# 自己反映的証明体系 $\mathcal{RPT}$ の理論と実現

亀山幸義　佐藤雅彦

体系 Reflective Proof Theory($\mathcal{RPT}$) [11] は，構成的プログラミングを行うための基礎となる体系のひとつである．本論文では，まず，意味論的体系として与えられた $\mathcal{RPT}$ の形式化 RPT を与え，その妥当性について述べる．ここで与えた形式化に基づき，計算機上に実現した RPT の証明システムの概要について述べる．このシステムを使って，RPT の項の Church-Rosser の性質を証明した例および構成的プログラミングの例について述べる．

## 1 はじめに

構成的プログラミングは，Curry-Howard の同型対応に基づき，構成的論理体系における証明とプログラミング言語におけるプログラムを同一視することにより，プログラム作成を証明作成で置き換える新しいプログラミング・スタイルである [2] [8] [10]．

構成的プログラミングにおいては，以下の手順でプログラムを作成する．

1. プログラムの仕様を $\forall x.A(x) \supset \exists y.B(x,y)$ の形の論理式で記述する．ここで，$x$ は入力，$A(x)$ は入力値に関する条件，$y$ は出力，$B(x,y)$ は入出力関係に関する条件である．
2. 構成的な論理体系でその論理式を証明する．
3. 実現可能性解釈などを用いて，証明からプログラムを自動的に抽出する．

この手順で作成されたプログラムは，仕様を満たすことが保証されている．

構成的プログラミングを行うシステムには，以下の 2 要素が重要である．

- 基礎となる構成的な論理体系
- この論理体系における証明作成，プログラム抽出を行うシステム

論理体系は，プログラム抽出を行うことのできる体系でなければならない．また，強力なプログラム言語を包含すること，その性質に関する豊かな表現力を持つことが必要である．

一方，上記の手順で作成されたプログラムは，正しいこと (仕様を満たすこと) が保証されている．しかし，証明が誤っていてはこのことは保証されない．従って，計算機による証明チェッカが必要である．また，大規模な証明作成を完全に人手で行うのは困難であるので，その観点からも計算機による証明作成支援システムが必要である．

本論文では，これらの要求を満たすべく，$\mathcal{RPT}$ [11] に基づく構成的プログラミングとそれを実現したシステムについて述べる．

$\mathcal{RPT}$ は (1) リフレクション (自己反映)[†1] の機構に

Reflective Proof Theory and its Proof System.

Yukiyoshi Kameyama, 東北大学電気通信研究所, Research Institute of Electrical Communication, Tohoku University.

Masahiko Sato, 東北大学電気通信研究所, Research Institute of Electrical Communication, Tohoku University.

コンピュータソフトウェア, Vol.x, No.x (199x), pp.xx–xx.

[論文] 1993 年 6 月 10 日受付.

†1 ここでいうリフレクションは論理学の言葉であり，オブジェクト指向言語などにおけるリフレクティブな計算機構とは直接の関係はない．

より，メタな言明を体系内で行うことができる．(2) 証明を表す項を体系内に持ち，証明を直接取り扱うことができる．(3) 広い範囲の帰納的定義が可能である，という特徴を持つ体系である．これらの特徴により，$\mathcal{RPT}$ は表現力が豊かになり，仕様記述が容易になる．

$\mathcal{RPT}$ はこのような特徴を持つ体系であるが，[11] においては意味論的体系 (semantical theory) として与えられている．これは，Martin-Löf の体系 [9]と同様に，数学の基礎付けを行うことを目標としているため，形式的体系の表現を取らず，それ自身の意味が明確になるよう自然言語で記述する，という方法を取ったためである．しかし，実際に構成的プログラミングを行うために，計算機上に証明システムを実現するためには，形式的体系を定義する必要がある．

本稿では，まず，2節で $\mathcal{RPT}$ の概要を説明する．3節で，$\mathcal{RPT}$ の形式化 RPT を提案し，その性質について述べる．次に，4節で，RPT での証明例について述べる．5節で，この形式化に基づいて計算機上に実現した証明システムについて述べる．RPT および，この証明システムの有効性を示すため，項の簡約の合流性を証明した実例について 6節で述べる．7節で，このシステムを用いた構成的プログラミングの方法について述べ，8節で結論を述べる．

## 2 意味論的体系 $\mathcal{RPT}$

本節では，[11]の Reflective Proof Theory ($\mathcal{RPT}$) の概要を説明する．

$\mathcal{RPT}$ は Aczel の Frege 構造 [1] を基に，構成的プログラミングを行うための拡張を施したものである．

Frege 構造は，ラムダ計算の任意のモデルから構成できる．モデルの領域を $M$ とし，$M$ には，$\wedge, \vee, \supset, \forall, \exists$ という論理記号に対応する項 $\dot{\wedge}, \dot{\vee}, \dot{\supset}, \dot{\forall}, \dot{\exists}$ が含まれているとする．$M$ の 2 つの部分集合 $P, Q$ (ただし $P \supseteq Q$) を，それぞれ，命題を表す項の集合，正しい命題を表す項の集合に対応するよう定義する．たとえば，$\dot{\vee}ab$ という項に対しては，

$$\dot{\vee}ab \in P \iff a \in P \text{ かつ } b \in P$$

$$\dot{\vee}ab \in Q \iff (a \in Q \text{ かつ } b \in P) \quad \text{または} \ (a \in P \text{ かつ } b \in Q)$$

という性質を満たすとする．

このような性質を持つ構造を Frege 構造と呼ぶ．Frege 構造では，命題関数 (1 引数の関数で値が命題となるもの) を集合と見なすことにより，Frege 流の集合論を展開することができる．Aczel はこれを，Russell の逆理の解析などに用いた[1]．

$\mathcal{RPT}$ は Frege 構造に以下の拡張を施したものである．

### 基礎となるラムダ計算の領域の拡張

$\mathcal{RPT}$ の項は純粋なラムダ計算に，対 (Lisp のコンス・ペア) や `if` $a$ `then` $b$ `else` $c$ `fi` などを加えたものである．純粋なラムダ計算でも適当なコーディングにより $\mathcal{RPT}$ の項を表現することができるので，この拡張は計算能力の点からは非本質的である．しかし，$\mathcal{RPT}$ の項をプログラムと見なす立場から，この拡張は非常に有用である．

### 証明を表す項を明示的に持つ

Frege 構造では，「$p$ が命題である」という表現のほかに，「$p$ が正しい命題である」という表現がある．$\mathcal{RPT}$ では，証明自身を体系内で取り扱うため，「$p$ が命題である」という表現のほかに，「$a$ が $p$ の証明である」ことを意味する表現を取りいれた．「$p$ が正しい命題である」ことを「$p$ に対する証明が存在する」ことと定義すれば，$\mathcal{RPT}$ は Frege 構造の拡張になっていることがわかる．

なお，$\mathcal{RPT}$ では，「$p$ が命題である」ことを $\models_i p$ と表し，「$a$ が $p$ の証明である」ことを $a \vdash_i p$ と表す．ここで添字の $i$ は，次項で述べる自己反映の階層のレベルを表す順序数である．通常，記号 $\models$ と $\vdash$ はそれぞれ，モデルで真となること，形式的体系で証明可能なことを表すが，$\mathcal{RPT}$ では異なる意味に使うことに注意されたい．$a \vdash_i p$ という表現は型理論における $a : p$ という表現，および，実現可能性解釈における「$a$ が論理式 $p$ の実現子である」という表現に対応するものである．

### 自己反映 (reflection) の階層

「$p$ が命題であれば $p \supset p$ は正しい」という表現は通常の論理体系の枠をはずれる．この種のメタな命題

(スキーマ) は，強力な内容を簡潔に表現できるため，非常に有用である．しかし，メタな命題を単に命題とすると，Russell の逆理など矛盾を生じる．$\mathcal{RPT}$ では，命題にレベルの概念を導入し，第 $i$ レベルの命題に関するメタ命題は第 $i+1$ レベルとすることにより，矛盾を回避しつつ，この種のメタ定理を体系内で扱うことを可能にしている．

自己反映階層は，構成的プログラミングにとっても有用な概念である．1 階述語論理に基づく体系 (たとえば，Beeson の EON [2]など) では，項の存在に関する性質 (Term Existence Property) と呼ばれる以下の性質が成立することが，証明からのプログラム抽出のために必要である．

**命題 1**論理式 $\exists x.A(x)$ が証明されれば，その証明から実効的に項 $t$ と論理式 $A(t)$ の証明とが得られる．□

この性質を証明するため，実現可能性解釈や証明図の正規化などの技法が用いられるが，いずれにせよ，証明論的な (メタな) レベルでの推論が必要である．一方，$\mathcal{RPT}$ では，この性質を体系内で表現した性質が成立する．

**命題 2**$\mathcal{RPT}$ において，ある項 $a$ に対して，

$$a \vdash_{i+1} \forall y.((y \vdash_i \exists x.p) \supset (\mathrm{cdr}(y) \vdash_i p_x[\mathrm{car}(y)]))$$

が成立する．□

これは，命題 $\exists x.p$ の証明 $y$ から項 $\mathrm{car}(y)$ と命題 $p_x[\mathrm{car}(y)]$ の証明 $\mathrm{cdr}(y)$ とが得られることを意味している．ここで用いた記号の詳細は次節で説明する．

帰納的定義

再帰データ型の定義やそれに関する命題の記述において，帰納的定義とそれに対応する帰納法が必要不可欠である．$\mathcal{RPT}$ では，述語の帰納的定義のための一般的枠組みが与えられている．

$\mathcal{RPT}$ は Frege 構造と同様に，意味論的体系 (semantical theory) である．すなわち，$\mathcal{RPT}$ 自身はいわゆる構造やモデルに相当するものであり，形式的推論をするための論理体系ではない．本論文では，$\mathcal{RPT}$ の証明システムを計算機上に実現して，構成的プログラミングを行うことが目的であるため，[11] で意味論的に与えられた $\mathcal{RPT}$ の形式化を行うことが必要である．$\mathcal{RPT}$ の形式化については次節で述べる．

## 3 形式的体系 RPT

本節では，$\mathcal{RPT}$ の形式化 (のひとつ) である RPT について述べる．以下では，混同を避けるため，意味論的体系 を $\mathcal{RPT}$ と表記し，形式的体系を RPT と表記する．

### 3.1 形式化の対象

$\mathcal{RPT}$ [11] では，$a \vdash_i p \wedge q$ が成立するとき，$a$ が対である (ある項 $b$ に対して $b \vdash_0 \mathtt{pair?}(a) = \mathtt{true}$ が成立する) かどうか定まっていない．しかし，$\mathcal{RPT}$ の本来の意図はこれが成立することであったので，以下では，$p \wedge q, p \ \&\ q, p \vee q, \exists p$ の証明項は対であること，$p \supset q, \forall p$ の証明項は関数であることが成立する構造のみを，$\mathcal{RPT}$ と呼ぶことにする．なお，命題 $p$ の証明項とは，$a \vdash_i p$ が成立する項 $a$ のことである．

$\mathcal{RPT}$ の形式化における我々の目標は，$\mathcal{RPT}$ の主要な部分をほぼカバーするような形式的体系を作ることである．

$\mathcal{RPT}$ と比較して，以下で述べる RPT における主要な制限，変更点は以下の通りである．

1. 自己反映階層におけるレベルに関する制限
   $\mathcal{RPT}$ において，レベルは，項として表現可能ないくらでも大きい順序数となり得る．また，レベルの上を動く変数を $\forall$ や $\exists$ で限量することができる．たとえば，$t \triangleq \lambda ixayz.z$ とおくと，
   $$t \quad \vdash_\omega \quad \forall i < \omega.\forall a.((\models_i a) \supset a \supset a)$$
   は正しい 判定である．この表現は，「$\omega$ より下の任意のレベルにおいて，$a$ が命題であれば $a \supset a$ は正しい」ということを形式化したものである．
   しかし，プログラミングの観点からは，そのように高いレベルやレベルに関する限量子は必要ないと判断し，RPT においては，レベルは自然数の定数のみとした．以下では，メタ変数 $i, j$ は，自然数の定数 (を RPT の適当な項でコーディングしたもの) とする．
2. 帰納的定義に関する制限
   $\mathcal{RPT}$ では，狭正 (strictly positive) であるという条件のもとで述語を帰納的に定義することがで

