Vol. 48 No. SIG 0(PRO 34) 情報処理学会論文誌：プログラミング 2007

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

鈴木輝信[†] 亀山幸義[†]

本論文は，関数型プログラム言語におけるコントロールオペレータを対象とし，これに対する型システムの構築とその性質について述べる．本論文で対象とする shift/reset は，限定継続を操作するコントロールオペレータであり，その意味が CPS 変換に基づいて厳密に定義されているため，形式的に扱いやすいという特徴がある．shift/reset は様々な制御構造を表現できるが，複数の異なる目的で shift/reset を用いると，それぞれの shift/reset が干渉しあい，プログラマの意図通りにプログラムが実行されない場合がある．そこで，それぞれの shift/reset を区別するため，shift/reset に自然数のレベルを付与して階層化することが提案されている．

階層化 shift/reset に対して，これまで，型システムやその性質について十分には検討されておらず，ML などの型をもつプログラム言語に階層化 shift/reset を直接導入することはできなかった．我々は，先行研究において，レベル 2 の shift/reset の型システムを構築したが，この方式では，1 つの項の型付けにおいて，レベルの指数関数の個数の型を必要とするため，レベル 3 以上の型システムに拡張することは困難であった．

本論文では，任意レベルの階層化 shift/reset に対する型システムを提案し，型システムの健全性などの望ましい性質が成立することを示す．これにより，型があるプログラム言語に階層化 shift/reset を導入できるようになると考えられる．

Construction of a Type System for Layered Control Operators

Terunobu Suzuki[†] and Yukiyoshi Kameyama[†]

We construct a type system for control operators in functional programming languages. Specifically, we treat the layered version of the control operators shift and reset whose semantics is defined in terms of (iterated) CPS translations. While shift and reset are capable of representing various control structures, we cannot have two or more different uses of shift and reset in a single program since they may interfere with each other. In order to avoid this interference, shift and reset should be layered, that is, we should assign to each occurrence of shift and reset a natural number which designates its level.

Although it is apparent that a sound type system for layered shift and reset is necessary to introduce them to practical programming languages, no work has ever tried to construct a sufficiently expressive type system for them. In this paper we refine our previous work which solved this problem for the case of level-2, and propose a type system for an arbitrary level. We show that desirable properties such as type soundness hold for this type system, which enables one to introduce the layered shift and reset to statically typed programming languages.

1. はじめに

コントロールオペレータは，関数型プログラムの制御を行うための言語プリミティブであり，様々なプログラム言語に含まれている．例えば，例外 (Standard ML の exception, Java の try-catch-finally など)，第一級継続 (Scheme や Standard ML/NJ の call/cc) が，これに該当する．

本研究が対象とする shift/reset は，様々な制御構造を表現できる☆コントロールオペレータであり，「継続」(continuation) すなわち「計算の残り」を表す概念をプログラマが直接操作できるようになる．shift/reset は，CPS 変換に基づいて意味が定義されているため，形式的に扱いやすく，近年，盛んに研究が行われている[?),?)]．shift/reset を含む計算体系に対する型システムとしては，Danvy と Filinski[?)] が提案したものがある．彼らの型システムでは，項の型付けの判断に，項

† 筑波大学システム情報工学研究科コンピュータサイエンス専攻
Department of Computer Science, Graduate School of Systems and Information Engineering, University of Tsukuba

☆ Filinski[?)] は，一定の条件のもとで，任意の制御効果 (control effect) が shift/reset を用いて表現できることを示した．

そのものの型だけでなくその継続の型を加えている．

shift/reset を複数の異なる目的で利用した場合にそれぞれの shift/reset の計算が干渉しあって，プログラマの意図通りに実行されないことがある．これを解決するために階層化 shift/reset [?] が提案された．階層化 shift/reset は，階層化されない shift/reset と同様に，その意味が CPS 変換により厳密に与えられているという利点があるが，その複雑さのため，型システムやその性質に関しては十分には検討されてこなかった．階層化 shift/reset に対する型システムは，型のある言語への導入のためだけではなく，型によってプログラムを抽象化することによりプログラムの理解を助けるという効果もある．我々は階層化 shift/reset に対する型システムについて研究しており，先行研究[?] においてレベル 2 の階層化 shift/reset に対して型システムを提案した．しかし，この型システムでは，1 つの型判断が，レベルの指数関数の個数の型を必要とし，この多数の型を適切に記述する記法がなかったため，レベル 3 以上の階層化 shift/reset に対応するのは難しかった．

本研究では，型スキームと呼ばれる記法を導入することにより上記の問題点を克服し，任意のレベルの shift/reset に対する型システムを提案する．さらに，この型システムが型の健全性や CPS 変換による型の保存など，型システムとして必要とされる良い性質を満たすことを証明する．これにより，階層化 shift/reset を静的な型システムをもつ言語に導入できるようになると考えている．

本論文の構成は以下の通りである．**??**章では，shift/reset の概要と階層化について述べる．**??**章では，対象とする λ 計算に階層化 shift/reset を追加した言語 λS_m と CPS 変換の定義を与えた後，本論文が提案すると型システムの形式的な定義を述べる．**??**章では，本論文のた型システムが満たす性質を示し，**??**章で，まとめと今後の課題を述べる．

2. shift/reset と階層化

この章では，shift/reset の概要，簡単なプログラム例，階層化の必要性について述べる．

2.1 コントロールオペレータ shift/reset

前章で，shift/reset は継続を扱うためのコントロールオペレータであると述べた．継続は，対象となる式の計算後からプログラム終了までの計算を表す概念であり，shift/reset はこの継続をプログラムの中から利用する機構を与える．

これと同様のコントロールオペレータとして，Scheme 等のプログラム言語における call/cc がある．call/cc は，継続を変数に束縛するため，継続をプログラム中で利用することができる．call/cc と shift/reset の違いは，call/cc が獲得する継続はプログラムが終了するまでの計算全てであるが，shift/reset が獲得する継続は，プログラムが終了するまでの計算の一部分にできることである．後者の継続を限定継続 (delimited continuation) と呼び，より精密にプログラムを制御することができる．

shift オペレータは，k を変数，M を式とするとき，$Sk.M$ という形で表現される．この式はその (限定) 継続を k に束縛して，M を計算するという式である．一方，継続を束縛する際にその範囲を限定するのが reset であり $\langle M \rangle$ と表す．shift で継続を捕捉する際には，reset までの範囲の継続が対象となる．

これらの式の意味を以下の例で考える．

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

最初の式は，shift で捕えた継続を利用しない場合である．$Sk.(3+4)$ を実行する際の残りの計算は，$\langle 2+[\,]\rangle+1$ であるが，shift が捕える継続は最も近い reset までであるため，$\langle 2 + [\,]\rangle$ (2 を加える) という継続 (を関数で表現したオブジェクト) が k に束縛される．次に，shift の内側の計算 $(3+4)$ を行い，この部分の計算が終わると reset の継続 ($[\,]+1$) に飛ぶ．結果として，2 を加える部分は実行されず，8 という計算結果を得る．

2 つ目の式は，shift の内側の計算で k を利用した場合である．k には，「2 を加える」という関数が束縛されており，この計算を 2 回行った後に 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い，計算結果は 12 になる．

2.2 shift/reset の階層化

階層化 shift/reset は，前節の shift/reset に自然数のレベルを割り当て，使用目的の異なる shift と reset を区別できるようにしたコントロールオペレータである[?]．階層化した shift は $S_n k.M$, reset は $\langle M \rangle_n$ という形になり，それぞれの n がその shift, reset のレベルになる．

階層化 shift/reset の例を以下に示す．

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

もし，この式の shift/reset のレベルが無いとすると，k に束縛される継続は 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

のレベルが 2 なので，レベルが 1 の reset を無視して，外側のレベルが 2 の reset までの継続を捕えることになり，計算結果は $1+((1+3+2)+3+2) = 12$ になる．この例ではレベル 2 までしかないが，一般に $n > 0$ なる任意の自然数に対して，レベル n の shift/reset が定義される．

2.3 階層化 shift/reset の例

階層化 shift/reset の利用法を説明するため，文献 **?**) の例を取り上げる．まず，非決定的な計算を行うプログラムを shift/reset で表現したものを以下に示す．

$$\begin{aligned}\mathrm{filp}(x) &= S_1k.(k(\mathrm{true}); k(\mathrm{false}); \mathrm{fail}(\_))\\ \mathrm{fail}(x) &= S_1k.\text{“no”}\\ \mathrm{choice}(n) &= \text{if } n < 1 \text{ then } \mathrm{fail}(\_)\\ &\quad\ \text{else if } \mathrm{filp}(\_) \text{ then } \mathrm{choice}(n-1)\\ &\quad\ \text{else } n\end{aligned}$$

ここで $\_$ は任意の値であり，$a; b$ は式 a と式 b を順番に実行して b の値を結果とする式である．

このような関数を定義し，$\langle \mathrm{print}(\mathrm{choice}(3))\rangle_1$ というプログラムを実行すると，1,2,3 がこの順で印刷される．すなわち，関数 flip は非決定的選択を実現する関数である．

