6J－5 表明を持つ項書き換え系（A－TRS）を用いたストリーム・プログラミング

○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

われわれは戦略の表明を持つ項書き換え系A-TRSを提案し，その処理系を実現したことをすでに報告している[1][2]．本論文ではトークン・モデルに基づくストリーム（並行プロセス）[3]を，A-TRSを用いて記述する．

ストリームは関数型言語に順序関係を導入するものであり，プログラムの構造化手法としても重要である．さらにストリームを並行に動作するモジュール間のデータの流れとして捉えることで，プログラムの中に並列性を陽に表現することが可能となる．また，この論文で紹介するプログラムはプロセスの動作をすべてリダクション（書き換え）でシミュレートしており，処理系に対して通信機能を加えるなどの変更を何ら施すことなく実行が可能である．

2．項書き換え系，A-TRS

2．1項書き換え系と戦略

項書き換え系（Term Rewriting System; TRS）は関数型言語の計算モデルとしても用いられるが，それ自体をプログラミング言語とする見方もできる．この場合，TRS は記述や意味の把握が容易であるといった優れた特長を持つ．

従来，TRSの処理系を構築する際には正しさを重視し正規化戦略を採用するか，または実行効率を重く見て最内戦略を採るという，二者択一的な立場が採られてきた．これに対し，TRSをプログラミング言語として有効に利用するためには単一の戦略を用いるのでは不十分であるという観点から，ユーザがプログラム中に記述した表明によって戦略が規定される項書き換え系 A-TRSを提案し，その処理系を定義および実現した[1][2]．

2．2 A-TRSにおける書き換え

書き換え戦略を規定するために，プログラムである書き換え規則に表明（Annotation）を付けられたTRSをA-TRSと呼ぶ．表明は規則の中の関数記号に対して与えられ，最外並行戦略を指定するlazy表明（"[]"）と最左最内戦略を指定する eager表明（"{}"）とがある．

A-TRSプログラムの実行順序，即ち書き換え戦略は，正規化戦略である最外並行戦略を原則としている．書き換えようとする項の部分項が表明項の場合には，これを指定された戦略により正規形まで書き換えたのち全体の書き換えを行なう．プログラムの例として，階乗を計算するfactorial，および二つの引数を取りその対を返すpairに関する規則を図1に示す．

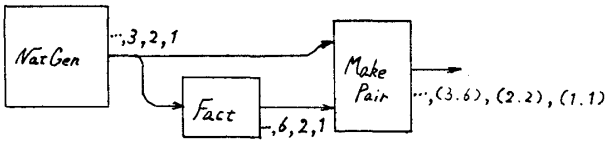
```
factorial(*x)
    ▷ if(eq(*x,0) ,1
                  ,(*x × factorial(*x - 1)))

pair(*x,*y) ▷ (*x . *y)

(実行例) pair(factorial[3],factorial(5))

        → pair(6,factorial(5))
        → (6 . factorial(5))
        ···→ (6 . 120)
```

図1　A-TRSプログラム（1）

一般に項は関数の適用を表わしており，対応する正規項をその値と考えれば，表明項はその関数適用の値を表わしていると見ることができる．

3．ストリームの記述

3．1ストラクチャ・モデルとトークン・モデル




```
MakePair(NatGen(),Fact(NatGen()))

  NatGen() ▷ NatStream(0)
    NatStream(*x) ▷ next(*x,NatStream([*x + 1]))

  Fact(next(*hd,*tl))
      ▷ next(factorial[*hd],Fact(*tl))

  MakePair(next(*hd1,*tl1),next(*hd2,*tl2))
        ▷ next(pair[*hd1,*hd2],MakePair(*tl1,*tl2))
```

図2　A-TRSプログラム（2）

ストリームのデータ構造をリスト状のものとして捉えると（ストラクチャ・モデル），プログラムは「リスト状のデータに対して先頭の要素から逐次的に処理を行なう操作」と解釈されるが，データ・トークンの流れとして捉えた場合には，いくつかのモジュールがデータの

Stream Programing in Strategy Annotated Term Rewriting System(A-TRS)

Shinya Koga,　Hiroshi Nunokawa,　Shoichi Noguchi

Research Institute of Electrical Communication Tohoku University

受渡しを行ないながら（パイプライン的に）処理を進めていく並行プロセスの記述と見ることができる（トークン・モデル）[3].