きる．狭正とは，帰納的定義の本体に含まれる $A \supset B$ の形の全ての部分論理式に対して，$A$ の中には，定義しようとしている述語変数があらわれないことである．[11]に述べられているように，この条件は，意味的に決定される．たとえば，$X$ を 1 引数の述語変数，$P$ を $(X[x] \supset X[x]) \wedge X[x]$ とすると，$X[x]$ が $\supset$ の前半に出現するので，構文的には 狭正ではないが，$P$ は $X[x]$ という狭正な論理式と同値なので，意味的には，狭正になる．従って，$\mathcal{RPT}$ においては，$P$ を利用して帰納的定義をすることができる．

しかし，実用上，構文的な判断のみで，自然数，リスト，木など十分広い範囲の帰納的述語を定義することができる．したがって，RPT においては，構文的に 狭正な場合にのみ帰納的定義を許している．これにより，狭正な命題に対する表現である $\models^+$ や $\vdash^+$ は使用しないこととなった．

3. $a\downarrow$ という命題の導入

RPT の推論では，「項 $a$ が値を持つ」という命題をしばしば利用する．RPT での計算は，遅延評価で行われるので，$a$ が値を持つことは，

$$a = \mathtt{nil}\ \vee\ a = \mathtt{true}\ \vee\ a = \mathtt{false}$$
$$\vee\, \mathtt{pair?}(a) = \mathtt{true} \vee \mathtt{fun?}(a) = \mathtt{true}$$

と表現される．この命題は 4 つの論理和記号を含むため，その証明項はかなり長いものである．しかし，この命題が証明を持つとき，証明項 (のひとつ) は $a$ から計算によって求めることができるので，証明項に有益な情報はない．

そこで RPT では，上述の命題と論理的には等価であるが，証明項が trivial な定数である命題として，[2]の記号にならい $a\downarrow$ を導入することとした．

### 3.2 項とその簡約化

**定義 1** (項)

$$
\begin{array}{rcl}
t & ::= & x \\
 & | & \mathtt{nil} \mid \mathtt{null?}(t) \\
 & | & \mathtt{true} \mid \mathtt{true?}(t) \\
 & | & \mathtt{false} \mid \mathtt{false?}(t) \\
 & | & [t\ .\ t] \mid \mathtt{car}(t) \mid \mathtt{cdr}(t) \mid \mathtt{pair?}(t) \\
 & | & \lambda x.t \mid t(t) \mid \mathtt{fun?}(t) \\
 & | & \mu t \\
 & | & \mathtt{if}\ \ t\ \ \mathtt{then}\ \ t\ \ \mathtt{else}\ \ t\ \ \mathtt{fi}
\end{array}
$$

ここで，$x$ は変数を表す．□

$\mathtt{nil}, \mathtt{car}(t), \mathtt{cdr}(t)$ は Lisp におけるそれらと同じ意味である．ただし，対は $[t\ .\ t]$ で表し，項の抽象，適用はラムダ計算と同様 $\lambda x.t$, $t(t)$ で表す．また，$\mu t$ という項は，再帰呼び出しのために使われる．

上記定義の中で，同じ行にある項を同種の項，異なる行にある項を異種の項と呼ぶ．たとえば，nil と $\mathtt{null?}(t)$ は同種であり，nil と $\mathtt{true?}(t)$ は異種の項である．項のうち，$\mathtt{null?}(t)$, $\mathtt{true?}(t)$, $\mathtt{false?}(t)$, $\mathtt{pair?}(t)$, $\mathtt{fun?}(t)$ の形の項を判別項と呼び，項の種類を判別するために用いられる．

束縛変数，自由変数，代入の概念は通常通り定義される．自由変数を持たない項を閉じた項と呼ぶ．項 $a$ 中の変数 $x_1, \cdots, x_n$ に項 $b_1, \cdots, b_n$ を同時に代入した結果を $a_{x_1,\cdots,x_n}[b_1, \cdots, b_n]$ と表す．項 $a$ の中の自由変数の集合を $FV(a)$ と表す．

また，

$$
\begin{array}{rcl}
ab & \triangleq & a(b) \\
a(b_1, \ldots, b_n) & \triangleq & a(b_1)\cdots(b_n) \\
[\,] & \triangleq & \mathtt{nil} \\
[a] & \triangleq & [a\ .\ \mathtt{nil}] \\
[a\ b] & \triangleq & [a\ .\ [b\ .\ \mathtt{nil}]] \\
[a\ b\ c] & \triangleq & [a\ .\ [b\ .\ [c\ .\ \mathtt{nil}]]]
\end{array}
$$

などの省略を用いる．

**定義 2** (正準項 (canonical term))

$$
\begin{array}{rcl}
c & ::= & \mathtt{nil} \mid \mathtt{true} \mid \mathtt{false} \\
 & | & [t\ .\ t] \mid \lambda x.t
\end{array}
$$

**定義 3** (正規項 (normal term))

$$
\begin{array}{rcl}
n & ::= & x \\
 & | & \mathtt{nil} \mid \mathtt{true} \mid \mathtt{false} \\
 & | & [n\ .\ n] \mid \lambda x.n
\end{array}
$$

項 $a$ を簡約化して正準項または正規項 $b$ が得られるとき，$b$ を項 $a$ の正準形または正規形と呼ぶ．□

RPT の簡約化規則では，たとえば，$\mathtt{pair?}(a)$ を計算する場合，$a$ が正準形になりさえすれば，全体の計算が可能になるので，$a$ を正規形になるまで計算する必要はない．従って，正準項は RPT の計算において重要な

表 1　項の簡約化規則

| 簡約化 | 条件 |
|---|---|
| $\#(a) \rhd$ true | $\#(a)$ は判別項，$a$ は 正準項で $\#(a)$ と $a$ は同種の項のとき |
| $\#(a) \rhd$ false | $\#(a)$ は判別項，$a$ は 正準項で $\#(a)$ と $a$ は異種の項のとき |
| car([$a$ . $b$]) $\rhd a$ | |
| cdr([$a$ . $b$]) $\rhd b$ | |
| $(\lambda x.a)b \rhd a_x[b]$ | |
| $\lambda x.(\lambda y.a)x \rhd \lambda y.a$ | $x \notin FV(a)$ のとき |
| $\mu a \rhd a(\mu a)$ | |
| if true then $a$ else $b$ fi $\rhd a$ | |
| if false then $a$ else $b$ fi $\rhd b$ | |
| $a_x[b] \rhd a_x[c]$ | $b \rhd c$ のとき |

概念である．

定義 4 (式項)

$$p \quad ::= \quad \texttt{nil} \mid [p\ .\ p]$$

また、自然数 $i$ を

$$0 \triangleq \texttt{nil}$$
$$i+1 \triangleq [\texttt{nil}\ .\ i]$$

と定義する．

後に述べる RPT の判定と命題は，以下のように適当な項でコーディングされている．

$$
\begin{aligned}
a = b &\triangleq [1\ a\ b] \\
a[b] &\triangleq [2\ a\ b] \\
\models_i b &\triangleq [3\ i\ b] \\
a \vdash_i b &\triangleq [5\ i\ a\ b] \\
a \wedge b &\triangleq [7\ a\ b] \\
a\ \&\ b &\triangleq [8\ a\ b] \\
a \vee b &\triangleq [9\ a\ b] \\
a \supset b &\triangleq [10\ a\ b] \\
\forall a &\triangleq [11\ a] \\
\exists a &\triangleq [12\ a]
\end{aligned}
$$

また，$\forall x.b, \exists x.b, \neg a$ は，それぞれ，$\forall(\lambda x.b)$, $\exists(\lambda x.b)$, $a \supset$ false の省略形であるとする．$a[b]$ は 1 引数の述語 $a$ に引数 $b$ を与えた命題をあらわす．

定義 5 (項の簡約化)項 $a, b$ に対し，$a \rhd b$ を表 *1*で定義する．□

$\rhd$ はいわゆる 1 ステップの簡約化に相当する．$\rhd$ の反射的推移的閉包を記号 $\rhd^*$ で表し，$\rhd^*$ を含む最小の同値関係を記号 $\approx$ で表す．簡約化が停止しない項の代表として，$\Omega \triangleq (\lambda x.xx)(\lambda x.xx)$ とおく．

以上の構文と簡約化規則で定義されるプログラム言語を $\Lambda$ と呼ぶ．$\Lambda$ の Church-Rosser の性質 (合流性) は[11]で証明されている．この証明の形式化については 6節で述べる．$\Lambda$ の計算は非決定的であるが，必要呼び (call by need) による戦略は正規化戦略になっており，これを計算機上に実現した．$\Lambda$ は，本論文で記述の対象となると同時に，計算機上にシステムを実現するためのプログラム言語でもある．

### 3.3　判定と推論規則

RPT では，Martin-Löf の体系 (の形式化) と同様，判定 (judgement) が推論の対象となる．

定義 6 (判定)$a, p$ を項，$i$ を自然数とするとき，以下のものは，判定である．

$$\models_i p$$
$$a \vdash_i p$$

$a \vdash_i p$ の形の判定に対して，項 $a$ を証明項と呼ぶ．以下，$a, b, c, p, q, s, t, \cdots$ を項，$i, j$ を自然数，$J, J_i$ を 判定とし，$f, g, x, y, z$ を変数をあらわすメタ変数として使う．

推論規則は，自然演繹法の形式で記述される．RPT の推論規則は以下の種類からなる．

1. レベルに関する規則
2. 等号と正準項に関する規則
3. 命題に関する規則
4. 証明項に関する規則
5. 述語定義に関する規則

ここでは，1, 5の全てと 2, 3, 4の主要なもののみ説明し，残りは付録で述べる．

**3.3.1　レベルに関する規則**

$$\frac{\models_j p}{\models_i p}\ (i > j) \qquad \frac{a \vdash_j p}{a \vdash_i p}\ (i > j)$$

$$\frac{a \vdash_i p \quad \models_j p}{a \vdash_j p}\ (i > j)$$

ここで $i, j$ は自然数の定数をあらわすメタ変数である．従って，これらは，それぞれ，単一の規則ではなく，規則のスキーマをあらわしている．以下，本論文を通じて $i, j$ はこの約束事に従う．

上記のうち，左の 2 つの規則は，あるレベルで成立する事実は，それより高いレベルでも成立することをあらわしている．一番右の規則は，逆に，あるレベルで成立する $a \vdash_i p$ の形の判定は，$p$ が命題となるレベルで既に成立していたことをあらわしている．言い換えると，$p$ が証明を持つかどうかは，それが命題となる一番低いレベルの時点で決まり，それより上のレベルはその状況を引き継いでいるだけである．

**3.3.2　等号と正準項に関する規則**

主な規則は以下の通りである．

$$\frac{}{0 \vdash_i a = b}\ (\text{ただし } a \approx b)$$

$$\frac{}{\lambda x.0 \vdash_i \neg(a = b)}\ (\text{ただし } a \text{ と } b \text{ は異種の正準項})$$

$$\frac{a \vdash_i b_x[d] \quad c \vdash_i d = e}{a \vdash_i b_x[e]} \qquad \frac{a_x[d] \vdash_i b \quad c \vdash_i d = e}{a_x[e] \vdash_i b}$$

最後の規則を左等号規則と呼ぶ．上記以外の規則は付録に記載する．

**3.3.3　命題に関する規則**

命題となる項は，通常の 1 階述語論理とほぼ同じ形をしている．原始命題として true, false, $a = b$, $\models_i p$, $a \vdash_i p$ を持ち，論理記号として $\wedge, \&, \vee, \supset, \forall, \exists$ を持つ．異なる点は (1) $\models_i p$ や $a \vdash_i p$ という表現自身も $i+1$ レベルの命題となる．ここで $a$ や $p$ はどんな項でもよい．(2) $p$ が証明を持たない命題であれば，$q$ が命題でなくても $p \supset q$ や $p \,\&\, q$ が命題となる, の 2 点である．

なお，$p \,\&\, q$ は，通常の論理積 $p \wedge q$ と似た意味を持つが，命題となるための条件が異なる．すなわち，$p \wedge q$ が命題になるためには $p, q$ ともに命題でなければならないが，$p \,\&\, q$ が命題になるのは，$p$ が命題であり，かつ，$p$ が正しい命題であるという仮定のもとで $q$ が命題になるときである．& の具体例は第 4.2節で述べる．

