# shift/reset 付き TDPE の抽出

廣田 知子 [1], 浅井 健一 [1]

[1] お茶の水女子大学

hirota.noriko@is.ocha.ac.jp, asai@is.ocha.ac.jp

**概 要** Danvy により提案された type-directed partial evaluator (TDPE) は，与えられた項を normal form に評価する関数であり，Filinski によって call-by-value 及び call-by-name の単純型付き λ 計算における TDPE の正当性が示されている．一方，対馬らの論文によって，shift/reset 付きに拡張された TDPE が提案されているが，未だ正当性は保証されていない．

本研究では，対馬らによって提案された TDPE を参考に，CPS 変換を行った shift/reset 付き TDPE を定義し，その正当性を示す試みを行った．証明は CPS 変換した論理述語を用いて行った．さらに，call-by-value の shift/reset を含んだ型付き λ 計算の正当性（本研究で定義した論理述語の定義を満たす）の証明を定理証明系システム Coq によって定式化することにより，OCaml 言語のプログラムを抽出した．抽出されたプログラムは，複雑にはなっているものの，概ね対馬らによって提案された shift/reset 付 TDPE の形を成していることが見て取れるような構造になっている．

## 1 はじめに

type-directed partial evaluator（TDPE）は，Danvy により提案された [3] 部分評価器であり，term とその term の型が渡されたとき，その型によって場合分けを行いながら，term を normal form へと評価する関数である．TDPE の利点として，term の中身を見ないが故に，term の構造における場合分け等を行わずに部分評価を行う為，高速であることが挙げられる．call-by-value 及び call-by-name の単純型付き λ 計算における TDPE においては，その正当性が [4] にて Filinski により既に示されている．Filinski はその論文にて，論理関係を用いた Tait 流の証明を行っている．call-by-value だけでなく，任意の monadic effect を持つ体系を対象としており，let-insertion を含めて証明が為されている．しかし，その証明は難解で，他の体系，特に shift/reset が入った場合にどうなるかを考えようとしても，そのまま拡張しようとするのは難しい．

一方，対馬らによって提案 [8] された shift/reset 付 TDPE においては，直感的な説明はされているものの，未だその正当性が示されていなかった．

[6] にて，Ilik が Coquand の研究 [2] を基に，性質の存在証明から shift/reset 付 TDPE を抽出する方法の足がかりを作っている．つまり，どのような性質の存在証明を行えば TDPE が抽出されるのかを明らかにした．定式化においては，（Ilik 自身の論文である）[5] を拡張した形となっている．しかしその論文 [6] にて扱われている限定継続は論理から見たものであり，我々が考える通常の限定継続とは異なっているものと考えられる．

そこで本研究では，対馬らが提案した shift/reset 付 TDPE を CPS 変換し，そのプログラムとの対応を付けながら，通常の限定継続を扱える形で Ilik の手法 [6] を定式化し直した．その際，

- 対馬らの TDPE のどの場所を CPS 変換すれば良いかは自明でなかった為，[6] が基になければ難しく，特に，TDPE に渡す static な term も CPS にしなければならないことが，Ilik の定式化を参考にすることにより分かった．
- その一方で、Ilik の定式化を通常の限定継続用に新たに定式化し直す際には，対馬らの TDPE と対応を付けつつ考えないと難しかった．

という意味において，対馬らと Ilik 双方の研究に融合させた形となっている．

本研究で行った定式化は，shift/reset なし $\lambda$ 計算における TDPE における定式化手法を，shift/reset 付きに拡張した形になっている為，本論文では，まず call-by-value の shift/reset なし $\lambda$ 計算における TDPE の抽出から説明する．本論文の構成は，まず 2 節にて call-by-value の shift/reset なし $\lambda$ 計算における性質の存在証明から TDPE の抽出を行う方法と抽出された TDPE の説明を行い，そして 2 節での手法を拡張することによって call-by-vaule の shift/reset 付 $\lambda$ 計算における性質の存在証明から shift/reset 付 TDPE が得られることを 3 節で述べる．そして 4 節にて本研究の今後の課題を考察する．

以降，本論文で定義する call-by-value の shift/reset なし $\lambda$ 計算を $\lambda_{cbv}$，call-by-value の shift/reset 付 $\lambda$ 計算を $\lambda_{cbv}^{S/R}$ と表記する．

## 2 $\lambda_{cbv}$ における TDPE の抽出

本節では，call-by-value の（shift/reset を含まない）$\lambda$ 計算 $\lambda_{cbv}$ において，どのように TDPE を抽出したかを説明する．対馬らが [8] にて記している call-by-value の $\lambda$ 計算における TDPE の定義は，let-insertion を用いて行われている．その let-insertion に shift/reset を用いた定義が以下である．

$$
\begin{array}{rcl}
t \in \mathrm{Type} & := & b \mid t_1 \to t_2 \\
v \in \mathrm{Value} & := & x \mid \overline{\lambda} x.v \mid v_0 \mathbin{\overline{@}} v_1 \\
e \in \mathrm{Expr} & := & x \mid \underline{\lambda} x.e \mid e_0 \mathbin{\underline{@}} e_1 \\
\\
\text{Reify } \text{“}\downarrow\text{”} & : & \mathrm{Type} \to \mathrm{Value} \to \mathrm{TLT} \\
\downarrow^{\mathsf{base}} v & = & v \\
\downarrow^{A \to B} v & = & \underline{\lambda} x^{\diamond}.\overline{\langle} \downarrow^{B} (v \mathbin{\overline{@}} \uparrow_{A} x_1^{\diamond}) \overline{\rangle} \\
\\
\text{Reflect } \text{“}\uparrow\text{”} & : & \mathrm{Type} \to \mathrm{Expr} \to \mathrm{TLT} \\
\uparrow_{\mathsf{base}} e & = & e \\
\uparrow_{A \to B} e & = & \overline{\lambda} v_1. \uparrow_{B} \overline{S} k_1.\underline{\mathrm{let}}\ g^{\diamond} = e \mathbin{\underline{@}} \downarrow^{A} v_1 \ \underline{\mathrm{in}}\ k_1 \mathbin{\overline{@}} g^{\diamond}
\end{array}
$$

Expr は dynamic term，つまり TDPE が出力する部分評価後のプログラム（構文木）を表し，Value は static term，つまり TDPE 時に（部分評価結果のプログラムを出力するために）実行されるプログラムを表す．また，TLT は Expr と Value が混ざった式を表していて，この中の static な部分を実行すると TDPE の結果が dynamic な式として得られることになる．アンダーラインの付いている式が dynamic な式を出力する命令，オーバーラインの付いている式が static な命令である．又，$^{\diamond}$ の付いている変数は，他の変数とぶつかることのないように選ばれた，fresh な変数である．Reflect の $A \to B$ のケースは，call-by-name では $\overline{\lambda} v_1. \uparrow_{B} (e \mathbin{\underline{@}} \downarrow^{A} v_1)$ となるところであるが，call-by-value では dynamic な $(e \mathbin{\underline{@}} \downarrow^{A} v_1)$ を let-insertion を使って出力している．$\overline{S} k$ で，その時点での継続を切り取り，fresh な変数 $g^{\diamond}$ を使って $(e \mathbin{\underline{@}} \downarrow^{A} v_1)$ を let 文に残し，以降の計算には $g^{\diamond}$ が渡されている．この let 文が出力されるのは，reify の $A \to B$ のケースで dynamic な束縛が作られる（static な reset の）直下である。

