プログラムに対する変換の正しさの形式検証

Formal Verification of Program Transformation

吉原宏之 †
Hiroyuki YOSHIHARA
† 筑波大学理工学研究科
Master's Program in Science and Engineering,
University of Tsukuba
yoshihara @ logic.cs.tsukuba.ac.jp

亀山幸義 ††
Yukiyoshi KAMEYAMA
†† 筑波大学システム情報工学研究科
Department of Computer Science,
University of Tsukuba
kameyama @ acm.org

本研究の目的は，CPS 変換などのプログラム変換に関する様々な性質を形式的に厳密に証明することである．λ 計算に基づく関数型プログラム言語のプログラム変換に対する形式検証は，変数束縛を含む高階のデータを正しく扱う必要があるため，一階のデータを扱うプログラムの検証に比べて困難である．変数束縛を持つデータ構造を表現するために，高階抽象構文 (HOAS) と呼ばれる表現法があり，高階抽象構文をサポートする定理証明システムとして Twelf がある．本研究では，プログラム変換の一種として，CPS 変換 (継続渡し方式への変換) を取り上げ，その健全性を Twelf で厳密に証明することを試みた．本稿では，この形式証明の手法について述べ，問題点と解決法を述べる．

1 はじめに

近年，バグのないソフトウェアを開発するために，適切に設計されたプログラム言語が必要とされている．例えば，プログラム言語 ML は 多相型システムに基づく λ 計算を核言語とした理論的な研究の成果が盛り込まれたプログラム言語である．ML は，定理証明のための体系 LCF に対するメタ言語として設計された．しかし，定理証明分野における成果をプログラム言語研究に直接取りいれる活動は，これまでは必ずしも盛んではなく，プログラム言語に関する研究論文の証明も，人手によるものがほとんどであった．

これに対して，近年，プログラム言語の研究に定理証明の研究成果を取り入れる試みがある ([1], POPLmark Challenge, 2005)．ここでは,「プログラム言語の論文における証明を全て形式化しよう」という壮大な夢への第一歩として，プログラム言語分野において形式化が難しいとされる数個の共通問題を様々な定理証明システムで解き，それらの解を比較する，という試みがなされている．

本研究の目的は，λ 計算に基づくプログラム言語で書かれたプログラムに対して，CPS 変換の健全性・完全性を形式的に証明することである．CPS 変換の健全性・完全性の証明では，変数の取り扱いが重要であり，また，λ 計算に関する証明では，変数の名前換え (α 同値性) を行っても種々の定義や定理が保たれる，という性質が必要であるが，人手による非形式的証明や，一階抽象構文による形式証明では，省略されてしまうことが多い．このような問題を解決するのが，Twelf がサポートする高階抽象構文である．本稿では，第一歩として，定理証明システム Twelf を用いて，値呼びの λ 計算に対する CPS 変換の健全性を形式的に証明した．

本稿の構成 本稿の構成は次の通りである．まず，2 節で表現に関する問題点について述べる．次に，3 節で具体的に DS 式，CPS 項， CPS 変換の表現し，表現の妥当性について述べ，CPS 変換の健全性について述べる．さらに，4 節では，Twelf で証明することの利点について述べ，最後に，まとめと今後の課題を述べる．

2 表現に関する問題点

本節では，対象となるプログラム言語を表現するために，よく用いられる一階抽象構文 (First-order Abstract Syntax, FOAS) ではなく，高階抽象構文 (Higher-order Abstract Syntax, HOAS) を採用した理由を述べる．

2.1 変数名を自然数や文字列で表現した一階抽象構文

多くの形式化で用いられる一階抽象構文による表現では，変数を表現するために，文字列や自然数を用いる．この場合，α同値や代入の扱いが問題になる．

例えば，変数名を自然数で表現すると，

$$\lambda x.\lambda y.x\ y\ z$$

は

$$\lambda 1.\lambda 2.1\ 2\ 0$$

や

$$\lambda 2.\lambda 1.2\ 1\ 3$$

となる．この表現では，代入の際に変数の名前の衝突を回避するため，α 同値性を考慮しなければならない．これは，どのようなプログラム言語でも生じる問題であるので，形式化ごとに定義を繰返すのは無駄であるし，間違いがおきやすい．

この表現法でさらに問題となるのは，すべての定義や定理が，α 同値による名前換えを行っても保たれる，という性質を証明しなければならない，という点である．人手による証明では，この性質を明示的に証明することはほとんどないが，厳密な形式証明の観点からは，省略することはできない．

南出・大熊 ([6], 2003) によるの検証では，一階抽象構文による形式化を行っている．この場合，代入される項が自由変数を持たない場合に限定されるため，名前換えを必要としない簡潔な代入の定義を採用している．我々の研究の目標は，自由変数を含む項に対するプログラム変換の形式検証であるため，このアプローチは利用できない．

