項書き換えシステムとその応用

外山　芳人[+]　　二木　厚吉[++]

（[+]日本電信電話公社　武蔵野電気通信研究所　　[++]電子技術総合研究所）

1．はじめに

関数的プログラムや代数的仕様のように，等式によって表現されたシステムが広く興味を集めている。これらのシステムにおいては，関数の意味は等式のみによって定められ，その実現方法や計算の詳細な記述を避けることによって簡明な論理構造を得ている。

これらのシステムに計算という操作的意味を付与する最も自然な方法のひとつは，等式を左辺から右辺への書き換え規則とみなすことである。このようにして得られたシステムを，項書き換えシステムという。項書き換えシステムは，等式で表現された論理の世界と書き換え規則で表現された計算の世界を，きわめて自然に結びつけるため，両者の世界にまたがるさまざまな事項をその上で統一的に取り扱うことが可能である。項書き換えシステムのこのような性質は，関数的プログラムのような等式システムの研究に，多くの示唆を与える。

ここでは項書き換えシステムを，理論的な側面と，関数的プログラムの計算モデルとしての応用的側面から紹介する。まず次章では，項書き換えシステムを説明する。3章では，Church-Rosserの性質とその成立条件について述べる。4章では，停止性について述べ，5章では，リダクションの戦術について紹介する。6章では，関数的プログラムへの応用として，項書き換えシステムによるＬＩＳＰインタプリタの記述を紹介する。

2．項書き換えシステム

2.1　再帰的プログラムの計算

再帰的プログラムの計算を例にとり項書き換えシステムの考え方をまず説明する。等式で記述されたシステムの例として階乗関数 $f(n)=n!$ の再帰的プログラムを考える。

$$f(X) = if(zero(X), 1, X \times f(X-1)) \quad [1]$$

ここで $if(A,B,C)$ は条件式 if A then B else C を表わし，$zero(X)$ はXが0のときのみ true となる論理式である。このとき $f(3)=6$ は再帰的プログラムによって次のように計算される。

$$f(3) \rightarrow if(zero(3), 1, 3 \times f(3-1)) \rightarrow 3 \times f(2) \rightarrow 3 \times if(zero(2), 1, 2 \times f(2-1)) \rightarrow 3 \times (2 \times f(1)) \rightarrow \cdots \rightarrow 6$$

この計算においては，等式[1]は左辺から右辺への書き換え規則としてのみもちいられていることに注意されたい。すなわち，計算という非可逆な過程を表わすためには，等式[1]は次のような左辺から右辺への書き換え規則と考えた方が適切である。

$$f(X) \triangleright if(zero(X), 1, X \times f(X-1)) \quad [2]$$

以上のように，システムを記述している等式Ｍ＝Ｎを，書き換え規則Ｍ▷Ｎとみなしたものが，項書き換えシステムである。

2.2 簡単な項書き換えシステムの例

前節の再帰的プログラムでは，＋，－，×，if，zero，の計算は自明なこととして特に述べなかった。しかし，これらもすべて書き換え規則で表わすことができる。たとえば自然数上の加算＋を項書き換えシステムで表わしてみよう。話を簡単にするために，0，1，2，…を0，S(0)，S(S(0))，…と書くことにする。すると加算＋は次の公理$\mathcal{E}_+$をもつ等号論理で表わされる。

$$\mathcal{E}_+:\quad x+0 = x$$
$$x+S(y) = S(x+y)$$

ここで$\mathcal{E}_+$のすべての等式を左辺から右辺への書き換え規則とみなすと，項書き換えシステム$\mathcal{R}_+$が得られる。

$$\mathcal{R}_+:\quad x+0 \rhd x$$
$$x+S(y) \rhd S(x+y)$$

2＋1＝3の計算を$\mathcal{R}_+$の書き換え規則をもちいて行うと次のようになる。

$$S(S(0))+S(0) \to S(S(S(0))+0)$$
$$\to S(S(S(0)))$$

項S(S(S(0)))はこれ以上書き換えることができないので計算の答となる。

同様な方法により，すべての計算可能な関数は項書き換えシステムで表わせることが知られている。したがって計算能力という点に関しては，十分な計算モデルとなっている。

2.3 基本的な用語の説明

項書き換えシステムは有限個の書き換え規則P⊳Qによって定められている。ここでQに出現している変数記号は必ずPにも出現しているものとする。

項Mの中で書き換え規則が適用可能な部分をリデックスという。項Mのリデックスを書き換えることによって項Nが得られるなら，MはNにリダクションされたといいM→Nと書く。Mから0回以上のリダクションによってNに到達できたときにも，同様にMはNにリダクションされたといいM$\overset{*}{\to}$Nと書く。

