解　説

項書き換え型計算モデルとその応用†

二木厚吉††　外山芳人†††

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近年活発に研究されている．これらのシステムでは等式のみで関数の意味が定められ，関数の実現方法や計算方法の詳細な記述を避けることにより簡明な論理構造を得ている．

等式自身にはもともと計算という意味はない．たとえば，等式 1+2=3 において左辺から右辺を得ることも右辺から左辺を得ることも，論理の世界では同じように考えることができる．一方計算の世界では 1+2 の計算結果として 3 が得られるのであって，3 の計算結果として 1+2 が得られることはない．このように計算の世界からながめた場合には，等式を複雑な式から単純な式へ変換する非可逆な書き換え規則とみなした方が都合がよい．すなわち，等式を左辺の複雑な式から右辺の単純な式への書き換え規則とみなすことによって，等式で記述された論理の世界を自然な形で計算の世界に結びつけることができる．このようにして論理の世界と計算の世界を結びつけることは，きわめて自然であり，プログラムの検証や変換といった論理と計算の 2 つの世界にまたがらざるを得ないような種種の問題に対し，有効なわく組みを提供する．この事情は，論理プログラミングの基礎をなす Horn 節の手続き的解釈に対してもまったく同じである*．

項書き換え型計算モデルは，上述のような，等式を書き換え規則とみなす考え方から発展してきた計算モデルであり，ふつう項書き換えシステムと呼ばれる．本稿では，項書き換えシステムとその応用に関する現在までの研究について解説する．まず 2 で項書き換えシステムとは何かを説明し，3 ではその理論的な性質について述べる．

4 では関数的プログラミングとその周辺に対象をしぼり，項書き換えシステムの応用について述べる．

2. 項書き換えシステムとは？

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．

$$f(x)=\mathrm{if}(\mathrm{zero}(x),1,x\times f(x-1)) \qquad (1)$$

ここで，$\mathrm{if}(A,B,C)$ は条件式 if A then B else C を表わし，$\mathrm{zero}(x)$ は x が 0 のときのみ true(真) となる論理式である．このとき $f(3)$ はこの再帰的プログラムによって以下のように計算されるだろう．

$f(3)\rightarrow\mathrm{if}(\mathrm{zero}(3),1,3\times f(3-1))\rightarrow 3\times f(2)\rightarrow$
$3\times\mathrm{if}(\mathrm{zero}(2),1,2\times f(2-1))\rightarrow 3\times(2\times f(1))\rightarrow\cdots\rightarrow$
$3\times(2\times(1\times f(0)))\rightarrow 3\times(2\times(1\times 1))\rightarrow 6$

この計算において等式(1)は左辺から右辺への書き換え規則としてもちいられていることに注意してほしい．すなわち，与えられた項 $f(3)$ を最終的な結果 6 に徐々に近づけて行くという計算の非可逆的な過程を表現するためには，等式よりも左辺から右辺への書き換え規則(2)の方が適切である．

$$f(x)\triangleright\mathrm{if}(\mathrm{zero}(x),1,x\times f(x-1)) \qquad (2)$$

このように，システムを記述している等式 $M=N$ を左辺から右辺への書き換え規則 (rewrite rule) $M\triangleright N$ とみなしたものが**項書き換えシステム** (term rewriting system) 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．たとえば自然数上での加算＋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, …をそれぞれ 0, $s(0)$, $s(s(0))$, …と書くことにする．すると x の次の自然数は $s(x)$ となるから加算＋は次の

† Term Rewriting Systems and Their Applications: A Survey by Kokichi FUTATSUGI (Electrotechnical Laboratory) and Yoshihito TOYAMA (Musashino Electrical Communication Laboratory, N. T. T.).

†† 電子技術総合研究所ソフトウェア部言語処理研究室
††† 日本電信電話公社武蔵野電気通信研究所基礎研究部第一研究室

* Horn 節の手続き的解釈については，本特集“論理型計算モデル”を参照．

公理 $\mathcal{E}_+$ をもつ**等号論理***(equational logic)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.

$$\mathcal{R}_+:\quad x+0\triangleright x$$
$$x+s(y)\triangleright s(x+y)$$

2+1=3 の計算を $\mathcal{R}_+$ の書き換え規則によっ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はできないから計算の答えとなる.

加算以外の関数も同様にして書き換え規則で表わすことにより,階乗関数 f を定める項書き換えシステム $\mathcal{R}_{\mathrm{fact}}$ が得られる.

$$\mathcal{R}_{\mathrm{fact}}:\quad x+0\triangleright x$$
$$x+s(y)\triangleright s(x+y)$$
$$x-0\triangleright x$$
$$s(x)-s(y)\triangleright x-y$$
$$x\times 0\triangleright 0$$
$$x\times s(y)\triangleright (x\times y)+x$$
$$\mathrm{if}(\mathrm{true},x,y)\triangleright x$$
$$\mathrm{if}(\mathrm{false},x,y)\triangleright y$$
$$\mathrm{zero}(0)\triangleright \mathrm{true}$$
$$\mathrm{zero}(s(x))\triangleright \mathrm{false}$$
$$f(x)\triangleright \mathrm{if}(\mathrm{zero}(x),s(0),x\times f(x-s(0)))$$

$\mathcal{R}_{\mathrm{fact}}$ ではすべての計算が書き換え規則のみで完全に実現されていることに注意されたい.このように項書き換えシステムではすべての計算を有限個の書き換え規則によって定める.有限個の書き換え規則のみでどの程度の計算能力が得られるのか疑問をもたれるかも知れない.実は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,項書き換えシステム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すでに見たように有限個の書き換え規則 $P\triangleright Q$ によって定められる.ここで Q に出現する変数記号は必ず P にも出現しているものとする.この制限は $f(x)\triangleright g(x,y)$ のようなあいまいな書き換え規則を避けるために必要である.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** (redex) とよぶ.たとえば $\mathcal{R}_{\mathrm{fact}}$ における項 $(s(0)+0)\times 0$ のリデックスは,$s(0)+0$ ($x+0\triangleright x$ が適用可能) と $(s(0)+0)\times 0$ ($x\times 0\triangleright 0$ が適用可能)である.項 M のリデックスを書き換えることによって項 N が得られたなら,M は N に**リダクション** (reduction) されたといい,$M\to N$ と書く.M から0回以上のリダクションで N に到達できるときも,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.特に1回のリダクション $M\to N$ を強調したい場合は**1ステップリダクション** (one step reduction) とよぶ.たとえば前述の例においては $(s(0)+0)\times 0\to s(0)\times 0\to 0$ あるいは $(s(0)+0)\times 0\overset{*}{\to}0$,$(s(0)+0)\times 0\to 0$ などのリダクションが存在する.項 N がリデックスをもたないなら**正規形** (normal form)(または**正規項** (normal term))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 $M{\downarrow}$ 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 $\mathcal{E}$ で $M=N$ が証明されることを意味する.$M\equiv N$ 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** (termination property) をみたすという.たとえば $\mathcal{R}_+$ は停止性をみたすが,$\mathcal{R}_{\mathrm{fact}}$ は関数記号 f の無限の書き換えが可能であるから停止性をみたさない.

項書き換えシステムは木の書き換えシステムとみなすこともできる.$\mathcal{R}_{\mathrm{fact}}$ における $f(s(s(0)))\overset{*}{\to}s(s(0))\times f(s(0))$ を**図-1** に示す.点線で囲んだ部分木は書き換えられるリデックスである.

---

* 等号論理の定理は,公理から次の推論規則をもちいて導かれた等式である.

