分散Lisp系を用いたメッセージ交換リダクション

1Q−4

寺島 貴之*　布川 博士**　野口 正一*

*東北大学応用情報学研究センター　**東北大学電気通信研究所

１．はじめに

本稿では，関数型言語の抽象インタプリタである表明付き項書き換え系Ａ−ＴＲＳ[4]のリダクションを学内ＬＡＮによるワークステーションネットワークを用いて分散的に実行するリデューサの実現について述べる．このリデューサは分散Lisp系により記述されており，ネットワーク上に分散された関数プロセス間のメッセージ交換によりリダクションを実行する．

２．メッセージ交換によるリダクション[3]

ＴＲＳプログラムにおいて，正規形はＴＲＳのリデューサにより求める．その際に従来広く用いられてきた方式はデータ交換方式である．これはリデューサに対してリダクションの対象となる項の形のみをデータとして与える方式である．例えば，並行最外戦略のリデューサがRpoであるとき，f(s0*ss0)の正規形はRpo(f(s0*ss0))で求められる．

メッセージ交換方式は，ＴＲＳ上の各関数記号fにそれぞれ関数プロセスproc(f)を割当て，それらの間のメッセージ交換によりリダクションを行う方式である．この方式では，戦略はメッセージにより指定することができる．例えばf(s0*ss0)の並行最外戦略による正規形はproc(f)に対してメッセージpo,s0*ss0を送ることにより求めることができる．

メッセージ交換を用いたリデューサの利点には以下がある．

(1)プロセスが分散されていても効率よく実行できる

(2)最外戦略のリダクションを効率的に実行できる

(3)メッセージの種類を増やすことにより，戦略以外にも種々の制御が可能になる

以後リダクションとは，特に断らないかぎり，メッセージ交換によるリダクションを指す．

３．リダクションシステムの実現

３．１リダクションシステムの構造

メッセージ交換を用いたリダクションシステムでは，複数の関数プロセスがネットワーク上に分散して配置され，それらは互いにプロセス間通信を行う．

関数プロセスは，ＴＲＳ上の１つの関数記号に対応した処理を行うプロセスであり，その関数記号に関する書き換え処理，表明処理を内部に持つ．

リダクションの実行時には，関数プロセスは，リダクションを行う項の構造を木構造で表したときの１つのノードに対応する．関数プロセスが外部から項の形と戦略を受信すると，戦略に従って項の書き換えや子ノードのリダクションのための実行制御を行う．

あるノードに対応した関数プロセスが，その子ノードのリダクションを行う際には，子ノードに対応した関数プロセスに対して，子ノードのリダクションのための項の形と戦略をメッセージとして送信し，リダクションの結果を待って受信すれば良い．

プロセス間通信を実現するためのネットワークは，自由な相手に，確実にプロセス間通信が行えればよく，特に媒体を問わない．

３．２リダクションシステムの仕様

本システムは，学内ＬＡＮを用いたワークステーションネットワーク上に構築されており，各ワークステーション上に複数の関数プロセスが分散して配置され，それらの間のメッセージ交換によりリダクションを行う．

記述言語は分散Lisp系[2]を用いている．分散Lisp系は通常のLispシステムに通信機能とＧＵＩの機能を付加したものであり，Lispにより分散システム上に様々な環境を構築することができる特徴を持つ．

３．３関数プロセスの構造

以前我々が作成した，トランスピュータを用いたリダクションシステム[5]では，関数プロセスへの関数記号の割り当てや表明処理の設定は，リダクションの実行前に静的に行った．

本システムは記述言語として分散Lisp系を用いたことにより，関数プロセス内部で実行される書き換え処理，表明処理，通信プロトコルのいずれをも外部からデータとして与えることができ，関数プロセスへの関数記号の割り当てや表明処理の追加などを動的に行うことができる．図１に本システムの関数プロセスの構造を示す．各関数プロセスは，自身を含む各関数プロセスに対して自由に通信を行うことができる．実際のシステムでは，TCP/IPのソケットを用いて通信を行う．

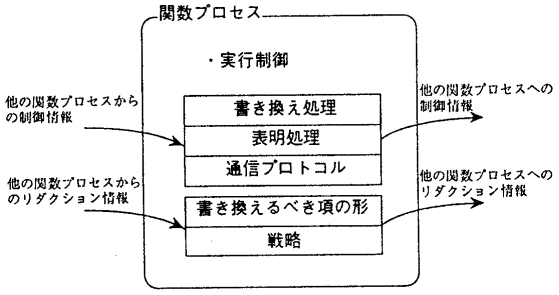

図１．関数プロセスの構造

Message Passing Reduction using Decentralized Lisp Interpreters

Takayuki TERASHIMA*,Hiroshi NUNOKAWA**,Shoichi NOGUCHI*

*Research Center for Applied Information Sciences,Tohoku University

**Research Institute of Electorical Communication,Tohoku University

3．4制御方式

実現したシステムは，[3]によるリデューサの定義に従って作成されているが，実現上の最大の問題は，関数プロセスの動的生成，消滅を行うか否か，という点にある．

リダクションを行う際に，リダクションを行っている項の木構造のノードのそれぞれに，1つの関数プロセスを対応させる．従って一般に，同一の関数記号に対応した関数プロセスが同時に複数個存在する．また，リダクションの過程において項の形は変化するため，必要とされる関数プロセスの個数も変化する．このような理由により，関数プロセスは動的に生成，消滅を行えるのが望ましい．