次に，生成された数を収集するための関数を以下のように定義する．

$$\mathrm{emit}(n) = S_1k.\mathrm{cons}(n, k(\mathrm{nil}))$$

例えば，$\langle \mathrm{emit}(1); \mathrm{emit}(2); \mathrm{emit}(3)\rangle_1$ という計算を行うと自然数のリスト $[1; 2; 3]$ を返す．

そこで，$\langle \mathrm{emit}(\mathrm{choice}(3))\rangle_1$ とすると，自然数のリスト $[1; 2; 3]$ を返すことを期待する．しかし，実際に計算してみると，[1], [2], [3] という別々のリストは生成されるが，これらが収集されて一つのリストになることはない．

このように，shift/reset を用いた関数を複数組み合わせることにより，これらが干渉して意図通りにプログラムが動かなくなってしまうことがある．そこで，階層化 shift/reset を用いて，emit の定義を以下のように変更する．

$$\mathrm{emit}(n) = S_2k.\mathrm{cons}(n, k(nil))$$

$\langle \mathrm{emit}(\mathrm{choice}(3))\rangle_2$ を実行すると，期待通り $[1; 2; 3]$ というリストが返る．

階層化していない shift/reset を複数用いる場合，shift/reset 同士の干渉は容易に起きる問題であり，その解決のために階層化することは，現実のプログラミングを行う際には非常に重要であると考える．

3. 階層化 shift/reset の形式化と型システム

この章では λ 計算に階層化 shift/reset を追加した言語 λS_m を形式的に定義し，さらにその CPS 変換と本論文で提案する型システムについて述べる．

3.1 構文と評価

m を正の自然数とする．本論文が対象とする言語 λS_m は，λ 計算に m 以下のレベルの shift と reset を追加した言語である．λS_m の項 M と値 V は以下のように定義される．

$$\begin{aligned}M ::=\ & x && \text{(変数)}\\ \mid\ & \lambda x.M && \text{(}\lambda\text{-抽象)}\\ \mid\ & M_1M_2 && \text{(適用)}\\ \mid\ & S_nk.M && (\mathrm{shift}, 1 \le n \le m)\\ \mid\ & \langle M\rangle_n && (\mathrm{reset}, 1 \le n \le m)\\ V ::=\ & x\\ \mid\ & \lambda x.M\end{aligned}$$

$S_nk.M$ はレベル n の shift を用いた項であり，この項の中で，変数 k が束縛される．shift が獲得する継続の範囲を限定するのが reset であり，$\langle M\rangle_n$ と表現される．変数は λ あるいは S_n により束縛され，α 同値性や代入 $M\{x := M'\}$ は，通常のラムダ計算と同様に定義する．また，α 同値な項は同一視する．

次に，λS_m に対する文脈を 3 種類導入する．

$$C ::= [\ ] \mid \lambda x.C \mid MC \mid CM \mid \langle C\rangle_n \mid S_nk.C$$
(文脈)

$$E ::= [\ ] \mid VE \mid EM \mid \langle E\rangle_n$$
(評価文脈)

$$E^n ::= [\ ] \mid VE^n \mid E^nM \mid \langle E^n\rangle_i \quad (i < n)$$
(レベル n の評価文脈)

C は一般の文脈であり，[ ] は穴 (hole) を表す．E は値呼び計算戦略における評価文脈である．ただし，穴を取り囲む reset があってもよいように拡張している．E^n は，穴を囲む reset のレベルを n より小さいレベルに制限した評価文脈であり，次に述べる shift の評価規則において利用する．文脈 C に対し，$C[M]$ は文脈 C の穴を項 M で埋めた項をあらわす．

λS_m の評価規則を次に示す．ただし，$n \le j$ とする．

$$E[(\lambda x.M)V] \to E[M\{x := V\}]$$
$$E[\langle E^n[S_nk.M]\rangle_j] \to E[\langle M\{k := \lambda x.\langle E^n[x]\rangle_n\}\rangle_j]$$
$$E[\langle V\rangle_n] \to E[V]$$

最初の規則は，値呼び λ 計算の通常の β 簡約である．

2 番目の規則は shift の計算を表すもので，shift から見て一番内側の reset(ただし，reset のレベルは n 以上) までの継続 E^n を捕捉し，それを関数 $\lambda x.\langle E^n[x]\rangle_n$ に変形して変数 k に束縛する．レベル n の shift が獲得する継続は，レベル n 以上の reset によって限定されるので，$n \le j$ という条件がついている．たとえば，

$\langle E^2[\langle E^1[S_2k.M]\rangle_1]\rangle_2$ という式における S_2 が獲得する継続は，外側のレベル 2 の reset までである．

k に束縛される関数として $\lambda x.\langle E^n[x]\rangle_n$ のかわりに $\lambda x.E^n[x]$ を採用すると，shift/reset とは意味が異なる control/prompt と呼ばれるコントロールオペレータになる[?]．control/prompt は，部分継続を表現する最初のコントロールオペレータであるが，その意味論が shift/reset より複雑であったことを一因として，今日では shift/reset が主に使われている．

評価規則の最後のものは，reset の内側の項が値になった場合に，その reset を除去する規則である．

3.2 CPS 変換

CPS 変換は，プログラムを CPS(Continuation Passing Style) とよばれる特別な形式に変換するプログラム変換である．CPS 形式のプログラムは，計算の途中結果全てに名前が付けられ，全ての関数呼出しが末尾呼出しであるという特徴があり，理論的な解析やコンパイラの中間言語等で幅広く利用されている．

λS_m のプログラムに対する CPS 変換は，通常の CPS 変換を $m+1$ 回繰返すことで与えられる[?]．レベル n の shift/reset に対して CPS 変換を行うと，レベルが $n-1$ である shift/reset に変換され，特に $n=1$ の時は，CPS 変換により shift/reset が消去され，純粋なラムダ項になる．そこで，λS_m のプログラムに対して，$m+1$ 回☆CPS 変換を繰返すことにより，純粋なラムダ項になる．この繰返し全体を 1 つの変換としたものが CPS 変換 C_{m+1} であり，その定義を**図??**で与える．

図**??**でわかるように，この CPS 変換は，$m+1$ 個の継続変数 $\kappa_1, \cdots, \kappa_{m+1}$ を受け取って変換後の項を生成している．κ_2 をメタ継続と呼び, κ_3 をメタメタ継続等と呼ぶ. この CPS 変換は，型のない λS_m の項全体に対して定義され，CPS 変換後の項は再帰型を利用して型をつけられる．図**??**の上部にその型を記述した．C_{m+1} は，変換前の項 (型は $Term$) と $m+1$ 個の継続 (型は $Cont_i$) を受けとり，$Answer$ 型の値を返す関数である．変換前の項に型がないことに対応して，値の型 Val の定義式で再帰型が現れている☆．

shift と reset 以外の項に対する CPS 変換においては，メタ継続など 2 個目以降の継続を η 変換により消去することができ，通常の CPS 変換を 1 回行って得られる項と同じである．shift は 1 個目から n 個目までの継続をまとめて k に割り当てて，M の 1 個目から n 個目までの継続は初期化される．また，reset では 1 個目から n 個目の継続を $n+1$ 個目の継続にまとめる．これにより，レベルが n 以下の shift では継続が束縛されない．

3.3 先行研究の型システム

本研究の型システムについて述べる前に，先行研究の shift/reset に対する型システムについて述べる．

Danvy と Filinski は文献**?**) において，階層化していない shift/reset に対する型システムを与えた．彼らの型システムでは，項の型を考える際に継続の型を考慮するものであり，型判断は以下の形をしている．

$$\Gamma, \alpha \vdash M : \tau, \beta$$

Γ は型環境，M は項，τ, α, β は型である．これは，M の継続の型が $\tau \to \alpha$ である時に，計算結果の型は β であることを表す．shift/reset が無ければ $\alpha = \beta$ であるが，shift/reset を導入すると，$\alpha \neq \beta$ のことがある．この型システムの shift と reset の型付け規則は以下の通りである．

$$\frac{\Gamma[k \mapsto (\tau/\delta \to \alpha/\delta)], \sigma \vdash M : \sigma, \beta}{\Gamma, \alpha \vdash Sk.M : \tau, \beta}\ \text{shift}$$

$$\frac{\Gamma, \sigma \vdash M : \sigma, \tau}{\Gamma, \alpha \vdash \langle M \rangle : \tau, \alpha}\ \text{reset}$$

shift の規則では，k の型は関数型であり，その引数の型が τ，関数本体の継続の型が $\alpha \to \delta$, 計算結果の型は δ である．この規則を見ると，shift の継続の型 $\tau \to \alpha$ を k の型として割り当て，M の継続の型が $\sigma \to \sigma$ となっており，M の継続が初期化されていることが分かる．reset の規則では，reset の型 τ を M の結果の型として，shift で捕える範囲外にして，M の継続の型は $\sigma \to \sigma$ として初期化している，と見ることができる．このように見ると，shift/reset の計算が型の上で表現されていることが分かる．