1) $\dfrac{}{M=M}$ (反射律)　2) $\dfrac{M=N}{N=M}$ (対称律)

3) $\dfrac{M=N,\ N=P}{M=P}$ (推移律)　4) $\dfrac{M=N}{f(\cdots M\cdots)=f(\cdots N\cdots)}$ (代入律)

** これは,一般再帰的関数 (general recursive function) の定義法が項書き換えシステムそのものであることから容易に示される.一般再帰的関数については,本特集"関数型計算モデル"を参照.

* $M\equiv N$ は M と N が構文的に等しいこと (syntactical equality) を表わす.したがって,$M\equiv N$ なら $M=N$ であるが,その逆は必ずしも成立しない.たとえば,$s(0)+s(0)=s(s(0))$ であるが,$s(0)+s(0)\not\equiv s(s(0))$.

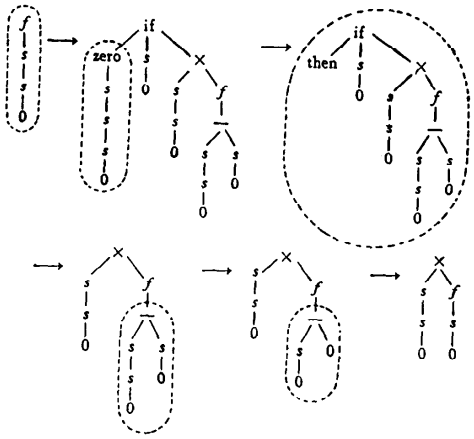


図-1　項書き換えシステムの木表現

3. 項書き換えシステムの性質

ここでは **Church-Rosser の性質**，**停止性**，**リダクションの戦術**を取り上げ，項書き換えシステムの理論的な側面について説明する．

3.1 Church-Rosser の性質

リダクションの合流性

項書き換えシステムでは，与えられた項にどのような順序で書き換え規則を適用してもかまわない．したがって項の中に複数のリデックスが存在する場合には何通りものリダクションが生ずることになる．たとえば $\mathcal{R}_{\text{fact}}$ における項 $(s(0)\times 0)+(s(0)+0)$ のリダクションは**図-2** に示すようになる．

この例では途中の道すじとは無関係に，すべてのリダクションは唯一の正規形 $s(0)$ で合流している．一般の項書き換えシステムでは，このようなリダクションの合流性は必ずしもみたされない．むしろリダクションが異なれば最終的な結果も異なることが多い．**図-2** のようなリダクションの合流性を常に保証するためには，項書き換えシステムが Church-Rosser の性質をみたす必要がある．

Church-Rosser の性質とその意味

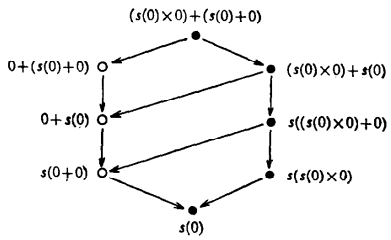


図-2　$\mathcal{R}_{\text{fact}}$ におけるリダクション

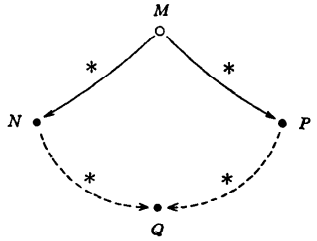


図-3　Church-Rosser の性質

Church-Rosser の性質 (Church-Rosser property) (あるいは合流性 (confluent property)) は λ 計算の重要な性質として Church, Rosser により 1936 年に発見された[9]．この性質は次のように述べることができる．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 $M \xrightarrow{*} N$, $M \xrightarrow{*} P$ ならば，適当な項 Q が存在して $N \xrightarrow{*} Q$, $P \xrightarrow{*} Q$ となることである(**図-3**)．

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 3.1**，**性質 3.2** が成立する[3),30)]．

[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 の意味は M が正規形 N をもつならば，どのようなリダクションで正規形を求めても，それは N に一致するということである．これは実際の計算を行う上で，どのような計算方法で答を求めても常に正しい答が得られるということに対応している．

性質 3.2 は等号論理 $\mathcal{E}$ における $M=N$ 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意味している．すなわち M と N の正規形 $M\downarrow$, $N\downarrow$ を求め，$M\downarrow \equiv N\downarrow$ なら $M=N$ が成立し，$M\downarrow \not\equiv N\downarrow$ なら $M=N$ は成立しないと判定することができる．これは計算機に適した等式の証明方法である．通常の証明が逆戻り（バックトラッキング）による試行錯誤をくり返して行われるのに対し，Church-Rosser の性質をみたす項書き換えシステムによる証明では，リダクションによって正規形を求めるだけでよく，逆戻りはまったく不要となる．

Church-Rosser の性質の成立条件-1 (停止性をみたす場合)

項書き換えシステム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** (critical pair) を調べることで Church-Rosser の性質の判定ができる[30),37)]．

危険対について説明するために簡単なシステム $\mathcal{R}_{ex}$ を考える.

$\mathcal{R}_{ex}$: (1) $f(h(x))\triangleright g(h(x), h(x))$
(2) $h(s(x))\triangleright a$
(3) $f(a)\triangleright g(a, a)$

ここで a は定数である. さて項 $M\equiv f(h(s(a)))$ を考える. M には書き換え規則(1), (2)がどちらも適用可能である. ここでリデックス $h(s(a))$ を書き換え規則(2)をもちいて a に書き換えてみる. すると書き換え規則(1)のリデックスは変形されてしまい, (1)を適用することは不可能となる. この原因は書き換え規則(1), (2)の左辺が関数記号 h の部分で重なっている点にあることは明らかであろう. このように, 左辺が部分的に重なり合う書き換え規則が存在するなら, 項書き換えシステムは**重なりをもつ** (overlapping) という*. ここで左辺を部分的に重ね合わせて得られる項の中でもっとも一般的な項 M を考える**. たとえば前述の例では, 書き換え規則(1)と(2)の左辺を部分的に重ね合わせて得られる項は, すべて項 $f(h(s(x)))$ の変数 x に適当な項を代入した形になっている. したがって $M\equiv f(h(s(x)))$ となる. この M にそれぞれの書き換え規則を適用することで $M\rightarrow P, M\rightarrow Q$ が得られるとき, $\langle P, Q\rangle$ を**危険対**という***. 前述の例では危険対は $\langle f(a), g(h(s(x)), h(s(x)))\rangle$ である. 項書き換えシステムの危険対は高々有限個しか存在しないことに注意されたい. このとき次の十分条件が知られている[30),37)]

[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 $P{\downarrow}\equiv Q{\downarrow}$ となるならば, $\mathcal{R}$ は Church-Rosser の性質をみたす.

前述の $\mathcal{R}_{ex}$ は停止性をみたしている. また $\mathcal{R}_{ex}$ の危険対は $\langle f(a), g(h(s(x)), h(s(x)))\rangle$ のみである. このとき $f(a){\downarrow}\equiv g(a, a)\equiv g(h(s(x)), h(s(x))){\downarrow}$ であるから $\mathcal{R}_{ex}$ は Church-Rosser の性質をみたしている. また **2.2** の $\mathcal{R}_{+}$ も停止性をみたしている. この場合には危険対が存在しないから, 当然 **条件 CR. 1** は成立する. したがって $\mathcal{R}_{+}$ も Church-Rosser の性質をみたしている.

Church-Rosser の性質の成立条件-2 (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 の性質の成立条件は簡単には求まらない. しかし, $\mathcal{R}$ の書き換え規則に**重なりがない** (nonoverlapping) 場合に制限すると簡単な成立条件が知られている. 以下ではこの条件を紹介する.

項 M が**線形** (linear) であるとは, M の中に同じ変数が2回以上出現しないことである. 項書き換えシステム $\mathcal{R}$ の任意の書き換え規則 $M\triangleright N$ において常に M が線形ならば, $\mathcal{R}$ は**線形** (あるいは**左線形** (left linear)) であるという. このとき次の十分条件が得られる[30),49)].

