rightsquigarrow E[e[v/x]]$：補題 3

$E[\mathsf{let}\ x = v\ \mathsf{in}\ e] \rightsquigarrow E[e[v/x]]$：補題 4

より証明できる。

$E[\langle F[\mathcal{S}k.\, e] \rangle] \rightsquigarrow E[\langle \mathsf{let}\ k = \lambda x.\, \langle F[x] \rangle\ \mathsf{in}\ e \rangle]$ については、F がいろいろな計算を含む可能性があり、直接証明することが難しい。そこで、以下の新しい簡約規則 $\to$ を導入する。

$$\begin{array}{rcl}
(\mathcal{S}k.\, e_1)\, e_2 & \to & \mathcal{S}k'.\, \mathsf{let}\ k = \lambda u.\, \langle k'\, (u\, e_2) \rangle\ \mathsf{in}\ e_1 \\
v\, (\mathcal{S}k.\, e_1) & \to & \mathcal{S}k'.\, \mathsf{let}\ k = \lambda u.\, \langle k'\, (v\, u) \rangle\ \mathsf{in}\ e_1 \\
\langle \mathcal{S}k.\, e \rangle & \to & \langle \mathsf{let}\ k = \lambda u.\, \langle u \rangle\ \mathsf{in}\ e \rangle \\
\langle (\lambda u.\, \langle F[u] \rangle)\, e \rangle & \to & \langle F[e] \rangle \\
\mathsf{let}\ x = v\ \mathsf{in}\ e & \to & e[v/x]
\end{array}$$

この簡約規則 $\to$ について、任意のコンテキスト C に対して、$R \to e$ ならば $C[R] \to C[e]$ が成り立つ。

$\to$ を用いて、以下の手順で型保存性を証明する。

(1) $E[\langle F[\mathcal{S}k.\, e] \rangle] \rightsquigarrow E[\langle \mathsf{let}\ k = \lambda x.\, \langle F[x] \rangle\ \mathsf{in}\ e \rangle]$ を $\to$ に分解する。つまり、$E[\langle F[\mathcal{S}k.\, e] \rangle] \to^* E[\langle \mathsf{let}\ k = \lambda x.\, \langle F[x] \rangle\ \mathsf{in}\ e \rangle]$ を示す。

(2) $\to$ がそれぞれ型保存性を満たすことを示す。

簡約規則 $\to$ では、コンテキスト F を分解し、少しずつ `shift` の本体に加える。例えば、$(\mathcal{S}k.\, e_1)\, e_2 \to \mathcal{S}k'.\, \mathsf{let}\ k = \lambda u.\, \langle k'\, (u\, e_2) \rangle\ \mathsf{in}\ e_1$ では、左辺では $\mathcal{S}k.\, e_1$ のコンテキストに含まれる e_2 を右辺では $\mathcal{S}k'. \cdots$ の本体に移動させている。k は引数に e_2 を適用し、その結果を k' に渡すことと等しい。

$\to$ の 1 ステップの簡約は変化が小さく、型保存性を示すことが比較的容易となる。

(1) は F に関する帰納法で証明できる。(2) はほとんどのケースにおいて左辺と右辺の導出を考えることで証明できる。(2) の証明において、$\langle (\lambda u.\, \langle F[u] \rangle)\, e \rangle \to \langle F[e] \rangle$ のケースのみ補題 5 を用いる。

補題 5 $\Gamma, u : \tau_1 \vdash F[u] : \alpha\ [\beta, \gamma]$ かつ $\Gamma \vdash e : \tau_1\ [\gamma, \tau]$ ならば $\Gamma \vdash F[e] : \alpha\ [\beta, \tau]$

この補題は次のように捉えることができる。u が値であるもとで、$F[u]$ は answer type γ を返す。e は answer type γ を τ に変化させるため、$F[e]$ の answer type は τ に変化する。

5 まとめと今後の展望

本研究では、`shift/reset` を含む体系の多相な型システムを拡張し、捕捉した継続の型のうち型環境に現れない型変数を多相にした型システムを与え、その型保存性を証明した。

今後の展望として、進行性や他の性質の証明が挙げられる。また、`control/prompt` を含む体系や 4 種類の限定継続演算子を含む体系において、多相な型システムの構築について考えたい。

参考文献

[1] Kenichi Asai and Yukiyoshi Kameyama. Polymorphic delimited continuations. In Zhong Shao, editor, *Programming Languages and Systems*, pp. 239–254, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[2] Oliver Danvy and Andrzej Filinski. Representing control: a study of the CPS transformation. *Mathematical Structures in Computer Science*, Vol. 2, No. 4, p. 361–391, 1992.