Murthy[?] は，階層化 shift/reset に対する型システムを与えた．彼の型システムでの型判断は

$$\Gamma \vdash M : K_{\overline{\alpha}}[T]$$

という形である．ただし，$\overline{\alpha}$ は型の列 $\alpha_m \ldots \alpha_1$ の略記であり，$K_{\overline{\alpha}}[T]$ は以下のように定義される．

$$K_{[]}[T] = T$$
$$K_{\overline{\alpha}\tau}[T] = (T \to K_{\overline{\alpha}}[\tau]) \to K_{\overline{\alpha}}[\tau]$$

☆ m 回の CPS 変換で，階層化 shift/reset を全て消すことができるが，CPS 形式にならないため，さらにもう 1 回余分に変換を行っている

☆ Val の定義の右辺に Val が negative な位置に出現している．変換前の項に本論文で述べる型が付いている場合は，変換後に再帰型は必要なく，単純型付きラムダ計算の範囲内で型が付けられる．

$$
\begin{aligned}
C_{m+1} &: \mathit{Term} \to \mathit{Cont}_1 \to \ldots \to \mathit{Cont}_{m+1} \to \mathit{Answer} \\
\kappa_i &\in \mathit{Cont}_i = \mathit{Val} \to \mathit{Cont}_{i+1} \to \ldots \to \mathit{Cont}_{m+1} \to \mathit{Answer} \qquad (1 \le i \le m) \\
\kappa_{m+1} &\in \mathit{Cont}_{m+1} = \mathit{Val} \to \mathit{Answer} \\
\mathit{Val} &::= \mathit{Val} \to \mathit{Cont}_1 \to \ldots \to \mathit{Cont}_{m+1} \to \mathit{Answer}
\end{aligned}
$$

$$
\begin{aligned}
C_{m+1}[\![x]\!] &= \lambda\kappa_1 \ldots \kappa_{m+1}.\ \kappa_1 x \kappa_2 \ldots \kappa_{m+1} \qquad (x \text{ は変数}) \\
C_{m+1}[\![\lambda x.M]\!] &= \lambda\kappa_1 \ldots \kappa_{m+1}.\kappa_1(\lambda v\kappa'_1 \ldots \kappa'_{m+1}.(C_{m+1}[\![M]\!]\{x := v\})\kappa'_1 \ldots \kappa'_{m+1})\kappa_2 \ldots \kappa_{m+1} \\
C_{m+1}[\![M_1 M_2]\!] &= \lambda\kappa_1 \ldots \kappa_{m+1}.C_{m+1}[\![M_1]\!](\lambda f\kappa'_2 \ldots \kappa'_{m+1}. \\
&\qquad C_{m+1}[\![M_2]\!](\lambda a\kappa''_2 \ldots \kappa''_{m+1}. \\
&\qquad\quad f\ a\kappa_1\kappa''_2 \ldots \kappa''_{m+1})\kappa'_2 \ldots \kappa'_{m+1})\kappa_2 \ldots \kappa_{m+1} \\
C_{m+1}[\![\langle M \rangle_n]\!] &= \lambda\kappa_1 \ldots \kappa_{m+1}.C_{m+1}[\![M]\!]\theta_1 \ldots \theta_n \\
&\qquad (\lambda v\kappa'_{n+2} \ldots \kappa'_{m+1}.\theta_0 v\kappa_1 \ldots \kappa_{n+1}\kappa'_{n+2} \ldots \kappa'_{m+1})\kappa_{n+2} \ldots \kappa_{m+1} \\
C_{m+1}[\![S_n k.M]\!] &= \lambda\kappa_1 \ldots \kappa_{m+1}.(C_{m+1}[\![M]\!]\{k := p\})\theta_1 \ldots \theta_n\kappa_{n+1} \ldots \kappa_{m+1}
\end{aligned}
$$

ただし，

$$
\begin{aligned}
p &= \lambda v\kappa'_1 \ldots \kappa'_{m+1}.\theta_0 v\kappa_1 \ldots \kappa_n(\lambda w\kappa''_{n+2} \ldots \kappa''_{m+1}.\theta_0 w\kappa'_1 \ldots \kappa'_{n+1}\kappa''_{n+2} \ldots \kappa''_{m+1})\kappa'_{n+2} \ldots \kappa'_{m+1} \\
\theta_i &= \lambda v\kappa_{i+1} \ldots \kappa_{m+1}.\kappa_{i+1} v\kappa_{i+2} \ldots \kappa_{m+1} \qquad (0 \le i \le m) \\
\theta_{m+1} &= \lambda v.v
\end{aligned}
$$

図 1 λS_m の CPS 変換
Fig. 1 CPS-translation for λS_m

表 1 先行研究との比較
Table 1 Comparison of our work and previous work

		Answer Type の変化	
		非対応	対応
階層化	非対応	Filinski の実装 (1994)	Danvy,Filinski (1989)
	対応	Murthy (1992)	本研究の型システム

例として $m = 1$ の場合は，

$$\Gamma \vdash M : (T \to \alpha_1) \to \alpha_1$$

となる．この例からわかるように，上記の Danvy-Filinski の型システムに比べ，継続の返す型と計算結果の型が同じという制約が加わっている．これらの型が異なるような項 (いわゆる Answer Type が変化する項) に対して，Murthy の型システムでは型をつけることができない．ML における printf の型付け[?)] を shift/reset で表現した場合など，興味深いプログラムで，Murthy の制約のもとでは型付けができないものが存在するため，より多くの項を型付けできる型システムが必要である．

これらの先行研究と今回提案する型システムの関係を**表??**にまとめる．

これまで我々は，Danvy-Filinski の型システムを拡張し，レベル 2 の階層化 shift/reset に対応した型システムを構築した[?)]．型判断は，レベル 2 に対応して 2 種類の継続の型と結果の型を含む必要があったため，以下の形になっている．

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

項 M の 1 個目の継続の型が $\tau \to (\alpha_1 \to \alpha_2) \to \alpha_3$ であり，2 個目の継続の型が $\beta_1 \to \beta_2$, 結果の型は β_3 となる．例として，この型システムにおけるレベル 2 の shift に対する型付け規則をあげる．

$$\frac{\Gamma'; \gamma_1, \gamma_2, \gamma_2 \vdash M : \gamma_1, \gamma_3, \gamma_3, \beta}{\Gamma; \overline{\alpha} \vdash S_2 k.M : \tau, \alpha_1, \alpha_2, \beta}\ \mathrm{shift}_2$$

ただし，$\Gamma' = \Gamma[k \mapsto (\tau/\overline{\delta} \to \alpha_3/\overline{\delta})]$ とする．

このようにして，レベル 2 の shift/reset に対応した型システムを構築したが，これをレベル n まで単純に拡張すると型判断に含まれる型の個数が非常に多くなり，型付け規則を表現することだけでも簡単なことではなくなってしまう．レベル 1 で 3 個, レベル 2 で 7 個の型が必要であり，一般にレベル n の型システムでは，必要な型の個数が $O(2^n)$ となる．型の爆発をおさえるには，これらの多くの型を適切に表現する記法が不可欠である．

3.4 λS_m に対する型システム

本研究では，型システム構築にあたって，以下の原則を設定した．

- 型システムの健全性 (Subject Reduction と Progress) が成立すること．
- 意味をもつ項は，できる限り多く型付けできるこ

と．少なくとも応用上，有用な項は型付けできること．

第一の要求は，静的な型システムをもつプログラム言語のコアとなる体系を目指している以上，当然のことであろう．

第二の要求も当然のことであるが,「意味をもつ項」が何かについては検討を要する．階層化 shift/reset の意味は，CPS 変換により与えられており，CPS 変換をほどこした後の項は，通常の (shift/reset のない) ラムダ項になる．そこで，我々は,「項 M を CPS 変換したときに，単純型付きラムダ計算の項になる (単純型付きラムダ計算で型が付く) こと」を「項 M が意味をもつ」ことと定めた．

この考え方は，Danvy-Filinski の型システムにおける考え方と同じであるが，上述したように，shift/reset の型システムに関する既存の研究の多くは，この要求を満たしていない．本論文で提案する型システムは，既存の研究で型付けできる範囲を全て含んだ上で，これまで考えられてきたプログラム例を (我々が試みた範囲では) すべて型付けできるという意味で既存の研究より強力である．

3.4.1 型と型スキーム

本論文では，レベル m の shift/reset に対する型システムを表現するため，型スキーム を導入した．これは，複数ある継続の型を木構造で表現するとともに，必要以上に型情報を型判断に加える必要が無い工夫を加えたものである．また，shift/reset の型付け規則が λS_m の m によらず同じ形である，という特徴を持っている．

型 τ と型スキーム T を以下のように定義する．

$$\begin{aligned} \tau &::= b && \text{(基本型)} \\ &\mid \tau \to T && \text{(関数型)} \\ T &::= * && \text{(基本型スキーム)} \\ &\mid (\tau, T_1, T_2) && \text{(3 つ組)} \end{aligned}$$

