記　号　処　理　36－8
プログラミング言語　5－8
（１９８６．３．１０）

項書換え言語Ｔ

相場亮（慶応大理工）外山芳人（ＮＴＴ通研）
井田哲雄（理研）

１．はじめに

数学の形式的体系に基づいて設計された言語の有効性を最近の関数型言語や論理型言語が実証しつつある。これら言語の計算のモデルとなる形式的体系として、ラムダ計算系や一階述語論理がよく知られている。　本稿では、項書換え系に基づくプログラミング言語Ｔについて述べる。項書換え系は、書換え規則Li->Ri　の集合｛Li->Ri ｜i∈I｝、ただしRiに現れる変数はすべてLiに現れる、によって与えられる。この種のシステムは、代数における語問題においても、あるいは、知識を表現する手段としてのプロダクションシステムとしても、現われる。また最近の関数型言語には、等式を基本とするものがあるが、これらの言語も、項書換え系として位置付けることもできる。

項書換え系についての研究は、最近まで主に理論的なものであったが、われわれは、項書換え系を実用的なプログラミング言語として用いることを考えた。そして、実用的言語として成立しうる条件を考察し、一つの簡単な言語を設計した。この言語をわれわれはＴ（ＴはTermの略）と呼んでいる。

２．Ｔの基本構造

本節ではＴの基本的なシンタックスとセマンティクスについて説明する。

2.1. 項とコンビネータ表記

Ｔではすべての対象を以下の項の形で表現する。

＜項＞::= ＜定数＞｜
＜変数＞｜
＜項＞　＜項＞｜
（＜項＞）

関数f(x)をＴの中で表現するには、コンビネータ表記　ｆ ｘ を用いる。ｆ ｘ はx に対してf を関数適用することを意味している。n(n≧0)個の引数を持つ関数 f(x1,‥,xn)　は、カリー化することによって、Ｔにおいては、

(‥((f x1) x2)‥xn)

あるいは

f x1 x2 ‥xn（左結合）

と表わされる。

たとえば、Ｔでは式 (1+2)*(3-4)　を

mult (add 1 2) (sub 3 4)

と表現する。Ｔはタイプフリーなシステムであるため、関数とデータを区別することなく、すべて定数として取り扱う。したがって，add １２という項において加算＋を意味するadd は、Ｔの中では単なる定数であるから、add add や　2 add　などもＴの項として意味をもつことに注意しなければならない。ただし、add １２を評価した結果は3 となるのに対し、残りの二つの項はそれ自身が評価した結果となる。

2.2. 定数

Ｔの定数は次の３種類に分類される。

＜定数＞::= ＜基本定数＞｜
＜基本コンビネータ＞｜

＜定義コンビネータ＞

このうち基本定数と基本コンビネータは、システムに組み込まれた定数であり、定義コンビネータはプログラミング時に新しく導入される定数である。

基本定数はTの基本的なデータ型である、ブール値、整数、実数、文字、文字列、およびリスト構造をつくるためのリスト構成子から成る。リスト

(a,b,c,d)

はリスト構成子 cons と nil をもちいて

cons a (cons b (cons c (cons d nil)))

と表わされる。

基本コンビネータは、基本的なデータ構造上での計算（たとえば整数上の四則演算等）を行うために用いられる。たとえば、基本コンビネータの例として、次のものがある。

if true M N --->> M,

if false M N --->> N,

and true false --->> false,

add 2.6 3.2 --->> 5.8,

hd (cons a (cons b (cons c nil))) --->> a,

等。

これらは、if、and 、add 、およびconsの例である。ただし、--->> は、左辺の項から右辺の項へ書換えられることを意味する。このとき、左辺の項が右辺の項にリダクションされるという。

プログラミングを容易にするために、基本コンビネータは、適当なシンタックスシュガーをもつ。たとえば、add 2.6 3.2 のかわりに中置表記2.6 + 3.2 と書くことも許される。

基本コンビネータを表１に示す。

プログラムという形で、書換え規則を定義することによって、新しいコンビネータを、Tの中に導入することができる。これを定義コンビネータという。定義コンビネータd の意味は、以下の形の書換え規則の集合によって定められる。

d a1 a2 ･･ an ---> E;

ここでa1,･･,an (n≧0) は引数であり、変数か、基本定数か、パターンでなければならない。書換え規則の詳細については後述する。

2.3. シンタックスシュガー

Tではプログラミングを容易にするために、さまざまなシンタックスシュガーを用意している。ここでは、すでに説明した中置表記以外のシンタックスシュガーについて説明する。以下では、E ≡ Fによって、Tの中では左辺と右辺が同一視されることを示す。

まず、リスト構造に関する中置表記を説明する。Tではconsの中置表記として：を許している。つまり、

E1:E2 ≡ (cons E1) E2

となる。さらに、：は右結合と約束する。したがって、

E1:E2:E3 ≡ E1:(E2:E3)

となる。

Tには、以下のような、リスト構造に関するシンタックスシュガーも用意されている。

{ } ≡nil

{E1, E2, E3, E4} ≡ E1 : E2 : E3 : E4 : nil.