項Nがリデックスをもたないならば正規形とよぶ。M$\overset{*}{\to}$NでNが正規形なら，Mは正規形Nをもつという。Mの任意の正規形をM↓で表わす。正規形は項書き換えシステムにおける計算結果とみなされる。

M＝Nは項書き換えシステム$\mathcal{R}$のもととなる等号論理$\mathcal{E}$において等式M＝Nが証明されることを意味する。M≡NはMとNが項としてまったく同じ形をしていることを表わす。

項書き換えシステム$\mathcal{R}$において，$M_0 \to M_1 \to M_2 \to \cdots$なる無限のリダクションが存在しないなら，$\mathcal{R}$は停止性をみたすという。

3. Church-Rosser の性質

3.1 Church-Rosser性とその意味

項書き換えシステムがChurch-Rosserの性質をみたすとは，任意の項M，N，PにおいてM$\overset{*}{\to}$N，M$\overset{*}{\to}$Pならば，適当な項Qが存在してN$\overset{*}{\to}$Q，P$\overset{*}{\to}$Qとなることである。[3]（図1）

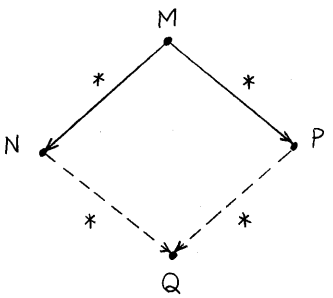


図1. Church-Rosserの性質

項書き換えシステムがChurch-Rosserの性質をみたすなら次の性質をもつ。[10]

〔性質3.1〕

項Mの正規形は高々一個しか存在しない。

〔性質3.2〕

M＝Nなら，適当な項Pが存在して $M \xrightarrow{*} P$，$N \xrightarrow{*} P$となる。

性質3.1はどのようなリダクションによって正規形を求めても，途中のリダクションの道すじとは無関係に唯一の正規形が得られることを保証している。性質3.2は，等号論理$\mathcal{E}$における等式M＝Nの証明を，項書き換えシステム$\mathcal{R}$のリダクションによって行えることを示している。

3.2 Church-Rosser性の成立条件

(A) 停止性をみたす場合

項書き換えシステム$\mathcal{R}$が停止性をみたす場合には，書き換え規則がつくる危険対を調べることでChurch-Rosserの性質を判定できる。まず危険対について説明する。[10]

$\mathcal{R}$の書き換え規則$A \triangleright A'$と$B \triangleright B'$が重なっているとはAとBを部分的に重ね合わせることが可能なことである。このときAとBを重ね合わせて得られる項をMとする。Mは$A \triangleright A'$と$B \triangleright B'$をもちいて，それぞれ$M \to P$，$M \to Q$とリダクションできる。このとき$\langle P, Q \rangle$を危険対という。$\mathcal{R}$が危険対をもつなら，$\mathcal{R}$は重なりをもつという。このとき次の条件が知られている。[14]

〔条件CR.1〕（Knuth, Bendix）

項書き換えシステム$\mathcal{R}$が停止性をみたし，かつ任意の危険対$\langle P, Q \rangle$に対してP↓≡Q↓となるならば，$\mathcal{R}$はChurch-Rosserの性質をみたす。

(B) 停止性をみたさない場合

項書き換えシステム$\mathcal{R}$が停止性をみたさない場合には，Church-Rosserの成立条件は簡単には求まらない。しかし，$\mathcal{R}$が重なりをもたない場合には，簡単な成立条件が知られている。次にこの条件を紹介しよう。

項Mの中に同じ変数が2回以上出現しないなら，Mは線形であるという。項書き換えシステム$\mathcal{R}$の任意の書き換え規則$M \triangleright N$において常にMが線形ならば，$\mathcal{R}$は線形であるという。このとき次の条件が知られている。[10][17]

