解 説

項書き換えによる関数型プログラミング†

稲垣康善†† 直井 徹††

1. はじめに

関数型プログラミングが広く認められるようになってきたのは,

(1) 表示的な記述が可能で,簡明かつ記述性と理解性に優れたプログラムを書くことができる,

(2) 数学的な形式性と厳密性に優れており,そのために,プログラム検証やプログラム変換などのプログラミングに関する基本的な諸問題を形式的に扱うことができる,

(3) 関数型プログラムは,本質的に並列性を内包しており,その利用が可能である,

などのような特長による.

関数型プログラミングの歴史は古く J. McCarthy が純 Lisp を発表した 1960 年代初頭にまでさかのぼる[18]. J. Backus の 1978 年の論文[4]以来,関数型プログラミングの概念は特に注目されるようになり,この 10 年間に,多くの研究が行われ,いくつかの関数型言語や関数型計算向き計算アーキテクチャが提案され,実験システムが開発されるとともに,コンビネータ,カテゴリ,型推論などの理論研究が行われてきている.

最近,これらの成果を踏まえて,多数の解説論文[14),26)]や関連の書物[7),10),11),15),25)]が出版されている.本特集号もその一つであり,関数型プログラミング言語と計算モデル,関数型プログラムの実行方式,関数型プログラムの作成支援および検証,関数型計算モデルの応用と,理論から応用まで幅広い解説が含まれている.本稿では,関数型プログラミングの理論的枠組みに関する解説の一つとして,項書き換えによる関数型プログラムの意味論に関する基本的な概念と問題点について著者らの最近の成果を含めて解説する.

† Term Rewriting Systems and Functional Programming by Yasuyoshi INAGAKI and Tohru NAOI (Faculty of Engineering, Nagoya University).

†† 名古屋大学工学部

2. 項書き換え系の理論の位置づけ

関数型プログラミングの理論的な計算モデルとしては,λ 計算,コンビネータ論理,項書き換え系,再帰的プログラム,CCC (Cartesian Closed Category) などがある.λ 計算,コンビネータ論理は古く 1930 年代から計算理論モデルとして研究されてきた.関数型プログラミングの計算モデルとして,λ 計算 (コンビネータ論理)[5)] と再帰的プログラム[8)]は 1970 年代以降よく研究されてきている.これらは,ともに項の簡約を計算原理としている.両者を較べると,λ 計算は高階関数の記述が容易であるがその簡約操作の抽象度が高く直観的な理解性に欠ける.一方,再帰的プログラムは,簡約操作が λ 計算より具体的な水準で行われるため,分かりやすい.それらの構文論的性質,ならびに意味論的性質に関しては,相互に影響を与えながら多くの成果が蓄積されてきている[5),8)].

再帰的プログラムは,簡約系として考えれば,項書き換え系の特別な場合とみなすことができる.したがって,再帰プログラムの理論を項書き換え系に対して拡張することは,関数型言語の統一的な理論的枠組みを構成する上で自然でかつ重要な課題である.項書き換え系の構文論的な性質についてはすでに多くの成果が得られてよく理解されているので,上の課題のうち残されているのは主に意味論的な問題である.

項書き換え系の意味論は,関数型言語の理論的枠組みの統一だけでなく,一方で,抽象データ型の代数的仕様記述[1),16)]に対して,操作的意味論と完全に整合した表示的意味論を与えることを可能にし,関数型プログラムと抽象データ型の概念を自然な形で融合するという点からも意味のあることである.

このように項書き換え系を位置づけて,以下では,プログラミング体系としての項書き換え系とそれに望まれる性質について述べ,項書き換え系の意味論の基本的な概念と問題点を解説する.また,意味論の一例として著者らが文献[20)]に提案したものを紹介し,それ

と再帰的プログラム、λ 計算の意味論との関連について簡単に触れる.