条件式については、以下のシンタックスシュガーを使うことができる。

```
if E1 than E2 else E3 ≡ if E1 E2 E3,

E1 if F1,      (if F1 then E1 else
E2 if F2,  ≡        (if F2 then E2 else
  :                    ...
En-1 if Fn-1;         (if Fn-1 then En-1
En otherwise;                 else En)...))
```

次に、整数に対するシンタックスシュガーを示す。Tでは、式の中にE'が現われているなら、

$$E' \equiv E + 1$$

と解釈する。(注意：E'が書換え規則の左辺に現われた場合には後述する数値パターンになる。)

2.4. セマンティクス

Tの中での計算過程は、書換えシステムを用いて説明される。与えられた項M0は、基本コンビネータおよび定義コンビネータに対するTの書換え規則に従ってリダクションされ、リダクション系列

M0 --->>M1--->> M2 --->> ･･･

が生成される。リダクションは、Tの書換え規則が適用できなくなるまで続けられる。その結果得られた項を、M0の正規形とよぶ。M0の正規形は、計算結果、あるいは答とみなされる。

たとえば、階乗関数fac(n) = n! は、次の書換え規則によって定義される。

```
fac x ---> 1              if x = 0,
           x * (fac x-1) otherwise;
```

このとき fac(3) = 6 は以下のリダクション系列によって計算される。

```
fac 3 --->>
1 if 3 = 0, 3 * (fac 3-1) otherwise --->>
3 * (fac 2) --->> ･･･
--->> 3 * 2 * 1 --->> 6 。
```

3．パターン引数

Tの書換え規則を定義するとき、左辺にパターン引数を用いることができる。パターン引数は、特定のデータ構造に対してのみ適用される規則を定義するために利用される。Tの中で使用できるパターンは、自然数に対する数値パターンと、リスト構造に対するリストパターンである。

数値パターンはx', x'', x''', ･･･で表わされ、それぞれ $n \geq 1$, $n \geq 2$, $n \geq 3$ なる自然数にマッチングする。したがって、書換え規則

```
foo x'' ---> x;
```

は以下の無限個の規則と等価である。

```
foo 2 ---> 0;
foo 3 ---> 1;
foo 4 ---> 2;
   :
   :
```

数値パターンをもちいることにより、先ほど説明した階乗関数は以下のように定義することができる。

```
fac 0 ---> 1;
fac x' ---> x' * (fac x);
```

ここで、書換え規則の右辺に出現しているx'は数値パターンではなくて、x' ≡ x + 1の意味である。

リストパターンは以下のように定義される。

＜リストパターン＞::=
　　＜パターン要素＞：＜パターン要素＞
＜パターン要素＞::= ＜変数＞ |
　　＜基本定数＞ |
　　＜リストパターン＞

リストパターンをもちいたプログラム例として、整数リスト上のクイックソートを以下に示す。

```
quick { } -> { };
quick x:y -> append (quick (S x y))
                    x:(quick (L x y));
S x y -> { }                    if (null y),
         (hd y):(S x (tl y))    if (hd y)=<x,
         (S x (tl y))           otherwise;
L x y -> { }                    if (null y),
         (hd y):(L x (tl y))    if (hd y)>x,
         (L x (tl y))           otherwise;
append { } z -> z;
append x:y z -> x:(append y z);
```

ここで、書換え規則の左辺に出現しているx:y はリストパターンで、任意の空でないリストとマッチングする。その際、リストの前部と後部は、右辺においては、それぞれx 、およびy によって、参照できる。

数値パターンとリストパターン以外の引数パターンも、理論的には可能である。しかし、パターンマッチングのためのアルゴリズムを高速、かつ効率的に実現するために、Tの引数パターンは、あえて上記の二つに制限している。

4．書換え規則

リダクションによる計算を矛盾なくかつ効率的に実行するために、Tの書換え規則にはいくつかの制限がつけられている。ここでは、どのような書換え規則がTの中で許されるのかを説明する。まず、Tの書換え規則は以下の(1)，(2)をみたさねばならない。

（1）線形性

書換え規則の左辺に、同一の変数が2回以上出現することがない。たとえば、以下の規則は、許されない。

```
g x x ---> x;
```

（2）重なりなし

二つの書換え規則の左辺が、同一の項にマッチングすることがない。たとえば、以下の規則

```
por two x  ---> true;
por x true ---> true;
```

は、 por true true に対してどちらの左辺もマッチングするから、重なりなしの制限に反している。

制限(1)，(2)より、Tのリダクション関係がChurch-Rossor 性を満足することが示される。この性質は次のことを意味している。もし、項MがNとPにリダクションされたならば、NとPに対してそれぞれ適当なリダクションを行うことによって、ある項Qにおいて、両者のリダクション系列を合流させることができる。Church- Rosser性をみたすなら、リダクションの順序には関係なく正規形は一意に定まることが知られている。したがって、非決定型計算、並列計算、遅延計算、部分計算など、さまざまな計算方法の正当性がTでは保証されている。