型スキームにおける 3 つ組 (τ, T_1, T_2) は，2 分木を表現しており，木の根は型 τ であり，左部分木は T_1, 右部分木は T_2 である．これは，直感的には，CPS 変換により，$(\tau' \to T_1') \to T_2'$ と変換される型を表現したものである．(ただし，τ' 等は τ 等を CPS 変換して得られる型である)．型スキーム $*$ は，“don't care” を意図した定数であり，Answer Type (図**??**の型 *Answer*) を含む任意の型に CPS 変換される型を表現している．すなわち，n 未満のレベルの shift/reset しか持たないプログラムを型付けするとき，(厳密には，さらに，そのプログラムに含まれる自由変数があらわすエフェクトも レベル n 未満であるとき)，$n+1$ 以上の深さの型の構造を記述する必要はないという事実に基づき，複雑な型を $*$ を使って簡潔に記述するために使われる．

型スキーム T に対して $T[l]$ は左部分木，$T[r]$ は右部分木を表す．また，$T[r^l]$ を以下のように定義する ($T[l^l]$ も同様に定義する)．

$$\begin{aligned} T[r^0] &= T \\ T[r^l] &= (T[r^{l-1}])[r] \quad (l > 0) \end{aligned}$$

型スキーム T と $s = l, r, l^n, r^n$ に対して，その部分木 $T[s]$ を型スキーム S で置き換えた型スキームを $T[s \leftarrow S]$ と表記する．よく現れる型スキームの略記として $init(n,T,\vec{\gamma},\vec{S})$ を以下のように定義する．ただし，$0 \le n \le m$, T は型スキーム, $\vec{\gamma}$ は n 個の型 $\gamma_1, \cdots, \gamma_n$ の列，$\vec{S}$ は n 個の 型スキーム $S_1, \cdots, S_n$ の列である．

$$\begin{aligned} init(0, T, \vec{\gamma}, \vec{S}) &= T \\ init(n+1, T, \vec{\gamma}, \vec{S}) &= \\ & (\gamma_1, (\gamma_1, S_1, S_1), init(n, T, \vec{\gamma}', \vec{S}')) \end{aligned}$$

ただし，$\vec{\gamma}'$ と $\vec{S}'$ は，それぞれ $\vec{\gamma}$ と $\vec{S}$ の先頭要素を取り除いた列である．後に述べる型付け規則において $init$ を使うときは，第 3, 第 4 引数は，型や型スキームをあらわすフレッシュなメタ変数の列を取る．これは，レベル n の shift や reset の型付けにおいて，レベル n 未満の部分は，$(\gamma, (\gamma, S, S), \_)$ の形であることを意味する．

型スキームの例を**図??**に示す．ここで，a, b, c, d, γ_i は型，A, B, C, D, X, S_i は 型スキーム, $T[r]$ は T の右の部分木を，$T[r^2]$ はそのさらに右部分木を取る操作である．また，$T[r \leftarrow X]$ は T の右部分木を X に置き換えた木を表し，同様に $T[r^2 \leftarrow X]$ は $T[r^2]$ を X に置き換えた木を表す．$init(2, X, \_, \_)$ は図**??**のような木を構成する．$init$ の第一引数が大きくなると，木がより下へ伸びることになる．

3.4.2 型付け規則

本研究で提案する λS_m に対する型システムの型判断と型文脈を，それぞれ以下のように定義する．ただし，M は項，T は型スキーム, x は変数，t は型である．

$$\Gamma \vdash M : T$$

$$\Gamma ::= [\,] \mid \Gamma[x \mapsto \tau]$$

型判断 $\Gamma \vdash M : T$ は，型環境 Γ 下で項 M の型スキームが T であることを表す．

型付け規則の定義を**図??**に与える．

図**??**において，shift と reset の規則で現れる $init$ は，n 番目までの継続が初期化されることに対応する型である．shift や reset に対する型スキームが右部分木と左部分木で共通しているところは，1 番目の継続

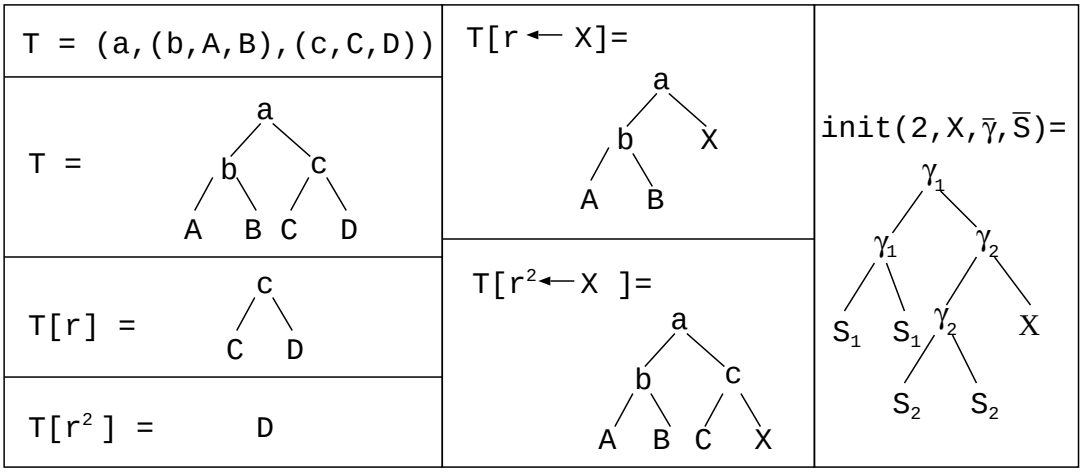


図 2 型スキームの例
Fig. 2 Examples of Type Scheme

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash x : (\tau, T, T)}\ var \qquad \frac{\Gamma[x \mapsto \tau] \vdash M : T}{\Gamma \vdash \lambda x.M : (\tau \to T, S, S)}\ \lambda$$

$$\frac{\Gamma \vdash M_1 : (\alpha \to (\tau, T, S), U, W) \quad \Gamma \vdash M_2 : (\alpha, S, U)}{\Gamma \vdash M_1 M_2 : (\tau, T, W)}\ app$$

$$\frac{\Gamma \vdash M : init(n, (\tau, T[r^n], S), \vec{\gamma}, \vec{S})}{\Gamma \vdash \langle M \rangle_n : (\tau, T, T[r^n \leftarrow S])}\ reset_n,\ n \le m$$

$$\frac{\Gamma' \vdash M : init(n, U[r^{n-1}], \vec{\gamma}, \vec{S})}{\Gamma \vdash S_n k.M : (\tau, U[r^{n-1} \leftarrow (\alpha, T, S)], U)}\ shift_n,\ n \le m$$

$$\Gamma' = \Gamma[k \mapsto \tau \to (\alpha, W[r^n \leftarrow T], W[r^n \leftarrow S])]$$

図 3 型付け規則
Fig. 3 Typing Rules

に 2 番目以降の継続を適用できることに対応している．

3.4.3 型付けの例

前節で定義した型システムを用いて，実際のプログラムの型付けを行った例を示す．ここで型付けを行ったプログラム M は以下のものである．

$$M = \langle \langle emit_2(choice\ 3); \text{“no”} \rangle_1; nil \rangle_2$$

choice は**??** 節で定義した関数であり，$emit_2$ はレベル 2 の shift を用いた emit を表している．ここでは，整数などの基本型や if-then-else 等に対する型付け規則が適宜追加されていると仮定する．

まず，関数 choice に対して次の型が導出できる．

$$\vdash \text{choice} : (int \to (int, (string, T, T), (string, T, T)), T, T)$$

この時，**図??**に示す導出により，$\vdash M : (\text{list}\ int, U, U)$ が導出できる．ただし，list int は自然数のリスト型であり，U は任意の型スキームである．この導出における emit_2 の本体の型を見ると，深さ 1 の Answer Type ($string$) と深さ 2 の Answer Type (list int) が異なっており，階層化 shift/reset に対応した型システムであることがわかる．

この例は階層化レベルが 2 であるが，Biernacka らによる Normalization by Evaluation[?)] は，階層化レ

ベルが 5 の例題であり，我々の型システムはこの例に対しても型がつけられることを確認している．このような例は，実用的なプログラムにおいては頻出するものと考えられるが，従来研究では型付けすることができず，本研究による型システムの優位性を示すものである．

4. 型システムの性質

この章では，前章で定義した λS_m に対する型システムに対して望ましい性質が成立することを示す．

4.1 λS_m と CPS の対応

前節の型システムが，CPS 変換と整合していることを示す．すなわち，λS_m で型付けされた項は，CPS 変換により単純型付きラムダ計算 $\lambda^{\to}$ の項に変換され，また，λS_m での計算は，CPS 変換により単純型付きラムダ計算 $\lambda^{\to}$ における計算に変換されることを示す．