2.2 de Bruijn インデックスに基づく抽象構文

de Bruijn インデックスに基づく抽象構文では，変数束縛の構造はインデックスによって決定される．例えば，

$$\lambda x.\lambda y.(x\ y)$$

は

$$\lambda.\lambda.(2\ 1)$$

と表現され，α 同値な他の表現は存在しないため，α 同値性を考慮する必要はない．しかし，よく知られているようにインデックスは文脈によって変化するので，煩雑なインデックス操作が必要となり，また直感的でもない．例えば，

$$\lambda x.(\lambda y.x\ y)\ x$$

は

$$\lambda.(\lambda.(2\ 1))\ 1$$

と表現され，同じ変数が別のインデックスを持つ場合が生じる．この問題点は，自動証明ではなく人手による形式証明を行なう際には大きな問題点となる．さらに，前項の表現と同様，形式化の対象であるプログラム言語ごとに代入を定義し，その性質を検証する必要がある，という問題点がある．

2.3 HOAS (高階抽象構文)

HOAS で鍵となるアイディアは，束縛変数の名前のみ異る，すべての項を同一視し，束縛変数を導入する際，その変数は新しい変数であることを保証するように表現することである．

以上のような HOAS の条件を満すように，オブジェクト言語を表現する手段を提供するメタ言語に LF (Edinburgh Logical Framework) がある．LF では，オブジェクト言語の変数をメタ言語の変数に写すことで，オブジェクトレベルのすべての束縛構造をメタレベルの束縛演算子で表現することができる．これにより，新しい変数の生成や α 同値性，名前換え，代入をメタ言語である LF が提供するため，変数の名前を使った一階抽象構文での問題は生じない．本研究の目的は定理を厳密に証明することであるので，HOAS が適している．

2.4 Logical Framework

Edinburgh Logical Framework ([3], Harper *et al.*, 1993) は，プログラム言語や論理における演繹システムを定義するためのフレームワークであり，LF を実装したシステムの一つに Twelf ([7], Pfenning & Schürmann, 1999) がある．

LF は，依存 (関数) 型，型の族，$\beta\eta$-簡約を持つ λ 計算であり，その構文と主な判断は，以下の通りである．詳細は文献 [3] を参照されたい．

$$\begin{array}{rrcl}
\text{(種)} & K & ::= & \mathsf{type} \mid \Pi x{:}A.K \\
\text{(型)} & A & ::= & a \mid A\ M \mid \Pi x{:}A_1.A_2 \\
\text{(対象)} & M & ::= & c \mid x \mid \lambda x{:}A.M \mid M_1\ M_2 \\
\\
\text{(シグネチャ)} & \Sigma & ::= & \cdot \mid \Sigma, a{:}K \mid \Sigma, c{:}A \\
\text{(文脈)} & \Gamma & ::= & \cdot \mid \Gamma, x{:}A
\end{array}$$

$$
\begin{array}{rl}
\text{(型付けの判断)} & \Gamma \vdash_\Sigma U : V \\
\text{(等しさの判断)} & \Gamma \vdash_\Sigma U = U', \\
 & \Gamma \vdash_\Sigma V = V' \\
\text{(Canonical form の判断)} & \Gamma \vdash_\Sigma M \Uparrow A, \\
 & \Gamma \vdash_\Sigma A \Uparrow \mathsf{type}
\end{array}
$$

ここで，U は対象または型，V は型または種を表わし，a, c はそれぞれ型レベルの定数，対象レベルの定数であり，x は 対象レベルの変数である．A_2 の中に x が出現しない場合は，$\Pi x\!:\!A_1.A_2$ を $A_1 \to A_2$ と書く．シグネシャは，定数を宣言するために使われる．これは演繹システムを定義することに対応している．等しさ (definitional equality) の判断は，$\beta\eta$-簡約の規則を含んでおり，Canonical form の判断は，対象 M は A 型の canonical form であること，型 K は種 type の canonical form であることを主張している．LF での canonical form は，β-normal η-long normal form のことであり，β 簡約と η 拡張を繰返して得た正規形である．また， LF の型検査は決定可能であることが知られている．

2.5 Twelf

Twelf は LF の実装の 1 つであるが，LF 型理論に対する型検査・型推論の機能だけでなく，LF で表現されたメタ定理の証明のための強力な機構を備えている点に特徴がある．

