1N-7 戦略の表明を持つ項書き換え系 A－TRSの実現と評価

○古賀 信哉, 布川 博士, 野口 正一

(東北大学電気通信研究所)

1. はじめに

項書き換え系(TRS)は，プログラミング言語として見ると記述の容易性や意味の把握のしやすさなどの特長を持つ，優れた言語である．TRSにおいてプログラムの実行順序を与えるものである書き換え戦略は，本来特定されるものではないが従来は単一の戦略を持つ reducerを採用することにより一つに固定されていた．

正しさと実行速度という二つの排反する要件に対し，一方を犠牲にすることで他方を実現する従来の方式では柔軟性に欠け，TRSをプログラミング言語として有効に利用するには不十分である．われわれは，書き換え規則の項に表明を付けることによりプログラマが個々のTRSプログラムに対して書き換え戦略を自由に指定できる，表明付きの項書き換え系A－TRSを既に提案している[1]．

本論文では，Lispにより直接作成したA－TRS及び従来方式の処理系(reducer)と，これを用いて行なったA－TRSの評価について報告する．

2. 項書き換え系(TRS)，及びreducer

項の定義は通常通り[2]行なう(a,b,c,..により定数を，x,y,z,...により変数を表わす)．項の集合をTerm，変数を含まない項の集合をTerm0で表わす．書き換え規則，TRSについても通常通り[2]である．本論文では特に，重なりがなく且つ線形なTRSのみを扱い，これを単にTRSと呼ぶ．ここで設けた制限は，TRSが合流性をもつための十分条件である．

書き換え戦略には種々のものがあるが，平行最外戦略，最左最内戦略を持つreducerは比較的容易に定義できる．それぞれのreducer定義を以下に示す．

[定義] (並行最外戦略を持つreducer)

```
Rpo : Term0 → Term0
Rpo(t) = if nf(t) then t
         elseif match(t) then Rpo(rewrite(t))
         else   Rpo(f(Rpo-one(t1),...,Rpo-one(tn)))
                        where t≡f(t1,...,tn)

Rpo-one(t) = if nf(t) then t
         elseif match(t) then rewrite(t)
         else f(Rpo-one(t1),...,Rpo-one(tn))
                        where t≡f(t1,...,tn)
```

定義のために用いている言語は，通常のLispのような評価順序を持つものとする．

[定義] (最左最内戦略を持つreducer)

```
Rli : Term0 → Term0
Rli(t) = if nf(t) then t
         elseif t≡a then Rli-one(a)
         else   Rli-one(f(Rli(t1),...,Rli(tn))
                        where t≡f(t1,...,tn)
Rli-one(t) = if nf(t) then t
         elseif match(t) then Rli(rewrite(t))
         else t
```

Rliの定義において，t≡a は項tが定数か否かを確かめるためのものである．

3. 表明付き項書き換え系(A－TRS)

3. 1 リダクションの制御

```
x×0 ▷ 0                 f(0) ▷ s0
x×s(y) ▷ x+(x×y)        f(s(x)) ▷ f(x)×s(x)
```

TRSプログラム1

A－TRSでは，リダクション(書き換え)が項の中に示された表明によって制御される．例えばTRSプログラム1を用いたとき，項 f([s0×ss0])はまず部分項 s0×ss0が平行最外戦略で書き換えられ，f({s0×ss0})の場合にはそれが最左最内戦略で書き換えられる．

TRSプログラムの中に表明を入れることも可能であり，この場合プログラマはTRSを実行の制御が可能なプログラムとして扱うことが出来る．

```
f(0)▷ s0                    f(0)▷ s0
f(s[x])▷ f(x)×s(x)          f(s{x})▷ f(x)×s(x)
```

A－TRSプログラム2　　A－TRSプログラム3

```
f(0)▷ s0
f(s{x})▷ [f(x)]×s(x)
```

A－TRSプログラム4

```
f(ss0)
→ [f(s0)]×ss0
→ ([f(0)]×s0)×ss0
→ (s0×s0)×ss0
→ (s0×s0)+((s0×s0)×s0)
```

3. 2 A－TRSのreducer

A－TRSプログラムの実行を行なうreducerは．表明に従った戦略を実現するArpと，基本的に最外平行戦略を用いながらArpによって表明の処理を行なうRatから成る．各々の定義を以下に示す．

[定義] (Arp；Annotation replacement)

```
Arp: ATerm0 → Term0
Arp(at) = a, Rat-one(a), Rat-li(a)
                        at≡a, [a], {a} resp.
          f(Arp(at'1),...,Arp(at'n))
                        at≡f(at'1,...,(at'n))
          Rat-one(f(Arp(at'1,...,Arp(at'n)))
                        at≡f[at'1,...,(at'n)]
          Rat-li(f(Arp(at'1,...,Arp(at'n)))
                        at≡f{at'1,...,(at'n)}
```

The Implementation and Evaluation of Strategy Annotated Term Rewriting System(A-TRS)