〔条件CR.2〕（Rosem, Huet）

項書き換えシステム$\mathcal{R}$が重なりをもたず，かつ線形であるならば，$\mathcal{R}$はChurch-Rosserの性質をみたす。

3.3 Knuth-Bendixのアルゴリズム

等号論理$\mathcal{E}$の公理に，$\mathcal{E}$から導かれる等式を付け加えたり，あるいは他の公理から導かれる公理を取り除いたりして新しい等号論理$\tilde{\mathcal{E}}$をつくっても，$\mathcal{E}$と$\tilde{\mathcal{E}}$は同じ意味をもつ。このとき，$\mathcal{E}$から得られる項書き換えシステム$\mathcal{R}$がChurch-Rosserの性質をみたさなくても，上記の変形をくり返すことで新しい等号論理$\tilde{\mathcal{E}}$をつくり，$\tilde{\mathcal{E}}$から得られる$\tilde{\mathcal{R}}$が〔条件CR.1〕をみたすようにできる場合がある。与えられた$\mathcal{E}$からこのような$\tilde{\mathcal{R}}$を求めるためにはKnuth-Bendixのアルゴリズム（正確には半アルゴリズム）をもちいればよい。[14]以下にアルゴリズムを示す。ここで＞はすべての書き換え規則$M \triangleright N$がM＞Nをみたせば，項書き換えシステムの停止性を保証する適当な半順序関係である。

Knuth-Bendixのアルゴリズム

$\mathcal{E}$ ･･･ 等式の集合

$\tilde{\mathcal{R}}$, $\mathcal{R}'$ ･･･ 書き換え規則の集合

初期条件：$\mathcal{E}$は与えられた等号論理の公理，

$\tilde{\mathcal{R}}$は空集合

(1) if $\mathcal{E}=\phi$ then stop … 成功
(2) $\mathcal{E}$の中から任意の $M=N$ を選ぶ。
(3) $\mathcal{E} := \mathcal{E} - \{M=N\}$
(4) $\tilde{\mathcal{R}}$をもちいて $M\downarrow$, $N\downarrow$ を求める。
(5) if $M\downarrow \equiv N\downarrow$ then go to (1)
(6) if $M\downarrow > N\downarrow$ then $M_\ell := M\downarrow$; $M_r := N\downarrow$; go to (9)
(7) if $N\downarrow > M\downarrow$ then $M_\ell := N\downarrow$; $M_r := M\downarrow$; go to (9)
(8) stop … 失敗
(9) $\mathcal{R}' := \{M' \triangleright N' \in \tilde{\mathcal{R}} \mid M'$あるいは$N'$が $M_\ell \triangleright M_r$ で書き換え可能 $\}$
(10) $\tilde{\mathcal{R}} := (\tilde{\mathcal{R}} - \mathcal{R}') \cup \{M_\ell \triangleright M_r\}$
(11) $\mathcal{E} := \mathcal{E} \cup \{M'=N' \mid M' \triangleright N' \in \mathcal{R}'\} \cup \{P=Q \mid \langle P,Q\rangle$は$\tilde{\mathcal{R}}$の危険対$\}$
(12) go to (1)

このアルゴリズムが停止して成功ならば次のことが成立する。[12]
i) $\tilde{\mathcal{R}}$から得られる等号論理 $\tilde{\mathcal{E}}$ は $\mathcal{E}$ と等しい。
ii) $\tilde{\mathcal{R}}$は[条件CR.1]をみたす。
iii) $\tilde{\mathcal{R}}$の任意の書き換え規則 $M \triangleright N$ において，M，N は $\tilde{\mathcal{R}} - \{M \triangleright N\}$ に対して正規形となっている。

4. 停止性

項書き換えシステムの停止性を判定する一般的な方法は存在しない。しかし与えられた特定のシステムの停止性を判定する手法はいくつか提案されている。以下ではこれらの手法を紹介する。

4.1 整礎集合

集合Sとその上の半順序$>$からなる半順序集合$(S, >)$が整礎集合であるとはSの要素の無限列 $s_0 > s_1 > s_2 > \cdots$ が存在しないことである。項書き換えシステムが停止性をみたすためには，項の集合$\mathcal{T}$上に適当な半順序$>$を選び以下を示せば十分である。