Isabelle/HOL など多くの定理証明システムでは，帰納的定義により型や集合を定義することができ，推論方法として，その定義に対応する構造的帰納法を用いることができる．しかし，λ 計算の項を自然に HOAS で定義しようとすると，定義したい述語の負 (negative) の出現を含むため，帰納的定義として表現することはできない．例えば，λ 項を HOAS で定義すると以下のようになる．

$$
\frac{\begin{array}{c}\overline{x : term} \\ \vdots \\ M\,x : term\end{array}}{\lambda M : term}\ \mathrm{lam}^x
\qquad
\frac{M_0 : term \quad M_1 : term}{(@\, M_0\, M_1) : term}\ \mathrm{app}
$$

この定義は以下の論理式に対応している．

$$
\begin{array}{l}
M : term \triangleq \\
\quad \exists f.(\forall v.((v : term \supset f\, v : term)\,\&\,(M = \lambda f)))\,\& \\
\quad \exists f.\exists g.(f : term\,\&\,g : term\,\&\,(M = (@\, f\, g)))
\end{array}
$$

ここで，述語 $term$ は負の出現を含むため，帰納的定義にならない．

Twelf ではこの問題点を克服するため，証明の大きさに関する帰納法等を使うことができる．例えば，型付き λ 計算を Twelf で表現した場合，Subject Reduction 定理は以下の形のメタ定理となる．

メタ定理 (Subject Recution): $\Gamma \triangleright E : T$ が証明可能で，$E \longrightarrow E'$ が証明可能ならば，$\Gamma \triangleright E' : T$ が証明可能である．

ここで,「$\Gamma \triangleright E : T$ が証明可能である」という表現は，体系内の定理ではない点に注意されたい．従って，Subject Reduction は，定理ではなくメタ定理である．

Twelf は，メタ定理の証明チェックをするため，以下の検査を提供しており，これらの検査を全てパスすることにより，メタ定理が証明される．

- 型推論および型検査
- mode 検査
- world 検査
- termination 検査
- coverage 検査

mode 検査は関数，型の族の引数の入出力が指定された通りに振舞うかを検査し，termination 検査は，関数や型の族の呼び出しが停止するかどうかを検査し，coverage 検査は，関数や型の族のすべての入力に対して出力が決まるかどうかを検査する．また，world 検査については，文献 [4] を参照されたい．

なお，文献 [4] では，Twelf meta theorem prover と記述されているが，自動証明機能はなく，証明チェック機能のみ提供されている．文献 [10] の Twelf への自動証明機能の追加とは異なることに注意されたい．

また，Twelf における推論機構は,「証明の大きさに関する帰納法」には限定されない．一般には，LF 型理論で表現できる任意の項の canonical form に関する帰納法を用いることができる．

3 CPS 変換の形式化と検証

本節では，本稿で扱う CPS 変換とその性質の形式化について説明する．

変換の対象とする言語 (DS, Direct-Style) は，純粋な λ 計算であり，その構文を図 1 に示す．本稿で扱う CPS 変換は，Plotkin-Style の値呼びの CPS 変

換である．この変換を関数として定義したものを図 2 に示す．また，CPS 変換後の構文を図 3 に示す．

文献 [9] において，CPS 項に対する $\beta\eta$-equality と，DS 式に対する Moggi の computational lambda calculus による equality が一致する (健全かつ完全である) ことが証明されている．本稿では，この証明のうち健全性を形式的に表現し，証明した．

証明の形式化において特に重要なのは，非形式的な項や証明をどのように形式的な表現に対応付けるか，という表現方法である．概念や定義の表現が妥当 (adequate) でなければ，それに対する証明の形式化は意味を持たない．Twelf を用いた形式証明においては，非形式的な表現を LF の canonical object に対応付ける関数が合成的 (compositional) な全単射 (bijection) のとき，この表現が妥当であるとされる．

我々は，これに従い，DS 式，CPS 項，CPS 変換などを，合成的な全単射により LF の項や型に対応付けた．本稿では，DS 式の表現とその妥当性のみについて述べる．

3.1 LF での DS 式の形式化

図 1 の構文は，以下のような関数 $\ulcorner\cdot\urcorner$ を帰納的に定義することで LF で表現できる．

DExp:

$$\begin{aligned} \ulcorner e_0\, e_1\urcorner &\triangleq \mathtt{dapp}\ulcorner e_0\urcorner\ulcorner e_1\urcorner \\ \ulcorner t\urcorner &\triangleq \mathtt{dtriv\text{->}dexp}\ulcorner t\urcorner \end{aligned}$$

DTriv:

$$\ulcorner \mathsf{lam}\, x.e\urcorner \triangleq \mathtt{dlam}\,(\lambda x : \mathtt{dtriv}.\ulcorner e\urcorner)$$