この TDPE の定義は，static な shift/reset が使われている．Coq は shift/reset を提供していない為，上の TDPE 定義を直接 Coq で定式化することは出来ない。そこで本研究では，その shift/reset を CPS 変換して削除することとした．しかし TDPE のどの場所をどのようにして CPS 変換すれば良いのかは自明には分からなかった．そこで Ilik の研究 [6] を参考にし，どの箇所が CPS 変換に相当しているのかをみた．その論文において，抽出すると TDPE となると説明されている定理の定義を調べると，reify は型と CPS 変換された value を受け取り，CPS 変換されていないような term を返すこと，reflect は型と（CPS 変換されていない）term を受け取り，CPS 変換されている value

を返す関数となっていることが分かった．上の定義の static な term を CPS 変換したものが，以下に定義する関数である：

$$
\begin{array}{rcl}
\text{Reify } \text{"}\downarrow\text{"} & : & \text{Type} \to \text{Value} \to \text{TLT} \\
\downarrow^{\mathsf{base}} v & = & v \\
\downarrow^{A \to B} v & = & \underline{\lambda} x_1^{\diamond}.v \,\overline{@}\, (\uparrow_A x_1^{\diamond}) \,\overline{@}\, (\overline{\lambda} v.\downarrow^B v)
\end{array}
$$

$$
\begin{array}{rcl}
\text{Reflect } \text{"}\uparrow\text{"} & : & \text{Type} \to \text{Expr} \to \text{TLT} \\
\uparrow_{\mathsf{base}} e & = & e \\
\uparrow_{A \to B} e & = & \overline{\lambda} v_1.\overline{\lambda} k_1.(\underline{\lambda} g^{\diamond}.k_1 \,\overline{@}\, (\uparrow_B g^{\diamond}))\underline{@}(e\underline{@} \downarrow^A v_1).
\end{array}
$$

この定義は，先の call-by-value の TDPE のうち，reify に渡される引数 $v$ と reflect が返す static な値を CPS 変換したものである．reify に渡される引数 $v$ は CPS になったので，それを使う際には引数 $(\uparrow_A x_1^{\diamond})$ に加えて，継続が渡されている．また、reflect が返す値は，$\overline{\lambda} k_1$ のように継続を受け取る格好になっている．このように CPS 変換を施すと，shift/reset を使って行っていた let-insertion を shift/reset を使わずに行うことができる．上記の CPS の TDPE では，それを Coq での実装に合わせて let 文を展開した $(\lambda g^{\diamond}....)\,\underline{@}\,(...)$ という形で表している．

このように CPS 変換をすると，その副作用として TDPE が受け取る static な term も CPS でなくてはならなくなる．例えば [8] における CPS 変換前の TDPE では，$\lambda x.((\lambda y.y)@x) : \mathsf{base} \to \mathsf{base}$ における TDPE の結果を得たければ，この項をそのまま reify に渡せば良かった．しかし CPS 変換した TDPE にて $\lambda x.((\lambda y.y)@x) : \mathsf{base} \to \mathsf{base}$ を実行したい場合，この項を CPS 変換した式を reify に渡さねばならなくなっているのである．TDPE が受け取るものは CPS になるが，出力は直接形式のままである．上の例では，$\lambda x.((\lambda y.y)x)$ を CPS 変換したものを TDPE に渡すと，その直接形式での正規形 $\lambda x.x$ が得られる．

### 2.1 $\lambda_{cbv}$ の構文

まず，本研究で定義した $\lambda_{cbv}$ の構文を以下に示す．

$$
\begin{array}{rcccccl}
\text{type} & : & \mathsf{typ} & \ni & A & := & \mathsf{base} \mid A_1 \to A_2 \\
\text{environment} & : & \mathsf{world} & \ni & w, \Gamma & := & \text{list of } \mathsf{typ} \\
\text{variable term} & : & \vdash_{\mathsf{v}} & \ni & v & := & \mathsf{hyp} \mid \mathsf{wkn}(v) \\
\text{term} & : & \vdash_{\mathsf{t}} & \ni & p & := & v \mid \mathsf{lam}(p) \mid \mathsf{app}(p_1, p_2)
\end{array}
$$

本論文においては，束縛変数は de Bruijn index で表される．直感的には，$\mathsf{hyp}$ は一番内側の binder で束縛されている変数を，$\mathsf{wkn}(v)$ は $v$ の中の束縛変数の値を 1 繰り上げるような命令を意味している．例えば id 関数は，$\mathsf{lam}(\mathsf{hyp})$，$\lambda x.\lambda y.y@x$ は $\mathsf{lam}(\mathsf{lam}(\mathsf{app}(\mathsf{hyp}, \mathsf{wkn}(\mathsf{hyp}))))$ と書ける．

この構文では任意の term を書くことが出来るが，以下，型の付いている term だけを考える．次のような型規則を使い term を定義すると，型の付く term のみしか書けなくなる．

$$
\frac{}{\mathsf{hyp} : A, w \vdash_{\mathsf{v}} A} \qquad \frac{p : w \vdash_{\mathsf{v}} A}{\mathsf{wkn}(p) : B, w \vdash_{\mathsf{v}} A}
$$

$$
\frac{v : w \vdash_{\mathsf{v}} A}{v : w \vdash_{\mathsf{t}} A} \qquad \frac{p : A, w \vdash_{\mathsf{t}} B}{\mathsf{lam}(p) : w \vdash_{\mathsf{t}} A \to B} \qquad \frac{p : w \vdash_{\mathsf{t}} A \to B \quad q : w \vdash_{\mathsf{t}} A}{\mathsf{app}(p, q) : w \vdash_{\mathsf{t}} B}
$$

一見，一般的な型推論規則と差異がないようにも見えるが，この規則では $e : w \vdash_{\mathsf{t}} A$ の $e$ が term，$w \vdash_{\mathsf{t}} A$ が $e$ の型となっている．一般的に定義される term の型とは異なり環境 $w$ も型情報に含まれており，加えて一般的な型推論規則の条件をも満たしていなければ term として成立しなくなっていることが上の定義の特徴である．上の定義をみれば分かる様に，$\mathsf{wkn}(p)$ の型規則は weakening を表すものとなっている．このように表現することにより，$w \vdash_{\mathsf{t}} A$ は論理式「$w \Rightarrow A$」と捉えることが出来る．よって Curry Howard 同型により，$w \vdash_{\mathsf{t}} A$ の成立と，この型の term が存在すること

が同値となる．又，型情報に環境を加えることと，de Bruijn index の使用により，$\alpha$ 同値問題を考える必要がなくなっている．Coq で term を定式化する際は，以下のように推論規則のみで定義した．尚，term の定式化に関しては [6] における定式化を参考にした．

```
set Implicit Arguments.
Inductive var : world -> typ -> Set :=
| var_Hyp : forall w A, var (A :: w) A
| var_Wkn : forall w A B, var w A -> var (B :: w) A.
Inductive tm : world -> typ -> Set :=
| tm_var : forall w A, var w A -> tm w A
| tm_Lam : forall w A B,
           tm (A :: w) B -> tm w (arrow A B)
| tm_App : forall w A B,
           tm w (arrow A B) ->
           tm w A ->
           tm w B.
```

この定義を用いて，例えば id 関数は（world $w$ において (base $\to$ base) 型を持つとすると,）(tm_Lam (tm_Hyp w base)) と記述できる.（Set Implicit Arguments を使用しない場合は，全ての情報を記述しないといけない為，(tm_Lam w base base (tm_Hyp w base)) と記述する.）