(T.1) $(\mathcal{T}, >)$ は整礎集合

(T.2) $M \to N$ なら $M > N$

4.2 整礎写像法

各項を適当な整礎集合の元へ写像することで停止性を示す方法を整礎写像法という。整礎写像法では(T.1)は常に成立しているから(T.2)をみたす適当な写像を定めれば十分である。このような写像の定め方については[5][12][14]を参照されたい。

4.3 単純化順序

項の集合$\mathcal{T}$上の半順序$>$が単純化順序であるとは以下が成立することである。

(S.1) $M > N$ なら $f(\cdots M \cdots) > f(\cdots N \cdots)$
(S.2) $f(\cdots M \cdots) > M$

このとき次の停止定理が成立する。[4]

[停止定理] (Dershowitz)
$>$を単純化順序とする。項書き換えシステム$\mathcal{R}$の任意の書き換え規則 $M \triangleright N$ に対して，その中に出現している変数に任意の項を代入して得られた規則 $\tilde{M} \triangleright \tilde{N}$ が常に $\tilde{M} > \tilde{N}$ をみたすなら，$\mathcal{R}$は停止性をみたす。

4.4 一般帰納的経路順序法

前節の停止定理をもちいることにより，強力な停止性判定法が得られる。以下ではこれを紹介する。

関数記号の集合の上の適当な半順序を$>$とする。このとき項の集合$\mathcal{T}$上の一般帰納的経路順序 $\overset{*}{\gg}$ は以下のように定められる。[4][12] ただし $\overset{*}{\gg}$ は $\overset{*}{>}$ の多重集合順序である。[5]

[一般帰納的経路順序]
項M，Nが $M \overset{*}{\gg} N$ であるとは以下のどれかをみたすことである。

(I) $M \equiv f(M_1 \cdots M_m)$, $N \equiv g(N_1 \cdots N_n)$ の場合
(I-1) $f = g$ の場合 $\{M_1, \cdots, M_m\} \overset{*}{\gg} \{N_1, \cdots, N_n\}$
(I-2) $f > g$ の場合 $\{M\} \overset{*}{\gg} \{N_1, \cdots, N_n\}$

(I-3) 一般の場合
$\{M_1,\cdots,M_m\} \overset{*}{\geq} \{N\}$
(II) $N \equiv x$ の場合
$M \neq x$ でかつ変数 x は M に出現している。

このとき次の停止性判定法が得られる。[4][12]
[一般帰納的経路順序法]
項書き換えシステム $\mathcal{R}$ の任意の書き換え規則 $M \triangleright N$ に対して $M \overset{*}{>} N$ なら，$\mathcal{R}$ は停止性をみたす。

5. リダクションの戦術

5.1 正しさ問題と最適問題

項 M は一般に複数のリデックスをもつから，どのリデックスを書き換えるかによって，得られるリダクションは異なってくる。このとき，リダクションの各ステップで書き換えるべきリデックスを指示するための規則を，リダクションの戦術という。リダクションの戦術に関する問題として次のものが良く知られている。[2][22]

(1) 正しさ問題 (*correctness problem*)
項 M が正規形をもつならば，必ず正規形が得られる戦術は何か。

(2) 最適問題 (*optimality problem*)
項 M が正規形をもつならば，最小の書き換え回数で正規形の得られる戦術は何か。

上記の問題に対する答は重なりをもたない線形項書き換えシステムの場合には，ある程度明らかになっている。一言で述べると次のようになる。

(1) 正しさ問題の答
外側のリデックスを常に書き換える。(最外リダクション)

(2) 最適問題の答
外側のリデックスを常に書き換える。ただし書き換えの際に部分項はコピーせずに，ポインタをもちいて共有する。(コピー無し最外リダクション)

本章では考察の対象を，重なりをもたない線形項書き換えシステムに制限して上記のことを示す。

5.2 最外リダクション

リダクション $M_0 \to M_1 \to M_2 \to \cdots$ の各ステップで常に各 M_i の最外(最内)リデックスのどれかを書き換えるならば最外(最内)リダクションという。最外リダクションは次の重要な性質をもつ。[11][13][16]