[条件 CR. 2] (Rosen, Huet)
項書き換えシステム $\mathcal{R}$ が重なりをもたず, かつ線形ならば, $\mathcal{R}$ は Church-Rosser の性質をみたす.

2.2 の $\mathcal{R}_{fact}$ は停止性をみたさないが, 重なりをもたず, 線形であるから Church-Rosser の性質をみたしている. 再帰的プログラムの多くは **条件 CR. 2** をもちいて Church-Rosser の性質を導くことができる.

$\mathcal{R}$ が重なりをもつ場合, あるいは非線形の場合の成立条件については文献 30), 36), 43), 49), 56), 57) を参照されたい.

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. 与えられた等号論理 $\mathcal{E}$ から得られる項書き換えシステム $\mathcal{R}$ が Church-Rosser の性質をみたさないとき, 上記の変形をくり返すことによって新しい等号論理 $\tilde{\mathcal{E}}$ 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. このときには, 等式 $M=N$ を $\mathcal{E}$ の上で証明するかわりに, $\tilde{\mathcal{R}}$ の上で **性質 3.2** をもちいてはるかに簡単に証明することができる.

与えられた $\mathcal{E}$ から上記の $\tilde{\mathcal{R}}$ を求めるためには次の **Knuth-Bendix のアルゴリズム** (Knuth-Bendix Algorithm) をもちいればよい[31),37)]. **図-4** にアルゴリズムを示す. (このアルゴリズムは停止しない場合があるので正確には半アルゴリズムである.) 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

* 2つの書き換え規則 $A\triangleright A'$, $B\triangleright B'$ において, A の部分項 C と B が単一化可能 (unifiable) なことである. もし, 2つの書き換え規則が同じ場合には, C として A の真部分項のみを考えるとする.

** 前脚注の C と B の最も一般的な単一化作用素を θ とすると, $M\equiv A\theta$ である. このとき $B\theta$ は M の部分項となっている.

*** 前脚注における M を $B\triangleright B'$ で書き換えた項を P, $A\triangleright A'$ で書き換えた左項を Q とすると, $\langle P, Q\rangle$ が危険対である.

Knuth-Bendix アルゴリズム
$\mathcal{E}$…等式の集合
$\mathcal{R}$, $\mathcal{R}'$…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，
$\mathcal{R}$は空集合
(1) if $\mathcal{E}=\phi$ then stop…成功
(2) $\mathcal{E}$ の中から任意の等式 $M=N$ を選ぶ．
(3) $\mathcal{E}=\mathcal{E}-\{M=N\}$
(4) $\mathcal{R}$ をもちいて $M\downarrow$, $N\downarrow$ を求める．
(5) if $M\downarrow\equiv N\downarrow$ then goto(1)
(6) if $M\downarrow>N\downarrow$ then $M_l=M\downarrow$; $M_r=N\downarrow$; goto(9)
(7) if $N\downarrow>M\downarrow$ then $M_l=N\downarrow$; $M_r=M\downarrow$; goto(9)
(8) stop…失敗
(9) $\mathcal{R}'=\{M'\triangleright N'\in\mathcal{R}\mid M'$ あるい N' が $M_l\triangleright M_r$ で書き換え可能$\}$
(10) $\mathcal{R}=(\mathcal{R}-\mathcal{R}')\cup\{M_l\triangleright M_r\}$
(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) goto(1)

図-4　Knuth-Bendix のアルゴリズム

(このアルゴリズムは文献 34) を参考にした．ただし，(5)～(7)をつけ加えて本来の Knuth, Bendix のアルゴリズムに近い形に直してある．)

な半順序関係である．

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．

i) 等号論理 $\mathcal{E}$ と，$\tilde{\mathcal{R}}$ から得られる等号論理 $\tilde{\mathcal{E}}$ は等しい

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，項 M, N は $\tilde{\mathcal{R}}-\{M\triangleright N\}$ に対して正規形となる．

次の等号論理 $\mathcal{E}_{\text{list}}$ はリスト処理を定めている．ここで $\mathcal{E}_{\text{list}}$ から得られる書き換えシステム $\mathcal{R}_{\text{list}}$ が **条件 CR.1** をみたさないことは，書き換え規則(4)と(5)* から得られる危険対 $\langle$rev(append(rev(y), cons(x, nil))), cons(x, y)$\rangle$ を考えれば明らかである．

$\mathcal{E}_{\text{list}}$:
(1) append(nil, x)$=x$
(2) append(cons(x, y), z)$=$ cons(x, append(y, z))
(3) rev(nil)$=$nil
(4) rev(cons(x, y))$=$ append(rev(y), cons (x, nil))
(5) rev(rev(x))$=x$

$\mathcal{E}_{\text{list}}$ から $\tilde{\mathcal{R}}_{\text{list}}$ を Knuth-Bendix アルゴリズムをもちいて求めると以下のようになる．前述した **i**) **ii**) **iii**) が成立していることに注意されたい．

* 等式(4)，(5)から得られる書き換え規則の意味．以後，文脈から明らかな場合には，いちいち断わらない．

$\tilde{\mathcal{R}}_{\text{list}}$:
(1) append(nil, x)$\triangleright x$
(2) append(cons(x, y), z)$\triangleright$ cons(x, append(y, z))
(3) rev(nil)$\triangleright$nil
(4) rev(cons(x, y))$\triangleright$ append(rev(y), cons(x, nil))
(5) rev(rev(x))$\triangleright x$
(6) rev(append(x, cons(y, nil)))$\triangleright$ cons(y, rev(x))

Knuth-Bendix のアルゴリズムの応用としては，定理の自動証明[37),46)]．抽象データ構造の無矛盾性の自動検証[21),32)]，論理プログラムの性質の検証[2)] などがある．またアルゴリズムの拡張，改良については文献31)，46)を参照されたい．

3.2 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項書き換え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．

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** (well-founded set) 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．たとえば自然数の集合 $N=\{0,1,2,\cdots\}$ は普通の意味での大きさの順序づけを行うと整礎集合となる．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 $M>N$.

整礎写像法

項書き換えシステム $\mathcal{R}_{\text{asso}}$ を考える．

$\mathcal{R}_{\text{asso}}$: $(x\times y)\times z\triangleright x\times(y\times z)$

これは積の計算順序を結合則によって右側から計算する式に変形する書き換え規則である．たとえば，

$$(a\times b)\times((c\times d)\times e)\xrightarrow{*}a\times(b\times(c\times(d\times e)))$$

となる．このシステムの停止性を示すために，各項Mに対して自然数 $w(M)$ を対応させる[14)]．

$$w(M)=\begin{cases}2\cdot w(P)+w(Q)\cdots M\equiv P\times Q\text{ の場合}\\ 1\cdots\text{それ以外の場合}\end{cases}$$

たとえば $w((a\times b)\times((c\times d)\times e))=13$ となる．ここで項の集合 $\mathcal{T}$ 上での順序を $M>N\Leftrightarrow w(M)>w(N)$

によって定める．すると，この順序付けは(T 1)(T 2)をみたすことが示される．よって $\mathcal{R}_{asso}$ は停止性をみたす．

以上のように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** (wellfounded mapping method) という．整礎写像法では常に(T 1)は成立しているから，(T 2) をみたす適当な写像を定めれば十分である．さまざまな写像の定め方については文献 14), 34), 37) を参照されたい．

単純化順序

項書き換えシステムが複雑になると，整礎写像法をもちいて直接的に停止性を示すことは一般に困難である．この場合には項の木構造に注目した間接的な方法をもちいた方がよい．まず一般的な停止定理を示す．