DIde:

$$\ulcorner x\urcorner \triangleq x$$

ここで，以下の定数宣言からなるシグネチャ Σ を使用した．

$$\begin{aligned} \mathtt{dexp} &: \mathtt{type} \\ \mathtt{dtriv} &: \mathtt{type} \\ \mathtt{dapp} &: \mathtt{dexp} \to \mathtt{dexp} \to \mathtt{dexp} \\ \mathtt{dtriv\text{->}dexp} &: \mathtt{dtriv} \to \mathtt{dexp} \\ \mathtt{dlam} &: (\mathtt{dtriv} \to \mathtt{dexp}) \to \mathtt{dtriv} \end{aligned}$$

補題：

任意の DExp 式 e，DTriv 式 t に対して，それらの自由変数がたかだか $x_1, \cdots, x_n$ であれば，LF において，

$$\Gamma \vdash_\Sigma \ulcorner e\urcorner \Uparrow \mathtt{dexp},$$
$$\Gamma \vdash_\Sigma \ulcorner t\urcorner \Uparrow \mathtt{dtriv}$$

が導出可能である．ただし，$\Gamma = x_1 : \mathtt{dtriv}, \cdots, x_n : \mathtt{dtriv}$ とする．

次に，$\ulcorner\cdot\urcorner$ の逆関数 $\llcorner\cdot\lrcorner$ を以下のように定義する．

dexp:

$$\begin{aligned} \llcorner \mathtt{dapp}\, E_0\, E_1\lrcorner &\triangleq \llcorner E_0\lrcorner\llcorner E_1\lrcorner \\ \llcorner \mathtt{dtriv\text{->}dexp}\, T\lrcorner &\triangleq \llcorner T\lrcorner \end{aligned}$$

dtriv:

$$\begin{aligned} \llcorner \mathtt{dlam}\,(\lambda x : \mathtt{dtriv}.E)\lrcorner &\triangleq \mathsf{lam}\, x.\llcorner E\lrcorner \\ \llcorner x\lrcorner &\triangleq x \end{aligned}$$

補題：

$\Gamma = x_1 : \mathtt{dtriv}, \cdots, x_n : \mathtt{dtriv}$ とする．LF において，

$$\Gamma \vdash_\Sigma E \Uparrow \mathtt{dexp},$$
$$\Gamma \vdash_\Sigma T \Uparrow \mathtt{dtriv}$$

が導出可能であれば，$\llcorner E\lrcorner \in$ DExp，かつ $\llcorner T\lrcorner \in$ DTriv であり，また，これらの項の自由変数はたかだか $x_1, \cdots, x_n$ である．

定理 (DS 項の表現の妥当性)：

たかだか $x_1, \cdots, x_n$ を自由変数として含む DExp 式 e， DTriv 式 t を，以下の判断が導出可能な LF 項 E, T にそれぞれ対応付ける関数 $\ulcorner\cdot\urcorner$ は全単射である．

$$x_1 : \mathtt{dtriv}, \cdots, x_n : \mathtt{dtriv} \vdash_\Sigma E \Uparrow \mathtt{dexp},$$
$$x_1 : \mathtt{dtriv}, \cdots, x_n : \mathtt{dtriv} \vdash_\Sigma T \Uparrow \mathtt{dtriv}$$

また，この全単射は合成的である．すなわち，任意の DExp 式 e, DTriv 式 t, t' に対して，以下の判断が導出可能である．

$$\Gamma \vdash_\Sigma \ulcorner [t/x]e\urcorner = [\ulcorner t\urcorner/x]\ulcorner e\urcorner,$$
$$\Gamma \vdash_\Sigma \ulcorner [t/x]t'\urcorner = [\ulcorner t\urcorner/x]\ulcorner t'\urcorner.$$

DExp	$\ni$	$e ::= e_0\, e_1 \mid t$	(DS expressions)
DTriv	$\ni$	$t ::= x \mid \mathsf{lam}\, x.e$	(DS trivial expressions)
DIde	$\ni$	x	(DS identifiers)

図 1: The BNF of DS expressions

$$
\begin{array}{rcl}
[\![e_0\, e_1]\!]^{\mathrm{DExp}} & = & \mathsf{lam}\, k.[\![e_0]\!]^{\mathrm{DExp}}\, \mathsf{lam}\, v_0.[\![e_1]\!]^{\mathrm{DExp}}\, \mathsf{lam}\, v_1.v_0\, v_1\, k \\
 & & \text{where } k,\ v_0 \text{ and } v_1 \text{ are fresh} \\
[\![t]\!]^{\mathrm{DExp}} & = & \mathsf{lam}\, k.k\, [\![t]\!]^{\mathrm{DTriv}} \\
 & & \text{where } k \text{ is fresh} \\
[\![x]\!]^{\mathrm{DTriv}} & = & x \\
[\![\mathsf{lam}\, x.e]\!]^{\mathrm{DTriv}} & = & \mathsf{lam}\, x.[\![e]\!]^{\mathrm{DExp}}
\end{array}
$$