[最外リダクション定理]
項 M が正規形 N をもつなら，M から N への最外リダクションが存在する。

5.3 シーケンシャルシステム

項 M のインデックスとは簡単にいえば，正規形を得るためには書き換えが不可欠な，M の最外リデックスの出現位置である。[11][22] たとえば $M \equiv f(A, B, C)$ において，A, B, C を M の最外リデックスとすると，M の正規形を得るためには A の書き換えは不可欠であるが，B, C の書き換えは A の値が *true*, *false* になる場合には必ずしも必要ではない。したがって A の出現位置のみが M のインデックスとなる。

項書き換えシステム $\mathcal{R}$ において，任意の項 M が常にインデックスをもつなら，$\mathcal{R}$ をシーケンシャルシステムという。[11][22] シーケンシャルシステムにおいては，インデックスに出現している最外リデックスを常に書き換えることにより，正規形が存在するなら必ずそこへ到達することができる。[11][22]

5.4 左システム

項書き換えシステム $\mathcal{R}$ の任意の書き換え規則 $M \triangleright N$ において，M に出現している関数記号(定数記号も含む)の

左側に変数が出現しないなら，$\mathcal{R}$を左システムという。[11] 左システムはシーケンシャルシステムであり，最も左側の最外リデックスの出現位置がインデックスとなる。したがって，左システムにおいては，正規形が存在するなら必ず最左最外リダクション（すなわち名前よび規則）によって，それを得ることができる。[11][13][22]

5.5 最適リダクション

最外リダクションをもちいることによって，項Mの正規形が存在するなら必ず求まることはすでに述べた。しかし，最内リダクションと比較すると，最外リダクションは正規形を得るまでの書き換え回数が多くなりやすい。

最外リダクションのこのような欠点を取り除くひとつの方法は，コピー無し規則をもちいて書き換えを行うことである。[2][15][16][18][22] この方法では，書き換えの際に生ずる部分項のコピーをせず，かわりに一個の部分項をポインタによって共有する。たとえば，名前よび規則と，コピー無し名前よび規則によるリダクションの例を次の$\mathcal{R}_{cost}$によって示そう。

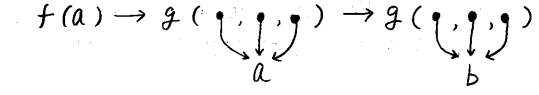
$\mathcal{R}_{cost}$： $f(x) \triangleright g(x,x,x)$
$a \triangleright b$

[名前よび]

$f(a) \to g(a,a,a) \to g(b,a,a) \to g(b,b,a) \to g(b,b,b)$

[コピー無し名前よび]

$f(a) \to g(\bullet,\bullet,\bullet) \to g(\bullet,\bullet,\bullet)$



書き換えをすべてコピー無し規則で行うものとする。このとき項Mが正規形をもつなら，最小の書き換え回数で正規形に致達できるコピー無し最外リダクションが存在する。[2][15][16][18][22] たとえば左システムの場合には，コピー無し名前よびが最適リダクションとなる。

6. 項書き換えシステムによるLISPインタプリタの記述

本章では項書き換えシステムを関数的プログラムへ応用した例として，LISPインタプリタの記述を紹介する。記述はHISP言語をもちいて階層的に行われる。HISP言語については文献[6]~[8]を参照されたい。記述は文献[9]を参考にして，[条件CR.2]をみたすようにした。この結果，得られたオブジェクトが定義する項書き換えシステムはChurch-Rosserの性質をみたしている。

まず特別な名前を表わすオブジェクトSPECIAL∅と通常の名前を表わすNAME∅を定義する。各々の名前は同等性を判定できるようにする。（図2）

```
SPECIAL0 ::
 create
  sub  BOOL /* BOOL is built-in */
  sort Special
  op   'QUOTE', 'LABEL', 'LAMBDA',
       'COND',
       'EQUAL', 'ATOM',
       'CONS', 'CAR', 'CDR': -> Special
       _=sp_: Special,Special -> Bool
  eq
   ( 'QUOTE' =sp 'QUOTE' = true )
   ( 'LABEL' =sp  'LABEL' = true )
   ( 'LAMBDA' =sp 'LAMBDA' = true )
   ( 'COND' =sp 'COND' = true )
   ( 'EQUAL' =sp 'EQUAL' = true )
   ( 'ATOM' =sp 'ATOM' = true )
   ( 'CONS' =sp 'CONS' = true )
   ( 'CAR' =sp 'CAR' = true )
   ( 'CDR' =sp 'CDR' = true )
   ( 'QUOTE' =sp 'LABEL' = false )
   /* ... as expected */
 end

NAME0 ::
 create
  sub  STRING /* STRING is built-in */,
       BOOL
  sort Name
  op   _: String -> Name
       _=nm_: Name,Name -> Bool
  eq var s,t:String
   ( s =nm t = s =string t )
 end
```

図2.