3. 項書き換え系の意味の領域としての代数

計算機構としての項書き換え系の意味論は，従来，次のように素朴に考えられてきた．すなわち，項書き換え系の意味とは，項を入力とし，その項の正規形（それ以上書き換えできない項）を出力とするような項集合上の（部分）関数とするのである．たとえば，次の項書き換え系を考えてみよう．

[例1]

$R=\{\text{add}(x,0)\rightarrow x,$
$\quad \text{add}(x,s(y))\rightarrow s(\text{add}(x,y))\}$

この項書き換え系は，add $(s(s(0)), s(0))$ が入力されると，

$\text{add}(s(s(0)), s(0))$
$\rightarrow s(\text{add}(s(s(0)), 0))$
$\rightarrow s(s(s(0)))$

のように，2ステップでその正規形に書き換える．自然数 $0, 1, 2, 3, \cdots$ を $0, s(0), s(s(0)), s(s(s(0))), \cdots$ で表せば，上の項書き換えは $2+1=3$ の計算をしていると意味づけられる．すなわち，上の項書き換え系は，自然数のたし算をするシステムであると考えることができる．□

すなわち，この項書き換え系の素朴な意味論は，入力として与えられた項を簡約していって，それ以上書き換えられない項（正規形）に到達して書き換えが停止したとき，その正規形を計算結果とするようなものである．

しかしながら，このような意味論がわれわれの直観を十分に形式化したものとはいい難い．われわれは，項を眺めるときそこに現れる関数記号（たとえば，add）に対し，書き換えを通じてある領域上の関数（たとえば，自然数のたし算）が定まることを暗黙のうちに期待し，またそのようにしてプログラムを理解しようとする．したがって，項書き換え系の意味論は，単に項書き換え系に関数としての意味を付与するだけでなく，個々の関数記号に対してもやはりそれを関数として説明できるようなものでなければならない．上の意味論は，このことに対してはなにも言及していない．

与えられた関数記号に対する解釈の構造は代数と呼ばれる．すなわち，これは，適当な集合 D と各関数記号にその意味として割り当てられるべき D 上の関数の集合である．抽象データ型の従来の理論[1),16)]では，古典的な等式論理の成果に従い，データ型の仕様として与えられた等式集合から（それを公理系として）代数を定める．これは，仕様の表示的な意味論である．一方，仕様を一つのプログラムとみなして実行するために，等式集合を項書き換え系とみなした．これが仕様の操作的意味論である．しかし，この両者の間には当然のことながらギャップが存在する．たとえば，例1の項書き換え系 R の各書き換え規則の両辺を交換すれば新たな項書き換え系 R^{-1} が得られるが，等式の集合とみれば変化していない．ところが，実際に加算を計算できるのは R であって，R^{-1} ではない．このギャップを克服するために，従来の抽象データ型の理論では項書き換え系に対して合流性と停止性（たとえば，文献[12)]）の二つの性質を前提とした．前者の条件は，それぞれの項に正規形が存在するならば一意である（すなわち，計算結果が一意である）ための十分条件になっている．また，後者の性質は書き換えが無限には続かず，必ず有限のステップで停止するということをいっている．よって，これらの前提を満たす項書き換え系 R の下ではすべての項に唯一の正規形が存在することが保証され，さらに，二つの項が同一の正規形をもつことと，それらの項が公理系としての等式集合 R の下で等しいこととが同値である．そこで，項書き換え系 R の下でそれぞれの項の意味はその正規形であるとしたときに，R を等式集合とみなして定まる代数と同型な構造を導くことができ，仕様の操作的意味は表示的意味に一致する．

しかしながら，項書き換え系に対して合流性と停止性をともに仮定するとその計算能力が真に制限される（任意の計算可能な関数を記述することができない）ことが知られている[3)]．よって，このような前提はプログラミング体系としての項書き換え系にとっては強すぎるものである．そこで，項書き換え系に対して（もし，合流性を仮定するならば）停止性は仮定せずに議論を進めなければならないが，この場合はもはや等式論理による従来の意味論に依存することはできず，新たな枠組みによって代数を構成することが必要となる．