Tの中では、項Mは一般に複数の書換え可能な部分（リデックスとよぶ）を持つので、リダクションを行う部分が異なれば、得られるリダクションの系列も当然異なってくる。このとき、リダクションの各ステップで書換えるべきリデックスを指示する規則を、リダクションの戦略という。戦略の中でとくに重要なものは正規戦略である。これは、項Mの正規形が存在するならば、必ず正規形が得られるような戦略である。

正規戦略として、これまでさまざまな戦略が提案されてきた。Tでは戦略の簡明さと実現の容易さを考慮して、最左最外リダクションが正規戦略となるように、以下の制限を書換え規則に付け加えている。

（3）左順序

書換え規則の左辺のすべての定数は、変数の左側に出現しなければならない。（ただし、中置表記は、前置表記に直して考える。）

5．配列

Tには、配列の操作を柔軟に行うために、二種類の配列、対象配列と関数配列とがある。対象配列は多くのプログラミング言語の配列と同様に、項E1, E2, ‥, Enを一列に並べたものである。たとえば

2, 4, 6, 8

を対象配列で表現すると

obarray{1:4} 2 4 6 8

と表現される。ここで、obarray{1:4}は配列構成子で、インデックス領域が1から4であることを意味している。一方、関数配列では、関数によってインデックスから要素を計算する。たとえば、上記の配列は以下のようになる。

funarray{1:4} F

ただし、 F x ---> 2 * x ; とする。このとき、関数配列のi番目の要素はF iによって計算される。

配列に対するさまざまな基本コンビネータは、対象配列と関数配列の両者に対して、まったく同じ形で定められている。さらに、対象配列と関数配列の間を相互に変換するためのコンビネータも用意されている。したがって、Tでは必要に応じて、対象配列と関数配列の間を行ききしながら、柔軟なプログラミングを行うことが可能となる。

なお、プログラミングを容易とするために、Tでは配列に対する以下のシンタックスシュガーを使うことが許される。

[{1:4}, 2, 4, 6, 8]≡obarray{1:4} 2 4 6 8,

[fun, {1:4}, F]≡funarray{1:4} F

6．Tによるプログラミングの例

以下に，Tで記述したプログラムを、例としてあげる。このプログラムは，オランダ国旗問題として知られる問題を、Tを用いて記述したものである。ここでは、上でのべたような，様々のシンタックスシュガーを用いている。

```
DNF { } -> { };
DNF {x} -> {x};
DNF {R, x:y} -> R:(DNF x:y};
DNF {B, x:y} -> append (DNF x:y) {B};
DNF {W, x:y} -> R: (DNF W:y) if x = R,
                append (DNF W:y) {B}
                                if x =B,
                DNF2 (DNF y) otherwise;
DNF2 { } -> { };
DNF2 x:y -> {R: DNF {W, W:y}} if  x=R,
            {W, W:(x:y)}      otherwise;
```

［謝辞］

Tは、昨年四月から八月頃にかけて、筆者等およびユトレヒト大学の Barendregt氏との議論によって、作られたものである。Barendregt氏との共同研究は、日本学術振興会の招へい研究費の補助を得て、行なわれた。

[参考文献]

[Augustsson 84] Augustsson, L. A compiler for lazy ML 1984 LISP and functional programming conference, Austin Texas, Aug. 1984

[Burstall 80] Burstall, R. et.al HOPE: An experimental applicative language LISP Conference, Stanford 1980

[Huet 77] Huet G., Confluent Reductions: Abstract Properties an Applications to Term Rewriting Systems. 18th IEEE Symposium on Foundastions of Computer Science (1977), 30-45

[Rosen 73] Rosen B.K. Tree-Manipulation Systems and Church-Rosser Theorems JACM 20 (1973), 160-187

[Turner 85] Turner,D.A. Miranda:A non-strict functional language with polymorphic types LNCS 201, Springer-Verleg, 1985

表１．基本コンビネータ

(前置表記／中置表記－シンタックスシュガー)

bool	integer	float	character	string	list
	add/+	add/+		str.add/str+	
	sub/-	sub/-			
	-	-			
	mult/*	mult/*			
	div//	div//			
	abs	abs			
	lt/<	lt/<	char.lt/char<	str.lt/str<	
	gt/>	gt/>	char.gt/char>	str.gt/str>	
	eq/=	eq/=	char.eq/char=	str.eq/str=	
	le/=<	le/=<	char.le/char=<	str.le/str=<	
	ge/=>	ge/=>	char.ge/char=>	str.ge/str=>	
	ne/<>	ne/<>	char.ne/char<>	str.ne/str<>	
	mod				
		fix			
	float				
				str.cons	
				str.hd	hd
				str.tl	tail
				str.null	null
if					
not					
and/&					
or/|					
				str.lh	lh
boolp	intp	floatp	charp	strp	atom consp listp