次に Special, Name, Bool の3種類の集合（これらを sort とよぶ）の元をアトムとして，通常の5つの基本演算 cons, car, cdr, atom, equal をもつリスト LISTØ を定義する。（図3）

```
LISTO ::
 create
  sub  SPECIALO, BOOL, NAMEO
  sort List
  op   _: Special -> List
       _: Bool -> List
       _: Name -> List
       nil: -> List
       [ _ _ ]: List, List -> List /*cons*/
       car: List -> List
       cdr: List -> List
       atom: List -> Bool
       equal: List,List -> Bool
  eq var v,w,x,y:List; sp,sp1:Special;
         b,b1:Bool; st,st1:Name
   ( car([ x y ]) = x )
   ( cdr([ x y ]) = y )
   ( atom(nil) = true )
   ( atom(sp) = true )
   ( atom(b) = true )
   ( atom(st) = true )
   ( atom([ x y ]) = false )
   ( equal(nil,nil) = true )
   ( equal(sp,sp1) = sp =sp sp1 )
   ( equal(b,b1)
       = (b and b1) or not(b or b1) )
   ( equal(st,st1) = st =nm st1 )
   ( equal([ v w ],[ x y ])
       = equal(v,x) and equal(w,y) )
 end
```

図3.

さらに，インタプリタの定義に必要な演算 append, pair, assoc が List 上に定義されたオブジェクトを LISTaØとする。（図4）

```
LISTaO ::
 refine LISTO
  op   append: List,List -> List
       pair: List,List -> List
       assoc: Name,List -> List
  eq var s,t:Name; v,w,x,y,z:List
   ( append(nil,y) = y )
   ( append([ w x ],y) = [ w append(x,y) ] )
   ( pair(nil,nil) = nil )
   ( pair([ v w ],[ x y ])
     = [ [ v  x ]  pair(w,y) ] )
   ( assoc(s,[ [ t  x ]  z ])
         = if s =nm t then x
           else assoc(s,z) fi )
 end
```

図4.

最後にインタプリタの本体を構成する4つの演算 evlis, evcon, apply, eval を定義し，オブジェクト LISPØ を得る。（図5）

```
LISPO ::
 refine LISTaO
  op   evlis: List,List -> List
       evcon: List,List -> List
       apply: List,List,List -> List
       eval: List,List -> List
  eq var sp:Special; b:Bool; st:Name;
         v,w,x,y,z:List
   ( evlis(nil,z) = nil )
   ( evlis([ x y ],z)
         = [ eval(x,z) evlis(y,z) ] )
   ( evcon([ [ v [ w x ] ] y ],z)
         = if equal(eval(v,z),true)
           then eval(w,z)
           else evcon(y,z) fi )
   ( eval(sp,z) = sp )
   ( eval(b,z) = b )
   ( eval(st,z) = assoc(st,z) )
   ( eval([ x y ],z) = apply(x,y,z) )
   ( apply(st,y,z) = apply(eval(st,z),y,z) )
   ( apply('COND',x,z) = evcon(x,z) )
   ( apply('CONS',[ v [ w x ] ],z)
              = [ eval(v,z) eval(w,z) ] )
   ( apply('CAR',[ x y ],z) = car(eval(x,z)) )
   ( apply('CDR',[ x y ],z) = cdr(eval(x,z)) )
   ( apply('EQUAL',[ v [ w x ] ],z)
            = equal(eval(v,z),eval(w,z)) )
   ( apply('ATOM',[ x y ],z)
            = atom(eval(x,z)) )
   ( apply('QUOTE',[ x y ],z) = x )
   ( apply([ 'LABEL' [ v [ w x ] ] ],y,z)
         = apply(w,y,[ [ v w ] z ]) )
   ( apply([ 'LAMBDA' [ v [ w x ] ] ],y,z)
     = eval(w,append(pair(v,evlis(y,z)),z)) )
 end
```

図5.