次に $\lambda_{cbv}$ における normal form, neutral term を定義する．normal form は正規形であり，これ以上簡約出来ない term となっている．neutral term は変数を正規形で呼び出した形で、変数の値が具体化しない限り、これ以上、実行を進められない term を意味している．

$$\begin{array}{lcccl} \text{normal form} & : & \vdash_{\mathsf{nf}} & \ni & nf \quad := \quad ne \mid \mathsf{lam}(nf) \\ \text{neutral term} & : & \vdash_{\mathsf{ne}} & \ni & ne \quad := \quad v \mid \mathsf{app}(ne, nf) \mid \mathsf{let}(nf_1, \mathsf{app}(ne, nf_2)) \end{array}$$

$\mathsf{let}(nf_1, \mathsf{app}(ne, nf_2))$ は，$\mathsf{app}(ne, nf_2)$ を（$nf_1$ の中の）0 番の変数に束縛して $nf_1$ を実行するような let 文であり，$\mathsf{app}(\mathsf{lam}(nf_1), \mathsf{app}(ne, nf_2))$ と同じことである．この式は call-by-name ではさらに $\beta$ 簡約出来てしまうが，call-by-value では $\mathsf{lam}(nf_1)$ に渡される項が value でないと簡約は出来ない．しかし $\mathsf{app}(ne, nf_2)$ は（これ以上簡約出来ないような）application であり，value ではない為，$\mathsf{app}(\mathsf{lam}(nf_1), \mathsf{app}(ne, nf_2))$ はこれ以上簡約出来ない項であり，neutral term として定義する必要がある．この式は，これ以上実行を進められない項 $\mathsf{app}(ne, nf_2)$ を，let 文に残していることに相当する．又，normal form，neutral term の推論規則は term の定義と同様にして以下の如く表現される：

$$\frac{p : w \vdash_{\mathsf{ne}} A}{p : w \vdash_{\mathsf{nf}} A} \qquad \frac{p : A, w \vdash_{\mathsf{nf}} B}{\mathsf{lam}(p) : w \vdash_{\mathsf{nf}} A \to B}$$

$$\frac{v : w \vdash_{\mathsf{v}} A}{v : w \vdash_{\mathsf{ne}} A} \qquad \frac{p : w \vdash_{\mathsf{ne}} A \to B \quad q : w \vdash_{\mathsf{nf}} A}{\mathsf{app}(p, q) : w \vdash_{\mathsf{ne}} B}$$

$$\frac{p : A, w \vdash_{\mathsf{nf}} B \quad q : w \vdash_{\mathsf{ne}} C \to A \quad q' : w \vdash_{\mathsf{nf}} C}{\mathsf{let}(p, \mathsf{app}(q, q')) : w \vdash_{\mathsf{ne}} B}$$

### 2.2 Soundness と Completeness の証明

本研究では Tait 流の論理述語を用いて，Kripke 意味論に対する soundness と completeness の定理を定義している．使用する論理述語は，call-by-value における一般的な定義（Pierce の著書 [7] に説明がある）と，[5]（$\lambda_{cbv}^{S/R}$ においては [6]）における Ilik の定義，そして上述の CPS 変換した TDPE を参考に定義している．$\lambda_{cbv}$ にて使用する論理述語は以下の如く定義する．

$$
\begin{array}{rcl}
R^1_w(\mathsf{base}) & := & w \vdash \mathsf{base} \\
R^1_w(A \to B) & := & \forall w_1, \forall T, w \leq w_1 \ \Rightarrow \ R^1_{w_1}(A) \ \Rightarrow \\
 & & (\forall w_2, w_1 \leq w_2 \ \Rightarrow \ R^1_{w_2}(B) \ \Rightarrow \ w_2 \vdash T) \ \Rightarrow \\
 & & w_1 \vdash T
\end{array}
$$

$$
Rs^1_w((A_1, A_2, ..., A_n)) \ := \ (R^1_w(A_1)) \wedge (R^1_w(A_2)) \wedge ... \wedge (R^1_w(A_n))
$$

ここで引数として渡されている $\vdash$ は，world と型における述語であれば何でも良い．又，$\leq$ は world の大小関係を表している．以下の様な $\leq$ の定義と公理を与えれば，$R^1$ における soundness を示すことが出来る．

$$
\begin{array}{ll}
(i) & w \leq w \\
(ii) & w \leq w' \ \Rightarrow \ w \leq (A, w')
\end{array}
$$

**公理 1** $\forall w, \forall w', \forall A, w \leq w' \ \Rightarrow \ w \vdash A \ \Rightarrow \ w' \vdash A$

**補題 1** $\forall w, \forall w', \forall \Gamma, w \leq w' \ \Rightarrow \ Rs^1_w(\Gamma) \ \Rightarrow \ Rs^1_{w'}(\Gamma)$

**定理 1 (Soundness, $\lambda_{cbv}$)**

$$
\forall A, \forall \Gamma, \forall w, \forall T, \Gamma \vdash_{\mathsf{t}} A \ \Rightarrow \ Rs^1_w(\Gamma) \ \Rightarrow \ (\forall w', w \leq w' \ \Rightarrow \ R^1_{w'}(A) \ \Rightarrow \ w' \vdash T) \ \Rightarrow \ w \vdash T
$$

Proof. $\Gamma \vdash_{\mathsf{t}} A$ における帰納法.（補題 1 を使用する.） □

定理 1 の定義が意味するところは,「$\Gamma$ のもとで $A$ が成り立つなら，$\Gamma$ を満たす任意の $w$ の元で $A$ が成り立つ.（言い換えると，$\Gamma$ のもとで $A$ 型になるような term があるなら，$\Gamma$ を満たす値環境の中で $A$ 型として振る舞う値が存在する）」である．ただし，その「$A$ 型として振る舞う値」というのは，ここでは CPS になっているので，$\Gamma$ を満たす環境の中で，$A$ 型の値を受け取ったら Ans になるような継続を受け取ったら，Ans が成り立つ，ということになる．この定理の証明を抽出すると，CPS 変換に相当している．実際，証明は CPS 変換のプログラム通りに進んでゆく．

completeness の証明には，$\vdash$ を $\vdash_{\mathsf{nf}}$ に置換した論理述語を用いる．この述語の名前を $R$，$Rs$ と置くこととする．定義は以下の如く．

$$
\begin{array}{rcl}
R_w(\mathsf{base}) & := & w \vdash_{\mathsf{nf}} \mathsf{base} \\
R_w(A \to B) & := & \forall w_1, \forall T, w \leq w_1 \ \Rightarrow \ R_{w_1}(A) \ \Rightarrow \\
 & & (\forall w_2, w_1 \leq w_2 \ \Rightarrow \ R_{w_2}(B) \ \Rightarrow \ w_2 \vdash_{\mathsf{nf}} T) \ \Rightarrow \\
 & & w_1 \vdash_{\mathsf{nf}} T
\end{array}
$$

$$
Rs_w((A_1, A_2, ..., A_n)) \ := \ (R_w(A_1)) \wedge (R_w(A_2)) \wedge ... \wedge (R_w(A_n))
$$