この章の終わりに，停止性を満たさないが意味のある項書き換え系の例を与えておこう．

[例2] リストの領域上の計算を行う，次のような項書き換え系 R を考える．

$R=\{\text{if}(\text{true}, x, y)\rightarrow x,$
$\quad \text{if}(\text{false}, x, y)\rightarrow y,$

car (cons (x, y))→x,
cdr (cons (x, y))→y,
isnil (nil)→true,
isnil (cons (x, y))→false,
$f(x)$→cons $(x, f(s(x))$,
$g(x)$→if (isnil (x),
nil,
cons $(s$(car $(x))$, g(cdr $(x)))$}

この項書き換え系 R の下では項 $f(0)$ は正規形をもたず，次のような無限書き換え列

$f(0)$→cons $(0, f(1))$
→cons $(0,$ cons $(1, f(2)))$
→… …(*)

が生じる．ここで，1, 2, 3, … は，$s(0)$, $s(s(0))$, $s(s(s(0)))$, … の略記法として理解されたい．以下でもこの略記法を用いる．また，項 g(cons (0, nil)) は正規形 cons (1, nil) をもつが，書き換える部分項の選び方によっては無限の書き換えを生じてしまう．たとえば，g で始まる部分項を常に書き換えるような場合がそうであり，これが R の停止性に対するもう一つの反例となっている． □

4. プログラミング体系としての項書き換え系

本章ではプログラミング体系としての項書き換え系について述べる．本稿では任意の項書き換え系を考えるのではなく，文献[12),13)]で扱われた，無あいまい性と線形性という二つの性質を満たすような項書き換え系を対象とする．以下では，これらの性質について説明するとともに，プログラミング体系としての項書き換え系が満たすべき性質について考察し，無あいまいかつ線形な項書き換え系がプログラミング体系として有望であることを説明する．

まず，項書き換え系の(左)線形性とは，任意の書き換え規則を一つとったとき，その左辺の項に同一の変数記号が2度以上現れないという性質である．また，無あいまい性とは，任意の規則の左辺が，もう一つの任意の規則の左辺の（変数でない）部分項とユニフィケーション不可能であるという性質である☆．

［例3］ 次の書き換え規則を含むような項書き換え系 R は，線形でない．

$f(x, x)$→0

また，次の二つの規則を含むような項書き換え系 R' は，$g(0, x)$ と $g(y, 1)$ とが $g(0, 1)$ にユニフィケーションされるので，あいまいである．

$g(0, x)$→0
$g(y, 1)$→1 □

以下に，線形かつ無あいまいという制約をおくことが，プログラミング体系としての項書き換え系に対してどのような意味をもつかについて検討する．

まず，計算能力の点からは，項書き換え系に線形で無あいまいであるという制約をおくことはなにも問題がない．実際，任意の決定性チューリング機械は，線形かつ無あいまいな項書き換え系により，時間コスト（書き換え回数とチューリング機械のステップ数）および空間コスト（項のサイズとテープの長さ）について定数係数の相違だけで模倣できる（容易な練習問題である）．

次に，プログラムの記述性に与える影響については次のことが指摘できる．

まず，線形性を要請する理由とこの要請が記述性に与える影響について述べる．たとえば，例3の非線形な項書き換え系 R の下で，項0は正規形であるとし，また，項 t, t' はともに正規形をもたないとする．処理系が項 $f(t, t')$ の書き換えを行ってその正規形を求めようとする際には，t と t' がある同一の項に書き換えられ得るかどうかを判定しなければならない（そのような場合には，$f(t, t')$ は正規形0をもつ）．しかし，この判定は一般には困難（決定不能）である．したがって，項に正規形が存在するなら必ずそれを求め得るという性質（後述）を処理系に対して期待すると，非線形項書き換え系に対する実用的効率をもつ処理系の実現は非常に難しくなる．

このように，非線形な項書き換え系においては書き換え操作（の実現）自体に二つの項のある種の等価性の判定が暗に含まれている．しかし，その判定手続きを明示的に（線形な）書き換え規則として与えるならば，非線形な規則をあえて用いる必要はない．たとえば，自然数に関するプログラミングでは，その判定手続きとして次のようなものがあれば十分である．

eq (0, 0)→true,
eq (0, $s(x)$)→false,
eq ($s(x)$, 0)→false,
eq ($s(x)$, $s(y)$)→eq (x, y)