図 2: The left-to-right, call-by-value CPS transformation formulated as a function

CRoot	$\ni$	$r ::= \mathsf{lam}\, k.e \mid t_0\, t_1$	(CPS terms)
CExp	$\ni$	$e ::= r\, cont \mid k\, t$	(CPS expressions)
CTriv	$\ni$	$t ::= x' \mid \mathsf{lam}\, x'.r \mid v$	(CPS trivial expressions)
CCont	$\ni$	$cont ::= \mathsf{lam}\, v.e \mid k$	(Continuations)
CIde	$\ni$	x'	(CPS identifiers)
CVar	$\ni$	v	(fresh parameters of continuations)
CContVar	$\ni$	k	(fresh variables denoting continuations)

図 3: The BNF of CPS terms

3.2 LF での CPS 項の形式化

図 3 の構文は，以下のような関数 $\ulcorner\cdot\urcorner$ を帰納的に定義することで LF で表現できる．

CRoot:

$$
\begin{array}{rcl}
\ulcorner \mathsf{lam}\, k.e \urcorner & \triangleq & \mathtt{klam}\, (\lambda k : \mathtt{ccont}.\ulcorner e \urcorner) \\
\ulcorner t_0\, t_1 \urcorner & \triangleq & \mathtt{capp1} \ulcorner t_0 \urcorner \ulcorner t_1 \urcorner
\end{array}
$$

CExp:

$$
\begin{array}{rcl}
\ulcorner r\, cont \urcorner & \triangleq & \mathtt{capp2} \ulcorner r \urcorner \ulcorner cont \urcorner \\
\ulcorner k\, t \urcorner & \triangleq & \mathtt{cret} \ulcorner k \urcorner \ulcorner t \urcorner \\
\ulcorner \mathsf{lam}\, x'.r \urcorner & \triangleq & \mathtt{xlam}\, (\lambda x' : \mathtt{ctriv}.\ulcorner r \urcorner)
\end{array}
$$

CTriv:

$$
\ulcorner \mathsf{lam}\, v.e \urcorner \triangleq \mathtt{vlam}\, (\lambda v : \mathtt{ctriv}.\ulcorner e \urcorner)
$$

CIde:

$$
\ulcorner x' \urcorner \triangleq x'
$$

CVar:

$$
\ulcorner v \urcorner \triangleq v
$$

CContVar:

$$
\ulcorner k \urcorner \triangleq k
$$

ここで使用した LF のシグネチャは以下の通りであり，シグネチャ Σ に追加される．

```
croot  :  type
 cexp  :  type
ctriv  :  type
ccont  :  type
 klam  :  (ccont → cexp) → croot
capp1  :  ctriv → ctriv → croot
capp2  :  croot → croot → cexp
 cret  :  ccont → ctriv → cexp
 xlam  :  (ctriv → croot) → ctriv
 vlam  :  (ctriv → cexp) → ccont
```

3.3 LF での CPS 変換の形式化

前節では CPS 変換を関数で定義したが (図 2)，Twelf では関数の帰納的定義はないので，CPS 変換を関係として定義しなければならない．再定義したものを図 4 に示す．ここで，$cpsT(t,\, t')$ は DTriv 式 t を CPS 変換すると CTriv 式 t' が得られることを表わし，$cpsE(e,\, k,\, e')$ は DExp 式 e を継続 k のもとで CPS 変換すると CExp 式 e' が得られることを表わしている．

LF では，この場合も関数 $\ulcorner\cdot\urcorner$ を帰納的に定義す

$$\frac{cpsT(t,\ t')}{cpsE(t,\ k,\ k\,t')}\ \text{cps\_dtriv}$$

$$\frac{cpsE(e_0,\ \mathsf{lam}\,v_0.(k_1\,v_0),\ e') \quad cpsE(e_1,\ \mathsf{lam}\,v_1.v_0\,v_1\,k,\ (k_1\,v_0))}{cpsE(e_0\,e_1,\ k,\ e')}\ \text{cps\_dapp}^{v_0,k_1}$$

$$\frac{\dfrac{}{cpsT(x,\ x')}\ u \\ \vdots \\ cpsE((e\,x),\ k,\ (e'\,k\,x'))}{cpsT(\mathsf{lam}\,x.(e\,x),\ \mathsf{lam}\,x'.\mathsf{lam}\,k.(e'\,x'\,k))}\ \text{cps\_dlam}^{x,x',u,k}$$

図 4: The left-to-right, call-by-value CPS transformation formulated as a relation