命題に関する規則は付録に記載する．

**3.3.4　証明項に関する規則**

証明項に関する規則は以下の通りである．ここでは $i > j$ と仮定する．

$$\frac{}{0 \vdash_i \texttt{true}} \qquad \frac{a \vdash_i \texttt{true}}{0 \vdash_i a = 0} \qquad \frac{a \vdash_i \texttt{false}}{a \vdash_i p}$$

$$\frac{\models_j p}{0 \vdash_i (\models_j p)} \qquad \frac{a \vdash_j p}{0 \vdash_i (a \vdash_j p)}$$

$$\frac{a \vdash_i (b \vdash_j p)}{b \vdash_j p} \qquad \frac{a \vdash_i (b \vdash_j p)}{0 \vdash_j a = 0}$$

$$\frac{a \vdash_i p \quad b \vdash_i q}{[a\ .\ b] \vdash_i p \wedge q} \qquad \frac{a \vdash_i p \wedge q}{0 \vdash_i \texttt{pair?}(a) = \texttt{true}}$$

$$\frac{a \vdash_i p \wedge q}{\texttt{car}(a) \vdash_i p} \qquad \frac{a \vdash_i p \wedge q}{\texttt{cdr}(a) \vdash_i q}$$

$$\frac{a \vdash_i p \quad \models_i q}{[\texttt{true}\ .\ a] \vdash_i p \vee q} \qquad \frac{\models_i p \quad b \vdash_i q}{[\texttt{false}\ .\ b] \vdash_i p \vee q}$$

$$\frac{a \vdash_i p \vee q}{0 \vdash_i \texttt{pair?}(a) = \texttt{true}}$$

$$\frac{a \vdash_i p \vee q \qquad \begin{array}{c}[P \quad x \vdash_i p]\\ \vdots \\ b \vdash_i r\end{array} \qquad \begin{array}{c}[Q \quad x \vdash_i q]\\ \vdots \\ c \vdash_i r\end{array}}{D \vdash_i r}\ (x \text{ は固有変数})$$

ただし，最後の規則において，

$P \overset{\triangle}{=} 0 \vdash_i \texttt{car}(a) = \texttt{true}$

$Q \overset{\triangle}{=} 0 \vdash_i \texttt{car}(a) = \texttt{false}$

$D \triangleq (\mathtt{if}\ \mathtt{car}(a)\ \mathtt{then}\ \lambda x.b\ \mathtt{else}\ \lambda x.c\ \mathtt{fi})(\mathtt{cdr}(a))$
とする．この規則を ∨ 除去規則と呼ぶ．

$$\dfrac{\begin{array}{c}[x \vdash_i p] \\ \vdots \\ \models_i p \quad a \vdash_i q\end{array}}{\lambda x.a \vdash_i p \supset q}\ (x \text{ は固有変数})$$

$$\dfrac{a \vdash_i p \supset q}{0 \vdash_i \mathtt{fun?}(a) = \mathtt{true}} \qquad \dfrac{a \vdash_i p \supset q \quad b \vdash_i p}{a(b) \vdash_i q}$$

$$\dfrac{a \vdash_i p \quad b \vdash_i q}{[a\ .\ b] \vdash_i p\ \&\ q} \qquad \dfrac{a \vdash_i p\ \&\ q}{0 \vdash_i \mathtt{pair?}(a) = \mathtt{true}}$$

$$\dfrac{a \vdash_i p\ \&\ q}{\mathtt{car}(a) \vdash_i p} \qquad \dfrac{a \vdash_i p\ \&\ q}{\mathtt{cdr}(a) \vdash_i q}$$

$$\dfrac{a \vdash_i p}{\lambda x.a \vdash_i \forall(\lambda x.p)}\ (x \text{ は固有変数})$$

$$\dfrac{a \vdash_i \forall p}{0 \vdash_i \mathtt{fun?}(a) = \mathtt{true}}$$

$$\dfrac{a \vdash_i p_x[b]}{[b\ .\ a] \vdash_i \exists(\lambda x.p)} \qquad \dfrac{a \vdash_i \exists p}{0 \vdash_i \mathtt{pair?}(a) = \mathtt{true}}$$

$$\dfrac{a \vdash_i \forall p}{a(b) \vdash_i p(b)} \qquad \dfrac{a \vdash_i \exists p}{\mathtt{cdr}(a) \vdash_i p(\mathtt{car}(a))}$$

固有変数は通常の論理と同様の固有変数条件を満たさなければならない．たとえば，∨ の除去規則では，$x \notin FV(p) \cup FV(q) \cup FV(r)$ であり，さらに，$x$ は右 2 つの部分証明の仮定中には，この規則で表示されているもの以外にはあらわれてはいけない．

これらの規則は，どういう項が命題の証明になるか，について推論する規則である．RPT では，証明が存在する命題を正しい命題と呼ぶので，どの命題が正しいかについての推論をする規則であるとも言える．証明となる項は，Curry-Howard の同型対応に従って自然に定まるものである．すなわち，論理積の形の命題は対の形の証明項をもち，その第 1, 2 要素はそれぞれ，論理積の第 1, 2 要素の証明となっている．他も同様である．

**3.3.5 述語定義に関する規則**

述語は全て帰納的述語の形で導入される．以下の推論規則で $P$ は $\lambda f.\lambda x.F$ のことである．また，$F$ は，$f$ に関して 狭正でなければならない，すなわち，$F$ 中の $p \supset q$ の形の全ての部分項に対し，$p$ が $f$ の自由な出現を含んではいけない．

$$\dfrac{\begin{array}{c}[w \vdash_{i+1} (\forall y.\ \models_i f[y])] \\ \vdots \\ \models_i F\end{array}}{\models_i P[a]}\ (x, f, w \text{ は固有変数})$$

この規則は，帰納的述語 $P$ が命題となる条件をあらわしている．

$$\dfrac{b \vdash_i F_{x,f}[a, P]}{b \vdash_i P[a]} \qquad \dfrac{b \vdash_i P[a]}{b \vdash_i F_{x,f}[a, P]}$$

これらの規則は，それぞれ，帰納的述語定義の fold, unfold に相当する．

$$\dfrac{b \vdash_i \forall x.(F_f[p] \supset p[x])}{ind_P(b) \vdash_i \forall x.(P[x] \supset p[x])}$$

この規則が帰納法に相当する．ここで $ind_P$ は $P$ に応じて定められる項である．詳細は [11] を参照のこと．

ここでは簡単のため，1 引数の述語定義についてのみ記述したが，2 引数以上の述語定義に対する推論規則も同様である．

述語定義の例

「$p$ が 1 引数の命題関数であること」は RPT では $\forall x.\models_i p(x)$ という命題で表現される．これを $\mathtt{PF}[p]$ という (メタな) 命題で表現するためには，述語PFを以下のように定義すればよい．

$$\mathtt{PF} \triangleq \lambda f.\lambda p.\ \forall x.\models_i p(x)$$

述語PFは，定義の中で $f$ を参照していないので帰納的でない．この場合，帰納法の推論規則は自明なものになる．

自然数を表す述語 Nat は以下のように定義される．

$$\mathtt{Nat} \triangleq \lambda f.\lambda x.\ x = 0\ \vee\ \exists y.\ (x = [0\ .\ y] \wedge f[y])$$

このように定義すると，先に定義した自然数 $n$ に対して，$\mathtt{Nat}[n]$ が成立することがわかる．帰納法の推論規則にNatの定義をあてはめると，通常の数学的帰納法と同値であることがわかる．ただし，$ind_{\mathtt{Nat}}$ は以下のように定義される．

$$\begin{array}{l} ind_{\mathtt{Nat}} \triangleq \lambda r.\mu(\lambda fxq.\ r(x)([\mathtt{car}(q). \\ \qquad \mathtt{if}\ \mathtt{car}(q)\ \mathtt{then}\ \mathtt{cdr}(q) \\ \qquad \mathtt{else}\ [\mathtt{cadr}(q).[\mathtt{caddr}(q)\ . \\ \qquad\qquad f(\mathtt{cadr}(q))(\mathtt{cdddr}(q))]]\ \mathtt{fi})) \end{array}$$

ここで Lisp と同様に cadr 等の略記を用いた．

### 3.4 $\mathcal{RPT}$ と RPT

本節で述べたように，形式化の対象とした $\mathcal{RPT}$ は [11] の $\mathcal{RPT}$ にわずかな変更を加えたものである．変更後の $\mathcal{RPT}$ が，ここで与えた形式的体系 RPT のモデルになることは容易に言える．

**定理 1** (モデル) $\mathcal{RPT}$ は RPT のモデルである．□

**系 1** (無矛盾性) RPT は無矛盾である．□

$\mathcal{RPT}$ は，項の集合を同値関係 $\approx$ で割ったものを領域とするモデルである．これが RPT のモデルとなることから，以下の系が成立する．

**系 2** RPT で $p \vdash_i a = b$ が仮定なしに証明されることと，$a \approx b$ は同値である．□

この系は，RPT の同値関係 $=$ が，$\mathcal{RPT}$ の同値関係 $\approx$ を忠実に表現していることをあらわしている．この同値関係は，操作的に定義される内包的なものである．7節では，RPT の外で別の同値関係を導入する．

### 3.5 RPT の性質

形式的体系 RPT から左等号規則を除いた体系では，証明図と，その証明図の結論にあたる判定の証明項とはほぼ同じ構造を持っている．すなわち，証明項における簡約化可能な部分は，証明図におけるそれに対応している．RPT の証明図 (あるいはその結論の証明項) は，通常の論理体系のように強正規化可能，つまり，その中に含まれる基をどのような順番で簡約化しても正規形に到達する，とは限らない．また，後に定理 3 で述べるように弱正規化可能ですらない．しかし，

- $\lambda$ 抽象の内部を簡約化しない．
- 存在記号の導入，全称記号の除去の規則で証明項に取り込んだ項は簡約化しない．

という制限つきの簡約化に関しては良い性質を持つ．ここで，存在記号の導入等で取り込んだ項というのは，たとえば，$[\Omega\,.\,0] \vdash_i \exists x.x = \Omega$ という判定の証明項における $\Omega$ の部分のことであり，これは，意味的には証明の構造をあらわすのではなく，対象の世界の要素を表している．

**定理 2** RPT から左等号規則を除いた体系において，仮定なしに判定 $a \vdash_i p$ が証明できたとすると，$a$ は制限つき簡約化に関して強正規化可能である．□

**系 3** RPT において，仮定なしに判定 $a \vdash_i p$ が証明できたとすると，$a$ は制限つき簡約化に関して弱正規化可能である．□

この定理は，高階の理論に対する Tait の手法を変形することにより証明される．また，この証明から，RPT の無矛盾性の別証や，RPT の証明図に関する結果がいくつか得られる．しかし，本稿では，体系 RPT の証明論的性質を調べるのが目的ではないので，詳細は省略する．

## 4 RPT での証明の例

### 4.1 S combinator

通常の命題論理において，$A, B, C$ が論理式のとき，

$$(A \supset B \supset C) \supset (A \supset B) \supset A \supset C$$

という論理式 (これを $F$ とする) が証明可能である．この主張はいわゆるメタ定理に相当するが，以下のように RPT では体系内で証明することができる．

$$P_1 \equiv \cfrac{\cfrac{x \vdash_0 A \supset B \supset C \quad z \vdash_0 A}{xz \vdash_0 B \supset C} \quad \cfrac{y \vdash_0 A \supset B \quad z \vdash_0 A}{yz \vdash_0 B}}{xz(yz) \vdash_0 C}$$

$$P_2 \equiv \cfrac{\models_0 A \supset B \supset C \quad \cfrac{\models_0 A \supset B \quad \cfrac{\models_0 A \quad P_1'}{\lambda z.xz(yz) \vdash_0 A \supset C}}{\lambda yz.xz(yz) \vdash_0 (A \supset B) \supset A \supset C}}{S \vdash_0 F}$$

ここで，$P_1'$ は $P_1$ 中の 3 つの仮定が $P_2$ における $\supset$ 導入によって除去された証明図，$S \triangleq \lambda xyz.xz(yz)$, $F \triangleq (A \supset B \supset C) \supset (A \supset B) \supset A \supset C$ である．

$$P_3 \equiv \cfrac{\begin{array}{c} \cfrac{[u \vdash_1 (\models_0 A)]}{\models_0 A} \quad \cfrac{[w \vdash_1 (\models_0 C)]}{\models_0 C} \quad \cfrac{[v \vdash_1 (\models_0 B)]}{\models_0 B} \\ \vdots \; P_2 \\ \cfrac{S \vdash_0 F}{0 \vdash_1 (S \vdash_0 F)} \\ \vdots \\ \lambda uvw.0 \vdash_1 T \end{array}}{\lambda ABC.\lambda uvw.0 \vdash_1 \forall ABC.T}$$