このようにすることは，規則を与えることによって等

☆ たとえば，項 cons (0, $f(1)$) の部分項とは，cons (0, $f(1)$) それ自身と，0, $f(1)$, 1 を指す．このとき，0, $f(1)$, 1 を真部分項とよぶ．定義中の任意の二つの規則としては，同一の規則をとる場合も考えなければならないが，その場合は，部分項として真部分項だけを考える．

しさの判定の対象となる項を制限することにより，非線形規則による書き換えに対する処理系の重い負担が回避された，と説明できる．

次に，無あいまい性の要請のプログラム記述性への影響について述べる．上の例3のあいまいな項書き換え系において，項 $g(0,1)$ に対する書き換えを考えると，二つの規則がともに同一の位置に適用可能であり，しかも，いずれの規則で書き換えるかによって結果はまったく異なる．この意味で，一般にあいまいな項書き換え系は本質的な非決定性をもつものと考えられる．書き換え規則を与えることが一つの関数（たとえば，例3の g）の場合分けによる定義と考えれば，あいまいさが生じることはその場合分けの不完全さを意味しており，非決定性を意図的に用いるのでなければ，あいまい性は容易に避けられる．

最後に，プログラミング体系としての項書き換え系の実現に関して重要な意味をもつ書き換え戦略について述べる．線形かつ無あいまいな項書き換え系は合流性を有しているが，より重要なことは，このような項書き換え系に対しては，必須呼び，並列最外戦略などの書き換え戦略（正規化戦略）が知られており[13]，項に正規形が存在するときには必ずそれに到達するように，書き換えるべき部分項を選択する手続き（書き換えの各ステップにおいて線形時間ないしは多項式時間で選択を行う）が存在することである．

Prolog の場合には，ホーン節論理としては証明可能なゴールであっても，現実の処理系に組み込まれた導出戦略の下では空節に到達しないことがある．このような事態を回避することはプログラマの責任であり，よって，処理系の導出戦略を細かに考慮せずにはプログラミングを行うことができない（カット演算子のような制御機構を用いればなおさらである）．

これに対して，正規化戦略が利用できるような項書き換え系においては，正規形の存在する場合には処理系が必ずそれを求めてくれるので，プログラミングに際して部分項の書き換え順序を意識する必要はない．この意味では，無あいまい線形項書き換え系によれば，Prolog よりも宣言的なプログラミング体系が自然に可能となると考えられる．

研究者によっては，無あいまい線形項書き換え系よりも一般的な項書き換え系をプログラミング体系として用いようという主張もある．たとえば，合流性だけを満たすような項書き換え系がしばしば言及される．合流性から，次のような性質を導くことはできる．すなわち，項が正規形をもつときに，その項に対してどのような書き換えを行った後もまだその時点から正規形に到達する書き換えが可能である（したがって，back-track は不要である）．しかし，実際に処理系を実現する際にはこのような消極的な保証では不十分であって，書き換えをどのように進めるかを具体的に指示する手続き（前述の正規化戦略）が要請されるのである*．すなわち，合流性は必要であるが十分ではない．

5. 停止しない書き換えの意味

前章では，無あいまい線形項書き換え系のプログラミング体系としての有望さについて述べたが，本章以降は再び意味論の問題に戻り，そのような項書き換え系が意味論的にも整った構造をもっていることを示す．

さて，無あいまい線形項書き換え系は合流性を満たすから，さらに停止性をも仮定するなら前述のようにその計算能力が制限されてしまう．したがって，以下では停止性の条件なしで議論を進めるが，このときなお，それぞれの項の意味を原則としてその正規形によって与えるという立場をとろうとすれば**，正規形が存在しない項の意味をどのように定めるかが意味論構成の一つの鍵になる．一つの候補は正規形をもたない項の意味を未定義とするものだが，代数を導く（すなわち，各関数記号に関数としての意味を与える）という立場からすればこの考え方は自然なものでなく，次の例に示すように，病的な構造を導くことがある．