前頁のA-TRSプログラム１は，自然数とその階乗の対を要素とする無限ストリームを生成するものである（"next"はストリームの構成子）．このプログラムをトークン・モデルに当てはめて解釈すると，例えばFactの規則で右辺の表明項"factorial[*hd]"はFactという名のモジュールが受け取った自然数の階乗値を表わしており，従って右辺"next(factorial[*hd],Fact(*tl))"は全体として「モジュールFactが，変数hdに束縛された自然数の階乗の値を出力する」という状況を書いたものとして理解される．

３．２ TRSプログラムの構造化と並行プロセス

前節では，一つのプログラムで使われる関数記号に対し，それらを具体的な計算を行なうものとデータの流れを司るものとに弁別する手段を，表明とストリームが与えている事について述べた．プログラムの構成要素として，値を他の値に変換する通常の意味での関数と，他と値のやり取りをしながら関数を用いて計算を進めるモジュールの二つを考えることで，関数定義としての規則の羅列になりがちなTRSプログラムの，モジュールを単位とした構造化が期待できる．

３．３ A-TRSによる並行プロセスの記述

より一般的な並行プロセスを扱うために，まずストリームを，チャネルを通じたモジュール間の交信として捉える．チャネルはデータの通路であり，プログラム中に名前を用いて明示される．交信の形態としては，送信および受信側の準備がともに完了した時点でデータの受渡しがなされる同期通信を考える．

そのような考え方に基づいて前節のプログラム１を書き直した．これを以下に示す．

```
Proc(NatGen(0,*init),Fact(0,()),MakePair(0,()))

  NatGen:
    NatGen(0,*x) ▷ out(ch1(*x),NatGen(1,*x))
    NatGen(1,*x):done
          ▷ out(ch2(*x),NatGen(2,[*x + 1]))
    NatGen(2,*x):done ▷ NatGen(0,*x)

  Fact:
    Fact(0,()) ▷ in(ch1(),Fact(1,()))
    Fact(1,()):receive(*x)
          ▷ out(ch3(factorial[*x],Fact(1,())))
    Fact(1,()):done ▷ Fact(0,())

  MakePair:
    MakePair(0,()) ▷ in(ch2(),MakePair(1,()))
    MakePair(1,()):receive(*x)
          ▷ in(ch3(),MakePair(2,(*x)))
    MakePair(2,()):done ▷ MakePair(0,())

  Proc(out(ch1(*x),*proc1),in(ch1(),*proc2),*proc3)
    ▷ Proc([*proc1:done],[*proc2:receive(*x)],*proc3)

  Proc(out(ch2(*x),*proc1),*proc2,in(ch2(),*proc3))
    ▷ Proc([*proc1:done],*proc2,[*proc3:receive(*x)])

  Proc(*proc1,out(ch3(*x),*proc2),in(ch3(),*proc3))
    ▷ Proc(*proc1,[*proc2:done],[*proc3:receive(*x)])

  Proc(*proc1,*proc2,out(ch4(*x),*proc3))
    ▷ next(*x
          ,Proc(*proc1,*proc2,[*proc3:done]))
```

図３　A-TRSプログラム（３）

プログラム２で，例えばNatGenの定義中 "out(ch1(*x),NatGen(1,*x))"とあるのは「モジュールNatGenが，チャネルch1を通して変数xに束縛されたデータを出力しようとしている」状況を書いたものである．outに関する規則が与えられていないためこの項単独では正規項だが，モジュール間のデータの受渡しを行なう関数Procによってデータが Factに渡され，その際NatGenには送出が完了した旨を知らせるメッセージ"done"が与えられる．

ここでは並行プロセスを，並行に動作するモジュール（プロセス単位）とモジュール間のデータの受渡しを行なうもの（プロセス・マネージャ）とから成ると考えた．プログラム（３）では，個々のプロセス単位からの出力及び入力要求"out","in"を受けてプロセス・マネージャProcがデータの受渡しを行なう．その際プロセス単位はProcからメッセージが与えられるまで次の計算を行なうことができない．つまり，Procによってモジュール間の同期通信がシミュレートされている．

４．まとめ

元来A-TRSでは，"f([b],{c})"のようにある項が複数の表明項を部分項として持つ場合，二通りの解釈ができた．一つは各々を並列に書き換えてよいという宣言，すなわち並列性の明示と見るものであり，もう一つは引数のすべてに表明が与えられていれば，それをもって関数fがストリクトであると見る解釈である．従って，TRSプログラムに適切な表明付けを行なって並列性を明示しようとする際には，ストリクトネス解析が必要な場合もある．一方，いくつかのモジュールが相互にデータをやり取りしながら処理を進めていく並行プロセスの考え方は，プロセス単位で陽に並列性が表現され，また現実の問題との親和性もよい．

本論文では並行プロセスの考えに基づいたTRSプログラムの構造化，及び並列性の明確な表現について述べた．今後は実際的な問題の設計や，プロセス・マネージャに対する意味付けなどについても考えてみたい．

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－，信学技報，Vol.88，No.143，pp.77-86（COMP 88-43）（1988）

[2] 古賀，布川，野口：戦略の表明を持つ項書き換え系A-TRSの実現と評価，情報処理学会研究報告Vol.89，No.12，89-PL-20（1989）

[3] 田中：関数型プログラムにおけるストリーム計算，情報処理，Vol.29，No.8，pp.836-844（1988）