ただし，$T \triangleq (\models_0 A) \supset (\models_0 B) \supset (\models_0 C) \supset (S \vdash_0 F)$ とする．

証明 $P_3$ は，最初の主張そのものを形式化した証明になっている．

### 4.2 集合の定義と Russell の逆理

$\mathcal{RPT}$ では，命題関数を集合と見なして構成的な集合論を展開することができる．すなわち，任意の項 $a$ に対して $p(a)$ が第 $i$ レベルの命題になるとき，$\{x|p(x)$ が成立する $\}$ という集合が定義されると考える．これは内包 (comprehension) の規則に対応している．このことは RPT では，以下のように表現される．

$$\mathtt{Set}_i[p] \triangleq \forall x.(\models_i p(x))$$

$$a \in_i p \triangleq \mathtt{Set}_i[p] \ \& \ p(a)$$

$\mathtt{Set}_i[p]$ は $p$ が $i$ レベルの集合であることを表し，$a \in_i p$ は，$p$ が集合であって $a$ がその要素であることを表す．$a \in_i p$ の定義で，通常の論理積 $\wedge$ ではなく，記号 $\&$ が使われている．仮に，$a \in_i p$ の定義を $\mathtt{Set}_i[p] \wedge p(a)$ とすると，$a$ の値によっては命題にならないことがある．一方，上記定義であれば，$a \in_i p$ はいつでも $i+1$ レベルの命題になることが証明できる．

集合の例として，自然数の集合 NatSet は

$$\mathtt{NatSet} \triangleq \lambda x.\mathtt{Nat}[x]$$

として定義できる．これより，$\mathtt{Set}_0[\mathtt{NatSet}]$ や任意の自然数 $n$ に対して，$n \in_0 \mathtt{NatSet}$ が言える．

素朴集合論における Russell の逆理は，集合 $R$ を $R \triangleq \{x|x \notin x\}$ として定義し，この定義式の $x$ に $R$ 自身を代入することにより，矛盾を導くものである．ZF 集合論においては，集合を構成する規則から comprehension の規則を排除して，$R$ を集合として構成できないようにすることにより，この逆理を避けている．一方，RPT では $R$ を集合として定義することができる．しかし，以下に述べるように，この集合はレベル付けされているため，逆理が生じないようになっている．

**定理 3** $R \triangleq \lambda f.\ \neg(f \in_0 f)$ とすると以下の *4* つの判定が証明可能である．

$$a \vdash_1 \neg\mathtt{Set}_0[R]$$

$$\lambda w.a(\mathtt{car}(w)) \vdash_1 \neg(R \in_0 R)$$

$$\lambda u.0 \vdash_2 \mathtt{Set}_1[R]$$

$$[\lambda u.0\ .\ \lambda w.a(\mathtt{car}(w))] \vdash_2 R \in_1 R$$

ただし，

$$a \triangleq \lambda u.(\lambda v.\mathtt{cdr}(v)v)[u\ .\ (\lambda v.\mathtt{cdr}(v)v)]$$

とする．□

$R$ は Russell 集合を命題関数の形で書いたものである．定理 3の意味は，$R$ はレベル 0 では集合ではなく，レベル 1 以上の世界で集合となる，ということである．従って，$R$ の定義で参照している $\neg(f \in_0 f)$ に $R$ を代入できず，矛盾は生じない．なお，項 $a$ は，RPT の閉じた証明の証明項であるが，正規形を持たない．言い換えると，RPT の証明図は必ずしも正規化可能ではない．

### 4.3 自己反映階層の有用性 -1

$\mathcal{RPT}$ の大きな特徴は，自己反映の階層により同一の体系内でメタ定理を取り扱うことができる点である．本節では，その例をあげる．

論理和に関する性質 (Disjunction Property) と項の存在に関する性質 (Term Existence Property) [2]は，構成的論理体系であれば成立することが望ましい性質である．2節で述べたように，1 階述語論理に基づく体系等では，これらの性質は，実現可能性解釈などを用いてメタな立場から証明されるが，RPT では，体系内で形式化したこれらの性質を証明することができる．

以下では $\exists x.\ x \vdash_i p$ を $\vdash_i p$ と略記する．

**定理 4** 論理和に関する性質「命題 $p, q$ に対して，$p \vee q$ が証明可能であれば，$p$ か $q$ が証明可能であること」を形式化した命題は RPT で証明できる．すなわち，ある項 $a$ に対して，

$$a \vdash_{i+1} \forall p.\forall q.\ \ (\models_i p) \supset (\models_i q) \supset (\vdash_i p \vee q) \supset (\vdash_i p) \vee (\vdash_i q)$$

が成立する．□

**定理 5** 項の存在に関する性質「命題 $\exists x.p$ が証明可能であれば，その証明から項 $t$ と命題 $p_x[t]$ の証明とが得られる」ことを形式化した命題は RPT で証明できる．すなわち，ある項 $a$ に対して，

$$a \vdash_{i+1} \forall f.(\mathtt{PF}[f] \supset \forall y.((y \vdash_i \exists x.f(x)) \supset (\vdash_i f(\mathtt{car}(y)))))$$

が成立する．ここで PF は先に定義した「*1* 引数命題関数であること」をあらわす述語である．□

### 4.4 自己反映階層の有用性 -2

前節でも自己反映階層を利用した例について述べたが，ここではより一般的な例として，ハロップ論理式に

関するメタ定理を形式化する．

1 階述語論理では，ハロップ論理式は以下のように定義される．

**定義 7** (ハロップ論理式)

$$H \quad ::= \quad \texttt{true} \mid \texttt{false} \mid t = t \mid H \wedge H \mid P \supset H \mid \forall x.H$$

ここで $P$ は任意の論理式である．□

直観主義の 1 階述語論理において，項 $a$ が論理式 $P$ の実現子であることを $a\ \mathbf{r}\ P$ と表現する[2]．実現可能性解釈 $a\ \mathbf{r}\ P$ は，項 $a$ が $P$ の証明と同じ構造を持っていることを表しているので，RPT における $a \vdash_i P$ という判定と意味的な共通性がある．$P$ がハロップ論理式である場合，以下の良く知られた定理が成立する．

**定理 6** $H$ をハロップ論理式，$FV(H) \subseteq \{x\}$ とすると，ある項 $t$ に対して $H \supset (t(x)\ \mathbf{r}\ H)$ が成立する．□

RPT では，このメタ定理に対応した定理を体系内で証明することができる．以下では，任意の項 $a$ に対し $p(a)$ がハロップ命題になる項 $p$ をハロップ命題関数と呼ぶことにする．

**定義 8** (ハロップ命題関数)

$$\begin{aligned}
\texttt{HPF}_i \triangleq \lambda f.\lambda p.\ & p = \lambda x.\ \texttt{true} \\
& \vee\, p = \lambda x.\ \texttt{false} \\
& \vee\, \exists y.\exists z.\ p = (\lambda x.\ y(x) = z(x)) \\
& \vee\, \exists q.\exists r.\ (p = (\lambda x.\ q(x) \wedge r(x))) \wedge f[q] \wedge f[r] \\
& \vee\, \exists q.\exists r.\ (p = (\lambda x.\ q(x) \supset r(x))) \\
& \qquad \wedge \forall x.\ (\models_i q(x)) \wedge f[r] \\
& \vee\, \exists q.\ (p = \lambda x.\ \forall y.\ q(\texttt{[}x\ .\ y\texttt{]})) \wedge f[q]
\end{aligned}$$

**定理 7** 適当な項 $a$ に対して

$$a \vdash_{i+2} \forall p.(\texttt{HPF}_i[p] \supset \forall x.(\models_{i+1} p(x)))$$

が成立する．□

**定理 8** 適当な項 $a$ に対して

$$a \vdash_{i+2} \forall p.(\texttt{HPF}_i[p] \supset \exists f.\forall x.(p(x) \supset (f(x) \vdash_{i+1} p(x))))$$

が成立する．□

いずれも，述語 $\texttt{HPF}_i$ に関する帰納法で証明される．定理 7, 8は，ハロップ命題関数の定義に $\lambda x.\ a(x)\downarrow$ や $\lambda x.\ a(x)\ \&\ b(x)$ を追加しても成立する．

## 5 証明システムの実現

本節では，計算機上に実現された RPT の証明システムについて述べる．

### 5.1 計算機による支援システムの必要性

1節で述べた手順に従って構成的プログラミングを行うには，仕様の証明を誤りなく与える必要がある．言うまでもなく，証明が誤っていれば抽出されたプログラムの正しさは保証されない．従って，構成的プログラミングの実行には，計算機上に証明チェッカを実現することが必要である．

また，一般に，仕様の証明は大きなものになるので，人間の手間をなるべく減らすため計算機システムの支援が必要である．これは，良好なユーザインターフェースや自動証明系を含む．本システムは，自動証明の可能性について検討することが目的ではないが，後述するようにある程度の自動化をはかっている．

### 5.2 実　現

本システムの実現には，プログラム言語 $\Lambda$ を用いた．ほとんどの証明支援システムでは，その体系の項の集合より，システムを実現するために使うプログラム言語がはるかに強力である．このようなシステムでは，システム (それ自体プログラムである) についての推論を行うためには，さらに強力な体系が必要になってしまう．

本システムでは，体系 RPT の推論対象である項の集合と，システムを実現するプログラム言語とを一致させることを目標とした．これにより，最終的には，証明システムのプログラム自身に関する推論をシステムで行うことが可能になる．

RPT の項や判定を $\Lambda$ 上にのせるため，quoteの機構を使った．quoteは，$\Lambda$ の全ての項を 式項に一対一に写像するものである．$\Lambda$ では，Lisp の quoteとは異なり，quote$(a)$ を計算した結果は quote$(a)$ 自身である．

### 5.3 システム構成

本システムは対話的に証明を作成するためのシステムである．ただし自動証明を重視したものではなく，証明図の葉の方から根の方へ順番に証明を生成していくものである．

システムのトップレベルで入力可能なコマンドは，現在のレベルや証明項を表示するもののほかは，RPT の推論規則に一対一に対応している．

```
RPT:0> true-intro
Result:0: |- T
RPT:1> proof
proof-term is 0
RPT:1>
```

あるのが全てだったと思います．この例で，RPT:0> はプロンプトであり，それに対して true-intro という推論規則を入力した結果，$0 \vdash_0$ true を結論に持つ証明が得られた．なお，システムでは true を T と表示する．証明を表す項およびレベルは煩雑になるので通常は表示されず，それらを表示するコマンドを入力した時のみ表示される．上の例では，RPT:1> というプロンプトに応じて proof というコマンドを入力し証明項を表示させている．なお，表示コマンドは証明を進めるものではないので，プロンプトの後の数字 (証明のステップ数にほぼ対応する) は 1 のままである．

推論規則に対応するコマンドは，引数をいくつか取り，それらを組み合わせた証明を返す．もし，正しく証明が生成できない場合はエラーを返す．入力の手間を軽減するため，引数を省略すると文脈により適宜補われる．たとえば，

```
RPT:72> (and-intro -2 -1)
```

と入力すれば，2 ステップ前の証明と 1 ステップ前の証明に，$\wedge$ 導入規則を適用するものであるが，これは，単に，

```
RPT:72> and-intro
```

と入力するだけ同じ効果が得られる．また，省略された引数が，証明ではなく，項を表すものであるときは，入力待ち状態となる．

```
RPT:74> prop-eql
term?
```

この例では，$\models_0 a = b$ の形の 判定を導くため，項 $a$ の入力待ち状態となっている．

本システムでは，証明項を表示したり，それに適当な値を適用して計算を行せることができる．また，RPT に含まれる推論規則以外に，いくつかの派生規則も利用できる．たとえば，存在記号除去規則に対応する推論は，RPT では派生規則であるが，実際の証明では有用なので，コマンドとして用意している．また，3 個以上の命題の論理積，論理和，2 個以上の変数による限量などを 1 つの派生規則で行えるようにした．ただし，これらはプリミティブな推論規則ではないので，RPT の推論規則の組み合わせを呼び出すだけである．