先述したように，Curry Howard 同型により「$w \vdash_{\mathsf{nf}} A$」は「$w \vdash_{\mathsf{nf}} A$ を型に持つ term が存在する」とみなすことが出来る．一般的に completeness，soundness の証明に使われる論理述語と比較して特に異なっているのは，論理述語が受け取る型が $A \to B$ のケースが複雑になっていることと，world $w$ を受け取っていることである．前者は CPS 変換を行っている為であり，特に，call-by-name における論理述語の定義ならば $R_w(A) \ \Rightarrow \ R_w(B)$ となるところを，$R_w(B) \ \Rightarrow$ Ans という形の継続をとるようになっている．後者による利点は，world を持ち歩くことで，上手く $\alpha$ 変換の問題を回避していることにある．尚，$R$ は TDPE の世界においては，実行済みの（中味を見ることが出来ない）値（或は関数）を意味している．

又，$\leq$ は，ここでは以下の様に定義する．

$$\forall A,\ \ w \leq (A, w)$$

soundness の証明をしたときとは違って，ここでは $\leq$ について推移律が成り立つ必要がない．次の補題は，変数の weakening が 1 回だけ行えることを示している．

**補題 2** $\forall w, \forall w', \forall A, w \leq w' \ \Rightarrow\ w \vdash_{\mathsf{v}} A \ \Rightarrow\ w' \vdash_{\mathsf{v}} A$

Coq で定式化する際には，$\leq$ は Set 型として定義している為，抽出された関数にも登場することになるが，その関数内においては変数名のカウンタをひとつ増やす役割をすることになる．

以下の completeness の定理を証明し，その証明を抽出したものが，$\lambda_{cbv}$ の TDPE となる．

**定理 2 (Completeness, $\lambda_{cbv}$)**

$$\begin{array}{lll} \forall A, & (i) & \forall w, R_w(A) \ \Rightarrow\ w \vdash_{\mathsf{nf}} A \\ & (ii) & \forall w, w \vdash_{\mathsf{v}} A \ \Rightarrow\ R_w(A) \end{array}$$

Proof. $A$ における帰納法.（補題 2 を使用する.） □

$(i)$ の定義は，$A$ 型の static な値が存在するなら，$A$ 型の正規形が存在する（つまり値を正規形に変換できる）ことを意味しており，$(ii)$ は，$A$ 型の変数は，いつでも $A$ 型の static な値に変換することができることを意味する．$(i)$ が，先に示した TDPE のモデルにおける Reify，$(ii)$ が Reflect にそれぞれ対応しており，モデルの定義通りに証明していくことが可能である．

## 2.3 TDPE の抽出

soundness，completeness の証明は，それぞれ CPS 変換，CPS の TDPE のプログラムと一対一の関係にあり，実際，証明はプログラムを見ながら，その通りに証明した．completeness の証明から得られる Coq 上の関数定義（Print コマンドを用いれば見る事が出来る）に登場する，型に関する帰納法の原理を展開して, 自明な $\beta$ 簡約を行うと，以下に示す関数が得られる．

```
Fixpoint reify (t : typ) : (forall w : world, R w t -> nf w t) :=
  match t as t0 return (forall w : world, R w t0 -> nf w t0) with
  | base => (fun (w : world) (X : R w base) => X)
  | arrow t0 t1 => (fun (w : world) (X : R w (arrow t0 t1)) =>
     nf_Lam
       (X (t0 :: w)
          t1
          (lew_cons t0 w)
          (reflect t0 (t0 :: w) (var_Hyp w t0))
          (fun (w2 : world) (_ : t0 :: w <== w2) (X0 : R w2 t1) =>
           reify t1 w2 X0)))
  end
with reflect (t : typ) : (forall w : world, var w t -> R w t) :=
  match t as t0 return (forall w : world, var w t0 -> R w t0) with
  | base => (fun (w : world) (H : var w base) => nf_ne (var_ne H))
  | arrow t0 t1 => (fun
       (w : world)
       (H : var w (arrow t0 t1))
       (w1 : world)
       (T : typ)
       (H0 : w <== w1)
       (X : R w1 t0)
       (X0 : forall w2 : world, w1 <== w2 -> R w2 t1 -> nf w2 T) =>
```

```
    nf_ne
      (ne_Let
         (X0 (t1 :: w1) (lew_cons t1 w1) (reflect t1 (t1 :: w1) (var_Hyp w1 t1)))
         (var_ne (wkn_var H0 H)) (reify t0 w1 X)))
  end.
```

lew_cons は completeness の証明にて使用した $\leq$ であり，lew_cons t0 w は $w \leq (t, w)$ を表している．これは，変数名のカウンタをひとつ増やす役割を担っている．又，nf_ne と var_ne は，受け取った neutral term を nomal form，variable term を neutoral term としてそれぞれ認識する為の命令である．<==は，lew_cons の型を表している．wkn_var は補題 2 の証明により得られた関数である．例えば wkn_var (H1: w <== w') (H2: var w A) といった式が与えられたとき，この式は var w' A を型として持つ項を返す.（H2 は variable term を表す）<==の定義により，w' は w の先頭に要素を一つ加えたものである．故に，var w' A は H2 の状態よりも,（一つ変数名が進んだ）variable term を表す型となる．よって上の式は var_Wkn H2 となる．以上を踏まえて上の関数をみると，本節の冒頭にて示した，CPS 変換を施した TDPE に等しい定義になっているであろうことが分かる．さらにこの関数に Extraction コマンドを用いて OCaml のプログラムを抽出したものが以下である．

```
(** val reify : typ -> world -> r -> nf **)

let rec reify t w x =
  match t with
  | Base -> Obj.magic x
  | Arrow (t0, t1) ->
    Nf_Lam (w, t0, t1,
      Obj.magic x (Cons (t0, w)) t1 (Lew_cons (t0, w))
       (reflect t0 (Cons (t0, w)) (Var_Hyp (w, t0)))
       (fun w2 x0 x1 -> reify t1 w2 x1))

(** val reflect : typ -> world -> var -> r **)

and reflect t w h =
  match t with
  | Base -> Obj.magic (Nf_ne (w, Base, (var_ne w Base h)))
  | Arrow (t0, t1) ->
    Obj.magic (fun w1 t2 h0 x x0 ->
      Nf_ne (w1, t2,
        Ne_Let (w1, t1, t2, t0,
          (x0 (Cons (t1, w1)) (Lew_cons (t1, w1))
            (reflect t1 (Cons (t1, w1)) (Var_Hyp (w1, t1)))),
          (var_ne w1 (Arrow (t0, t1)) (wkn_var w w1 (Arrow (t0, t1)) h0 h)),
          (reify t0 w1 x))))
```

Obj.magic が現れているのは，論理述語の定義において，base と $\rightarrow$ ケースにおける式の型が異なる為である．Obj.magic が現れてはいるものの，CPS 変換された TDPE に等しいであろうことが見てとれる定義となっていることが分かる．