型と型スキームに対する CPS 変換を次のように定義する．ただし，λS_m の基本型は，単純型付きラムダ計算の基本型であると仮定する．また，X は $\lambda^{\to}$ における特定の型を 1 つ選んで固定したものであり，$i \leq m+1$ とする ($i \geq 0$ とは限らない)．

$$[\![\ ]\!]_i : \text{型スキーム} \to \lambda^{\to}\text{の型}$$

$$[\![*]\!]_i = \begin{cases} X & i \leq 0 \text{ の時} \\ (X \to [\![*]\!]_{i-1}) \to [\![*]\!]_{i-1} & i > 0 \text{ の時} \end{cases}$$

$$[\![(\tau, T, S)]\!]_i = ([\![\tau]\!] \to [\![T]\!]_{i-1}) \to [\![S]\!]_{i-1}$$

$$[\![\ ]\!] : \text{型} \to \lambda^{\to}\text{の型}$$

$$[\![b]\!] = b$$

$$[\![\tau \to T]\!] = [\![\tau]\!] \to [\![T]\!]_{m+1}$$

$$[\![\ ]\!] : \text{型環境} \to \lambda^{\to}\text{の型環境}$$

$$[\![[\ ]]\!] = [\ ]$$

$$[\![\Gamma[x \mapsto \tau]]\!] = [\![\Gamma]\!][x \mapsto [\![\tau]\!]]$$

上記の定義で，型スキーム T に対する変換は，i というインデックスを持ち，$[\![T]\!]_i$ という形で与えられている．その理由は，我々の型システムにおいては，型スキームが煩雑になるのを避ける工夫として，$*$ を使って「任意の型スキーム」を表現できるようにしたが，CPS 変換の際にはそのような型 $*$ を展開して，高さ $m+1$ の木にする必要があるためである☆．なお，上記の定義を展開していくと，$i < 0$ となることもあるが，特に問題は生じない．

定理 1 (CPS 変換による型付けの保存) λS_m において $\Gamma \vdash M : T$ が導けるならば，

$$[\![\Gamma]\!] \vdash_{\lambda\to} C_{m+1}[\![M]\!] : [\![T]\!]_{m+1}$$

が導ける．ここで，$\vdash_{\lambda\to}$ は，単純型付きラムダ計算 $\lambda^{\to}$ における型の導出である．

定理**??**は，本論文の型システムで型が付く項は，CPS 変換すると $\lambda^{\to}$ で型が付く項になることを示している．証明は，付録の**??**節と**??**節に記載した．

定理 2 (CPS 変換による項の等しさの保存) λS_m に対して, $\Gamma \vdash M : T$ が導け，さらに $M \to N$ であるならば，

$$C_{m+1}[\![M]\!] =_{\beta\eta} C_{m+1}[\![N]\!]$$

である．ここで，$=_{\beta\eta}$ は単純型付きラムダ計算 $\lambda^{\to}$ における $\beta\eta$-equality をあらわす．

証明. 文献**?**) は，階層化 shift/reset に対する型のない体系 $\lambda S_m^{\mathrm{free}}$ を与え，本論文で述べた CPS 変換に関して健全かつ完全な等式系を与えた．この等式系の一部は以下の通りである．(ただし，$j \geq n$ とし，記法は本論文のものに合わせた．)

$$(\lambda x.M)V = M\{x := V\}$$

$$\langle V \rangle_n = V$$

$$\langle E^n[S_n k.M] \rangle_j = \langle M\{k := \lambda x.\langle E^n[x] \rangle_n\} \rangle_j$$

これらの等式から，λS_m で $M \to N$ であれば，$\lambda S_m^{\mathrm{free}}$ で $M = N$ である．さらに，$\lambda S_m^{\mathrm{free}}$ に対する CPS 変換の健全性**?**) より，型のない純粋なラムダ計算の体系において，$C_{m+1}[\![M]\!] =_{\beta\eta} C_{m+1}[\![N]\!]$ である．この式の両辺は，定理**??**と後述する定理**??**より $\lambda^{\to}$ で型付け可能なので，$\lambda^{\to}$ において $C_{m+1}[\![M]\!] =_{\beta\eta} C_{m+1}[\![N]\!]$ である．(証明終わり)

4.2 型システムの健全性

型システムの健全性は，型の保存 (Subject Reduction, Preservation) と計算の進行 (Progress) の 2 つの性質から構成される．

定理 3 (型の保存) λS_m の項 M, N, 型環境 Γ, 型スキーム T に対し，$\Gamma \vdash M : T$ が導出できて $M \to N$ であれば，$\Gamma \vdash N : T$ が導出できる．

この定理により，項に型が付くならば，その項の実行中に型エラーが起きないことが保証される．証明は付録**??**を参照されたい．

定理 4 (計算の進行) λS_m の項 M, 型スキーム T に対して $[\ ] \vdash \langle M \rangle_m : T$ が導けるならば，$\langle M \rangle_m$ は，評価規則を適用して計算を進めることができる．

この定理で，$\langle M \rangle_m$ の形をした項に限定しているのは，reset で囲われていない shift を持つ項は，いわば open term であり，計算が進まないためである．たと

☆ CPS 変換の定義からわかるように，CPS 変換後の項は必ず $(m+1)$ 個の継続を受けとるため，その型を木として見た場合，高さが $(m+1)$ 以上になる．

以下の導出における W と U は任意の型スキーム, T, S, Γ はそれぞれ以下の型スキームと型環境を表す.

$$T = (\text{list } int, (\text{list } int, W, W), (\text{list } int, W, W))$$
$$S = (string, (\text{list } int, W, W), (\text{list } int, W, W))$$
$$\Gamma = [n \mapsto int, k \mapsto (\text{list } int \to (\text{list } int, T, T))]$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\Pi \quad \vdash choice\ 3 : (int, S, S)}}{\vdash emit_2(choice\ 3) : (\text{list } int, S, S)} \quad \dfrac{}{\vdash \text{“no”} : (string, S, S)}}{\vdash emit_2(choice\ 3); \text{“no”} : (string, S, S)}}{\vdash \langle emit_2(choice\ 3); \text{“no”}\rangle_1 : (string, T, T)} \quad \dfrac{}{\vdash nil : (\text{list } int, T, T)}}{\vdash \langle emit_2(choice\ 3); \text{“no”}\rangle_1; nil : (\text{list } int, T, T)}}{\vdash \langle\langle emit_2(choice\ 3); \text{“no”}\rangle_1; nil\rangle_2 : (\text{list } int, U, U)}$$

$$\Pi = \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash n : (int, T, T)} \quad \dfrac{\dfrac{}{\Gamma \vdash k : (\text{list } int \to (\text{list } int, T, T), T, T)} \quad \dfrac{}{\Gamma \vdash nil : (\text{list } int, T, T)}}{\Gamma \vdash k(nil) : (\text{list } int, T, T)}}{\Gamma \vdash cons(n, k(nil)) : (\text{list } int, T, T)}}{[n \mapsto int] \vdash S_2 k.cons(n, k(nil)) : (\text{list } int, S, S)}}{\vdash emit_2 : (int \to (\text{list } int, S, S), S, S)}$$

図 4 型付けの例
Fig. 4 Example of Type Derivation

えば，$S_1 k.x$ は，そのままでは評価が進まない．この体系のプログラムは，自由変数がなく，かつ，プログラム全体がレベル m の reset で囲われているものと定めれば，上記の定理を適用することができる．

上記の定理により，$\langle M\rangle_m$ が型が付くならば，その項の計算が進むことが示される．証明は付録**??**を参照されたい．

これら 2 つの定理を組み合わせると，次の健全性定理が得られる．

定理 5 (型システムの健全性) λS_m の項 M と型スキーム T に対し，$[\,] \vdash \langle M\rangle_m : T$ が導けるならば，M は値であるか，あるいは，ある項 N が存在して $\langle M\rangle_m \to \langle N\rangle_m$ かつ $[\,] \vdash \langle N\rangle_m : T$ が導ける．

証明. $[\,] \vdash \langle M\rangle_m : T$ が導けたと仮定すると，定理**??**から，$\langle M\rangle_m$ は 1 ステップ計算可能である．M が値でなければ，この 1 ステップの計算によって一番外側の reset がなくなることはない．従って，ある N に対して，$\langle M\rangle_m \to \langle N\rangle_m$ となる．さらに定理**??**から，$[\,] \vdash \langle N\rangle_m : T$ が導ける．(証明おわり)

静的に型付けされたプログラミング言語にとって，型システムの健全性は必要最低限の性質であり，この性質だけで十分であるとは言えないが，本研究の型システムは，階層化 shift/reset を静的に型付けされたプログラム言語で用いるための基礎となると考えられる．

5. まとめと今後の課題

継続を扱うためのコントロールオペレータである shift/reset は様々なプログラムの制御を表現できる上，形式的に扱いやすいという特徴がある．1 つのプログラムの中で shift/reset を複数の目的に用いると，互いに干渉を起こし，意図通りに動作しないことがあるため，階層化 shift/reset が提案された．これまで，この階層化 shift/reset に対する型システムとして十分なものは提案されてこなかった．