本システムの使用例については，次節で述べる．

### 5.4 証明の自動化

本システムは，実際に証明を作成する上で有用ないくつかの簡単な自動化手続きを組みこんである．

- 命題であることの証明

  $\mathcal{RPT}$ の特徴の 1 つは，命題であることが正しさに依存することである．たとえば，nil という項は命題にならないので，$a \supset$ nil が命題になることと，$a$ が正しくない命題になることは同値である．従って，命題になることも証明を要する．このことは，$\mathcal{RPT}$ を表現力豊かな体系にしているが，証明チェッカの立場からは，通常の論理体系のように，論理式であることが簡単にチェックできない，という取り扱いにくい性質となる．

  しかし実際には，$a \supset b$ と $a \,\&\, b$ の形の項において，正しさに依存しないで命題であることが証明できる場合が多く，その場合，命題であることは決定可能である．この決定手続きを組み込んだ．

- 計算

  1 ステップの計算のほかに，何ステップでも正規形に至るまで計算する，というコマンドを作成した．

- 簡単化

  ここでいう簡単化は，true $\wedge A$ という形の命題を $A$ に変形する，正規項でない命題を計算する等の通常の簡単化手続きを，存在記号の除去規則と組み合わせたものである．

  いくつかの命題 (ただし，等式を 1 つ以上含むとする) の論理積を存在記号で限量した命題を $E$ 命題

とよぶことにする．たとえば，

$e_1 \triangleq \exists x.\exists y.\exists z.(u = [x \,.\, y] \wedge y = [0 \,.\, z] \wedge P[z])$

は $E$ 命題である．

命題 $p$ と $E$ 命題 $e$ に対して，$a \vdash_i p$ と $b \vdash_i e$ が証明されているとする．以下の手続きが成功すれば，$p$ を簡単化した命題 $q$ に対して，$a \vdash_i q$ の証明が得られる．この手続きを $E$ 簡単化とよぶ．

1. あらかじめ $e$ で存在記号で束縛されている変数を $p$, $e$ の自由変数とは異なる名前に変更しておく．
2. $e$ に含まれる等式の集合に対して，単一化を行い，その最汎単一化子 (mgu) を $\theta$ とする．
3. $p\theta$ に対し，$\texttt{pair?}([a \,.\, b])$ を $\texttt{true}$ で置き換える, $\texttt{true} \wedge A$ を $A$ で置き換えるなどの簡単化を行って得られる命題を $r$ とする．
4. $r$ 中に，$e$ で存在束縛されている変数が出現しないとき，この手続きは成功する．1 つでも出現するとき，失敗する．

例として，上記の $E$ 命題 $e_1$ と以下の命題 $p_1$ を組み合わせる．

$$p_1 \triangleq \texttt{pair?}(u) = \texttt{true} \ \wedge\ \texttt{fun?}(\texttt{cdr}(u)) = \texttt{true} \ \vee p_2$$

ここで $p_2$ は変数 $u$ を含まない命題とする．$e_1$ の等式集合から得られる mgu は，

$$\theta_1 \triangleq \{u := [x \,.\, [0 \,.\, z]], y := [0 \,.\, z]\}$$

である．$p_1\theta$ を計算すると $\texttt{true} = \texttt{true} \wedge \texttt{false} = \texttt{true} \vee p_2$ となり，さらに簡単化して $p_2$ が得られる．変数 $x, y, z$ を含まない命題を得られたので $p_2$ の証明が得られた．

これは単純な方法であるが，帰納的述語定義を自然に書くと，$E$ 命題の論理和の形になることが多く，帰納法のステップの証明に有効である．特に，次節で述べる Church-Rosser の定理の証明において有効であった．

## 6 Church-Rosser の定理の証明

### 6.1 証明手順

Church-Rosser の定理は，項書換えシステムや関数型プログラム言語にとって最も基本的な定理である．

**定義 9 (Diamond Property)** *2* 項関係 $R$ が *Diamond Property* を持つとは，$aRb$ かつ $aRc$ となる任意の項 $a, b, c$ に対して，$bRd$ かつ $cRd$ となる項 $d$ が存在することである．□

Diamond Property から Church-Rosser の定理は導かれるので，以下ではもっぱら Diamond Property について考察する．

計算機上での Church-Rosser の定理の証明には，純粋なラムダ計算に対する Shankar の仕事[13]がある．ここでは，本システムの主たる興味は，RPT や $\Lambda$ 自身に関する推論であるので，$\Lambda$ に対する Church-Rosser の定理を表現し，その証明を計算機上のシステムで証明することを試みた．

### 6.2 証明方法

純粋なラムダ計算に対する Church-Rosser の定理は，Tait, Martin-Löf の方法を高橋[14]が改良した．この方法は [11] により $\Lambda$ に適用されている．この方法は，以下の通りである．

1. 項の並行簡約 $\rightarrow$ を定義する．
   並行簡約は同時に簡約化できる基のうちのいくつかを簡約化する，という 2 項関係である．たとえば，$(\lambda x.xx)(\texttt{car}([y \,.\, z]))$ を並行簡約すると，この項自身，$(\texttt{car}([y \,.\, z]))(\texttt{car}([y \,.\, z]))$, $(\lambda x.xx)y$, $yy$ の 4 通りの項になり得る．
2. 項 $a$ に対して $a^*$ を定義する．
   項 $a$ に対して $a^*$ は，項 $a$ の中の同時に簡約化できる基の全てを簡約化したものである．たとえば，上の例では，
   $$((\lambda x.xx)(\texttt{car}([y \,.\, z])))^* \equiv yy$$
   となる．当然ながら，$a \rightarrow a^*$ が成立する．
3. $\rightarrow$ と $^*$ の以下の性質を証明する．
   - $a \rightarrow a$
   - $a \rightarrow c$ かつ $b \rightarrow d$ ならば $a_x[b] \rightarrow c_x[d]$
   - $a \rightarrow b$ ならば $b \rightarrow a^*$

   これによって $\rightarrow$ が Diamond Property (合流性) を持つことが言える．
4. 最後に，Diamond Property を持つ簡約化の反射的推移的閉包が Diamond Property を持つことを示す．

上記の証明を本システムを用いて実行した手順は以下の通りである．

1. 対象となる $\Lambda$ の項を，式項で形式化する．
   本システムはプログラム言語 $\Lambda$ の上に，体系 RPT を実現するため，RPT の要素は全て quote されているが，項の簡約化などを取り扱うためには更にもう一度 quote して形式化する必要がある．それに基づいて，「$\Lambda$ の項である」「並行簡約」などを帰納的述語として定義する．
2. $\rightarrow$ が Diamond Property を持つことを証明する．
3. $\rightarrow$ の推移的閉包が Diamond Property を持つことを証明する．

明らかに $\rightarrow$ の推移的閉包と $\rhd^*$ は一致するので，これによって $\rhd^*$ の Diamond Property も証明できたことになる．

$\rightarrow$ が Diamond Property を持つことの証明は，項の構成に関する帰納法を用いた．純粋なラムダ式に比べ，場合分けの数が非常に多いが，前節で述べた $E$ 命題に関する簡単化などを利用することにより，Diamond Property を証明することができた．

ここでは，証明の最後の部分について説明する．

### 6.3 Diamond Property の推移的閉包への拡張

**定理 9** 適当な項 $a$ に対して，

$$a \vdash_1 \forall S.\forall A.(\mathtt{Unary}(S) \supset \mathtt{Binary}(A) \supset \mathtt{Persis}(A,S) \supset \mathtt{Diamond}(A,S) \supset \mathtt{Diamond}(\mathtt{Trans}(A),S))$$

が証明可能である．

ただし，

$\mathtt{Unary}(S) \triangleq \forall x.\models_0 S[x]$

$\mathtt{Binary}(A) \triangleq \forall x.\forall y.\models_0 A[x,y]$

$\mathtt{Trans}(A) \triangleq \lambda f.\lambda x.\lambda y.(x=y \vee \exists z.A[x,z] \wedge f[z,y])$

$\mathtt{Persis}(A,S) \triangleq \lambda x,y.(A[x,y] \supset S[x] \supset S[y])$

$$\mathtt{Diamond}(A,S) \triangleq \forall x.(S[x] \supset \forall y.\forall z.(A[x,y] \wedge A[x,z] \supset \exists u.(A[y,u] \wedge A[z,u])))$$

とする．□

この定理は，二項関係 $A$ が Diamond Property を持てば，その推移的閉包 $\mathtt{Trans}(A)$ も Diamond Property を持つことを表している．なお，単項述語 $S$ は (形式化された) 項であることを意味する述語であり，$\mathtt{Persis}(A,S)$ は，項であるという性質が簡約化 $A$ で保存されることを意味している．

定理 9は，任意の二項関係に対する主張であるので，$\rightarrow$ 以外の簡約化に対しても適用できる一般性を有している．このような表現ができることが，RPT の長所の 1 つである．この証明は本システムを利用して約 170 ステップであった．証明の要点を付録に記す．

## 7 プログラム生成

本節では，証明からプログラム作成までを一貫して行った例としてリストの append プログラムについて述べる．

### 7.1 append の仕様とその証明

append プログラムの仕様は以下の通りである．

$$\forall x.(\mathtt{List}[x] \supset \forall y.(\mathtt{List}[y] \supset \exists z.\mathtt{Append}[x,y,z]))$$

ここで List, Appendは，それぞれ以下のように定義された帰納的述語である．

$$\mathtt{List} \triangleq \lambda f.\lambda x.(x=\mathtt{nil} \vee \exists x_1.\exists x_2.\ x=[x_1\ .\ x_2] \wedge f[x_2])$$

$$\mathtt{Append} \triangleq \lambda f.\lambda x.\lambda y.\lambda z.\ x=\mathtt{nil}\ \wedge\ y=z \ \vee\ \exists x_1.\exists x_2.(x=[x_1\ .\ x_2] \wedge \exists z_1.(f[x_2,y,z_1]\ \wedge\ z=[x_1\ .\ z_1]))$$

上記仕様の証明を本システムで行った．List や Append という述語の定義や途中で得られた定理への名前付けなど冗長なものも含め，80 ステップほどで証明が完成し証明項 $T_{app}$ が得られた．得られた項 $T_{app}$ を付録に記す．

$T_{app}$ は，

- リスト $x$
- $x$ がリストであることの証明 $q$
- リスト $y$
- $y$ がリストであることの証明 $l$

の 4 つの引数を取り，$\exists z.Append[x,y,z]$ の証明を返すプログラムである．従って，$x$ がリストであることの証明を $r_{list}(x)$ とすると，$\mathtt{car}(T_{app}x(r_{list}(x))y(r_{list}(y)))$ が $x$ と $y$ を append

した結果となる．

### 7.2 得られたプログラムの改善

明らかに $T_{app}$ は非常に効率が悪いプログラムである．また，入力のリスト $x$ と $y$ 以外に，それらがリストであることの証明を必要とするという意味で，満足できるプログラムとはいえない．第 1 の問題については次節で議論することとし，本節では第 2 の問題について考察する．

**定理 10** 適当な項 $a$ に対して以下が成立する．

$$a \vdash_1 (\texttt{List}[x] \supset r_{list}(x) \vdash_0 \texttt{List}[x])$$

ただし，

$$\begin{aligned} r_{list} \triangleq \mu(\lambda f.\lambda x.& \\ &\texttt{if null?}(x)\ \texttt{then [true . 0]} \\ &\quad \texttt{else [false.[car}(x)\texttt{.[cdr}(x) \\ &\qquad \texttt{.[0 .}\ f(\texttt{cdr}(x))\texttt{]]]]\ fi}) \end{aligned}$$

とする．□

この定理は $x$ がリストであることの証明 $r_{list}(x)$ が $x$ から計算することができるという主張である．自然数，リスト，木など実用上有用な多くのデータ型に対して，同様の定理が成立する．

### 7.3 プログラム変換による冗長性の除去

本節では，append プログラム $T_{app}$ から冗長性を除去することを目標とする．そのためにプログラムの等価性を保ちながらプログラム変換を行う．プログラム変換の目的のためには，操作的に与えられた同値関係 $\approx$ より粗い同値関係が必要である．

値呼び (call by value) の計算系では，正規形が計算の結果と考えられる． $F$ と $G$ を 1 引数の関数とすると，任意の正規項 $n$ と $m$ に対して，

$$F(n) \rhd^* m \text{ と } G(n) \rhd^* m \text{ は同値}$$

であるとき， $F$ と $G$ は外延的 (extensional) に等しい関数である． $F$ と $G$ は必ずしも内包的 (intensional) に等しくはないが，計算の答えに着目する限り， $F$ を $G$ で置き換えても構わないことが直観的に理解できる．$\Lambda$ のプログラム変換における項の同値性は基本的にはこの考えに基づくが，$\Lambda$ の計算は本質的に遅延評価であり，計算の値を正規形ではなく正準形でとらえるべきである．