ることで LF で表現できる．演繹システムの表現は，「判断=型」と「導出=対象」の原理を用いる．判断を LF の型で解釈し，導出を LF の対象と解釈することで，CPS 変換の導出を LF の型検査により得ることができる．

$$\ulcorner\ \frac{\mathcal{D}}{\dfrac{cpsT(t,\ t')}{cpsE(t,\ k,\ k\,t')}\ \text{cps\_dtriv}}\ \urcorner$$

$$\triangleq\ \texttt{cps\_dtriv}\ \ulcorner t\urcorner\ \ulcorner t'\urcorner\ \ulcorner k\urcorner\ \ulcorner\mathcal{D}\urcorner$$

$$\ulcorner\ \frac{\dfrac{\mathcal{D}_0}{cpsE(e_0,\ \mathsf{lam}\,v_0.k_1,\ e')} \quad \dfrac{\mathcal{D}_1}{cpsE(e_1,\ \mathsf{lam}\,v_1.v_0\,v_1\,k,\ k_1)}}{cpsE(e_0\,e_1,\ k,\ e')}\ \urcorner$$

$$\triangleq\ \texttt{cps\_dapp}\ \ulcorner e_0\urcorner\ \ulcorner k_1\urcorner\ \ulcorner e'\urcorner\ \ulcorner e_1\urcorner\ \ulcorner k\urcorner\ \ulcorner\mathcal{D}_0\urcorner\ (\lambda v_0 : \texttt{ctriv}.\ulcorner\mathcal{D}_1\urcorner)$$

$$\ulcorner\ \frac{\dfrac{\dfrac{}{cpsT(x,\ x')}\ u \quad \dfrac{\mathcal{D}}{cpsE(r,\ k,\ r')}}{cpsT(\mathsf{lam}\,x.e,\ \mathsf{lam}\,x'.\mathsf{lam}\,k.e')}\ \urcorner$$

$$\triangleq\ \texttt{cps\_dlam}\ \ulcorner\mathsf{lam}\,x.e\urcorner\ \ulcorner\mathsf{lam}\,x'.\mathsf{lam}\,k.e'\urcorner\ (\lambda x : \texttt{dtriv}.\lambda x' : \texttt{ctriv}.\lambda u : \texttt{cpsT}\,x\,x'.\ (\lambda k : \texttt{ccont}.\ulcorner\mathcal{D}\urcorner))$$

ここで使用した LF のシグネチャを図 5 に示す．また，このシグネチャは LF のシグネチャ Σ に追加される．

3.4 CPS 変換の健全性

CPS 変換の健全性は，以下の 5 つのメタ定理を証明すればよい．ここで，

```
-->T, -->E, -->K
```

は，CPS 項に対する $\beta\eta$ 簡約の LF での表現あり，

```
<-->T, <-->E, <-->K
```

は，CPS 項に対する $\beta\eta$ equality の LF での表現である．

メタ定理 (β)：

```
theorem_sound_beta :
  (cpsE (dapp (dtriv->dexp (dlam E))
              (dtriv->dexp V))
        K E1)
→ (cpsE (E V) K E2)
→ (E1 -->E E2)
→ type
```

```
     cpsE  :  dexp → ccont → cexp → type
     cpsT  :  dtriv → ctriv → type
cps_dtriv  :  ΠT : dtriv.ΠT′ : ctriv.ΠK : ccont.cpsT T T′
              → cpsE (dtriv->dexp T) K (cret K T′)
 cps_dapp  :  ΠE_fun : dexp.ΠK_1 : ctriv → cexp.ΠE′ : cexp.ΠE_arg : dexp.ΠK : ccont.
              cpsE E_fun (vlam K_1) E′
              → (Πv_0 : ctriv.cpsE E_arg (vlam (λv_1 : ctriv.(capp2 (capp1 v_0 v_1) K)) (K_1 v_0))
              → cpsE (dapp E_fun E_arg) K E′
 cps_dlam  :  ΠE : dtriv → dexp.ΠE′ : ctriv → ccont → cexp.
              (Πx : dtriv.Πx′ : ctriv.cpsT x x′ → (Πk : ccont.cpsE (E x) k (E′ x′ k)))
              → cpsT (dlam E) (xlam (λx′.(klam (E′ x′))))
```

図 5: The left-to-right, call-by-value CPS transformation in LF

メタ定理 (η)：

```
theorem_sound_eta :
  (cpsT (dlam λx : dtriv.
          (dapp (dtriv->dexp V)
                (dtriv->dexp x)))
        CT1′)
→ (cpsT V CT2′)
→ (CT1′ -->T CT2′)
→ type
```

メタ定理：

```
sound_moggi1 :
  (cpsE (dapp (dtriv->dexp
          (dlam λx : dtriv.
          (dtriv->dexp x))) E)
        K CE1)
→ (cpsE E K CE2)
→ CE1 -->E CE2)
→ type
```