## 3 $\lambda_{cbv}^{S/R}$ における TDPE の抽出

2 節を受けて，どのように $\lambda_{cbv}^{S/R}$ における TDPE の抽出を行ったかを説明する．対馬らの call-by-value の shift/reset 付き $\lambda$ 計算における TDPE の定義は以下となっている．

$$
\begin{array}{lcl}
t \in \mathrm{Type} & := & b \mid t_1/\alpha \to t_2/\beta \\
v \in \mathrm{Value} & := & x \mid \overline{\lambda} x.v \mid v_0 \,\overline{@}\, v_1 \mid \overline{\langle v \rangle} \mid \overline{S} k.v \\
e \in \mathrm{Expr} & := & x \mid \underline{\lambda} x.e \mid e_0 \,\underline{@}\, e_1 \mid \underline{\langle e \rangle} \mid \underline{S} k.e
\end{array}
$$

$$
\begin{array}{lcl}
\text{Reify ``}\downarrow\text{''} & : & \mathrm{Type} \to \mathrm{Value} \to \mathrm{TLT} \\
\downarrow^{\mathsf{base}} v & := & v \\
\downarrow^{A/a \to B/b} v & := & \underline{\lambda} x^{\diamond}.\underline{S} k_1^{\diamond}.\, \overline{\langle_2} \downarrow^b \overline{\langle \mathrm{let}\ v_1 = (v \,\overline{@}\, \uparrow_A x^{\diamond}) \overline{\mathrm{in}}\ (\downarrow^a (\overline{S_2} k.\underline{\langle} \mathrm{let}\ x^{\diamond} = (k_1^{\diamond} \,\underline{@}\, \downarrow^B v_1)\ \underline{\mathrm{in}}\ k \,\overline{@}\, x^{\diamond} \underline{\rangle}))\rangle} \overline{\rangle_2}
\end{array}
$$

$$
\begin{array}{lcl}
\text{Reify ``}\downarrow\text{''} & : & \mathrm{Type} \to \mathrm{Expr} \to \mathrm{TLT} \\
\uparrow_{\mathsf{base}} e & := & e \\
\uparrow_{A/r \to B/r} e & := & \overline{\lambda} v_1.\overline{S} k_1.\, \uparrow_b \overline{S_2} k.\underline{\langle} \mathrm{let}\ x^{\diamond} = (\underline{\mathrm{let}}\ x_1^{\diamond} = (e \,\underline{@}\, \downarrow^A v_1)\ \underline{\mathrm{in}}\ \overline{\langle_2} \downarrow^a k_1 \,\overline{@}\, \uparrow_B x_1^{\diamond} \overline{\rangle_2})\ \underline{\mathrm{in}}\ k \,\overline{@}\, x^{\diamond} \underline{\rangle}
\end{array}
$$

$\overline{S}$ と $\overline{\langle\rangle}$ は 2 節でのケースと同様，let-insertion の定義の為の shift/reset である．又，$\overline{S_2}$ と $\overline{\langle_2\ \rangle_2}$ は，$\overline{S}$ と $\overline{\langle\rangle}$ よりも一段階上の shift/reset となっている．よって，この TDPE から static な shift/reset を外すには，まず $\overline{S}$ と $\overline{\langle\rangle}$ を CPS 変換した後，さらに $\overline{S_2}$ と $\overline{\langle_2\ \rangle_2}$ を CPS 変換する必要がある．そのようにして二回 CPS 変換をかけて，static な shift/reset をはずしたのが次の定義である：

$$
\begin{array}{lcl}
\text{Reify ``}\downarrow\text{''} & : & \mathrm{Type} \to \mathrm{Value} \to \mathrm{TLT} \\
\downarrow^{\mathsf{base}} v & := & v \\
\downarrow^{A/a \to B/b} v & := & \underline{\lambda} x_1^{\diamond}.\underline{S} k_1^{\diamond}.v \,\overline{@}\, (\downarrow^A x_1^{\diamond}) \,\overline{@}\, \overline{\lambda} v_1.\overline{\lambda} k_2.((\underline{\lambda} g^{\diamond}.k_2 \,\overline{@}\, \uparrow_a g^{\diamond}) \,\underline{@}\, (k_1^{\diamond} \,\underline{@}\, \downarrow^B v_1)) \\
& & \qquad \overline{@}\, \overline{\lambda} v_2.\downarrow^b v_2 \\
\text{Reify ``}\downarrow\text{''} & : & \mathrm{Type} \to \mathrm{Expr} \to \mathrm{TLT} \\
\uparrow_{\mathsf{base}} e & := & e \\
\uparrow_{A/r \to B/r} e & := & \overline{\lambda} v_1.\overline{\lambda} k_1.\overline{\lambda} k_2.((\underline{\lambda} g^{\diamond}.k_2 \,\overline{@}\, \uparrow_b g^{\diamond}) \,\underline{@}\, \\
& & \qquad \underline{\langle} (\underline{\lambda} x_1^{\diamond}.k_1 \,\overline{@}\, (\uparrow_B x_1^{\diamond}) \,\overline{@}\, (\overline{\lambda} v_2.\downarrow^a v_2)) \,\underline{@}\, (e \,\underline{@}\, \downarrow^A v_1) \underline{\rangle})
\end{array}
$$

関数の型は 2 節にて示したモデルの型と同じである．$k_1$ が continuation，$k_2$ が meta continuation を表す変数となっている．このように CPS 変換を 2 回施して，2 階層の継続が出てくる形式を 2CPS のプログラムと呼ぶ．このモデルを参考にして定式化を行っていく．$\lambda_{cbv}^{S/R}$ のシステムや証明の定式化全てが $\lambda_{cbv}$ のそれと全く同じ流れになっており，$\lambda_{cbv}$ の定式化の拡張となっていることに注意されたい．

### 3.1 $\lambda_{cbv}^{S/R}$ の構文

まず $\lambda_{cbv}^{S/R}$ の構文を定義する：

$$
\begin{array}{rcccccl}
\text{monomorphic type} & : & \mathsf{typ}_m & \ni & A & := & \mathsf{base} \mid A_1/A_3 \to A_2/A_4 \\
\text{polymorphic type} & : & \mathsf{typ}_p & \ni & \alpha & := & \mathsf{mono}\ A \mid A_1 \to_p A_2 \\
\text{environment} & : & \mathsf{world} & \ni & w, \Gamma & := & \text{list of } \mathsf{typ}_p \\
\text{variable term} & : & \vdash_{\mathsf{v}} & \ni & v & := & \mathsf{hyp} \mid \mathsf{hyp'} \mid \mathsf{wkn}(v) \\
\text{term} & : & \vdash_{\mathsf{t}} & \ni & p & := & v \mid \mathsf{lam}(p) \mid \mathsf{app}(p_1, p_2) \mid \mathsf{shift}(p) \mid \mathsf{reset}(p)
\end{array}
$$

monomorphic type の関数型 $A_1/A_3 \to A_2/A_4$ においては，$A_1$ 型の式を受け取ると $A_2$ 型の式を返し，その実行に伴い継続の返す型 $A_3$ を $A_4$ へと変化させる様な関数の型である．polymorphic type $A_1 \to_p A_2$ は $\forall r, A_1/r \to A_2/r$ を意味する関数型である．polymorphic type も定義している理由は，completeness の証明において型の整合性をとる為に必要となるからである．又，hyp と hyp' はそれぞれ answer type が monomorophic と polymorphic であるような，(一番内側の binder で束縛されている）変数である．

term の推論規則は以下の如く定義する．$e : w; a \vdash_{\mathsf{t}} A; b$ とあるとき，term $e$ の型が $w; a \vdash_{\mathsf{t}} A; b$ となる．そしてこの型は，world $w$ において $A$ 型を持つ式 $e$ を実行すると継続の返す型が $a$ から $b$ へと変化することを意味として持つ．

$$\frac{}{\mathsf{hyp} : \forall r, (\mathsf{mono}\ A, w; r \vdash_{\mathsf{v}} A; r)} \quad \frac{}{\mathsf{hyp'} : \forall r, (A \to_p B, w; r \vdash_{\mathsf{v}} A/a \to B/a; r)}$$

$$\frac{v : w; a \vdash_{\mathsf{v}} A; b}{\mathsf{wkn}(v) : B, w; a \vdash_{\mathsf{v}} A; b} \quad \frac{v : w; a \vdash_{\mathsf{v}} A; b}{v : w; a \vdash_{\mathsf{t}} A; b}$$

$$\frac{p : \mathsf{mono}\ A, w; a \vdash_{\mathsf{t}} B; b}{\mathsf{lam}(p) : \forall r, (w; r \vdash_{\mathsf{t}} A/a \to B/b; r)} \quad \frac{p : w; r \vdash_{\mathsf{t}} c/a \to A/b; d \quad q : w; b \vdash_{\mathsf{t}} c; r}{\mathsf{app}(p, q) : w; a \vdash_{\mathsf{t}} A; d}$$

$$\frac{p : A \to_p a, w; c \vdash_{\mathsf{t}} c; b}{\mathsf{shift}(p) : w; a \vdash_{\mathsf{t}} A; b} \quad \frac{p : w; a \vdash_{\mathsf{t}} a; A}{\mathsf{reset}(p) : \forall r, (w; r \vdash_{\mathsf{t}} A; r)}$$