［例4］ 以下では，関数記号 f に対してそれが表す関数を $\underline{f}$ のように表す．例2で与えた項書き換え系に次の規則を追加して考える．

$h(x) \rightarrow h(x)$

まず，項0はそれ自身正規形だから $\underline{0}=0$ であり，項 $f(0)$ は正規形をもたないから $\underline{f}(\underline{0})$ は未定義である．ところが，$\mathrm{car}(f(0))$ は正規形0をもつから，写像 $\underline{\mathrm{car}}$ はパラメータの値 $\underline{f}(\underline{0})$ が定まらないのに結果の値を返すことになり，集合論に基づく写像の概念からみて不合理である．これを避けるため，$\underline{f}(\underline{0})$ は真に未定義なのでなく，未定義を象徴するある特別の値

---

* 正規化戦略の存在を前提とするとき，線形性の制約を緩めることの困難さはすでに示唆したが，無あいまい性の制約を緩めることは一部可能である[23],[24]．

** これとは異なる立場も可能である．その一つは，文献[11]のような古典的動作意味論であり，3. の冒頭で述べた単純な意味論を代数の構成という意図の下に改良したものである．同文献では，本稿で以下に解説する意味論と動作意味論の関連について議論している．

⊥が存在してそれと同一視されていると考える．すると，項 $h(0)$ もやはり正規形をもたないから，$\underline{f}(\underline{0})=\underline{h}(\underline{0})=\perp$ が成り立つ．この各辺に写像 car を適用すれば，$\underline{\mathrm{car}}(\underline{f}(\underline{0}))=\underline{\mathrm{car}}(\underline{h}(\underline{0}))$ が導かれる．ところが $\mathrm{car}(f(0))$ は正規形0をもち，$\mathrm{car}(h(0))$ は正規形をもたないから，結局 $\underline{0}=\perp$ が導かれてしまう．同様にして，任意の自然数が⊥に等しいことが導かれる．□

上の例から分かるように，正規形をもたない項を一律に未定義とするのは適当でない．ここで，$f(0)$ から始まる無限の書き換え（＊）をもう一度観察してみると，その極限として得られる無限リストを想像することは難しくない．実際，car 演算を何度か用いると $f(0)$ から任意の自然数を取り出すことができるから，$f(0)$ の「正規形」を，すべての自然数を含むリストと考えることは自然に思われる．そこで，以下では，このような無限の書き換えの極限としての正規形をいかに定式化するかを考える．

6. 書き換えの極限と近似正規形

無限の書き換えの極限は，Scott の順序論的手法を用いて定式化される．実際，このような定式化は再帰的プログラムの意味論においてはすでに 1970 年代半ばに成されており[8),9)]，本章で述べるものはその一般化である．なお，本稿では，Scott の提案した連続写像や完備順序集合 (complete partial order) などの概念については説明しないが，文献[5)] に十分な要約があるのでこれを参照されたい．

また，本章と次章の内容は，最近著者らによって得られた項書き換え系の代数的意味論の要点の紹介である．詳しくは，文献[20)~22)]を参照されたい．

まず，項をつくる関数記号の集合に特別な定数記号Ωが含まれているものとする．これを用いて項の集合上に次のような順序 $\preceq$ を定義する．項 t の部分項の出現を任意個選び，それらをΩで置き換えると項 t' が得られるとき，$t'\preceq t$．この順序の下での最小元は定数記号Ωであり，これは構文論的にみてなんら情報をもたないことを象徴する．たとえば，次の無限列(＊＊)はこの順序の下で上昇鎖を成す．

$$\begin{aligned}&\Omega \preceq \mathrm{cons}(0,\Omega)\\&\quad\preceq \mathrm{cons}(0,\mathrm{cons}(1,\Omega))\\&\quad\preceq \mathrm{cons}(0,\mathrm{cons}(1,\mathrm{cons}(2,\Omega)))\\&\quad\cdots \qquad\cdots(**)\end{aligned}$$