メタ定理：

```
sound_moggi2 :
  (cpsE (dapp (dtriv->dexp
          (dlam λx : dtriv.
            (dapp (E1 x) E2))) E)
        K CE1)
→ (cpsE (dapp (dapp (dtriv->dexp
      (dlam λx : dtriv E1 x)) E) E2)
        K CE2)
→ (CE1<-->E CE2)
→ type
```

メタ定理：

```
sound_moggi3 :
 (cpsE (dapp (dtriv->dexp
         (dlam λx : dtriv.
           (dapp (dtriv->dexp V)
           (E1 x)))) E) K CE1)
→ (cpsE (dapp (dtriv->dexp V)
    (dapp (dtriv->dexp
     (dlam λx : dtriv.(E1 x)))
     E)) K CE2)
→ (CE1 <-->E CE2)
→ type
```

本研究では，以上のメタ定理を証明した．Twelf による証明では，証明は簡潔に記述できるが，証明作業が大変である．Twelf による証明作業は，対話的なユーザ補助機能がなく，証明すべきことや仮定していることをユーザが管理しなければならない．

4 Twelf で証明することの利点

ここでは，Twelf で証明することの利点について述べる．

4.1 代入に関する補題

Twelf での証明は，代入に関する補題が必要にならない，という利点がある．これは，形式化の際に定義した関数 $\ulcorner\cdot\urcorner$ が全単射であり合成的であるため，オブジェクトレベルの代入がメタレベルの代入になるからである．

FOAS では，例えば，メタ定理 (β) を証明するために以下の代入に関する補題が必要となる．

補題 (代入)：

$$[\![[V/x]E]\!]^{\mathrm{DExp}} = [V'/x][\![E]\!]^{\mathrm{DExp}}$$

ここで，$[\![V]\!]^{\mathrm{DTriv}} = V'$ である．

一方，Twelf では，このメタ定理は次のメタレベルの代入に対応している．

$$\begin{array}{c} \overline{cpsT(x,\ x')}\ u \\ \mathcal{D}' \\ cpsE(E,\ k,\ E') \end{array}$$

と

$$\begin{array}{c} \mathcal{D} \\ cpsT(V,\ V') \end{array}$$

から直ちに

$$\begin{array}{c} \mathcal{D} \\ cpsT(V,\ V') \\ \mathcal{D}' \\ cpsE(E,\ k,\ E') \end{array}$$

が得られる．推論規則は，LF では関数や型の族で表現されるため，この操作は β 簡約に対応している．

4.2 Twelf の厳密な推論

本稿では，変数を厳密に扱うために，HOAS をサポートする Twelf を採用した．Twelf における束縛変数の扱いは厳密であり，手による証明の際には気付かないような一般的なケースについての推論が必要になることがある．以下では，このことの証明への影響について述べる．

例として `thorem_sound_beta` の証明を考える．Twelf での証明を図 6 に示す．この例では，二つの演繹：

```
        D4 : {x:dtriv}{x':ctriv}
             {q:cpsT x x'}{k:ccont}
             cpsE (E x) k (E' x' k)
(D5  t0) : {t0} cpsT V (T2' t0)
```

から

```
(D51 t0) : {t0} (T2' t0) ==T T21
 D10     : (E' T21 K) ==E E2
```

を示すことで証明が完成する．基本的には，次の補題:

```
uniq_cpsE : cpsE E K1 E1'
         -> cpsE E K2 E2'
         -> K1 ==K K2
         -> E1' ==E E2'
         -> type.
```

を用いて証明を行う．これは，CPS 変換の結果の一意性を主張している．具体的には D4 の x，x'，k に V，T21，K を代入した

```
(D4 V T21 q K) : cpsE (E V) K (E' T21 K)
```

と

```
D2 : cpsE (E V) K E2
```

について補題:uniq_cpsE を使う．また，D4 の q には

```
cpsT V T21
```

の演繹 (証明) を代入しなければならない．ところで，この証明は

```
(D5 t0)  : cpsT V (T2' t0)
(D51 t0) : (T2' t0) ==T T21
```

に対して，補題：

```
lemma_cpst_eq :
       (cpsT T T')
    -> (T' ==T T2')
    -> (cpsT T T2')
    -> type.
```

を使うことにより証明が得られそうである．ところが，Twelf は，(D51 t0) の型の中の T21 が (T21 t0) のように変数 t0 を含んでいる，と厳密に推論するため，証明が得られない．この変数の問題の原因は，Twelf が図 6 の (D5 t0)，(D51 t0) やその型の中の (T21 t0) のように，証明の中で導入された束縛変数はそのスコープ内で必ず出現することを仮定し