本研究は，型スキームのアイディアに基づき，階層化 shift/reset に対する型システムを与えた．また，その性質について検討し，今回定義した型システムが CPS 変換と整合し，かつ，健全であることを示した．この型システムは，階層化 shift/reset と Answer Type の変化の両方に対応した初めてのものである．

本研究は，今後さまざまな研究に発展していくことが考えられる．以下では，そのような課題のいくつかに言及する．

- 型推論: 本研究の型システムは主型 (principal type) を持つ．これに基づき，第 1 著者はこの型システムに対する型推論アルゴリズムを実装した[?]．しかし，これはプロトタイプ実装であり，本格的プログラムに対する有効性の検証は今後の

課題である．

- 多相性との共存: 多相型と call/cc などの副作用が共存すると，型の健全性に問題が生じることが古くから論じられており，ML 言語では多相性を制限する方法が取られている．本論文の shift/reset についても同様の制限のもとで多相性と共存できると予想している．また，ソース言語の多相性だけでなく，CPS 変換後の多相性 (いわゆる Answer Type 多相性) との関連も興味深い課題である．
- 強正規化性:
本論文では，停止性 (強正規化可能性) について議論することができなかった．これは，いわゆる管理レデックス (CPS 変換の際に生じるレデックス) を生成しない CPS 変換を定義することにより証明できるという見通しを持っており，検討中である．

謝辞 第 2 著者は，科学研究費補助金 基盤研究 (C) (No. 16500004) の援助を受けている．

参 考 文 献

1) Biernacka, M., Biernacki, D. and Danvy, O.: An Operational Foundation for Delimited Continuations in the CPS Hierarchy., *Logical Methods in Computer Science*, Vol.1, No.2 (2005).
2) Danvy, O.: Functional Unparsing, *Journal of Functional Programming*, Vol.8, No.6, pp.621–625 (1998).
3) Danvy, O. and Filinski, A.: A Functional Abstraction of Typed Contexts, Technical Report 89/12, DIKU, University of Copenhagen (1989).
4) Danvy, O. and Filinski, A.: Abstracting Control, *Proceedings of the 1990 ACM Conference on LISP and Functional Programming, Nice*, New York, NY, ACM, pp.151–160 (1990).
5) Felleisen, M.: The Theory and Practice of First-Class Prompts, *Proc. 15th Symposium on Principles of Programming Languages*, pp.180–190 (1988).
6) Filinski, A.: Representing Layered Monads, *Proc. 26th Symposium on Principles of Programming Languages*, pp.175–188 (1999).
7) Kameyama, Y.: Axioms for Delimited Continuations in the CPS Hierarchy, *CSL'04, Lecture Notes in Compuer Science 3210*, Karpacz, Poland, pp.442–457 (2004).
8) Kameyama, Y. and Hasegawa, M.: A Sound and Complete Axiomatization of Delimited Continuations., *Proc. International Conference on Functional Programming*, pp.177–188 (2003).
9) Kiselyov, O., c. Shan, C. and Sabry, A.: Delimited Dynamic Binding, *Proc. International Conference on Functional Programming*, pp. 26–37 (2006).
10) Murthy, C.: Control Operators, Hierarchies, and Pseudo-Classical Type Systems: A-Translation at Work, *Proc. First ACM Workshop on Continuations, Technical Report STAN-CS-92-1426, Stanford University*, pp. 49–72 (1992).
11) 鈴木輝信：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，修士論文，筑波大学大学院システム情報工学研究科 (2007).
12) 鈴木輝信，亀山幸義：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，日本ソフトウェア科学会第 23 回大会講演論文集, 8 ページ (2006).

付 録

A.1 準 備

まず，定理の証明で必要となる定義と補題を述べる．

単純型付きラムダの型 t に対して，t のアリティ $\mathrm{arity}(t)$ を以下のように定義する．

$$\mathrm{arity}(b) = 0 \quad \text{(b が基本型の時)}$$
$$\mathrm{arity}(t_1 \to t_2) = 1 + \mathrm{arity}(t_2) \quad \text{(それ以外の時)}$$

t のアリティは，型 t をもつ項が受け取る引数の個数を表す．

補題 1 型スキーム T と $i > 0$ に対して，$\mathrm{arity}([\![T]\!]_i) \geq i$ である．

証明. T に関する帰納法で証明できる．

補題 2 値 V に対して，$\Gamma \vdash V : (\phi, S, S)$ が導ければ，任意の型スキーム T に対して $\Gamma \vdash V : (\phi, T, T)$ を導くことができる．

証明. 値は変数かラムダ式であり，任意の T に対して上記の形の型付けができることがわかる．

補題 3 $\Gamma[x \mapsto \phi] \vdash M : T$ と $\Gamma \vdash V : (\phi, S, S)$ が導ければ，$\Gamma \vdash M\{x := V\} : T$ が導ける．

証明. M に関する帰納法で証明できる．

補題 4 レベル n の評価文脈 E^n と，型環境 Γ, 項 M, 型 τ, 型スキーム T, S に対して，$\Gamma \vdash E^n[M] : (\tau, T, S)$ が導出できれば，型 ϕ と型スキーム U, W が存在して，$\Gamma \vdash M : (\phi, U, W[r^{n-1} \leftarrow S[r^{n-1}]])$ が導出できる．

証明. レベル n の評価文脈 E^n に関する帰納法で証明できる．

A.2 定理??の証明

$\Gamma \vdash M : T$ の導出の大きさに関する帰納法で証明す

る．スペースの都合上，$M = x$, shift, reset のケースのみを掲載する．

(i) $M = x$ の時

仮定より，$\Gamma \vdash x : (\tau, T, T)$ が言える．このとき $\Gamma(x) = \tau$ となる．よって，$(x \mapsto [\![\tau]\!]) \in [\![\Gamma]\!]$ である．

任意の型スキーム T に対して，次の型判断を導出したい．

$$[\![\Gamma]\!] \vdash_{\lambda\to} C_{m+1}[\![x]\!] : ([\![\tau]\!] \to [\![T]\!]_m) \to [\![T]\!]_m$$

CPS 変換の定義から，$C_{m+1}[\![x]\!]$ は以下のようになる．

$$C_{m+1}[\![x]\!] = \lambda\kappa_1 \ldots \kappa_{m+1}.\kappa_1 x \kappa_2 \ldots \kappa_{m+1}$$

補題**??**より，arity$([\![T]\!]_m) \geq m$ なので，適当な型 $a_1, \ldots, a_m, b$ に対して，$[\![T]\!]_m = a_1 \to \ldots \to a_m \to b$ とおける．そこで，

$$\Sigma = [\kappa_1 \mapsto [\![\tau]\!] \to [\![T]\!]_m,\ \kappa_2 \mapsto a_1, \ldots \kappa_{n+1} \mapsto a_n]$$

とおくと，

$$[\![\Gamma]\!], \Sigma \vdash_{\lambda\to} \kappa_1 x \kappa_2 \ldots \kappa_{m+1} : b$$

となり，

$$[\![\Gamma]\!] \vdash_{\lambda\to} C_{m+1}[\![x]\!] : ([\![\tau]\!] \to [\![T]\!]_m) \to [\![T]\!]_m$$

が言える．

(ii) $M = \lambda x.M'$ の時，省略．

(iii) $M = M_1\ M_2$ の時，省略．

(iv) $M = \langle M' \rangle_n$ の時

仮定から，$\Gamma \vdash \langle M' \rangle_n : (\tau, T, T[r^n \leftarrow S])$ および $\Gamma \vdash M' : init(n, (\tau, T[r^n], S), \vec{\gamma}, \vec{W})$ が導出できる．

帰納法の仮定により，

$$[\![\Gamma]\!] \vdash_{\lambda\to} C_{m+1}[\![M']\!] : [\![init(n, (\tau, T[r^n], S), \vec{\gamma}, \vec{W})]\!]_{m+1}$$

が得られる．ここで，$1 \leq i \leq n$ に対して，

$$w_i = [\![\gamma_i]\!] \to [\![(\gamma_i, W_i, W_i)]\!]_{m-i+1}$$

とすると，

$$[\![init(n, (\tau, T[r^n], S), \vec{\gamma}, \vec{W})]\!]_{m+1}$$
$$= w_1 \to \ldots \to w_n \to ([\![\tau]\!] \to [\![T[r^n]]\!]_{m-n}) \to [\![S]\!]_{m-n}$$

と表せる．補題 **??**より，$[\![T]\!]_m$ と $[\![S]\!]_{m-n}$ はそれぞれ m 個以上と $m-n$ 個以上の引数をもつので，

$$[\![T]\!]_m = t_1 \to t_2 \to \ldots \to t_m \to t_0$$
$$[\![S]\!]_{m-n} = s_1 \to s_2 \to \ldots \to s_{m-n} \to s_0$$

と置けて，以下のようになる．

$$[\![T[r^n]]\!]_{m-n} = t_{n+1} \to t_{n+2} \to \ldots \to t_m \to t_0$$
$$[\![T[r^n \leftarrow S]]\!]_m = t_1 \to \ldots \to t_n \to s_1 \to \ldots \to s_{m-n} \to s_0$$