さらに，以下では，前述の「無限リスト」を表すような無限の大きさをもつ項を考える．これは，形式的には無限木と呼ばれる（その厳密な取り扱いについては，文献[2),6)] を参照されたい）．この観点からは項とは有限な木であり，無限木の特別な場合とみなされる．先ほどの順序 $\preceq$ の定義を無限木の集合にまで適用すれば，そのとき，無限木の集合は完備順序集合を成すことが知られている．このことから，上記の上昇鎖(＊＊)にもその上限 (least upper bound) の存在が保証される．実際，その上限は，図-1 に示すような無限木である．そこで，最初に述べた無限書き換えの極限をこの上限の概念を用いて定式化することが考えられる．例2の項 $f(0)$ の場合，これを次のように行う．まず，$f(0)$ から始まる無限書き換え系列（＊）に現れるそれぞれの項の，書き換え可能な部分項 $f(0), f(1), \cdots$ をすべて Ω で置き換える．すると，無限上昇鎖(＊＊)が得られるので，その上限を(＊)の「極限」と考えることができる．以上のことを，一般的には次のように定義する．なお，以下では，T, T' などは無限木を表すものとする．さらに，項書き換え系による書き換えも項（有限木）から無限木に拡張して考える．

［定義1］ 項書き換え系 R における木 T の記号的な値 $\mathrm{Val}_R(T)$ を次のように定義する．

$$\mathrm{Val}_R(T)=\mathrm{lub}\{\omega_R(T')\mid T\to_R{}^*T'\}$$

ここに，$T\to_R{}^*T'$ は，木 T から R による有限回の書き換えによって T' が得られることを表し，写像 ω_R は，大雑把には書き換え可能な部分木をΩで置き換える操作を表す（正確な定義は文献[20)]を参照されたい）．また，lub は集合の上限を表す．□

線形性と無あいまい性を用いると，写像 Val_R の全域性が保証される．写像 ω_R の値域を近似正規形の集合と呼び，ANF_R で表す．近似正規形は正規形（すなわち，書き換え可能な部分木を含まない無限木）でもある．また，$\omega_R(T)$ を T の直接近似正規形といい，さらに，$T\to_R{}^*T'$ のとき，$\omega_R(T')$ を T の近似正規

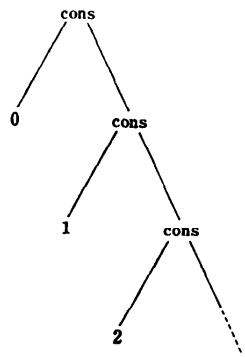

図-1 上限としての無限木

形という．上の定義では，書き換えの結果を，その各段階で得られる木から計算の中間結果としての近似正規形を抽出して，それらの情報を集積したものであるとしている．

写像 Val_R に関して次のような結果が証明される．

［定理1］ 写像 Val_R は連続であり，べき等である．すなわち，$\mathrm{Val}_R = \mathrm{Val}_R \circ \mathrm{Val}_R$. □

［命題1］ Val_R の値域は ANF_R に一致し，しかも，それは完備順序集合をなす． □

［命題2］ t が有限でΩを含まない（すなわち，通常の意味での）正規形であるとき，

$$T \to_R^* t \Leftrightarrow \mathrm{Val}_R(T) = t$$ □

結局，項書き換え系 R の意味は，上述のような連続写像 Val_R である．すなわち，これは，それぞれの項を，もしその項が（通常の意味における）正規形をもつならばその正規形に写し（命題2），そうでないならばその項から始まる書き換えの極限としての近似正規形に写す（命題1）．さらに，Val_R のべき等性はいったん評価した木を再度評価しても結果が変わらないことを意味するから，写像 Val_R 自体，一つの「計算機械」とみなすことができる．

7. 連続代数

前章では項書き換え系に対して連続写像としての意味を与えたが，**3.** で述べたように，これだけでは項書き換え系の意味論としては不十分で，さらに，この定式化を通じて代数構造（すなわち，関数記号の意味）が定められる必要がある．本章では，代数の構成を行ってその性質の一端を示す．