ているからである．そこで，このような変数を消去する方法が必要となる．

以上のような変数の問題は次の補題により解決することができる．

```
closed_lemmaT :
   ({y':ctriv} cpsT T (T' y'))
-> {T1:ctriv}
   (({y':ctriv} (T' y') ==T T1)
     -> type).
```

この補題は，T を CPS 変換した結果が (T' y') のように変数 y' を含んでいる場合，変数を含まない T1 が存在し，(T' y') ==T T1 が成り立つ，という意味である．この補題を

```
(D5 t0) : cpsT V (T2' t0)
```

と T21 に対して使うと，

```
(D51 t0) : (T2' t0) ==T T21
```

のように，(T21 t0) から t0 を消した式が得られる．さらに，これと補題 lemma_cpst_eq から

```
D52 : cpsT V T21
```

が得られる．最後に，これを D4 の q に代入すると

```
(D4 V T21 D52 K)
 : cpsE (E V) K (E' T21 K)
```

が得られ， uniq_cpsE により証明が完成する．

5 まとめと今後の課題

本稿では， Twelf を使って，CPS 項に対する $\beta\eta$-equality と DS 式に対する Moggi の computational lambda calculus による equality の健全性・完全性のうち，健全性を形式的に表現し，証明した．今後の課題として，完全性の形式化と証明を考えている．さらに，変換の対象となる言語にコントロールオペレータ call/cc や shift/reset (文献 [5]) を入れた CPS 変換の形式化を考えている．また，Twelf では，シグネチャを論理プログラムとして記述するため，CPS 変換を関係として定義しなければならないが，これを関数として定義できるようにしたいと考えている．これにより，従来の CPS 変換の定義がそのまま利用でき，また，証明の際に変数を消去する必要がなくなると考えられる．

参考文献

[1] Aydemir, B. E., A. Bohannon, M. Fairbairn, J. N. Foster, B. C. Pierce, P. Sewell, D. VVytiniotis, G. Washburn, S. Weirich, and S. Zdancewic. “Mechanized Metatheory for the Masses: The PoplMark Challenge.” 2005.

[2] Danvy, O. and F. Pfenning. “The occurrence of continuation parameters in CPS terms.” Technical Report CMU-CS-95-121, Department of Computer Science, Carnegie Mellon University. Feb. 1995.

[3] Harper, R., F. Honsell, and G. Plotkin. ”A framework for defining logics.” Journal of the Association for Computing Machinery, 40(1):142-184. Jan. 1993.

[4] Harper, R. and K. Crary. “How to Believe a Twelf Proof.” 2005.

[5] Kameyama, Y. “Axioms for Delimited Continuations in the CPS Hierarchy.” Proc. Annual Conference of the European Association for Computer Science Logic (CSL'04), Karpacz, Poland. Sep. 2004.

[6] Okuma, K. and Y. Minamide. “Executing Verified Compiler Specification.” In Proc. of the First Asian Symposium on Programming Languages and Systems, LNCS 2895. 2003.

[7] Pfenning, F. and C. Schürmann. “Twelf User's Guide: Version 1.4.” 2002.

[8] Pfenning, F. “Computation and Deduction.” Cambridge University Press, 2003. In preparation.

[9] Sabry, A. and M. Felleisen. “Reasoning About Programs in Continuation-Passing Style.” LISP and Functional Programming: 288-298. 1992.

[10] Schürmann, C. “Automating the Meta-Theory of Deductive Systems.” PhD thesis, Carnegie Mellon University. 2000.

```
pf_sound_beta :
  theorem_sound_beta
    (cps_dapp
      (cps_dtriv
        (cps_dlam
          (D4 : {x:dtriv}{x':ctriv}{q:cpsT x x'}{k:ccont}
                cpsE (E x) k (E' x' k))
        ))
      ([t0:ctriv] (cps_dtriv ((D5 t0) : (cpsT V (T2' t0))))))
    (D2 : (cpsE (E V) K E2))
    (-->E_trans
      (-->E_cret (-->T_refl)
                 (-->K_vlam [t0:ctriv]
                   (-->E_cret (-->T_refl2
                                (D51 t0)
                              )
                              (-->K_refl))))
      (-->E_trans
        (-->E_trans (-->E_beta1)
                    (-->E_trans (-->E_beta1)
                                (-->E_trans (-->E_capp2 -->K_refl -->R_beta)
                                            (-->E_beta2))))
        (-->E_refl2 D10)))
  <- (closed_lemmaT D5 T21 D51)
  <- ({t0:ctriv} lemma_cpst_eq (D5 t0) (D51 t0) (D52))
  <- (uniq_cpsE (D4 V T21 (D52) K) D2 (==K_refl) D10).
```

図 6: The Proof of thorem_sound_beta