HISP言語で書かれた記述はこのように階層的にモジュール化されているので，それにもとづいて新しい記述（ソフトウェア）を作りやすいという性質をもつ。たとえば，整数もアトムとして許すようなリスプ LISP 1 は図6のように簡単に定義できる。

```
LIST1 ::
 refine LISTO
  sub INT /* INTeger is built-in */
  op  _: Int -> List
  eq var i,j:Int
   ( atom(i) = true )
   ( equal(i,j) = i =int j )
 end

LISP1 :: LISPO($ LISTO <- LIST1 $)
```

図6.

LISP 1によって定義された項書き換えシステムでのリダクションの一例を図7に示す。

```
 term is ...
apply([ 'LABEL'
      [ "FIRSTATOM"
      [ [ 'LAMBDA'
        [ [ "X" nil ]
        [ [ 'COND'
          [ [ [ 'ATOM' [ "X" nil ] ]
            [ "X" nil ] ]
          [ [ true
            [ [ "FIRSTATOM"
              [ [ 'CAR' [ "X" nil ] ] nil ] ]
            nil ] ]
          nil ] ] ]
        nil ] ] ],
      [ [ 'QUOTE'
        [ [ [ "A" [ "B" nil ] ] [ "C" nil ] ]
        nil ] ]
      nil ],
      nil)

 canonical term is ...
"A"

total  = 162 times
```

図7.

7. おわりに

項書き換えシステムの理論を紹介し，その応用としてLISPインタプリタの記述を示した。項書き換えシステムの基本的な概念は非常に簡明であるが，その中に見いだせる多くの性質は，関数的プログラムの構造を考察する際に，大変参考になると考える。

参考文献

[1] Barendregt: "The lambda calculus, its syntax and semantics", North-Holland (1981)

[2] Berry / Levy: "Minimal and optimal computations of recursive programs", J.ACM 26-1 (1979)

[3] Church / Rosser: "Some properties of conversion", Trans. AMS 39 (1936)

[4] Dershowitz: "Orderings for term-rewriting systems", Theor. Comput. Sci. 17 (1982)

[5] Dershowitz / Manna: "Proving termination with multiset orderings", C.ACM 22 (1979)

[6] Futatsugi: "Hierachical software development in HISP", Computer Science & Technologies 1982, Ed. Kitagawa, OHM / North-Holland, (1982)

[7] Futatsugi / Okada: "Specification writing as construction of hierarchically structured clusters of operators", Proc. IFIP (1980)

[8] Futatsugi / Okada: "A hierachical structuring method for functional software systems", Proc. Software Eng. (1982)

[9] Hoffmann / O'Donnell: "Programing with equations", ACM TOPLAS 4-1 (1982)

[10] Huet: "Confluent reduction: abstract properties and applications to term rewriting systems", J. ACM 27-4 (1980)

[11] Huet / Levy: "Call by need computations in non-ambiguous linear term rewriting systems", Rapport Laboria 359, IRIA (1979)

[12] Huet / Oppen: "Equations and rewrite rules: a survey", Formal languages: Perspectives and Open Problems, Ed. Book, Academic Press (1980)

[13] Klop: "Combinatory reduction systems", Dissertation, Univ. of Utrecht (1980)

[14] Knuth / Bendix: "Simple word problems in universal algebras", Computer Problems in Abstract Algebra, Ed. Leech, Pergamon Press (1970)

[15] Levy: "Optimal reductions in lambda-calculus", Essays on Combinatory Logic, Eds. Soldin / Hindley, Academic Press (1980)

[16] O'Donnell: "Computing in systems described by equations", Lecture Notes in Comput. Sci. 58 (1977)

[17] Rosen: "Tree-manipulation systems and Church-Rosser theorems", J.ACM 20 (1973)

[18] Staples: "Optimal evaluations of graph-like expressions", Theor. Comput. Sci. 10 (1980)

[19] 杉山 / 鈴木 / 谷口 / 嵩: "あるクラスの項書き換え系の効率のよい実行", 信学論 J65-D-7 (1982)

[20] 外山: "項書き換えシステムの直和について", 信学技報 AL 82-39 (1982)

[21] 外山: "ラベル付項書き換えシステム", AL82-40 (1982)

[22] Vuillemin: "Correct and Optimal implementation of recursion in a simple programing language", J. Comput. Syst. Sci. 9-3 (1974)