前章の最後に述べたように，Val_R はある意味で計算機械を表しており，その値域 ANF_R は計算結果の全体に相当する．そこで，ANF_R 上に代数を構成することにする．f を n 変数の関数記号とするとき，その解釈を次のような関数 $\underline{f} : \mathrm{ANF}_R^n \to \mathrm{ANF}_R$ として与える．

$$\underline{f}(T_1, \cdots, T_n) = \mathrm{Val}_R(f(T_1, \cdots, T_n))$$

この定式化によって，例4で指摘した問題点は次の例で示すように解決される．

［例5］ まず，$\mathrm{Val}_R(0) = 0$ より $\underline{0} = 0$ であり，$\mathrm{Val}_R(f(0))$，すなわち，$\underline{f}(\underline{0})$ は図-1 の無限木である．これに対して，$h(0)$ から始まる書き換えは

$$h(0) \to h(0) \to h(0) \to \cdots$$

だけであるから，$\mathrm{Val}_R(h(0)) = \mathrm{lub}\{\omega_R(h(0))\} = \mathrm{lub}\{\Omega\}$ Ω．すなわち，$\underline{h}(\underline{0}) = \Omega$ である．したがって，$f(\underline{0}) \neq \underline{h}(\underline{0})$ であるから，例4のように $\underline{\mathrm{car}}(\underline{f}(\underline{0})) = \underline{\mathrm{car}}(\underline{h}(\underline{0}))$ が成立する必要はない．このために，0 が「未定義」と同一視されるような事態は回避される． □

Val_R に基づく関数記号の解釈に関して，次の性質が得られる．

［定理2］ 各関数記号 f に対して，関数 $\underline{f}$ は連続である（命題1参照）． □

このように，各関数記号が連続関数として解釈されるような代数を連続代数[2),6),8),9),20),22)]という．上の解釈によって，関数記号は近似正規形集合という構文的領域の写像を表すものとみなされるが，われわれが真に要請するのは，これがさらに，自然数やリストなどの個別的な領域上の写像として解釈されることである．本稿ではこの問題を指摘するにとどめるが，これについては，文献[20),22)]を参照されたい．

8. むすび

項書き換え系に対して停止性を仮定しないときにも，無限の書き換えの極限としての書き換え結果を定式化することで，項書き換え系から直接に代数が構成できることを示した．

本稿で解説した意味論を項書き換え系の代数的意味論という．これは再帰的プログラムの代数的意味論[8),9)]を一般化したものとなっている．なお，再帰的プログラムについてはいわゆる不動点意味論が著名で，またこれが代数的意味論と等価な定式化であることが導かれている[8)]．この結果も項書き換え系に対して一般化することができる[19)]．また，このような項書き換え系，再帰的プログラムの代数的意味論と類似する手法は λ 計算の意味論においてもすでに用いられており[5),17),27)]，これらの定式化に共通する数学的構造についても著者らは結果を得ている[22)]．以上のことは，抽象データ型の概念を包含するような関数型プログラムの統一的枠組みに近づきつつあることを示しているといえるであろう．

本稿ではプログラミング体系としての項書き換え系についてその意味論を中心に検討してきたが，これ以外に残された課題としては，まず，意味論の応用としての検証や等価変換，合成などの問題があげられる．また，このような処理を自動化しさらにそれを実用技術化するためには，項書き換え系による計算の複雑さに関する研究が重要となろう．あるいは，処理系の実現の観点から考えれば，従来から指摘されるグラフ書き換えの効率的実現などの問題だけでなく，項書き換

え系に内在する並列性を明示的にし，多重プロセッサによって扱いやすい形に表現した項書き換えの理論モデルについてさらに検討する必要があると思われる．

関数型プログラミングの立場は禁欲的に過ぎるという指摘がある．これに関して著者らの観点を述べたい．将来において人間と計算機のコミュニケーションは，現在のプログラミングという形態よりも高い次元で成されるべきである．そのためには，計算機が人間の要求する処理の意味について高度にかつ客観的に認識できる必要がある．このとき計算機言語の最も重要な特性は人間の意図を素直に表現でき，かつ，計算機にとって取り扱い容易な，形式的でかつ単純な意味論を備えることである．関数型プログラムの特長はそのような局面において真に活かされるであろう．関数型言語を現在の計算機とその利用形態に適合させ複雑化させるよりは，むしろ，その単純さから多くの理論的成果を実らせて，将来の計算機上での収穫に備えるべきであると考える．