ここで，

$$\Sigma = [k_1 \mapsto [\![\tau]\!] \to [\![T]\!]_m, \quad k_2 \mapsto t_1, k_3 \mapsto t_2, \ldots k_{n+1} \mapsto t_n]$$
$$\Pi = [v \mapsto [\![\tau]\!], \quad k'_{n+2} \mapsto t_{n+1}, k'_{n+3} \mapsto t_{n+2}, \ldots k'_{m+1} \mapsto t_m]$$
$$N = \lambda v \kappa'_{n+2} \ldots \kappa'_{m+1}.\theta_0 v \kappa_1 \ldots \kappa_{n+1} \kappa'_{n+2} \ldots \kappa'_{m+1}$$

とすると，

$$\Sigma, \Pi \vdash_{\lambda\to} \theta_0 v \kappa_1 \ldots \kappa_{n+1} \kappa'_{n+2} \ldots \kappa'_{m+1} : t_0$$
$$\Sigma \vdash_{\lambda\to} N : [\![\tau]\!] \to [\![T[r^n]]\!]_{m-n}$$

が言える．さらに，

$$\Delta = [k_{n+2} \mapsto s_1, k_{n+3} \mapsto s_2, \ldots k_{m+1} \mapsto s_{m-n}]$$

とおき，$\vdash_{\lambda\to} \theta_i : w_i\ (1 \leq i \leq n)$ に注意すると，

$$[\![\Gamma]\!], \Sigma, \Delta \vdash_{\lambda\to} C_{m+1}[\![M']\!]\theta_1 \ldots \theta_n N \kappa_{n+2} \ldots \kappa_{m+1} : s_0$$

となる．以上から，

$$[\![\Gamma]\!] \vdash_{\lambda\to} C_{m+1}[\![\langle M' \rangle_n]\!] : [\![(\tau, T, T[r^n \leftarrow S])]\!]_{m+1}$$

が導けて，このケースの証明を終了する．

(v)$M = S_n k.M'$ の場合

仮定から

$$\Gamma \vdash S_n k.M' : (\tau, U[r^{n-1} \leftarrow (\alpha, T, S)], U)$$

かつ，

$$\Gamma' \vdash M' : init(n, U[r^{n-1}], \vec{\gamma}, \vec{Z})$$
$$\Gamma' = \Gamma[k \mapsto \tau \to (\alpha, W[r^n \leftarrow T], W[r^n \leftarrow S])]$$

が言える．これに帰納法の仮定を適用すると，以下が得られる．

$$[\![\Gamma']\!] \vdash C_{m+1}[\![M']\!] : [\![init(n, U[r^{n-1}], \vec{\gamma}, \vec{Z})]\!]_{m+1}$$

$y_i = [\![\gamma_i]\!] \to [\![(\gamma_i, Z_i, Z_i)]\!]_{m+1-i}$ とすると，

$$[\![init(n, U[r^{n-1}], \vec{\gamma}, \vec{Z})]\!]_{m+1}$$
$$= y_1 \to y_2 \to \ldots \to y_n \to [\![U[r^{n-1}]]\!]_{m-n+1}$$

となる．補題 **??**より，$[\![U]\!]_m$,$[\![W]\!]_m$ はそれぞれ m 個以上，$[\![T]\!]_{m-n}$,$[\![S]\!]_{m-n}$ はそれぞれ $m-n$ 個以上の引数を取るので以下のように表現できる．

$$[\![U]\!]_m = u_1 \to u_2 \to \ldots \to u_m \to u_0$$
$$[\![W]\!]_m = w_1 \to w_2 \to \ldots \to w_m \to w_0$$
$$[\![T]\!]_{m-n} = t_1 \to t_2 \to \ldots \to t_{m-n} \to t_0$$
$$[\![S]\!]_{m-n} = s_1 \to s_2 \to \ldots \to s_{m-n} \to s_0$$

この時，

$$[\![U[r^{n-1}]]\!]_{m+1-n} = u_n \to u_{n+1} \to \ldots \to u_m \to u_0$$
$$[\![U[r^{n-1} \leftarrow (\alpha, T, S)]]\!]_m = u_1 \to u_2 \to \ldots \to u_{n-1} \to ([\![\alpha]\!] \to [\![T]\!]_{m-n}) \to [\![S]\!]_{m-n}$$
$$[\![W[r^n \leftarrow T]]\!]_m = w_1 \to w_2 \to \ldots \to w_n \to [\![T]\!]_{m-n}$$
$$[\![W[r^n \leftarrow S]]\!]_m = w_1 \to w_2 \to \ldots \to w_n \to [\![S]\!]_{m-n}$$

となる．ここで，

$$\Sigma = [v \mapsto [\![\tau]\!], \quad \kappa'_1 \mapsto [\![\alpha]\!] \to [\![W[r^n \leftarrow T]]\!]_m, \quad \kappa'_2 \mapsto w_1, \ldots, \kappa'_{n+1} \mapsto w_n, \quad \kappa'_{n+2} \mapsto s_1, \ldots, \kappa'_{m+1} \mapsto s_{m-n}]$$
$$\Pi = [w \mapsto [\![\alpha]\!], \quad \kappa''_{n+2} \mapsto t_1, \ldots, \kappa''_{m+1} \mapsto t_{m-n}]$$

とすると，

$$\Sigma, \Pi \vdash_{\lambda\rightarrow} \theta_0 w \kappa'_1 \ldots \kappa'_{n+1} \kappa''_{n+2} \ldots \kappa''_{m+1} : t_0$$

が言える．ここで，

$N = \lambda w \kappa''_{n+2} \ldots \kappa''_{m+1}.\theta_0 w \kappa'_1 \ldots \kappa'_{n+1} \kappa''_{n+2} \ldots \kappa''_{m+1}$

とすると，以下の型判断が導出できる．

$$\Sigma \vdash_{\lambda\rightarrow} N : [\![\alpha]\!] \rightarrow [\![T]\!]_{m-n}$$

さらに，

$$\Delta = [\kappa_1 \mapsto [\![\tau]\!] \rightarrow [\![U[r^{n-1} \leftarrow (\alpha, T, S)]]\!]_m, \quad \kappa_2 \mapsto u_1, \ldots, \kappa_n \mapsto u_{n-1}]$$

とすると，

$$[\![\Gamma]\!], \Delta, \Sigma \vdash_{\lambda\rightarrow} \theta_0 v \kappa_1 \ldots \kappa_n N \kappa'_{n+2} \ldots \kappa'_{m+1} : s_0$$

が得られる．ここで，

$$p = \lambda v \kappa'_1 \ldots \kappa'_{m+1}.\theta_0 v \kappa_1 \ldots \kappa_n N \kappa'_{n+2} \ldots \kappa'_{m+1}$$

とすると，以下の型判断が得られる．

$$[\![\Gamma]\!], \Delta \vdash_{\lambda\rightarrow} p : [\![\tau]\!] \rightarrow ([\![\alpha]\!] \rightarrow [\![W[r^n \leftarrow T]]\!]_m) \rightarrow [\![W[r^n \leftarrow S]]\!]_m$$

となる．一方，

$[\![\Gamma']\!], \Delta \vdash_{\lambda\rightarrow} C_{m+1}[\![M']\!] : [\![init(n, U[r^{n-1}], \vec{\gamma}, \vec{Z})]\!]_{m+1}$

が成立するので，$L = (C_{m+1}[\![M']\!])\{k := p\}$ と置くと，上記の型判断から以下が得られる．

$$[\![\Gamma]\!], \Delta \vdash_{\lambda\rightarrow} L : [\![init(n, U[r^{n-1}], \vec{\gamma}, \vec{Z})]\!]_{m+1}$$

さらに，$\vdash_{\lambda\rightarrow} \theta_i : y_i$ であり，

$$\Psi = [\kappa_{n+1} \mapsto u_n, \ldots, \kappa_{m+1} \mapsto u_m]$$

とすると，

$$[\![\Gamma]\!], \Delta, \Psi \vdash_{\lambda\rightarrow} L\theta_1 \ldots \theta_n \kappa_{n+1} \ldots \kappa_{m+1} : u_0$$

が成立するので，

$$[\![\Gamma]\!] \vdash_{\lambda\rightarrow} C_{m+1}[\![S_n k.M']\!] : [\![(\tau, U[r^{n-1} \leftarrow (\alpha, T, S)], U)]\!]_{m+1}$$

となるため，このケースでも定理は成立する．

(証明終わり)

A.3 定理??の証明

それぞれの評価規則ごとに証明する．なお，$E[R] \rightarrow E[M]$ の形の評価規則では，外側の評価文脈 E を除去して ($E = [\,]$ と仮定して) $R \rightarrow M$ に対する型の保存を証明すればよいことがすぐにわかるので，以下ではその形で証明する．