しかしながら，リダクションシステムの実現の際には，関数プロセスの動的な生成にはかなりの時間的コストを要するので，実際には関数プロセスを生成せずにリダクションを実行できるような実現の手法が必要となる．

そこで本システムでは，あるノードに対応した関数プロセスが，その子ノードのリダクションを行う際に，リダクションの結果を単に待ち続けるのではなく，子ノードにメッセージを送信した時点で一旦そのノードのリダクションを中断し，子ノードからリダクションの結果が返ってきたら，そのノードのリダクションを再開する，という方法をとった．

これは，リダクションの途中における関数プロセスのうち，実際にリダクションを実行しているのは木構造の末端にあたるノードの関数プロセスだけであり，他は子ノードのリダクションの結果を待っている状態にある，という事実に基づく方法である．

このような方法をとると，本来，関数プロセスが通信待ち状態にあるときに他のノードのリダクションを行えるようになり，1関数記号に対して1関数プロセスを用意すれば，リダクションの実行ができるようになる．

制御の例として図2に最内リダクションの実行制御の手順を示す．

ステージ1

(1)リダクションのためのメッセージの受信

(2)子ノードのリダクションのためのメッセージ送信

ステージ2

(1)子ノードからのリダクション結果の受信

(2)自ノードのリダクション

(3)自ノードのリダクション結果の送信

a)書き換えが行われたとき
自ノードへのメッセージ送信

b)書き換えが行われなかったとき
親ノードへのメッセージ送信

図2．最内リダクションの実行制御

3．5プロトコル

3.4節に述べた制御方式をとったことにより，関数プロセス間で通信を行う際には，親ノードに当たる関数プロセス（親関数プロセス）にリダクションの結果が返送されたときに，それを元の項に戻す必要がある．このため，単に項の形と戦略を送る他に，親関数プロセス内で使用される情報を付加して送る必要がある．

4．分散Lisp系による実現

図3に示す書き換え規則の関数記号fを，分散Lisp系を用いて実現した関数プロセスproc(f)の例を図4に示す．注釈にあるステージとは，図2の実行制御のステージを表す．また，変数dirは，通信の方向を表し，dnなら子ノードへの，upなら親ノードへの通信路を表す．

```
x＋0▷x          x＋s（y）▷s（x＋y）
x＊0▷0          x＊s（y）▷（x＊y）＋x
f（0）▷s（0）   f（s（x））▷f（x）＊s（x）
```

図3．書き換え規則の例

```
(defun funcprocess ()
(progn
 (setq link (OpenConnections 'susuki))  ;ワークステーションsusukiへのリンク
 (setq loop (cond ((null link) nil) (t t)))
 (setq newid 0)
 (setq waitque ())
 (setq fpid 'funcF)
 (while loop
  (progn
      (setq commingmsg (recvmsg link))
      (setq newid (% (inc newid) 1000))
      (getinformation commingmsg)     ;id情報の取り出し
      (cond
       ((equal stat 'break) (setq loop nil))
       ((equal dir 'dn)
        (cond
         ((equal stat 'inm)
           (innermost_stage1))          ;最内戦略のステージ1の実行
         ((or (equal stat 'otm) (equal stat 'otm_one))
           (outermost_stage1))))       ;最外戦略のステージ1の実行
       ((equal dir 'up)
        (progn
         (setq tdata (reconstruct tdata fpid))
         (cond
          ((equal stat 'inm)
            (innermost_stage2))        ;最内戦略のステージ2の実行
          ((or (equal stat 'otm) (equal stat 'otm_one))
            (outermost_stage2))))))) ;最外戦略のステージ2の実行
(close link)))
```

図4．分散Lisp系による関数プロセスの実現例

5．まとめ

本稿では，表明付き項書き換え系A−TRSのリダクションをワークステーション上で分散的に実行するリデューサの実現について述べた．

現時点では，3.4節，3.5節に述べた制御方式とプロトコルにより，リダクションの実行が行えることを確認している．

今後は，分散並列リダクションを実現し，さらに，関数プロセスに外部からメッセージを与えることによる関数プロセスの動的な配置を実現し，その効果の確認を行う予定である．

参考文献

[1]布川博士,古賀信哉,野口正一:戦略の表明を持つ項書き換え系A−TRSの実現と評価,情処研報Vol.89,No.12(1989)89-PL-20

[2]布川博士,三石大,宮崎正俊,野口正一:分散環境記述のための言語系,第45回情処全大,発表予定

[3]布川博士,野口正一:プロセス間でのメッセージ交換を用いた項書き換え系のリダクション,信学技報ss89-28(1990)pp.29-38(ソフトウェア科学会関数プログラミング研究会資料FP-90-04(1990))

[4]布川博士,富樫敦,野口正一:表明付き項書き換え系A−TRSーリダクション戦略の表明,信学技報COMP88-43,Vol.88,No.143(1988)pp.77-86

[5]寺島貴之,布川博士,野口正一:トランスピュータを用いた項書き換え系の分散リダクション,情処研報Vol.91,No.60(1991)pp.141-148