Shinya Koga, Hiroshi Nunokawa, Shoichi Noguchi

Research Institute of Electrical Communication Tohoku University

［定義］(A-TRSのreducer；reduce annotated term)

```
Rat: ATerm0 → Term0
Rat(at) = if nf(at) then at
          elseif match(at) then Rat(Arp(AQ·Arp(θ)))
                where match(at)=θ, at=AP·θ, AP▷AQ
          else Rat(f(Rat-one(at1),···,Rat-one(atn)))
                where    at≡f(at'1,···,at'n)
                     or  at≡f[at'1,···,at'n]
                     or  at≡f{at'1,···,at'n}

Rat-one(at) = if nf(at) then at
              elseif match(at) then Arp(AQ·Arp(θ))
                   where match(at)=θ, at=AP·θ, AP▷AQ
              else f(Rat-one(at1),···,Rat-one(atn)))
                    where    at≡f(at'1,···,at'n)
                         or  at≡f[at'1,···,at'n]
                         or  at≡f{at'1,···,at'n}

Rat-li(at) = if nf(at) then at
             elseif at∈AConst then  Rat-li(Rat-one(at))
              else  Rat-li(Rat-one(f(Rat-li(at1),
                                  ···,Rat-li(atn))))
                    where    at≡f(at'1,···,at'n)
                         or  at≡f[at'1,···,at'n]
                         or  at≡f{at'1,···,at'n}
```

3．3　A－TRSの実現と評価

A－TRS及び従来方式のreducerは，LispマシンFACOM－α上にUTILispを用いてインタプリタ方式で作成した．実現に際しては項をLispのリストを用いて表現しており，また処理の効率を図るために，一度正規形であることを確かめた項にはその旨のタグを付けて区別している．

初期項や書き換え規則に付けた表明の効果を，従来方式の処理系と実行速度を比較することで実験した．その結果を図1，図2に示す．

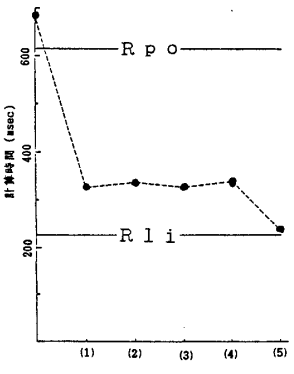

図1．f(s0×ss0)

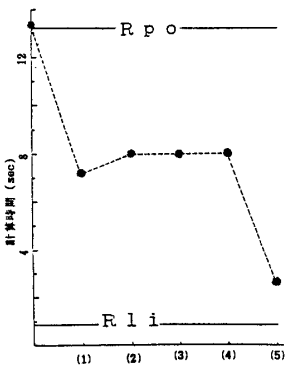

図2．f(ss0×ss0)

（1）初期項に表明（□）
（2）初期項に表明（{}）
（3）プログラム2
（4）プログラム3
（5）プログラム4

f(s0×ss0)を評価した場合の計算時間（単位：msec）を図1に，f(ss0×ss0)の場合の計算時間（単位：sec）を図2にそれぞれ示す．縦軸上にある測定点は表明をまったく用いなかった場合のものである．

図より，表明がない場合に多少のオーバヘッドが認められるものの，全体としては計算時間がRpoとRliの中間にあることが分かる．これは，表明の効果によって効率が向上した場合の例である．

3．4　応用

A－TRSの応用の一つとして，表明を利用したLisp関数の呼び出し機能を付加した．この機能を使えば，組み込み関数の自由な定義が可能である．Lispによる組み込み関数の応用例として図3に示すストリーム処理を記述したのが，次のプログラム5である．

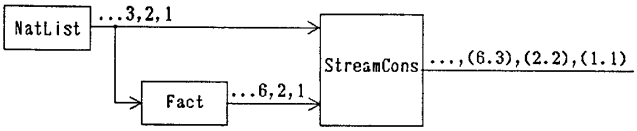

図3

```
NatList ▷ NumberList(1)          ；自然数発生部分
        NumberList(x) ▷ Next(x,NumberList({x+1}))
        [x]+[y] ▷ (PLUS x y)
Fact(Next(x,y)) ▷ Next(FactLisp[(x)],Fact(y))   ；階乗の計算部分
StreamCons(Next(x,y),Next(z,w)) ▷ Next(cons(x,z),StreamCons(y,w))
                                        ；ストリームの結合部分
PrintStream(Next(x,y)) ▷ Next(Print[(x)],PrintStream(y))
        Print[x] ▷ (PRINT x)             ；ストリームの印字部分
        cons([x],[y]) ▷ (CONS x y)
```

A－TRSプログラム5

4．まとめ

本論文では，TRSプログラムの中に記述する表明によって，書き換え戦略の制御を行なうことが可能な項書き換え系，A－TRSについてその処理系を実現し，評価を行なった．同じプログラムに対する実行時間の比較から，速度面についても従来方式より有利であることが確かめられた．

また，表明を利用することでLispによる組み込み関数の定義を行ない，これを用いてLispだけでは自然に記述することのできない処理をA－TRSによって記述した．

参考文献

[1]布川，富樫，野口：表明付き項書き換え系A－TRS－リダクション戦略の表明－，信学技法，Vol.88，No.143 (1988)，pp.77-86(COMP 88-43)

[2]布川，黒田，富樫，野口：項書き換え系の関数型言語への変換による実現，コンピュータソフトウェア，Vol.4，No.4(1987)

[3]二木，外山：項書き換え型計算モデルとその応用，情報処理，Vol.24，No.2(1983)，pp.133-146