たとえば，

$$\texttt{I} \triangleq (\mu(\lambda f.\lambda n.[n\ .\ f(suc(n))]))0$$

(ここで， $suc(n)$ は $n+1$ をあらわす項とする) とおくと，I は自然数の無限リストをあらわす．項 $\Omega$ と I はどちらも止まらない計算を表しており，上の値呼び計算系に対する外延性の定義では同値になる．しかし，これらは任意の文脈で同じ意味を持つわけではない．たとえば，pair?$(x)$ の $x$ 中にこれらの項を代入すると，前者は正準形を持たないが後者はtrueを返す．これらを区別するため，上記の単純な定義を工夫する必要がある．

ここでは，$\Lambda$ を[7]の遅延評価系 (lazy computation system) の一例と見なして，同値関係 $\sim$ を導入する．

**定義 10** (遅延評価系における前順序[7]) $R$ を閉じた項の上の *2* 項関係 $R$ とするとき， *2* 項関係 $T[R]$ を以下で定義する．

$$\begin{aligned} s\ T[R]\ t \triangleq& \\ &(s \rhd^* \texttt{nil} \supset t \rhd^* \texttt{nil}) \\ &\wedge (s \rhd^* \texttt{true} \supset t \rhd^* \texttt{true}) \\ &\wedge (s \rhd^* \texttt{false} \supset t \rhd^* \texttt{false}) \\ &\wedge \forall s_1, s_2.\ (s \rhd^* [s_1\ .\ s_2] \supset \\ &\qquad \exists t_1, t_2.\ (t \rhd^* [t_1\ .\ t_2] \wedge (s_1\ R\ t_1) \\ &\qquad\qquad \wedge (s_2\ R\ t_2))) \\ &\wedge \forall a.\ (s \rhd^* \lambda x.a \supset \\ &\qquad \exists b.\ (t \rhd^* \lambda y.b \wedge \forall u.\ (a_x[u]\ R\ b_y[u]))) \end{aligned}$$

ここで， $s_1, s_2, t_1, t_2, u$ の動く範囲は任意の閉じた項， $a, b$ の動く範囲は任意の項である†2．閉じた項の上の前順序 *(preorder)* $\leq$ は， $R = T[R]$ という等式の最大不動点として定義される．

自由変数を持つ項 $F, G$ に対しては， $FV(F) \cup FV(G) = \{x_1, \cdots, x_n\}$ とすると，任意の閉じた項の列 $d_1, \cdots, d_n$ に対して， $F_{\vec{x}}[\vec{d}] \leq G_{\vec{x}}[\vec{d}]$ が成立するとき $F \leq G$ であると定義する．□

上記の等式の右辺は $R$ が狭正にのみあらわれるので単調であり，最大不動点はたしかに存在する．

$\leq$ の例として，任意の項 $t$ に対して， $\Omega \leq t$ が成立する．また，$\texttt{I} \leq [0\ .\ [x\ .\ y]]$ が成立する．

**定義 11** (遅延評価系における外延的同値性[7]) *2* つ の項 $F, G$ が外延的に同値とは， $F \leq G$ かつ $G \leq F$ が

†2 本節で用いる論理記号 $\wedge$ や $\forall$ などは RPT の体系内ではなく体系の外でのメタな記号である．

成立することである．このとき，$F \sim G$ と書く．□

**定理 11** ($\le$ と $\sim$ の性質[7]) 以下が成立する．

*1.* $\le$ は反射的かつ推移的である．

*2.* $a \approx b$ ならば $a \le b$ である．

*3.* $\sim$ は同値関係である．□

1 の反射律の証明には，$\le$ が不動点の中で最大のものであることを使っている．定理の証明は容易である．

$\Lambda$ は[7]の用語で，関数記号に関して外延的 (operator extensional) な系であるので，$\sim$ は合同関係 (congruence relation) になっている．すなわち，$\sim$ は項の構成と可換である．さらに，$\Lambda$ は[7]の決定的 (deterministic) などの条件を満たすので，$\sim$ は観測同値 (observational equivalence)†3 の概念と一致することがわかる．

以下では，$\sim$ に関して，$\Lambda$ におけるいくつかの有用な変形が正当化されることを示す．

**定理 12** 変数 $x, y, z, u, v, w,$ 項 $a, b$ に対して以下が成立する．

$\texttt{car}(\texttt{if}\ x\ \texttt{then}\ y\ \texttt{else}\ z\ \texttt{fi})$
$\sim \texttt{if}\ x\ \texttt{then}\ \texttt{car}(y)\ \texttt{else}\ \texttt{car}(z)\ \texttt{fi}$

この式の car を cdr, null?, true?, false?, pair?, fun? に置き換えた式も成立する．

$(\texttt{if}\ x\ \texttt{then}\ y\ \texttt{else}\ z\ \texttt{fi})(w)$
$\sim \texttt{if}\ x\ \texttt{then}\ y(w)\ \texttt{else}\ z(w)\ \texttt{fi}$

$\texttt{if}\ (\texttt{if}\ x\ \texttt{then}\ y\ \texttt{else}\ z\ \texttt{fi})\ \texttt{then}\ u\ \texttt{else}\ v\ \texttt{fi}$
$\sim \texttt{if}\ x\ \texttt{then}\ (\texttt{if}\ y\ \texttt{then}\ u\ \texttt{else}\ v\ \texttt{fi})$
$\texttt{else}\ (\texttt{if}\ z\ \texttt{then}\ u\ \texttt{else}\ v\ \texttt{fi})\ \texttt{fi}$

$\texttt{if}\ x\ \texttt{then}\ a\ \texttt{else}\ b\ \texttt{fi}$
$\sim \texttt{if}\ x\ \texttt{then}\ a_x[\texttt{true}]\ \texttt{else}\ b_x[\texttt{false}]\ \texttt{fi}$

□

**定理 12の証明**

car に対する等式だけを証明する．

$\theta$ を $x, y, z$ に閉じた項をいれる代入
$L \triangleq \texttt{car}(\texttt{if}\ x\ \texttt{then}\ y\ \texttt{else}\ z\ \texttt{fi})\theta$
$M \triangleq (\texttt{if}\ x\ \texttt{then}\ \texttt{car}(y)\ \texttt{else}\ \texttt{car}(z)\ \texttt{fi})\theta$
とする．

まず $L \le M$ を示す．$L$ が正準形を持たない場合，$L \le M$ は成立するので，$L$ は正準形を持つと仮定する．すると，$\Lambda$ の簡約化規則より，(i) $x\theta \rhd^* \texttt{true}$ かつ 適当な項 $a, b$ に対して $y\theta \rhd^* [a\ .\ b]$ となるか，(ii) $x\theta \rhd^* \texttt{false}$ かつ 適当な項 $a, b$ に対して $z\theta \rhd^* [a\ .\ b]$ となるか，のいずれかである．いずれにせよ，$L \rhd^* a$ かつ $M \rhd^* a$ がいえるので，$L \le M$ は成立する．

$M \le L$ も同様に示すことができる．□

定理 12の $\sim$ の両辺の項は $\approx$ の意味では等しくない．従って，$F \sim G$ ならば $F \approx G$ という命題は必ずしも成立しない．

次に正格 (strict) な項という概念を定義する．直観的には，変数 $f$ に関して項 $c$ が正格であるとは，$c$ の計算に $f$ の値が必要とされるということである．

**定義 12** (正格な項)変数 $f$ に関して 正格な項 $c$ は以下で定義される．

$$
\begin{array}{rcl}
c & ::= & f \\
& | & \texttt{null?}(c) \mid \texttt{true?}(c) \mid \texttt{false?}(c) \\
& | & \texttt{pair?}(c) \mid \texttt{fun?}(c) \\
& | & \texttt{car}(c) \mid \texttt{cdr}(c) \\
& | & c(t) \\
& | & \texttt{if}\ c\ \texttt{then}\ t\ \texttt{else}\ t\ \texttt{fi}
\end{array}
$$

ここで $t$ は項を表す．□

**定理 13** 項 $F$, 相異なる変数 $f, x_1, \cdots, x_n,$ 項 $c, a, b_i^j$ $(1 \le i \le n, 1 \le j \le k)$ に対して，

$FV(F) = \{f\}$
$FV(a) = \emptyset$
$FV(b_i^j) \subseteq \{x_1, \cdots, x_n\}$
$FV(c) \subseteq \{f, x_1, \cdots, x_n\}$
$c$ は $f$ に関して 正格
$c_f[F] \sim a(c_{\vec{x}}[\vec{b^1}], \cdots, c_{\vec{x}}[\vec{b^k}], \vec{x})$

ならば，

$$c_f[\mu(\lambda f.F)] \sim G(\vec{x})$$

が成立する．ただし，$G$ は以下で定義される項である．

$$G \triangleq \mu(\lambda g.\lambda \vec{x}.a(g(\vec{b^1}), \cdots, g(\vec{b^k}), \vec{x}))$$

□

定理 13 は，$\mu$ による再帰呼出しにおいて，計算の無駄をなくすための中心となる定理である．たとえば，

$F \triangleq \lambda x.\texttt{if}\ \ \texttt{null?}(x)\ \ \texttt{then}\ \ [b\ .\ c]$

†3 任意の文脈 $C\langle\cdot\rangle$ に対して，$C\langle a\rangle$ と $C\langle b\rangle$ が同種の正準項になるか，あるいは正準形を持たないとき，$a$ と $b$ は観測同値である，と言われる．

$\quad\quad\quad\quad$ else $[p(\mathtt{car}(f(pred(x))))$

$\quad\quad\quad\quad\quad\quad . \ q(f(pred(x)))]$ fi

という項を考える．ここで，$pred(x)$ は $x-1$ をあらわし，$p$ の自由変数は $x$ のみであると仮定する．$F' \triangleq \mu(\lambda f.F)$ として，$F'$ を自然数 $n$ に適用すると，$\mu$ による再帰呼び出しの過程で途中結果は対の形で保持されている．$F'(n)$ の計算は $F'(pred(n))$ の値を参照しているが，対の第 1 要素は $\mathtt{car}(F'(pred(n)))$ のみを参照しているので，最終結果として $\mathtt{car}(F'(n))$ のみが欲しい場合には，再帰呼び出し自体を第 1 要素に限定しても良いと思われる．すなわち，

$H \triangleq \lambda x.$if null?$(x)$ then $b$

$\quad\quad\quad\quad$ else $p(g(pred(x)))$ fi

とおくと，$\mathtt{car}(\mu(\lambda f.F)x)$ を $\mu(\lambda g.H)x$ で置き換えても良いと考えられる．$\mathtt{car}(\mu(\lambda f.F)x) \approx \mu(\lambda g.H)x$ は言えないが，$\mathtt{car}(\mu(\lambda f.F)x) \sim \mu(\lambda g.H)x$ は証明できるので，$\sim$ に関して上記の置き換えは正当化される．

この例は，定理 13で，$c$ を $\mathtt{car}(fx)$ と置き，$F, G$ を上記の $F, \mu(\lambda g.H)$ と置いたことに相当する．また，

$c_f[F] \sim$ if null?$(x)$ then $b$

$\quad\quad\quad\quad$ else $p(\mathtt{car}(f(pred(x))))$ fi

の変形で定理 12を用いた．

定理 13の証明は付録に記す．

**定理 14** 相異なる変数 $f, y, x_1, \cdots, x_m$, 項 $F, G, a, b_i^j, c^j$ $(1 \le i \le m, 1 \le j \le k)$ に対して，

$$F \equiv \lambda\vec{x}.\lambda y.a(f(\vec{b^1}, c^1), \cdots, f(\vec{b^k}, c^k), \vec{x})$$

$$G \equiv \mu(\lambda f.F)$$

$$FV(a) = \emptyset$$

$$FV(b_i^j) \subseteq \{x_1, \cdots, x_m\}$$

$$FV(c^j) \subseteq \{x_1, \cdots, x_m, y\}$$

ならば，

$$G(\vec{x}, y) \sim \mu(\lambda g.\lambda\vec{x}.a(g(\vec{b^1}), \cdots, g(\vec{b^k}), \vec{x}))(\vec{x})$$

が成立する．□

この定理は，$\mu$ による再帰呼出しで，使われていない変数 (この場合は $y$) があれば，それを除去できることを意味している．この定理も定理 13と同様に証明される．