項の集合 $\mathcal{T}$ 上の半順序 $>$ が**単純化順序** (simplification ordering) であるとは (S 1), (S 2) が成立することである．

(S 1) $M>N$ ならば $f(\cdots, M, \cdots)>f(\cdots, N, \cdots)$

(S 2) $f(\cdots, M, \cdots)>M$

このとき次のことが成立する[13]．

[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 $\tilde{M}>\tilde{N}$ をみたすならば，$\mathcal{R}$ は停止性をみたす．

多重集合順序

停止定理をみたす単純化順序の一般的な構成法として，**一般帰納的経路順序** (generalized recursive path ordering) が知られている[13),34)]．ここではまず一般帰納的経路順序を定めるために必要となる**多重集合順序** (multiset ordering) について説明する[13)]．

多重集合 (multiset) は集合と似た概念であるが，集合の中に同じ要素が2個以上出現することも許される．たとえば集合では $\{1,2\}=\{1,1,2\}$ であるが，多重集合では $\{1,2\}\neq\{1,1,2\}$ である．集合 S の要素からつくられるすべての有限多重集合の集合を $\mathcal{M}(S)$ と書く．半順序集合 $(S,>)$ に対して $\mathcal{M}(S)$ 上の多重集合順序 $\gg$ を次のように定める．

多重集合 A, B が $A\gg B$ となるのは，多重集合 X, $Y(X\neq\phi)$ が存在して(1)，(2)をみたすことである．

(1) $B=(A-X)\cup Y$

(2) $\forall y\in Y,\ \exists x\in X,\ x>y$

たとえば自然数の多重集合において $\{3,1\}\gg\{2,2,1,1\}$ となることは $X=\{3\}$, $Y=\{2,2,1\}$ とおけば明らかである．

一般帰納的経路順序法

関数記号の集合の上の適当な半順序を $>$ とする．このとき項の集合の上の一般帰納的経路順序 $\overset{*}{>}$ は以下のように定められる[13),34)]．ただし $\overset{*}{\gg}$ は $>$ の多重集合順序である．

[一般帰納的経路順序]

項 M, N が $M\overset{*}{>}N$ 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) $f\not\geq g$ の場合 $\{M_1,\cdots,M_m\}\overset{*}{\geqq}\{N\}$

(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．

一般帰納的経路順序 $\overset{*}{>}$ は単純化順序であり，停止定理をもちいて次の停止性判定法が得られる[13),34)]．

[一般帰納的経路順序法] (generalized recursive path ordering method)

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 $M\overset{*}{>}N$ ならば，$\mathcal{R}$ は停止性をみたす．

次の項書き換えシステム $\mathcal{R}_{bool}$ の停止性は，関数記号 $\neg$, $\wedge$, $\vee$ に関して $\neg>\wedge>\vee$ なる順序付けを行えば一般帰納的経路順序法によって容易に示すことができる[13)]．

$\mathcal{R}_{bool}$:
(1) $\neg\neg x\triangleright x$
(2) $\neg(x\vee y)\triangleright\neg x\wedge\neg y$
(3) $\neg(x\wedge y)\triangleright\neg x\vee\neg y$
(4) $x\wedge(y\vee z)\triangleright(x\wedge y)\vee(x\wedge z)$
(5) $(x\vee y)\wedge z\triangleright(x\wedge z)\vee(y\wedge z)$

項書き換えシステムの停止性を示す他の方法については文献 12), 34), 35), 47) を参照されたい．

3.3 リダクションの戦術

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，書き換えられるリデックスが異なればその結果得られるリダクションも異なってくる．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** (reduction strategy) 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[5),59)].

（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が得られる戦術は何か．

上記の問題に対する答は，重なりのない線形項書き換えシステムに制限した場合には，ある程度わかっている．一言で述べると次の戦術が答えである．

（1） **正しさ問題の答**（最外リダクション）

外側のリデックスを常に書き換える．

（2） **最適問題の答**（コピー無し最外リダクション）

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．

同様な戦術は再帰的プログラム，λ 計算にも有効であることが知られている[3),5),15),36),38),39),59)]．

以下では比較的研究の進んでいる重なりのない線形項書き換えシステムに制限して，正しさ問題と最適問題を考える．それ以外の項書き換えシステムに対して，本節の内容は成立しない場合があることを注意されたい．

最外リダクションと最内リダクション

項 M のリデックス A が他のリデックス B の部分項となっていないなら，A を**最外リデックス** (outermost redex) という．逆にリデックス A の中に他のリデックス B が部分項として出現していないなら，A を**最内リデックス** (innermost redex) という．

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 M_i の最外（最内）リデックスのどれかが書き換えられるならば**最外（最内）リダクション** (outermost (innermost) reduction) という．最外リダクションは次の重要な性質をもつ[33),36),43)]．

［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．

シーケンシャル・システム

項 M の正規形を得るためには最外リダクションを行えばよい．しかし，リダクションの各ステップで，最外リデックスが常に一個とは限らない．この場合には複数の最外リデックスの中から書き換えるべき最外リデックスを指示する必要がある．このために，**インデックス** (index) という概念を導入する[33),59)]．

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*．

たとえば，項書き換えシステム $\mathcal{R}_{\text{fact}}$ において $M \equiv \text{if}(A, B, C)$ を考える．ここで，A, B, C は最外リデックスとする．もし M が正規形をもつなら，正規形を得るためには A を必ず書き換える必要がある．しかし B, C は，A の値が true あるいは false になるなら，どちらか一方を書き換えなくても M の正規形が得られる．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，正規形でない任意の項 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** (sequential system) という**．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[33),59)]．

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 $M_l \triangleright M_r$ において，M_l 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** (left sytem) という[33),36)]．左システムはシーケンシャル・システムとなる．さらに最も左の最外リデックスの出現位置は常に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，**最左最外リダクション** (leftmost outermost reduction) により，必ずそれを得ることができる[33),36),43)]．

次の $\mathcal{R}_{\text{if}}$ は代表的な左システムである．

$\mathcal{R}_{\text{if}}$:　$\text{if}(\text{true}, x, y) \triangleright x$
　　$\text{if}(\text{false}, x, y) \triangleright y$

名前よび，値よび，必須よび

ALGOL の手続き文におけるパラメータの引き渡し方式に対応させて，**最左最外リダクション**を**名前よび (call by name)**，**最左最内リダクション** (leftmost innermost reduction) を**値よび (call by value)** という．答えが存在するなら，値よびでは答が求まらな

* Scott の位相を使わずに，インデックスの正確な定義を述べることは難しい．ここでは，多少不正確であるが，直観的にわかり易い説明にとどめる．詳しくは文献を参照されたい．

** シーケンシャル・システムではインデックスを順番に計算することで必ず答が得られるから，本質的には並列計算が不要なシステムである．シーケンシャル・システムでない場合（たとえば並列論理和のように，順序を付けて計算したのでは収束しない可能性のある場合）には，答を得るために複数のリデックスのリダクションを並列して行うことが必要になる．

い場合でも，名前よびなら答が必ず求まるという良く知られた結果は，対象とする計算システムが左システムとなっているからにほかならない．たとえば再帰プログラムにおいては，条件文の計算が左システム $\mathcal{R}_{if}$ によって定められるために，名前よびが**不動点計算規則** (fixed point computation rule) となっている[39]．

シーケンシャル・システムのインデックスに出現する最外リデックスのように，項 M が正規形へ到達するためには必ず書き換える必要のあるリデックス* を**必須リデックス** (needed redex) という[33),38]．リダクションの各ステップにおいて必須リデックスのみを書き換えるなら**必須よび (Call by need)** という**．正規形が存在するなら，必須よびによって必ずそれを求めることができる．必須よびは，具体的に書き換えるべきリデックスを指示する戦術ではないが，リダクションの戦術を統一的に考察する場合に役立つ概念である．