(i) $(\lambda x.M)V \rightarrow M\{x := V\}$

$\Gamma \vdash (\lambda x.M)V : T$ より，以下の導出を得る．ただし，$T = (\alpha, S, U)$，$X^* = (X, U, U)$ とする．

$$\frac{\dfrac{\Gamma[x \mapsto \beta] \vdash M : (\alpha, S, U)}{\Gamma \vdash \lambda x.M : (\beta \rightarrow (\alpha, S, U))^*} \quad \dfrac{}{\Gamma \vdash V : \beta^*}}{\Gamma \vdash (\lambda x.M)V : (\alpha, S, U)}$$

補題??と $\Gamma[x \mapsto \beta] \vdash M : (\alpha, S, U), \Gamma \vdash V : (\beta, U, U)$ より，以下が導ける．

$$\Gamma \vdash M\{x := V\} : (\alpha, S, U)$$

よって，定理は成立する．

(ii) $\langle V \rangle_n \rightarrow V$

仮定より，$\Gamma \vdash \langle V \rangle_n : (\alpha, T, T[r^n \leftarrow U])$ である．reset の規則より，以下が得られる．

$$\frac{\Gamma \vdash V : init(n, (\alpha, T[r^n], U), \vec{\gamma}, \vec{W})}{\Gamma \vdash \langle V \rangle_n : (\alpha, T, T[r^n \leftarrow U])}$$

ここで，$init(n, (\alpha, T[r^n], U), \vec{\gamma}, \vec{W})$ の部分は，値に対する型スキーム なので，その左部分木と右部分木の型スキームは等しくなる．よって，以下の関係が得られる．

$$(\gamma_1, W_1, W_1) = init(n-1, (\alpha, T[r^n], U), \vec{\gamma_2}, \vec{W_2})$$

ただし，$\vec{\gamma_i}, \vec{W_i}$ はそれぞれ $\vec{\gamma}, \vec{W}$ の i 番目以降の変数列とする．ここで，

$$\begin{aligned} & init(n-1, (\alpha, T[r^n], U), \vec{\gamma_2}, \vec{W_2}) \\ =& (\gamma_2, (\gamma_2, W_2, W_2), \\ & \quad init(n-2, (\alpha, T[r^n], U), \vec{\gamma_3}, \vec{W_3})) \end{aligned}$$

であるから，これらの関係式から以下の等式が得られる．

$$\begin{aligned} \gamma_1 &= \gamma_2 \\ W_1 &= (\gamma_2, W_2, W_2) \\ &= init(n-2, (\alpha, T[r^n], U), \vec{\gamma_3}, \vec{W_3}) \end{aligned}$$

この W_1 に対しての等式より，最初と同じような関係が出てくる．このような計算を同様に繰り返していくと，結局，以下の等式が得られる．

$$\begin{aligned} \gamma_i &= \gamma_{i+1} \\ W_i &= (\gamma_{i+1}, W_{i+1}, W_{i+1}) \quad (1 \le i \le n-1) \\ \gamma_n &= \alpha \\ W_n &= T[r^n] = U \end{aligned}$$

この関係より，それぞれの型判断は以下のように書き換えられる．

$$\Gamma \vdash \langle V \rangle_n : (\alpha, T, T)$$
$$\Gamma \vdash V : (\alpha, W_1, W_1)$$

補題??より，$\Gamma \vdash V : (\alpha, W_1, W_1)$ から $\Gamma \vdash V : (\alpha, T, T)$ が言える．よって，定理は成立する．

(iii) $\langle E^n[S_n k.M] \rangle_j \rightarrow \langle M\{k := \lambda x.\langle E^n[x] \rangle_n\} \rangle_j$ $(n \le j)$

まず，仮定と型付け規則から，以下の導出を得る．

$$\frac{\Gamma \vdash E^n[S_n k.M] : init(j, (\tau, T[r^j], X), \vec{\gamma}, \vec{S})}{\Gamma \vdash \langle E^n[S_n k.M] \rangle_j : (\tau, T, T[r^j \leftarrow X])}$$

以下では, 煩雑さを避けるため, init の第 3, 第 4 引数 $(\vec{\gamma}, \vec{S})$ の記述を省略する．補題??から，

$$\Gamma \vdash S_n k.M : (\phi, U, V[r^{n-1} \leftarrow init(j-n, (\tau, T[r^j], X))])$$

を得る．この判断を結論とする導出を考えると，最後に適用する規則は shift であるので，

$$U = V[r^{n-1} \leftarrow (\psi, W, Y)] \qquad (1)$$

となり，以下の判断を得る．

$$\Gamma' \vdash M : init(n, init(j-n, (\tau, T[r^j], X)))$$

$\Gamma' = \Gamma[k \mapsto \phi \to (\psi, Z[r^n \leftarrow W], Z[r^n \leftarrow Y])]$

k に $\lambda x.\langle E^n[x]\rangle_n$ を代入しなくてはいけないので，それぞれが同じ型を持たなくてはいけないが，それは，以下の導出で確認できる．ただし，$A = (\psi, Z[r^n \leftarrow W], Z[r^n \leftarrow Y])$ とする．

$$\dfrac{\dfrac{\dfrac{}{\Gamma[x \mapsto \phi] \vdash x : (\phi, U, V[r^{n-1} \leftarrow (\psi, W, Y)])}}{\Gamma[x \mapsto \phi] \vdash E^n[x] : init(n, (\psi, W, Y))}}{\dfrac{\Gamma[x \mapsto \phi] \vdash \langle E^n[x]\rangle_n : A}{\Gamma \vdash \lambda x.\langle E^n[x]\rangle_n : (\phi \to A, Z_0, Z_0)}}$$

(**??**) より，$U = V[r^{n-1} \leftarrow (\psi, W, Y)]$ なので，この型判断は正しい．補題**??**と，

$\Gamma' \vdash M : init(n, init(j-n, (\tau, T[r^j], X)))$

$\Gamma \vdash \lambda x.\langle E^n[x]\rangle_n$

$\quad : (\phi \to (\psi, Z[r^n \leftarrow W], Z[r^n \leftarrow Y]), Z_0, Z_0)$

より，以下の判断が得られる．

$$\Gamma \vdash M\{k := \lambda x.\langle E^n[x]\rangle_n\} : init(n, init(j-n, (\tau, T[r^j], X)))$$

これに reset 規則を適用すると，

$$\Gamma \vdash \langle M\{k := \lambda x.\langle E^n[x]\rangle_n\}\rangle_j : (\tau, T, T[r^j \leftarrow X])$$

となり，定理は成立する．(証明終わり)

A.4 定理??の証明

この定理をやや一般化した以下の補題を証明する．

補題 5 項 M, 型スキーム T に対して $[\,] \vdash M : T$ が導けるならば，以下のいずれかが成立する．

1. M は値.
2. M は $E[(\lambda x.M')V]$ の形.
3. M は $E[\langle E^n[S_n k.M']\rangle_j]$ の形 (ただし，$j \geq n$).
4. M は $E^n[S_n k.M']$ の形.
5. M は $E[\langle V\rangle_n]$ の形.

証明. M に関する帰納法で証明する．

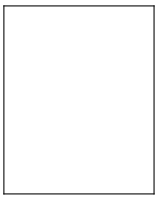
$M = N_1 N_2$ の場合だけを述べる．M が型付け可能なので，適当な型スキーム T_1, T_2 に対して，$[\,] \vdash N_1 : T_1$ と $[\,] \vdash N_2 : T_2$ が導ける．そこで帰納法の仮定を使うと，N_1, N_2 は，上記の 1, 2, 3, 4, 5 のいずれかの形である．

N_1 が 2, 3, 4, 5 のいずれかの形のときは，評価文脈 E を EN_2 に置き換えれば M 全体が 2, 3, 4, 5 の形であることが言える．N_1 が 1 の形で，N_2 が 2, 3, 4, 5 のいずれかの形のときは，E を $N_1 E$ に置き換えればよい．N_1 も N_2 も 1 の形のときは，M に自由変数がないので N_1 は変数ではあり得ず，ラムダ抽象である．従って，$N_1 N_2$ は 2 の形になっている．(証明おわり)

定理??の証明:

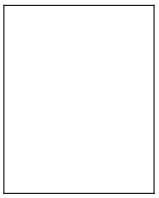
$[\,] \vdash \langle M\rangle_m : T$ が導けると仮定する．$\langle M\rangle_m$ は，上記の補題で述べた 1, 2, 3, 4, 5 のいずれかの形であるが，1 と 4 はあり得ない．従って，2, 3, 5 のいずれかの形であり，これらはいずれも評価規則の左辺になっているので，1 ステップの計算が可能である．(証明おわり)

(平成 18 年 12 月 16 日受付)
(平成 19 年 4 月 1 日採録)



鈴木 輝信

筑波大学大学院システム情報工学研究科コンピュータサイエンス専攻博士前期課程在学．プログラム言語の型システムに関する研究を行う．日本ソフトウェア科学会会員．



亀山 幸義（正会員）

筑波大学大学院システム情報工学研究科コンピュータサイエンス専攻助教授．プログラムの論理とソフトウェア検証に興味をもつ．ACM，日本ソフトウェア科学会 各会員．