参考文献

1) ADJ: An Initial Algebra Approach to the Specification, Correctness and Implementation of Abstract Data Types, in: Current Trends in Programming Methodology, Ed. R. T. Yeh, 4, pp. 80-149, Prentice-Hall (1978).
2) ADJ: Initial Algebra Semantics and Continuous Algebras, J. ACM 24, pp. 68-95 (1977).
3) Avenhaus, J.: On the Descriptive Power of Term Rewriting Systems, J. of Symbolic Computation 2, 2, pp. 109-122 (1986).
4) Backus J.: Can Programming be Liberated from Von Neumann Style? A Functional Style and its Algebra of Programs, C. ACM 21, 8, pp. 613-641 (1978).
5) Barendregt, H. P.: The Lambda Calculus-Its Syntax and Semantics, 2nd Ed., North-Holland (1984).
6) Courcelle, B.: Fundamental Properties of Infinite Trees, Theor. Comput. Sci. 25, 2, pp. 95-169 (1983).
7) Eisenbach, S. (ed.): Functional Programming: Languages, Tools and Architectures, Ellis Hollywood Limited (1978).
8) Guessarian, I.: Algebraic Semantics, LNCS 99, Springer-Verlag (1981).
9) Guessarian, I.: Survey on Classes of Interpretations and Some of Their Applications, in: M. Nivat and J.C. Reynolds, Eds., Algebraic Methods in Semantics, Cambridge University Press, pp. 383-410 (1985).
10) Henderson, P.: Functional Programming, Application and Implementation, Prentice-Hall (1980).
11) Henson, M.C.: Elements of Functional Languages, Blackwell Scientific Publications (1987).
12) Huet, G.: Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems, J. ACM 27, 4, pp. 797-821 (1980).
13) Huet, G. and Lévy, J.-J.: Call by Need Computations in Non-Ambiguous Linear Term Rewriting Systems, Raport Laboria 359, IRIA (1979).
14) 井田哲雄，田中二郎：関数型言語の計算モデル，コンピュータソフトウェア 3, 2, pp. 2-18 (1986).
15) 井田哲雄：プログラミング言語の新潮流，共立出版 (1988).
16) 稲垣，坂部：抽象データタイプの代数的仕様記述の基礎(1)-(4)，情報処理 Vol. 25, No. 1, 5, 7, 9 (1984).
17) Lévy, J.-J.: An Algebraic Interpretation of the $\lambda\beta$K-calculus; and an Application of a Labeled λ-calculus, Theor. Comput. Sci. 2, No. 1, pp. 97-114 (1976).
18) McCarthy, J.: Recursive Functions of Symbolic Expressions and their Computation by Machines, C. ACM 3, 4, pp. 184-195 (1960).
19) Naoi, T. and Inagaki, Y.: The Relation between Algebraic and Fixedpoint Semantics of Term Rewriting Systems, Report of Tech. Group on Computation, COMP 86-37, IECEJ (1986).
20) 直井，稲垣：項書換え系の意味論と自由連続代数，信学論 D, J71-D, 6. pp. 942-949 (1988年6月).
21) 直井，稲垣：項書換え系とその保存的拡大における代数的意味論と動作的意味論の関連について，信学論投稿中.
22) 直井，稲垣：連続動作代数とその項書き換え系の意味論への応用，信学論投稿中.
23) 直井，山下，茨木，本多：必須呼びが正規化戦略となるあいまいな線形項書き換えシステムのクラス，信学論 J 69-D, 9, pp. 1236-1245 (1986).
24) O'Donnell, M. J.: Computing in Systems Described by Equations, LNCS 58, Springer-Verlag (1977).
25) Revesz, G.: Lambda-Calculus, Combinators, and Functional Programming, Cambridge University Press (1988).
26) 米澤明憲：関数型計算モデル，情報処理，Vol. 24, No. 2, pp. 113-122 (1983).
27) Wadsworth, C. P.: The Relation between Computational and Denotational Properties for Scott's D∞-models of the Lambda-Calculus, SIAM J. Comput. 5, 3, pp. 488-521 (1976).

(昭和63年5月9日受付)