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り，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，最外リダクションは正規形を得るまでの書き換え回数が大きくなり易い．たとえば次の例を考えてみよう．

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

このとき項 $f(a)$ の正規形を名前よびと値よびで求める．

[名前よび]

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

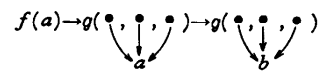
[値よび]

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

この例では，リデックス a を複数コピーしてしまうために，名前よびの書き換え回数は，値よびと比較して大きくなってしまう．

最外リダクションの上記の欠点を取り除くひとつの方法は，**コピー無し規則** (noncopying rule) をもちいて書き換えを行うことである．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．たとえば前述の例を**コピー無し名前よび** (noncopying call by name)* によってリダクションすると次のようになる．

[コピー無し名前よび]



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到達できる**コピー無し最外リダクション** (noncopying outermost reduction) が存在する[5),38),45),50)-52),59]．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．

コピー無し規則は項の書き換えというよりもグラフの書き換えとなっていることに注意してほしい．最外リダクションの振舞いから明らかなように，項書き換えシステムのわく組の中では，最適問題の答は得ることができない．したがって，最適問題を解くためには，コピー無し規則のような**グラフ書き換えシステム (グラフ・リダクション・システム)** まで，前述したような形に問題を拡張して考える必要がある．

コピー無し規則のようなグラフ書き換えシステムは，計算機上でリダクション・システムを実現する場合に広く使われている[29),54),58]．

4. 項書き換えシステムの応用

今まで見てきた通り，項書き換えシステムは等式で記述されたシステムに構成的／操作的な意味を与える簡明な計算モデルである．等式（再帰方程式系）で記述されたシステムに形式的で厳密な意味を与える方法としては，いわゆる Scott 流の不動点 (fixed point) を用いる方法（たとえば Manna[39]の 5 章を参照）や，抽象データ型の代数的仕様記述の意味定義法として最近注目されている始代数 (initial algebra) や終代数 (final algebra) を用いる方法[22),23),60]などもあり，計算機科学の中でも比較的整備が進んでいる分野である．

しかし，不動点や始／終代数に基づく意味定義が非構成的になり易いのに較べ，項書き換えシステムに基づく意味定義はより構成的であり，次のような特徴を持つといえる．

（1） 必要とされる数学的予備概念が少なく，意味

* 必ず書き換える必要のあるリデックスとは，「そのリデックス自身か，あるいは，そのリデックスがリダクションによって変型されたもの（リデックスの剰余）の中のどれかを，必ず書き換える必要がある．」という意味である．文献を参照のこと．

** call by need という術語を，研究者によっては，次節で述べるコピー無し名前呼びの意味でもちいることがあるので注意されたい．

* コピー無し名前よびは，遅延規則 (delay rule) とよばれることがある[59]．

定義自体が簡潔である．

（2） 3で見たとおり，実義される関数の種々の性質（値の唯一性，停止性など）を具体的に調べるアルゴリズムがよく研究されており，また時間／空間計算量を議論する基礎となり得る．

（3） LISP でのリスト処理および PROLOG での項の処理などで蓄積された種々の技法を形式化して取り込むことにより，それに基づく仕様記述法，プログラミング技法を比較的容易に整備，体系化できることが期待される．

以下では，いわゆる関数的 (functional/applicative) プログラミングとその周辺に対象をしぼり，ソフトウェア作成のいろいろな局面において項書き換えシステムを基礎とする研究の現状を概観し，その可能性を展望することにする．

4.1 等式による仕様記述

等式のみを用いてソフトウェアの仕様を記述しようとする試みは，いわゆる代数的仕様記述法に代表されるものであり，近年，形式的仕様記述法としての有効性が注目され，多くの研究が行われている[7),17)~19),20)~28),40)~42),53),55)]．

ここでは，代数的仕様記述法の一例として，停止性と Church-Russer 性をともに持つ項書き換えシステム（これを**収束性を持つシステム**という）に基づく仕様記述法[40),42)]を，簡単な例を用いて説明する．

キュー (Queue : First-In-First-Out List) を形式的に定義したい（仕様記述したい）とする．簡単のためにキューに入れられる要素は整数とし，整数の等価性判定演算＝int とブール値 (true, false) は自由に使ってよいとする．

代数的仕様記述法では，仕様記述すべき対象の基本構造を，

・ **ソート (sort)** と呼ばれるいく種類かの集合の集まりと，

・ それら集合の間に定義された**演算 (operator)** の集まり，

として捉える．この基本構造が**多ソート代数系**のそれであることが，このような仕様記述法が代数的と呼ばれるゆえんである．

キューに対するこの基本構造は**図-5** で表わされる．ここで，丸はソートを表わし，それぞれブール値の集合 Bool，キューの集合 Queue，整数の集合 Int を示している．また，一般に複数本の尾を持つ矢印で演算を表し，その尾で各演算の引数のソースを示し，その頭で各演算の値のソートを示している．この基本構造は通常ソートと演算を次のように宣言することで記述され，**シグネチャ** (signature) と呼ばれる．

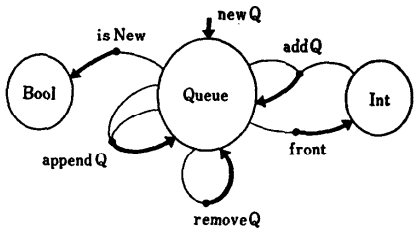


図-5 Queue の基本構造（シグネチャ）
（記法は Goguen[23)] に従った）

sorts

Queue, Int, Bool

operators

newQ : —〉 Queue
addQ : Int, Queue —〉 Queue
removeQ : Queue —〉 Queue
front : Queue —〉 Int
isNew : Queue —〉 Bool
appendQ : Queue, Queue —〉 Queue

このように宣言された，removeQ, appendQ, front, isNew といった演算の働きを示せれば，キューが仕様記述できたと考えられる．これは次のような等式システム* で記述できる．

equations var $q1, q2$: Queue ; i : Int
isNew(newQ)＝true
isNew(addQ($i, q1$))＝false
removeQ(newQ)＝newQ
removeQ(addQ($i, q1$))＝
　　if(isNew($q1$)) then newQ
　　else (addQ(i, (removeQ($q1$)))) fi
front(newQ)＝0
front(addQ ($i, q1$))＝
　　if (isNew($q1$)) then i
　　else (front($q1$)) fi
appendQ(newQ, $q1$)＝$q1$
appendQ(addQ($i, q1$), $q2$)＝
　　addQ (i, appendQ($q1, q2$))

ここで，**var** 以下は変数宣言であり，変数 $q1, q2, i$ のソートを指定している．さらに各等式について，各

* 2.2 の等号論理に対応する．以下等式で記述されたシステムをこのように総称する．

演算の引数に先に宣言した正しいソートの値が入っており，等式の両辺の値のソートは等しい，という意味でソートの不整合（いわゆるタイプのミスマッチ）は存在しない．

この等式システムが収束性を持つ項書き換えシステムになっていることは容易に示せる．このことから，この仕様（項書き換えシステム）は次のような性質を持つことが分かる．すなわち，ソート Queue のすべての元は $i_1, \cdots, i_n$: Int に対して

addQ(i_n, addQ(i_{n-1}, $\cdots$addQ(i_1, newQ)$\cdots$))

の形をした正規項のいずれか（$n \geq 0$）に等しい，ということである．したがって，すべての q: Quene に対してある性質 PROP(q)が成立することを証明するためには，