#### 7.3.1 append プログラムの効率化

この項に，定理 13を適用するため，$T_{app}$ の $\mu$ の中身を $F$, $\mathtt{car}(fx(r_{list}(x))yl)$ を $c$ とおく．すると，$c_f[F]$ は，定理 12を使って簡単化することができ，定理 13から次の式を得る．

$\mathtt{car}(T_{app}x(r_{list}(x))yl)$

$\quad \sim \mu(\lambda g.\lambda x, y, l.$

$\quad\quad\quad$ if null?$(x)$ then $y$

$\quad\quad\quad$ else $[\mathtt{car}(x) \ . \ g(\mathtt{cdr}(x), y, l)]$ fi$)$

が得られる．右辺の変数 $l$ は無駄な変数であるので，定理 14を適用することにより，

$\mathtt{car}(T_{app}x(r_{list}(x))yl)$

$\quad \sim \mu(\lambda g.\lambda x, y.$

$\quad\quad\quad$ if null?$(x)$ then $y$

$\quad\quad\quad$ else $[\mathtt{car}(x) \ . \ g(\mathtt{cdr}(x), y)]$ fi$)$

が得られる．最後の式の右辺は人間が直接作成する append のプログラムと同じである．

本節で述べた定理に基づく効率化の手法は実現されたシステムに組み込まれている．実際，項 $\mathtt{car}(T_{app}x(r_{list}(x))yl)$ から最終の append プログラムを得るまでの手順は，完全に自動的に行うことができた．従って，本システムでの方法により，効率良い append プログラムを，正しさの保証付きで抽出することができたと言える．ただし，定理 13などのメタ定理および $\sim$ の意味で等しい項に置き換えることの正当化は体系の外にあるため，体系に最大不動点に関する規則追加した上で，Church-Rosser の定理の証明と同様の符号化により体系内に取りこんで証明する必要がある．

一般的に証明から抽出した項の効率化を自動的に行うのは困難である．しかし，本節で述べた方法は原理的に [15] に対応したプログラムの効率化を行うことができ，かなり多くの場合に適用可能であると考えられる．

## 8 まとめ

本稿では，構成的プログラミングを行うために，基礎となる $\mathcal{RPT}$ の形式化 RPT, その性質，RPT を実現した証明システム，その例としての Church-Rosser の定理の証明例，RPT と証明システムを用いた構成的プログラミングについて述べた．

構成的数学に基づいた理論体系としては，Martin-Löf の型理論 $ITT_n$ [9], Feferman の $T_0$ [6], 林らの $PX$ [8], 佐藤らの $\mathcal{SST}$ [12], Coquand らの Calculus of Constructions (CoC) [4]などがある．$ITT_n$ およ

び CoC が型を持つ体系であるのに対し，$T_0$, $PX$, $\mathcal{SST}$ はいずれも型を持たない体系である．$\mathcal{RPT}$ は，型なしラムダ計算に基づくため，任意の計算可能関数を定義することができる．しかも，Curry-Howard の同型対応に基づき，直観主義の論理記号の解釈を与えている．従って，型あり，型なしの両方の体系の特徴を合わせ持つといえる．

計算機による構成的プログラミングのためのシステムとしては，Martin-Löf の理論の実現である Nuprl システム[3]，PX システム, 帰納的定義を追加した CoC を実現する Coq システム[5]などがある．これらは，自動証明，効率的なコードの生成，ユーザインターフェースなどにおいて特長を持つシステムである．本稿で述べた $\mathcal{RPT}$ 証明システムは，証明自動化および生成されたコードの効率化をある程度実行する手続きを含んでいる．また，証明システム自身の推論を行うことが可能である，という特徴を持っている．Nuprl システムに関する研究では，大規模な証明作成過程において，メタ理論の定理の有用性が指摘されているが，本稿で述べたように，$\mathcal{RPT}$ では，メタ理論を体系内で展開できるという利点がある．

今後の課題は，

- 本システムを用い，より多くの証明およびプログラミングの例題を作成すること，特に規模の大きなものを作成すること．
- 得られたプログラムの効率化について考察すること
- 構成的プログラミングを副作用や並列性を含む言語など，他の言語に適用すること

などである．

## 謝　辞

本稿をまとめるにあたり，東北大学龍田真博士にコメントをいただいた．また，査読者の方々には多くの重要な示唆をいただいた．ここに記して謝意を表します．

## 参考文献

[1] Aczel, P.: Frege Structures and the Notions of Proposition, Truth and Set, in *The Kleene Symposium* (Barwise, J., et al. eds.), North-Holland, 1980, pp. 31–59.

[2] Beeson, M.: *Foundations of Constructive Mathematics*, Springer-Verlag, 1985.

[3] Constable, R. L., et al.: *Implementing Mathematics with the Nuprl Proof Development System*, Prentice-Hall, 1986.

[4] Coquand, T. and Huet, G.: The Calculus of Constructions, *Information and Computation*, Vol. 76 (1988), pp. 95–120.

[5] Dowek, G., et al: The Coq Proof Assistant User's Guide, Version 5.8, Project Formel, INRIA-Rocquencourt, 1993.

[6] Feferman, S.: Constructive Theories of Functions and Classes, in *Logic Colloquium '78* (Boffa, M., et al. eds.), North-Holland, 1979, pp. 159–224.

[7] Howe, D. J.: Equality in Lazy Computation Systems, Proc. the Fourth Annual Symp. on Logic in Computer Science, IEEE Computer Society Press, 1989, pp. 198–203.

[8] Hayashi, S. and Nakano, H.: *PX: A Computational Logic*, The MIT Press, 1988.

[9] Martin-Löf, P.: *Intuitionistic Type Theory*, Bibliopolis, 1984.

[10] Sato, M. and Kameyama, Y.: Constructive Programming in SST, Proc. the Japanese-Czechoslovak Seminar on Theoretical Foundations of Knowledge Information Processing, Prague, 1989, pp. 23–30.

[11] Sato, M.: Adding Proof Objects and Inductive Definition Mechanisms to Frege Structures, Proc. International Conference on Theoretical Aspects of Computer Software, Lecture Notes in Computer Science, No. 526 (Ito, T. and Meyer, A. eds.), Sendai, Springer, 1991, pp. 53–87.

[12] 佐藤雅彦，桜井貴文: プログラムの基礎理論, 岩波講座ソフトウェア科学 13, 岩波書店, 1991.

[13] Shankar, N.: A Mechanical Proof of the Church-Rosser Theorem, *J. ACM*, Vol. 35, No. 3 (1988), pp. 475–522.

[14] Takahashi, M.: Parallel Reductions in λ-Calculus, *J. Symbolic Computation*, Vol. 7, (1989), pp. 113–123.

[15] 高山幸秀: 証明コンパイラにおける拡張射影手法, コンピュータソフトウェア, Vol. 7, No. 4 (1990), pp. 19–38.

## A RPT の推論規則

RPT の推論規則のうち，本文中で述べられなかったものについて述べる．

### 等号と正準項に関する規則

$$\frac{p \vdash_i \texttt{[}a \;.\; b\texttt{]} = \texttt{[}c \;.\; d\texttt{]}}{p \vdash_i a = c} \qquad \frac{p \vdash_i \texttt{[}a \;.\; b\texttt{]} = \texttt{[}c \;.\; d\texttt{]}}{p \vdash_i b = d}$$

$$\frac{p \vdash_i \texttt{null?}(a) = \texttt{true}}{0 \vdash_i a = \texttt{nil}} \qquad \frac{p \vdash_i \texttt{true?}(a) = \texttt{true}}{0 \vdash_i a = \texttt{true}}$$

$$\frac{p \vdash_i \texttt{false?}(a) = \texttt{true}}{0 \vdash_i a = \texttt{false}} \qquad \frac{p \vdash_i \texttt{pair?}(a) = \texttt{true}}{0 \vdash_i a = \texttt{[car}(a) \;.\; \texttt{cdr}(a)\texttt{]}}$$

$$\frac{p \vdash_i \texttt{fun?}(a) = \texttt{true}}{0 \vdash_i \#(a) = \texttt{false}} \ (\#(a) \text{ は } \texttt{fun?}(a) \text{ の形でない判別項})$$

$$\frac{a \vdash_i b = c}{0 \vdash_i a = 0} \qquad \frac{\models_i a_x[d] \quad c \vdash_i d = e}{\models_i a_x[e]}$$

$$\frac{a \vdash_i p}{0 \vdash_i a \downarrow}$$

$$\frac{}{0 \vdash_i a \downarrow} \ (\text{ただし } a \text{ は 正準項})$$

$$\frac{p \vdash_i \#(a) \downarrow}{0 \vdash_i a \downarrow} \ (\text{ただし } \#(a) \text{ は判別項})$$

$$\frac{p \vdash_i \texttt{car}(a) \downarrow}{0 \vdash_i \texttt{pair?}(a) = \texttt{true}} \qquad \frac{p \vdash_i \texttt{cdr}(a) \downarrow}{0 \vdash_i \texttt{pair?}(a) = \texttt{true}}$$

$$\frac{p \vdash_i a(b) \downarrow}{0 \vdash_i \texttt{fun?}(a) = \texttt{true}} \qquad \frac{p \vdash_i \mu a \downarrow}{0 \vdash_i \texttt{fun?}(a) = \texttt{true}}$$

$$\frac{p \vdash_i \texttt{if } a \texttt{ then } b \texttt{ else } c \texttt{ fi} \downarrow}{[a \ . \ 0] \vdash_i a = \texttt{true} \vee a = \texttt{false}}$$

$$\frac{p \vdash_i a \downarrow}{0 \vdash_i p = 0}$$

$$\frac{p \vdash_i a \downarrow}{D(a) \vdash_i a = \texttt{nil} \vee a = \texttt{true} \vee a = \texttt{false} \vee \texttt{pair?}(a) = \texttt{true} \vee \texttt{fun?}(a) = \texttt{true}}$$

ただし

$$\begin{aligned}
D(a) \triangleq\ & \texttt{if null?}(a) \texttt{ then [true . 0]} \\
& \texttt{else if true?}(a) \texttt{ then } c_1 \\
& \texttt{else if false?}(a) \texttt{ then } c_2 \\
& \texttt{else if pair?}(a) \texttt{ then } c_3 \\
& \texttt{else } c_4 \\
& \texttt{fi fi fi fi}
\end{aligned}$$

$c_1 \triangleq \texttt{[false . [true . 0]]}$
$c_2 \triangleq \texttt{[false . [false . [true . 0]]]}$
$c_3 \triangleq \texttt{[false . [false . [false . [true . 0]]]]}$
$c_4 \triangleq \texttt{[false . [false . [false . [false . 0]]]]}$

とする．

命題に関する規則

$$\frac{}{\models_i a = b} \qquad \frac{}{\models_i a \downarrow}$$

$$\frac{}{\models_i \texttt{true}} \qquad \frac{}{\models_i \texttt{false}}$$

$$\frac{}{\models_i (\models_j p)} \ (i > j) \qquad \frac{}{\models_i (p \vdash_j q)} \ (i > j)$$

$$\frac{\models_i p \quad \models_i q}{\models_i p \wedge q} \qquad \frac{\models_i p \wedge q}{\models_i p}$$

$$\frac{\models_i p \wedge q}{\models_i q} \qquad \frac{\models_i p \quad \models_i q}{\models_i p \vee q}$$

$$\frac{\models_i p \vee q}{\models_i p} \qquad \frac{\models_i p \vee q}{\models_i q}$$

$$\frac{\begin{matrix}[x \vdash_i p] \\ \models_i p \quad \models_i q\end{matrix}}{\models_i p \supset q} \ (x \text{ は固有変数}) \qquad \frac{\models_i p \supset q}{\models_i p}$$

$$\frac{\models_i p \supset q \quad a \vdash_i p}{\models_i q} \qquad \frac{\begin{matrix}[x \vdash_i p] \\ \models_i p \quad \models_i q\end{matrix}}{\models_i p \ \& \ q} \ (x \notin FV(q))$$

$$\frac{\models_i p \ \& \ q}{\models_i p} \qquad \frac{\models_i p \ \& \ q \quad a \vdash_i p}{\models_i q}$$

$$\frac{\models_i p}{\models_i \forall(\lambda x.p)} \ (x \text{ は固有変数}) \qquad \frac{\models_i \forall p}{\models_i p(t)}$$

$$\frac{\models_i p}{\models_i \exists(\lambda x.p)} \ (x \text{ は固有変数}) \qquad \frac{\models_i \exists p}{\models_i p(t)}$$

$$\frac{a \vdash_i (\models_j p)}{\models_j p} \ (i > j) \qquad \frac{a \vdash_i (\models_j p)}{0 \vdash_i a = 0} \ (i > j)$$