そして本研究では，shift/reset 付 $\lambda$ 計算における normal form と neutral term を以下の如く定めた：

$$\begin{array}{lcccll} \text{normal form} & : & \vdash_{\mathsf{nf}} & \ni & nf & := \mathsf{lam}(\mathsf{shift}(nf)) \mid ne \\ \text{neutral term} & : & \vdash_{\mathsf{ne}} & \ni & ne & := v \mid \mathsf{app}(ne, nf) \\ & & & & & \quad \mid \mathsf{let}(nf_1, \mathsf{app}(ne, nf_2)) \mid \mathsf{let}(nf, \mathsf{reset}(ne)) \end{array}$$

normal form は neutral term 又は $\lambda$ の直下に shift が入っている形の項であると定義した。$\mathsf{lam}(\mathsf{shift}())$ は，TDPE の結果の $\lambda$ の直下に shift が入っている項を意味している．この $\lambda$ と shift の関係に対する考え方は，shift/reset の通常の部分評価 [1] においても同様である．一方，neutral term は $\lambda_{cbv}$ での定義に加えて reset を let 文に残すものが入っている．これらの型規則を以下に示す．

$$\frac{p : w; a \vdash_{\mathsf{nf}} A; b}{\mathsf{wkn}(p) : B, w; a \vdash_{\mathsf{nf}} A; b} \quad \frac{p : B \to_p a, \mathsf{mono}\ A, w; c \vdash_{\mathsf{nf}} c; b}{\mathsf{lam}(\mathsf{shift}(p)) : \forall r, (w; r \vdash_{\mathsf{nf}} A/a \to B/b; r)} \quad \frac{p : w; a \vdash_{\mathsf{ne}} A; b}{p : w; a \vdash_{\mathsf{nf}} A; b}$$

$$\frac{p : w; a \vdash_{\mathsf{ne}} A; b}{\mathsf{wkn}(p) : B, w; a \vdash_{\mathsf{ne}} A; b} \quad \frac{}{\mathsf{hyp} : A, w; a \vdash_{\mathsf{ne}} A; b} \quad \frac{}{\mathsf{hyp} : \forall r, (A \to_p B, w; r \vdash_{\mathsf{ne}} A/a \to B/a; r)}$$

$$\frac{p : w; r \vdash_{\mathsf{ne}} c/a \to A/b; d \quad q : w; b \vdash_{\mathsf{nf}} c; r}{\mathsf{app}(p, q) : w; a \vdash_{\mathsf{ne}} A; d}$$

$$\frac{p : \mathsf{mono}\ c, w; a \vdash_{\mathsf{nf}} A; b \quad q : w; r' \vdash_{\mathsf{ne}} c'/b \to A/b'; r \quad q' : w; b' \vdash_{\mathsf{nf}} c'; r'}{\mathsf{let}(p, \mathsf{app}(q, q')) : w; a \vdash_{\mathsf{ne}} A; r}$$

$$\frac{p : \mathsf{mono}\ c, w; a \vdash_{\mathsf{nf}} A; r \quad q : w; a' \vdash_{\mathsf{nf}} a'; c}{\mathsf{let}(p, \mathsf{reset}(q)) : w; a \vdash_{\mathsf{ne}} A; r}$$

### 3.2 Soundness と Completeness の証明

2 節での soundness と completeness の証明と同様に，論理述語を用いて証明を行う．まずは，$\vdash$ を用いて monomorphic type と polymorphic type に関する論理述語をそれぞれ定義する．

$$\begin{array}{lcl} R^2_w(\mathsf{base}) & := & \forall r, w; r \vdash \mathsf{base}; r \\ R^2_w(A/a \to B/b) & := & \forall w_1, \forall T_1, w \le w_1 \Rightarrow R^2_{w_1}(A) \Rightarrow \\ & & (\forall w_2, \forall T, w_1 \le w_2 \Rightarrow R^2_{w_2}(B) \Rightarrow \\ & & (\forall w_3, w_2 \le w_3 \Rightarrow R^2_{w_3}(a) \Rightarrow \forall r, w_3; r \vdash T; r) \Rightarrow \\ & & \forall r, w_2; r \vdash T; r) \Rightarrow \\ & & (\forall w_2, w_1 \le w_2 \Rightarrow R^2_{w_2}(b) \Rightarrow \forall r, w_2; r \vdash T_1; r) \Rightarrow \\ & & \forall r, w_1; r \vdash T_1; r \end{array}$$

$$\frac{R^2_w(A)}{R^3_w(\mathsf{mono}\ A)} \quad \frac{\forall r, R^2_w(A/r \to B/r)}{R^3_w(A \to_p B)}$$

$$Rs^2_w((\alpha_1,\alpha_2,...,\alpha_n)) \;:=\; (R^3_w(\alpha_1)) \wedge (R^3_w(\alpha_2)) \wedge ... \wedge (R^3_w(\alpha_n))$$

$R^2$ の定義が 2CPS で書かれていることに注意されたい．$R^2_w(A/a \to B/b)$ の二行目から四行目にかけて括弧でくくられている式が continuation，五行目の式が meta continuation にそれぞれ対応している．この論理述語においても，2 節と同様にして，soundness を証明することが出来る．ここで $\leq$ は，定理 1 の証明にて使用した定義と同じものを使うこととする．

**公理 2** $\forall w, \forall w', \forall A, \forall a, \forall b, w \leq w' \;\Rightarrow\; w; a \vdash A; b \;\Rightarrow\; w'; a \vdash A; b$

**補題 3** $\forall w, \forall w', \forall \Gamma, \forall a, \forall b, w \leq w' \;\Rightarrow\; Rs^2_w(\Gamma) \;\Rightarrow\; Rs^2_{w'}(\Gamma)$

Proof. 公理 2 を用いる． □

**定理 3 (Soundness, $\lambda^{S/R}_{cbv}$)**

$$\begin{array}{l}
\forall A, \forall a, \forall b, \forall \Gamma, \forall T, \Gamma; a \vdash_{\mathsf{t}} A; b \;\Rightarrow\; \forall w, \\
(Rs'_w(\Gamma) \;\Rightarrow \\
(\forall w_1, \forall T_1, w \leq w_1 \;\Rightarrow\; R^2_{w_1}(A) \;\Rightarrow \\
\;(\forall w_2, w_1 \leq w_2 \;\Rightarrow\; R^2_{w_2}(a) \;\Rightarrow\; \forall r, w_2; r \vdash T_1; r) \;\Rightarrow \\
\;\forall r, w_1; r \vdash T_1; r) \;\Rightarrow \\
(\forall w_1, w \leq w_1 \;\Rightarrow\; R'_{w_1}(b) \;\Rightarrow\; \forall r, w_1; r \vdash T; r) \;\Rightarrow \\
\forall r, w; r \vdash T; r)
\end{array}$$