(i) PROP(newQ) が成り立つ，

(ii) $\forall q$: Queue, i: Int,

PROP(q)⇒PROP(addQ (i, q)) が成り立つ，

の2つを示せばよい．たとえば，$q1, q2, q3$: Queue として，

appendQ(appendQ($q1$, $q2$), $q3$))

=appendQ(appendQ($q1$, $q2$), $q3$)

が成立する（appendQ が結合律を満たす）ことをこのようにして示すことができる．この仕様における newQ と addQ のように，それらの演算のみから構成された項が正規形となるような演算は**構成子** (constructor) と呼ばれ，特に区別されるのが普通である．

収束性を持つ項書き換えシステムでいろいろなソフトウェアの仕様記述を行うことは必ずしも易しくないが，出来上がった仕様は検証し易いといったよい性質を多く持つ．たとえば，このような項書き換えシステムでは，上述の，appendQ が結合律を満たすといった，一般には帰納法を用いなければ証明できない性質を，**3.1** で説明した Knuth-Bendix アルゴリズムを用いて帰納法を使わずに証明できることが知られている*．

代数的方法に基づく仕様記述システムで，検証システムを含み，実際的なシステムの仕様記述に使われているのは，収束性を持つ項書き換えシステムを基礎とする AFFIRM[20),42),55)] だけである．しかし，いわゆる代数的仕様記述の中では，収束性を持つシステムに限定する立場はかなり制限がきついといえる．仕様がそれに続くべき実動化のための原器としての形式性を保ち，かつ実動化への各ステップの正しさの検証が複雑になりすぎない範囲で，この制限を緩和していくのが今後の課題であろう．

4.2 等式に基づく変換プログラミング

仕様に定形的な変換を繰り返し施すことにより目的とするプログラムを得ようとするプログラミング法を，**変換プログラミング**と呼ぶ．変換プログラミングに関する研究は数多く存在する[1),4),6),16)]が，Burstall & Darlington[6)] に始まる等式システム（再帰方程式系）に基づく方法は，比較的単純な変換を組み合わせるだけで，多くの有用な変換が行えることが特徴であり[10),11)]，関数／論理型などと称される将来のプログラミング・システムにおいて重要な役割を果たすと考えられる．

ここでは，この等式に基づく変換プログラミングを項書き換えシステムの応用という立場から眺めてみる．簡単な例を用いて説明する．（整数の）リストを，nil と cons を構成子とする項で表わすとする．（記法は **4.1** の代数的仕様記述法に従う．）

sorts

List, Int

operators

nil : —〉 List

cons : Int, List —〉 List

List の長さを値とする演算：

operator

length : List —〉 Int

は，

equations var l: List ; i: Int

length(nil)=0 (1)

length(cons(i, l))=1+length(l) (2)

で定義され，2つのリストをつないだリストを作る演算：

operator

append : List, List —〉 List

は，

equations var $l1, l2$ List ; i: Int

append(nil, $l2$)=$l2$ (3)

append(cons(i, l1), $l2$)

=cons(i, append($l1$, $l2$))

---

* 等号論理 $\mathcal{E}$ の中に証明したい等式 $M=N$ を公理として付け加えて $\mathcal{E}'$ を作る．このとき $\mathcal{E}'$ から Knuth-Bendix アルゴリズムによって Church-Rosser の性質をみたす項書き換えシステム $\mathcal{R}$ が構成でき，$\mathcal{R}$ が適当な条件をみたすならば，$M=N$ は $\mathcal{E}$ の中でも成立することが示される．詳しくは文献 21), 32), 41) などを参照されたい．

(4)

で定義される.

2つのリストをつないで得られるリストの長さを計算する演算:

operator

lengthOf 2 : List, List —〉 Int

対するプログラムを作りたいとしよう. 明らかに,

equation var l1, l2 : List

lingthOf 2(l1, l2)＝length(append (*l*1, *l*2))

(5)

であるが, これは定義そのものを書き下した仕様でありプログラムとしては効率が悪い. これを改善して効率的なプログラムを作るのが目的である.

(1)〜(5)の等式システムは, 収束性を持つ項書き換えシステムになっている. このシステムの“意味”, つまり各項に対し唯一定まる正規形, を変えることなく適当な等式を付加していくことで, 効率のよいプログラムを求めてみよう.

まず, (5)の $l1$ を nil で具体化して, (3)を使うと,

lengthOf 2(nil, *l*2)＝length(*l*2)　　(6)

を得る. さらに, (5)で $l1$ を cons(i, $l1$) に具体化して(4)を使うと,

lengthOf 2(cons(*i*, *l*1), *l*2)
＝length(cons(*i*, append(*l*1, *l*2)))

となり, さらに(2)を使うと,

lengthOf 2(cons(*i*, *l*1), *l*2)
＝1＋length(append(*l*1, *l*2))

を得る. 最後に, この等式の右辺を(5)を使ってlengthOf 2 で表わせば,

lengthOf 2(cons(*i*, *l*1), *l*2)
＝1＋lengthOf 2(*l*1, *l*2)　　(7)

を得る.

(6), (7)は lengthOf 2 に対して成り立つ等式であると同時に, (5)で定義されたその値を計算する効率のよいプログラムになっている.

Darlington らは, 上述のような変換プログラミングが, 次のような簡単な規則に従って新しい等式を適当に作り出していくことで行えることを, いくつかの重要な例について示している.

(i) **定義**: 補助演算などを, 新しい等式を導入して, 定義する. (上記(5))

(ii) **具体化**: すでにある等式の変数に適当な定数項を入れた新しい等式を加える.

(iii) **展開**: $E＝E'$ と $F＝F'$ をすでにある等式とする. E を具体化した項が F' の部分項として現われており, それを, E を同様に具体化した項で置き換えて得られる項が F'' のとき, $F＝F''$ を新しい等式として加える. (上記(6))

(iv) **たたみ込み**: $E＝E'$ と $F＝F'$ をすでにある等式とする. E' を具体化した項が F' の部分項として現われており, それを, E を同様に具体化した項で置き換えて得られる項が F'' のとき, $F＝F''$ を新しい等式として加える. (上記(7))

(v) **抽象化**: すでにある等式 $E＝E'$ から次のような **where** 定義を右辺に持つ等式を作り出して付け加える.

$E＝E'[c_1/F_1]$ **where** c_1 **is** F_1

ここで, $E'[c_1/F_1]$ は E' 中に現われるすべての部分項 F_1 を c_1 で置き換えて得られる項であり, c_1 は新たに作られた定数である.

(vi) **法則の適用**: 成り立つことが分かっている法則, (組み込み演算の性質など) を, すでに存在する等式の右辺に適用して得られる新しい等式を使け加える.

これらの規則の内(iv), (vi)を除いたものは, それに従って等式を付け加えても収束性が保たれることは明らかであるが, (iv), (vi)については一般には個別に確かめる必要がある. これは正当性を確認しながら変換プログラミングを進める際に重要な問題である[44].

Darlington らは, 彼らの変換法を再帰方程式系の直観的意味に基づいて議論しており, 項書き換えシステムを基礎にしているわけではない. しかし, 彼らが未解決の問題点として指摘している[6], 変換の結果得られプログラムが確かに改善されているかどうかを客観的に評価するための形式化として, 項書き換えシステムの具体化としてのグラフ・リダクション・システムは有望な候補と考えられる[50)〜52)].

4.3 等式によるプログラミング

等式そのものを用いてアルゴリズムを直接記述しようとする考え方は, 新しいものではないが, 関数的プログラミングの利点が見直されるにつれていろいろな立場からの研究が行われている.