## B　証明システムを用いた定理 9の証明例

ここでは，RPT 証明システムを用いた定理 9の概要を記述する．以下では，セミコロン (;) で始まる行はコメントである．また，Result:$n$: ではじまる行は証明システムからの応答であるが，ここでは，証明項とレベルは一般に省略されて $\vdash p$ の形で出力される．

```
Result:1: |-  ∀ x.(|= S[x])
; この命題を仮定し S-is-unary と名付ける

Result:3: |-  ∀ x. ∀ y.(|= A[x,y])
; この命題を仮定し A-is-binary と名付ける

Result:5: |-  ∀ x. ∀ y.(A[x,y] ⊃ S[x] ⊃ S[y])
; この命題を仮定し PersisProp と名付ける

Result:14: |-  ∀ x.(S[x] ⊃ ∀ z. ∀ y.(A[x,y]
∧ A[x,z] ⊃ ∃ u.(A[y,u] ∧ A[z,u])))
; この命題を仮定し Diamond と名付ける

RPT:21> (defindpred TransA (x y) it X)
pred-name: TransA
pred-body: x=y ∨ ∃ z.(A[x,z] ∧ TransA[z,y])
OK
; 述語 TransA を定義する．

RPT:26> (name BasicProp1 (all-intro it w))
Result:26: |-  ∀ w.TransA[w,w]
; w->*w の証明

RPT:35> (name BasicProp2 (all-intro (all-intro
it y) x))
Result:35: |-  ∀ x. ∀ y.(A[x,y] ⊃ TransA[x,y])
; x->y ⊃ x->*y の証明
```

```
RPT:69> (defpred Prop1 (x y) it)
pred-name: Prop1
pred-body: S[x] ⊃ ∀z.(A[x,z] ⊃ ∃u.(TransA[y,
u] ∧ TransA[z,u]))
OK
; この命題を Prop1 という述語とする．

RPT:80> (name IH (assume (prop-and (prop-pred
A (x z))
(prop-pred Prop1 (z y))) a3))
Result:80: |- A[x,z] ∧ Prop1[z,y]
; Prop1 の証明における帰納法の仮定

RPT:117> (name Proof-of-Prop1)
Result:117: |- ∀x.∀y.(TransA[x,y] ⊃ Prop1[x,
y])

RPT:123> (defpred Prop2 (x y) it)
pred-name: Prop2
pred-body: S[x] ⊃ ∀z.(TransA[x,z] ⊃ ∃u.(
TransA[y,u] ∧ TransA[z,u]))
OK
; この命題を Prop2 という述語とする．

RPT:133> (name IH2 (assume (prop-and (prop-pred
A (x z)) (prop-pred Prop2 (z y))) a13))
Result:133: |- A[x,z] ∧ Prop2[z,y]
; Prop2 の証明における帰納法の仮定

RPT:175> (name Proof-of-Prop2)
Result:175: |- ∀x.∀y.(TransA[x,y] ⊃ Prop2[x,
y])
; 目標の Diamond Property
```

## C 証明から得られた append プログラム

証明から得られた効率の悪い append プログラム $T_{app}$ は以下の通りである．ここでは，$\lambda x.\lambda y.t$ を $\lambda xy.t$ と書く等の省略を用いている．

```
μ(λfxq.
   (λr.if car(r) then
        λyl. [y true 0 . 0]
      else
        λyl.
         [[cadr(r) . car(cddddr(r)yl)]
          false
          cadr(r)
          caddr(r)
          0
          car(cddddr(r)yl)
          cdr(cddddr(r)yl)
          . 0]
        fi)
   [car(q) .
     if car(q) then cdr(q)
     else
       [cadr(q)
        caddr(q)
        cadddr(q) .
        f(caddr(q))(cddddr(q))]
     fi
   ]
  )
```

## D 定理 13の証明

### D.1 準　備

**定義 13** (文脈)文脈 $(context)$ は，穴 $(hole)$ とよばれる新しい定数 $\langle\,\rangle$ を持つ項である．$n$ 個の穴を持つ文脈 $C$ の穴に，項 $a_1,\cdots,a_n$ を代入してできた項を $C\langle a_1,\cdots,a_n\rangle$ または $C\langle\vec{a}\rangle$ と記述する．$C, D$ を同じ個数の穴を持つ文脈とするとき，$C \le D \triangleq C\langle\vec{y}\rangle \le D\langle\vec{y}\rangle$ と定義する．ここで，$\vec{y}$ はフレッシュな変数の列とする．□

**補題 1** $f$ を変数，$c$ を $f$ に関し正格な項, $C$ を文脈, $v$ を正準項, $\theta$ を $f$ 以外の変数に対する代入, $y$ をフレッシュな変数とすると, $C\langle c\theta\rangle \rhd^* v$ ならば，$C\langle y\rangle \rhd^* w$ かつ $v \approx w_y[c\theta]$ となる $v$ と同種の正準項 $w$ が存在する．□

この補題は $c$ の中の $f$ に，具体的な項が代入されない限り、$c$ は本質的に計算に使われないことを示している．証明は容易である．

**補題 2** 項 $F, e$, 変数 $f$, 正準項 $s'$, 代入 $\phi \triangleq \{f := \mu(\lambda f.F)\}$ に対して，$FV(F) = \{f\}$, $e\phi \rhd^* s'$ が成立するとする．このとき，適当な自然数 $m$ および $s'$ と同種の正準項 $v$ に対して、

$$e_f[\mathtt{unfold}(m,f,F)] \rhd^* v$$

かつ

$$s' \approx v\phi$$

が成立する．ただし，

$$\begin{aligned} \mathtt{unfold}(0,f,F) &\triangleq f \\ \mathtt{unfold}(k+1,f,F) &\triangleq F_f[\mathtt{unfold}(k,f,F)] \end{aligned}$$

とする．□

この補題は，簡約化列における $\mu(\lambda f.F)$ の形の項の書き換えの回数に関する帰納法で証明される．

### D.2 証　明

$F, f, x_1, \cdots, x_n, a, b_i^j, c$ を定理の前提条件を満たすものとする．簡単のため，$k = n = 1$ と仮定し、$b_1^1$ を $b$, $x_1$ を $x$ と書くことにする．一般の $k, n$ の場合も全く同様に証明できる．とくに前提条件のうち最後のものは，

$$c_f[F] \quad \sim \quad a(c_x[b], x) \qquad (1)$$

となる．

項の 2 項関係 $R_0, R_1$ を以下で定義する．

$s\ R_0\ t \stackrel{\triangle}{=} \exists C, D : context.\ \exists d : term.$
$$C \leq D \wedge s \approx C\langle (c\phi)_x[d]\rangle \wedge t \approx D\langle G(d)\rangle$$
$s\ R_1\ t \stackrel{\triangle}{=} \exists C, D : context.\ \exists d : term.$
$$C \leq D \wedge s \approx C\langle G(d)\rangle \wedge t \approx D\langle (c\phi)_x[d]\rangle$$

ただし，$\phi$ は、$\phi \stackrel{\triangle}{=} \{f := \mu(\lambda f.F)\}$ という代入をあらわすとする．また，$C, D$ は穴を 1 つ持つ文脈, $d$ は項の上を動くメタ変数である．

もし，$\forall s, t.\ (s\ R_0\ t \supset s\ T[R_0]\ t)$ を言うことができれば、$\leq$ が $T[\cdot]$ に関する最大不動点であることから，$\forall s, t.\ (s\ R_0\ t \supset s \leq t)$ が導かれ，特に，$s, t$ として $c\phi$ と $G(x)$ を取ることにより，$c\phi \leq G(x)$ が示される．同様に，もし，$\forall s, t.\ (s\ R_1\ t \supset s\ T[R_1]\ t)$ を言うことができれば、$G(x) \leq c\phi$ が示され，上と合わせて、定理の結論を得られる．

以下では，$\forall s, t.\ (s\ R_0\ t \supset s\ T[R_0]\ t)$ を示す．

$s, t$ を固定し，$s\ R_0\ t$ と仮定する．定義により，$C \leq D$, $s \approx C\langle c\phi\theta\rangle$, $t \approx D\langle G(d)\rangle$ となる文脈 $C, D$, 項 $d$ が存在する．ここで $\theta \stackrel{\triangle}{=} \{x := d\}$ とする．

$s\ T[R_0]\ t$ を言うため，$s$ が正準形 $s'$ を持つと仮定する．すると補題 2で，$e$ を $C\langle c\theta\rangle$ と置くことにより，$C\langle c\psi\theta\rangle \rhd^* v$ かつ $s' \approx v\phi$ となる正準項 $v$ および自然数 $m$ が存在する．ここで $\psi \stackrel{\triangle}{=} \{f := \mathtt{unfold}(m, f, F)\}$ とおいた．

**Case 1:** $v$ が **nil,true**または**false**のとき

Case 2 とほぼ同様にして証明されるので省略する．

**Case 2:** $v \equiv [v^1 . v^2]$ のとき

このとき，$s \approx [v^1\phi . v^2\phi]$ である．$s^i \stackrel{\triangle}{=} v^i\phi$ と置く．ある項 $t^1, t^2$ に対して，$t \rhd^* [t^1 . t^2]$ かつ $s^i\ R_0\ t^i\ (i = 1, 2)$ が成立することを，$m$ に関する帰納法で証明する．

**Base Case:** $m = 0$ のとき

$\mathtt{unfold}(0, f, F) \equiv f$ より，$C\langle c\theta\rangle \rhd^* [v^1 . v^2]$ となる．$c$ は $f$ に関して正格なので，補題1より $C\langle y\rangle \rhd^* [w^1 . w^2]$ かつ ${w^i}_y[c\theta] \approx v^i$ となる項 $w^1, w^2$ が存在する．このとき $C \leq D$ の仮定より，$D\langle y\rangle \rhd^* [p^1 . p^2]$ かつ $w^i \leq p^i\ (i = 1, 2)$ となる $p_1, p_2$ が存在する．$t^i \stackrel{\triangle}{=} {p^i}_y[G(d)]\ (i = 1, 2)$ とおくと，$t \rhd^* [t^1 . t^2]$ となる．

$s^i$ と $t^i$ の定義および $R_0$ の定義より $s^i\ R_0\ t^i\ (i = 1, 2)$ が成立するので，$s\ T[R_0]\ t$ が言えた．

**Step:** $m = m' + 1 (m' \geq 0)$ のとき

$\psi' \stackrel{\triangle}{=} \{f := \mathtt{unfold}(m', f, F)\}$,

$w \stackrel{\triangle}{=} C\langle (a(c_x[b], x))\psi'\theta\rangle$ とおく．$C\langle c\psi\theta\rangle \equiv C\langle (c_f[F\psi'])\theta\rangle$ と仮定 (1) から，$C\langle c\psi\theta\rangle \sim w$ である．これと，$C\langle c\psi\theta\rangle \rhd^* [v^1 . v^2]$ より、$w \rhd^* [w^1 . w^2]$ かつ $w^i\phi \sim s^i$ となる $w^i\ (i = 1, 2)$ が存在する．

文脈 $C', D'$ を $C' \stackrel{\triangle}{=} C\langle a(\langle\,\rangle, d)\rangle$, $D' \stackrel{\triangle}{=} D\langle a(\langle\,\rangle, d)\rangle$ で定義する．これらは，$C, D$ の穴に $a(\langle\,\rangle, d)$ を代入してできる文脈である．$D\langle G(d)\rangle \approx D\langle a(G(b_x[d]), d)\rangle$ より，帰納法の仮定を文脈 $C'$, $D'$ とそれらの穴に代入する項 $(c\phi)_x[b\theta]$, $G(b\theta)$ に適用することにより，$w\phi\ T[R_0]\ t$ がいえる．これと $s \sim w\phi$ より $s\ T[R_0]\ t$ がいえる．

**Case 3:** $v \equiv \lambda x.v^1$ のとき

Case 2 と同様にして言える．

いずれの Case も $s\ T[R_0]\ t$ が言えることが示された．従って，$\forall s, t.\ (s\ R_0\ t \supset s\ T[R_0]\ t)$ である．

$\forall s, t.\ (s\ R_1\ t \supset s\ T[R_1]\ t)$ も同様にして言える．(定理の証明終わり) □