Proof. $\Gamma; a \vdash_{\mathsf{t}} A; b$ における帰納法.（補題 3 を使用する.） □

これら補題・定理の証明は，ほとんどそのまま $\lambda_{cbv}$ のそれを拡張しているだけである．

次に，TDPE を抽出する為の completeness を行う．まずは，上の論理述語の定義における $\vdash$ を $\vdash_{\mathsf{nf}}$ に置き換える．

$$\begin{array}{lcl}
R'_w(\mathsf{base}) & := & \forall r, w; r \vdash_{\mathsf{nf}} \mathsf{base}; r \\
R'_w(A/a \to B/b) & := & \forall w_1, \forall T_1, w \leq w_1 \;\Rightarrow\; R_{w_1}(A) \;\Rightarrow \\
 & & (\forall w_2, \forall T, w_1 \leq w_2 \;\Rightarrow\; R_{w_2}(B) \;\Rightarrow \\
 & & \;(\forall w_3, w_2 \leq w_3 \;\Rightarrow\; R_{w_3}(a) \;\Rightarrow\; \forall r, w_3; r \vdash_{\mathsf{nf}} T; r) \;\Rightarrow \\
 & & \;\forall r, w_2; r \vdash_{\mathsf{nf}} T; r) \;\Rightarrow \\
 & & (\forall w_2, w_1 \leq w_2 \;\Rightarrow\; R_{w_2}(b) \;\Rightarrow\; \forall r, w_2; r \vdash_{\mathsf{nf}} T_1; r) \;\Rightarrow \\
 & & \forall r, w_1; r \vdash_{\mathsf{nf}} T_1; r
\end{array}$$

$$\frac{R'_w(A)}{R''_w(\mathsf{mono}\ A)} \qquad \frac{\forall r, R'_w(A/r \to B/r)}{R''_w(A \to_p B)}$$

$$Rs'_w((\alpha_1,\alpha_2,...,\alpha_n)) \;:=\; (R''_w(\alpha_1)) \wedge (R''_w(\alpha_2)) \wedge ... \wedge (R''_w(\alpha_n))$$

ここでの $\leq$ は以下の如く定義する．これは 2 節での completeness の証明に使用した $\leq$ を，world の要素を二つまで増やしても 2 項関係が成立する様に拡張しただけである．この様に定義することによって，2 節ではカウンタをひとつ増やすだけであったのが，今回はカウンタを二つ増やす役割をも持たせている．

$$\begin{array}{ll}
(i) & \forall A,\ \ w \leq (A, w) \\
(ii) & \forall A, \forall B,\ \ w \leq (A, B, w)
\end{array}$$

そして completeness の定義は以下の如くとなる．

**定理 4 (Completeness, $\lambda_{cbv}^{S/R}$)**

$$\forall A,\quad (i)\quad \forall w, R'_w(A) \;\Rightarrow\; \forall r, w; r \vdash_{\mathsf{nf}} A; r$$
$$(ii)\quad \forall w, \forall r, w; r \vdash_{\mathsf{v}} A; r \;\Rightarrow\; R'_w(A)$$

Proof. $A$ における帰納法.（補題 2 を使用する.） □

定理 4 の証明は，本節のはじめに示した TDPE のモデルを参考に，モデル定義にほぼ沿って証明を行うことが出来る.

### 3.3 TDPE の抽出

2 節と同様に，soundness と completeness の証明は,（shift/reset 付きにおける）CPS 変換と CPS の TDPE 一対一の関係にある．completeness の証明から得られた Coq 上の関数を 2 節と同様，型に関する帰納法の原理の展開と自明な $\beta$ 簡約を行ったものが以下である.

```
Fixpoint reify (t : typ) : (forall w : world, R w t -> forall r : typ, nf w t r r) :=
  match t as t0 return (forall w : world, R w t0 -> forall r : typ, nf w t0 r r) with
  | base => (fun (w : world) (X : R w base) (r : typ) => X r)
  | arrow t0 t1 t2 t3 => (fun (w : world) (X : R w (arrow t0 t1 t2 t3)) (r : typ) =>
     nf_LamShift w t0 t1 t2 t3 t3
       (X (arrowP t1 t2 :: mono t0 :: w) t3
          (lew_cons_2 (arrowP t1 t2) (mono t0) w)
          (reflect t0 (arrowP t1 t2 :: mono t0 :: w)
             (fun r0 : typ =>
              var_Wkn (mono t0 :: w) t0 (arrowP t1 t2) r0 r0
                (var_Hyp w t0 r0)))
          (fun (w2 : world) (T : typ)
             (H : arrowP t1 t2 :: mono t0 :: w <== w2)
             (X0 : R w2 t1)
             (X1 : forall w3 : world,
                   w2 <== w3 -> R w3 t2 -> forall r0 : typ, nf w3 T r0 r0)
             (r0 : typ) =>
           nf_ne w2 T r0 r0
             (ne_LetApp w2 T r0 r0 r0 t2 r0 t1 r0 r0
                (X1 (mono t2 :: w2) (lew_cons_1 (mono t2) w2)
                   (reflect t2 (mono t2 :: w2) (var_Hyp w2 t2)) r0)
                (ne_var w2 (arrow t1 t2 r0 r0) r0 r0
                   (wkn_var H (var_Hyp2 (mono t0 :: w) t1 t2 r0 r0)))
                (reify t1 w2 X0 r0)))
          (fun (w2 : world) (_ : arrowP t1 t2 :: mono t0 :: w <== w2)
             (X0 : R w2 t3) (r0 : typ) => reify t3 w2 X0 r0)
          t3)
       r)
  end
with reflect (t : typ) : (forall w : world, (forall r : typ, var w t r r) -> R w t) :=
  match t as t0 return (forall w : world, (forall r : typ, var w t0 r r) -> R w t0) with
  | base => (fun (w : world) (H : forall r : typ, var w base r r) (r : typ) =>
     nf_ne w base r r (var_ne (H r)))
  | arrow t0 t1 t2 t3 =>
    (fun
       (w : world)
       (H : forall r : typ, var w (arrow t0 t1 t2 t3) r r)
       (w1 : world)
       (T1 : typ)
       (H0 : w <== w1)
       (X : R w1 t0)
```

```
      (X0 : forall (w2 : world) (T : typ),
            w1 <== w2 ->
            R w2 t1 ->
            (forall w3 : world,
             w2 <== w3 -> R w3 t2 -> forall r : typ, nf w3 T r r) ->
            forall r : typ, nf w2 T r r)
      (X1 : forall w2 : world,
            w1 <== w2 -> R w2 t3 -> forall r : typ, nf w2 T1 r r)
      (r : typ) =>
    nf_ne w1 T1 r r
      (ne_LetReset w1 T1 r r t3 r t2
         (X1 (mono t3 :: w1) (lew_cons_1 (mono t3) w1)
            (reflect t3 (mono t3 :: w1) (var_Hyp w1 t3)) r)
         (ne_LetApp w1 t2 t2 t2 t3 t1 t3 t0 t3 t3
            (X0 (mono t1 :: w1) t2 (lew_cons_1 (mono t1) w1)
               (reflect2 t1 (mono t1 :: w1) (var_Hyp w1 t1))
               (fun (w3 : world) (_ : mono t1 :: w1 <== w3)
                  (X2 : R w3 t2) (r0 : typ) => reify t2 w3 X2 r0) t2)
            (ne_var w1 (arrow t0 t1 t2 t3) t3 t3 (wkn_var H0 (H t3)))
            (reify t0 w1 X t3))))
end.
```