ここでは, 項書き換えシステムを基礎にした, Hoffman らの研究[29]を紹介する. 彼らは等式を書き換え規則と見なし, それを用いて与えられた項の正規形を求めることが計算であり, そのような等式システ

```
Symbols
  QUOTE, LABEL, LAMBDA, ATOM, COND, CONS, CAR, CDR, EQ,
      NIL: 0; boolean; unspecified;
  cond, apply: 3; cons, eval, evlis, append, pair,
      assoc: 2;
  evcon, atom, car, cdr: 1.
Primitives
  eq.

For All V, W, X, Y, Z, Z1:
  car(cons(X, Y)) = X;
  cdr(cons(X, Y)) = Y;
  cond(T, X, Y) = X;
  cond(F, X, Y) = Y;
  atom(cons(X, Y)) = F;
  atom(X) = T where X in boolean ∪ unspecified
                        ∪ {QUOTE, EQ, LABEL, LAMBDA, NIL, ATOM, COND,
                           CONS, CAR, CDR, NIL};
  eval(X, Z) = assoc(X, Z) where X in unspecified;
  eval(cons(X, Y), Z) = apply(eval(X, Z), Y, Z);
  apply(eval(COND, Z), X, Z1) = evcon(evlis(X, Z1));
  apply(eval(CAR, Z), cons(X, Y), Z1) = car(eval(X, Z1));
  apply(eval(CDR, Z), cons(X, Y), Z1) = cdr(eval(X, Z1));
  apply(eval(ATOM, Z), cons(X, Y), Z1) = atom(eval(X, Z1));
  apply(eval(QUOTE, Z), cons(X, Y), Z1) = X;
  apply(eval(EQ, Z), cons(W, cons(X, Y)), Z1)
      = eq(eval(W, Z1), eval(X, Z1));
  apply(eval(CONS, Z), cons(W, cons(X, Y)), Z1)
      = cons(eval(W, Z1), eval(X, Z1));
  apply(apply(eval(LABEL, Z), cons(V, cons(W, X)), Z1), Y, Z2)
      = apply(eval(W, cons(cons(V, cons(cons(LABEL,
            cons(V, cons(W, X))), NIL)), Z)), Y, Z1);
  apply(apply(eval(LAMBDA, Z), cons(V, cons(W, X)), Z1), Y, Z2)
      = eval(W, append(pair(V, evlis(Y, Z1)), Z));
  evlis(NIL, Z) = NIL;
  evlis(cons(X, Y), Z) = cons(eval(X, Z), evlis(Y, Z));
  evcon(cons(cons(T, cons(X, Y)), Z)) = X;
  evcon(cons(cons(F, cons(X, Y)), Z)) = evcon(Y);
  append(NIL, Y) = Y;
  append(cons(W, X), Y) = cons(W, append(X, Y));
  pair(NIL, NIL) = NIL;
  pair(cons(V, W), cons(X, Y)) = cons(cons(V, cons(X, NIL)), pair(W, Y));
  assoc(X, Y) = cond(eq(car(car(Y)), X),
                     car(cdr(car(Y))),
                     assoc(X, cdr(Y))).
```

図-6 LISP インタプリタの等式プログラム[39]

ムを書くことがプログラミング（アルゴリズムの記述）であるとする立場を取る．その際，書かれたプログラム（等式システム）が意味のあるアルゴリズムを定めていることを保証するために，等式を，

（1） 左辺に同じ変数が2度以上現われない（線形），

（2） 2つの異なる等式の左辺が同一の項にマッチするとき，その2つの等式の右辺は等しい，

（3） 2つの等式（同じものでもよい）の左辺が同一の項の異なる部分にマッチすれば，その2つの部分は重ならない，

の3つの条件を満たすものに制限する．これは **3.1** の **条件 CR.2** に対応するものであり，このように制限した項書き換えシステムは，

（1） どんな項に対しても正規形が高々1つ定まる（正規形が存在しないことはある），

（2） 項の書き換えに際し，一番外側のリデックス（複数個あることもある）がいつかは必ず書き換えられるようにすれば，正規形が存在するのなら必ずそれが求められる，

という性質を持つことが示されている[43]．

Hoffman らはこのように制限した等式システムでも，従来のプログラミング言語よりはかなり高い記述能力を有し，見通しよくアルゴリズムを記述できることを，リスト処理，木探索，抽象データ型，言語インタプリタなどを例に用いて示している．たとえば，LISP インタプリタの等式プログラムは **図-6** のようになる．

ここで，**Symbols** 以下では項を構成する演算（記号）がその引数の数とともに宣言されている．定数は零引数の演算として宣言される．**boolean, unspecified** はそれぞれ，ブール値（T と F）と特別な意味を持たない文字列の集合であり，これらが定数として自由に使えることが宣言されている．**Primitives** 以下には，すでに定義されている定数の同等性を調べる演算 eq が宣言されている．

Hoffman らは，このような等式プログラムを項書き換えシステムとして直接実動化するために，Pascal でインタプリタを作成し，1秒間に数百回の書き換えが行えたと報告している[29]．

5. おわりに

理想的な計算モデルが備えるべき性質は，

（1） 人間が計算（機械に解かせるべき問題）を表現する際の良いモデルであり，

（2） 機械がそのモデルをある程度効率よく実現できること，

であろう．本稿で紹介した項書き換えシステムは，

（1） 人間がそれを用いて計算を表現しようとする際には，等式システム（等号論理）とみなすことができ，

（2） 機械がこれを実現する際には（グラフ）リダクション・システムとみなし得る，

という二面性を持ち，人間と機械の間を橋渡しする計算モデルとして基本的な要件を備えていると考えられる．

しかしながら，この計算モデルに基づいたプログラミング・システムが現実のものとなり，今日のソフトウェアが抱える問題点が解決されるには，

・ このような形式化に基づいた，仕様記述／プロ

グラミング法の整備,

・ 高効率のリダクション・マシンの実動化,

といった課題を克服しなければならない.

参考文献

1) Balzer, R., Goldman, N. and Wile, D.: On the transformational implementation approach to programming, Proc. 2nd Int. Conf. Software Eng., pp. 337-344 (Oct. 1976).
2) Barbuti, R., Degano, G. and Levi, G.: Toward an inductionless technique for proving properties of logic programs, Proc. 1st Int. Logic Prog. Conf., pp. 175-181 (1982).
3) Barendregt, H. P.: The lambda calculus, its syntax and semantics, North-Holland (1981).
4) Bauer, F. L.: Programming as an evolutionary process, Proc. 2nd Int. Conf. Software Eng., pp. 223-234 (Oct. 1976).
5) Berry, G. and Levy, J. J.: Minimal and optimal computations of recursive programs, J. ACM 26-1 (1979).
6) Burstall, R. M. and Darlington, J.: A transformation system for developing recursive programs, J. ACM 24-1, pp. 44-67 (1977).
7) Burstall, R. M. and Goguen, J. A.: Putting theories together to make specifications, Proc. 5th Int. Joint Conf. Artificial Intelligence, pp. 1045-1058 (1977).
8) Burstall, R. M., MacQueen D. B. and Sannella, D. T.: HOPE: An experimental applicative language, Proc. LISP Conf., Stanford, CA, pp. 136-143 (1980).
9) Church, A. and Rosser, J. B.: Some properties of conversion, Trans. AMS 39, pp. 472-482 (1936).
10) Darlington, J.: The structured description of algorithm derivations, *Algorithmic Languages*, Eds. de Bakker and van Vliet, North Holland, pp. 221-250 (1981).
11) Darlington, J.: Program transformation, *Functional programming and its application*, Eds. Darlington J. et al. Cambridge Univ. Press, pp. 193-215 (1982).
12) Dershowitz, N.: Termination of linear rewriting systems (preliminary version), Lecture Notes in Comput. Sci. 115, Springer-Verlag (1981).
13) Dershowitz, N.: Orderings for term-rewriting systems, Theor. Comput. Sci. 17, pp. 279-301 (1982).
14) Dershowitz, N. and Manna, Z.: Proving termination with multiset orderings, Comm. ACM 22, pp. 465-476 (1979).
15) Donway, P. J. and Sethi, R.: Correct computation rules for recursive languages, SIAM J. Comput. 5-3, pp. 378-401 (1976).
16) Feather, M. S.: A system for assisting program transformation, ACM TOPLAS 4-1, pp. 1-20 (1982).
17) Futatsugi, K.: Hierarchical software development in HISP, *Computer Science & Technologies 1982*, Ed. Kitagawa T., Japan Annual Reviews in Electronics, Computers & Telecommunications Series, OHM/North-Holland, pp. 151-174 (1982).
18) Futatsugi, K. and Okada, K.: Specification writing as construction of hierarchically structured clusters of operators, Proc. IFIP Congress 80, pp. 287-292 (Oct. 1980).
19) Futatsugi K. and Okada, K.: A hierarchical structuring method for functional software systems, Proc. 6th Int. Conf. Software Eng., pp. 393-402 (1982).
20) Gerhart, S. L. et al.: An overview of AFFIRM: A specification and verification system, Proc. IFIP Congress 80, pp. 343-347 (1980).
21) Goguen, J. A.: How to prove algebraic inductive hypotheses without induction, with applications to the correctness of data type implementation, Proc. 5th Conf. Automated Deduction, Les Arcs (Jul. 1981).
22) Goguen, J. A. and Tardo, J. J.: An introduction to OBJ: a language for writing and testing algebraic specifications, Proc. IEEE Conf. Specifications of Reliable Software, pp. 170-189 (1979).
23) Goguen, J. A., Thatcher, J. W. and Wagner, E. G.: An initial algebra approach to the specification, correctness, and implementation of abstract data types, *Current trends in programming methodology, Vol. IV: Data Structuring*, Ed. Yeh R. T., Prentice-Hall, pp. 80-149 (1978).
24) Goguen, J. A., Thatcher, J. W., Wagner, E. G. and Wright, J. B.: Initial algebra semantics and continuous algebras, J. ACM 24, pp. 68-95 (1977).
25) Guttag, J. V.: The specification and application to programming of abstract data types, CSRG-59, Univ. of Toronto (Sep. 1975).
26) Guttag, J. V. and Horning, J. J.: The algebraic specification of abstract data types, Acta Informatica 10, pp. 27-52 (1978).
27) Guttag, J. V. and Horning, J. J.: Formal specification as a design tool, Proc. ACM Sympo. Principles of Programming Languages, pp. 251-261 (1980).
28) Guttag, J. V, Horowitz, E. and Musser,

D. R.: Abstract Data Types and Software Validation, Comm. ACM 21-12, pp. 1048-1064 (Dec. 1978).
29) Hoffmann, C. M. and O'Donnell, M. J.: Programming with equations, ACM TOPLAS 4-1, pp. 83-112 (Jan. 1982).
30) Huet, G.: Confluent reductions: abstract properties and applications to term rewriting systems, J. ACM 27-4, pp. 797-821 (Oct. 1980).
31) Huet, G.: A complete proof of correctness of the Kunth-Bendix completion algorithm, J. Comput. Syst. Sci. 23, pp. 11-21 (1982).
32) Huet, G. and Hullot, J.-M.: Proofs by induction in equational theories with constructors, Proc. 21st IEEE Annual Sympo. Foundations of Computer Science, pp. 96-107 (Oct. 1980).
33) Huet, G. and Levy, J.-J.: Call by need computations in non-ambiguous linear term rewriting systems, Rapport Laboria 359, IRIA (1979).
34) Huet, G. and Oppen, D. C.: Equations and rewrite rules: a survey, *Formal Languages: Perspectives and Open Problems*, Ed. Book R., Academic Press, pp. 349-393 (Jan. 1980).
35) Jouannaud, J. P. and Lescanne, P.: On multiset orderings, Inf. Process. Lett. 15-2, pp. 57-63 (1982).
36) Klop, J. W.: *Combinatory reduction systems*, Dissertation, Univ. of Utrecht (1980).
37) Knuth, D. E. and Bendix, P. G.: Simple word problems in universal algebras, *Computational Probelms in Abstract Algebra*, Ed. Leech J., Pergamon Press, pp. 263-297 (1970).
38) Levy, J.-J.: Optimal reductions in the lambda-calculus, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, Eds. Seldin J. P. and Hindley J. R., Academic Press (1980).
39) Manna, Z.: *Mathematical theory of computation*, McGraw-Hill (1974) 邦訳 (五十嵐滋) プログラムの理論, 日本コンピュータ協会 (1975).
40) Musser, D. R.: A data type verification system based on rewrite rules, Proc. 6th Texas Conf. Computing Systems, Austin (Nov. 1978).
41) Musser, D. R.: On proving inductive properties of abstract data types, Proc. 7th ACM Sympo. Principles of Programming Languages (Jan. 1980).
42) Musser, D. R.: Abstract data type specification in the AFFIRM system, IEEE Trans. Softw. Eng., SE-6-1, pp. 24-32 (1980).
43) O'Donnell M.: *Computing in systems described by equations*, Lecture Notes in Comput. Sci. 58, Springer-Verlag (1977).
44) 岡田, 二木, 鳥居: 階層的仕様言語による変換プログラミング, 信学技報 AL82-42 (1982).
45) Pacini, G., Montangero, C. and Turini, F.: Graph representation and computation rules for typeless recursive languages, Lecture Notes in Comput. Sci. 14, Springer-Verlag, pp. 157-169 (1974).
46) Peterson, G. E. and Stickel, M. E.: Complete sets of reductions for equational theories, J. ACM 28-2, pp. 233-264 (1981).
47) Pettorossi, A.: Comparing and putting together recursive path ordering, simplification orderings and non-ascending property for termination proofs of term rewriting systems, Lecture Notes in Comput. Sci. 115, Springer-Verlag (1981).
48) Rosen, B. K.: Subtree replacement systems. Ph. D. Thesis, Harvard Univ. (1971).
49) Rosen, B. K.: Tree-manipulating systems and Church-Rosser theorems, J. ACM 20, pp. 160-187 (1973).
50) Staples, J.: Computation on graph-like expressions, Theor. Comput. Sci. 10, pp. 171-185 (1980).
51) Staples, J.: Optimal evaluations of graph-like expressions, Theor. Comput. Sci. 10, pp. 297-316 (1980).
52) Staples, J.: Speeding up subtree replacement systems, Theor. Comput. Sci. 11, pp. 39-47 (1980).
53) 杉山, 谷口, 嵩: 基底代数を前提とする代数的仕様, 信学論, J 64-D-4, pp. 324-331 (1981).
54) 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, J 65-D-7, pp. 858-865 (1982).
55) Sunshine, C. A. et al.: Specification and verification of communication protocols in AFFIRM using state transition models, IEEE Trans. Softw. Eng. SE-8-5, pp. 460-489 (1982).
56) 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).
57) 外山: ラベル付き項書き換えシステム, 信学技報 AL 82-40 (1982).
58) Turner, D. A.: A new implementation technique for applicative languages, Softw. Prac. Exper. 9, pp. 31-49 (1979).
59) Vuillemin, J.: Correct and optimal implementation of recursion in a simple programming language, J. Comput. Syst. Sci. 9-3, pp. 332-354 (1974).
60) Wand, M.: Final algebra semantics and data type extensions, J. Comput. Syst. Sci. 19-1, pp. 27-44 (1979).

(昭和57年11月9日受付)