nf_LamShift は normal form の定義における lam(shift(_)) に，ne_LetApp と ne_LetReset は neutral term の定義における let(_, app(_, _)) と let(_, reset(_)) にそれぞれ対応している．又,var_Hyp2, mono, arrowP,arrow はそれぞれ hyp', $(\_ \to_p \_)$, $(\_/\_ \to \_/\_)$ を表している．lew_cons_1 と lew_cons_2 は，completeness の証明で使用した $\leq$ の $(i)$ と $(ii)$ にそれぞれ対応している．この関数定義においても，上述した CPS の shift/reset 付 TDPE と等しいであろうことが見てとれる．この関数を OCaml のプログラムとして抽出すると以下の如くとなる．

```
(** val reify : typ -> world -> r -> typ -> nf **)

let rec reify t w x r0 =
  match t with
  | Base -> Obj.magic x r0
  | Arrow (t0, t1, t2, t3) ->
    Nf_LamShift (w, t0, t1, t2, t3, t3,
      (Obj.magic x (Cons ((ArrowP (t1, t2)), (Cons ((Mono t0), w)))) t3
        (Lew_cons_2 ((ArrowP (t1, t2)), (Mono t0), w))
        (reflect t0 (Cons ((ArrowP (t1, t2)), (Cons ((Mono t0), w))))
          (fun r1 -> Var_Wkn ((Cons ((Mono t0), w)), t0, (ArrowP (t1, t2)),
          r1, r1, (Var_Hyp (w, t0, r1)))))
        (fun w2 t4 h x0 x1 r1 ->
          Nf_ne (w2, t4, r1, r1,
            Ne_LetApp (w2, t4, r1, r1, r1, t2, r1, t1, r1, r1,
              x1 (Cons ((Mono t2), w2)) (Lew_cons_1 ((Mono t2), w2))
                 (reflect2 t2 (Cons ((Mono t2), w2))
                   (fun x2 -> Var_Hyp (w2, t2, x2)))
                 r1,
              Ne_var (w2, (Arrow (t1, t2, r1, r1)), r1, r1,
                wkn_var (Cons ((ArrowP (t1, t2)), (Cons ((Mono t0), w))))
                        w2 h (Arrow (t1, t2, r1, r1)) r1 r1
                        (Var_Hyp2 ((Cons ((Mono t0), w)), t1, t2, r1, r1))),
              reify t1 w2 x0 r1)))
        (fun w2 x0 x1 r1 -> reify t3 w2 x1 r1)
        t3),
      r0)
```

```
(** val reflect : typ -> world -> (typ -> var) -> r **)

and reflect t w h =
  match t with
  | Base ->
    Obj.magic (fun r0 ->
      Nf_ne (w, Base, r0, r0, (var_ne w Base r0 r0 (h r0))))
  | Arrow (t0, t1, t2, t3) ->
    Obj.magic (fun w1 t4 h0 x x0 x1 r0 ->
      Nf_ne (w1, t4, r0, r0,
        Ne_LetReset (w1, t4, r0, r0, t3, r0, t2,
          x1 (Cons ((Mono t3), w1)) (Lew_cons_1 ((Mono t3), w1))
             (reflect t3 (Cons ((Mono t3), w1))
                      (fun x2 -> Var_Hyp (w1, t3, x2)))
             r0,
          Ne_LetApp (w1, t2, t2, t2, t3, t1, t3, t0, t3, t3,
            x0 (Cons ((Mono t1), w1)) t2 (Lew_cons_1 ((Mono t1), w1))
               (reflect t1 (Cons ((Mono t1), w1))
                        (fun x2 -> Var_Hyp (w1, t1, x2)))
               (fun w3 x2 x3 r1 -> reify t2 w3 x3 r1)
               t2,
            Ne_var (w1, (Arrow (t0, t1, t2, t3)), t3, t3,
              wkn_var w w1 h0 (Arrow (t0, t1, t2, t3)) t3 t3 (h t3)),
              reify t0 w1 x t3))))
```

ここでも Obj.magic が現れる理由は，2 節でのそれと同じである．しかしこの事を除けば CPS の shift/reset 付 TDPE と同一の構造をしていること，そして [6] にて提示された shift/reset 付 TDPE とは全く異なる構造をしていることがみてとれる．

## 4 まとめと今後の課題

本研究では call-by-value の $\lambda$ 計算における TDPE の抽出を定式化し，その方法を拡張する形で call-by-value の shift/reset 付 $\lambda$ 計算における TDPE を抽出した．定式化にあたっては，shift/reset 付き $\lambda$ 計算における normal form，neutral term の構造を提示した．抽出されたプログラムは複雑にはなっていたものの，[8] によって提示された shift/reset 付 TDPE を CPS 変換したプログラムと同一であろうことがみてとれた．

completeness の証明に対応する TDPE は CPS で書かれており，入力として CPS された term を受け取る．一方，soundness の証明からは CPS 変換を得られる．そうすると，soundness の証明の継続に completeness の証明の前半部分を入れる形で両者を合成すると，直接形式の入力を受け取り，その正規形を返す形の TDPE が得られる可能性がある．単純に関数合成したのでは CPS 変換部分で入力の構文についての場合分けが入るので TDPE とは言いがたいが，両者を fusion などの手法を使って合成出来れば直接形式の TDPE を得られる可能性もある．この検討が今後の課題である．

## 謝辞

種々の有益なコメントを下さった査読者の方々に感謝いたします．

# 参考文献


[1] Asai, K. “Logical Relations for Call-by-value Delimited Continuations,” *Trends in Functional Programming (TFP 2005), Vol. 6*, pp. 63–78, Intellect (2007).

[2] Coquand, C. “A Formalised Proof of the Soundness and Completeness of a Simply Typed Lambda-Calculus with Explicit Substitutions” *Higher-Order and Symbolic Computation*, Volume 15, Issue 1, pp. 57-90 (March 2002).

[3] Danvy, O. “Type-Directed Partial Evaluation,” *Conference Record of the 23rd Annual ACM Symposium on Principles of Programming Languages*, pp. 242–257 (January 1996).

[4] Filinski, A. “Normalization by Evaluation for the Computational Lambda-Calculus,” In S. Abramsky, editor, *Typed Lambda Calculi and Applications (LNCS 2044)*, pp. 151–165 (May 2001).

[5] Ilik, D. “Continuation-passing style models complete for intuitionistic logic”, *Annals of Pure and Applied Logic, Special issue: Classical logic and computation 2010.*

[6] Ilik, D. “A formalized type-directed partial evaluator for shift and reset”, *http://arxiv.org/abs/1210.2094.*

[7] Pierce, B. C. *Types and Programming Languages*, Cambridge: MIT Press (2002).

[8] Tsushima, K., and K. Asai “Towards Type-Directed Partial Evaluation for Shift and Reset,” *Proceedings of the 2009 Workshop on Normalization by Evaluation*, pp. 57–64 (August 2009).