プログラミング言語　25－1
（１９９０．６．８）

AgentCellを用いたリダクションの視覚化

布川 博士　　野口 正一

東北大学電気通信研究所

本稿では，関数型言語の抽象インタプリタである項書き換え系（ＴＲＳ）のリダクションを各ステップごとに視覚化し，ユーザーと対話的に実行する対話的リダクションシステムの構成法，その試作システムの概要について述べる．本方式はＴＲＳ上の各関数記号に対してAgentCellと呼ばれるものを割り当てる．AgentCellはリダクションをつかさどるreductionAgent，図形的外観であるview，reductionAgent，view，ユーザーとの対話の制御を行なうviewAgentの３つからなる．リダクションは各reductionAgent間の通信で行なわれ，同時にview，viewAgentらとの通信により視覚化が行なわれる．

Visualization of Reductions Using AgentCell

Hiroshi Nunokawa and Shoichi NOGUCHI

Research Institute of Electorical Communcation TOHOKU University

2-1-1 Katahira, Sendai 980, JAPAN

Abstract

This paper shows a method of visualizing reductions in Term Rewriting Systems. We have used an AgentCell which consists of three components ,i.e. reductionAgent, view and viewAgent. These are customized instances of the MVC model in smalltalk80. The visualization and reduction is performed interactively by the internal/external communication of AgentCells. We implemented view and viewAgent by using HyperCard, and reductionAgent is made in Scheme on a SUN3 workstation. The communications between view/viewAgent and reductionAgent is based on actual local area network.

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項書き換え系（ＴＲＳ）のリダクションを各ステップごとに視覚化し，ユーザーと対話的に実行する対話的リダクションシステムの構成法，その試作システムの概要について述べる

ＴＲＳ自身は，それをプログラミング言語としてみると，記述の容易さ，意味の把握のしやすさ等優れた言語である．また，記述の抽象度が高いため，多くの関数的な言語の汎用の処理系として使うこともでき，ＴＲＳによるＬｉｓｐインタプリタの記述もすでに行なわれている[3]．また，等式論理で与えられる数学的意味[4]と操作的意味のセマンティックギャップがないため(半)自動証明システムや種々の検証システムのためのリダクションエンジンとしても用いられることが多い．本方式によって作成されたＴＲＳの対話的リダクションシステムは以上のようなＴＲＳに基づいて作成されているプログラミングシステム，証明／検証システムのユーザーインターフェースとして広く使用することが出来る．

本方式はＴＲＳ上の各関数記号に対してAgentCellと呼ばれるものを割り当てる．AgentCellはリダクションをつかさどるreductionAgent，図形的外観であるview，reductionAgent，view，ユーザーとの対話の制御を行なうviewAgentの３つからなる．リダクションは各reductionAgent間の通信で行なわれ，同時にview，viewAgentらとの通信により視覚化が行なわれる．本方式はreductionAgentに対して適当なviewが定められるものであればとくにＴＲＳのリダクションに限定する必要はなく，広く一般のリダクションシステムへの応用も容易である特長を有する．

2 項書き換え系

2．1 項書き換え系（ＴＲＳ）

本節では，通常行なわれるＴＲＳに関する厳密な定義は省略する．詳細は[ ]等を参考にされたい．

ＴＲＳにおける計算は与えられた項に対し適用できる規則をもとめ，次々と書き換えを行なう操作である．書き換え規則を適用できない項のことを正規形とよび，これがＴＲＳの計算における結果である．一般に，一つの項中には書き換え規則が適用できる部分項（リデックスという．部分項には自分自身も含む）が複数存在する．複数のリデックスから書き換えるべきリデックスを指定する方法をリダクション戦略という．リダクション戦略により，有限回の書き換えで求められる場合と求められない場合があり，また，正規形を求めるために要する書き換えの回数もパスも異なってくる．

本論文では以下の２つの条件を満たすＴＲＳを扱うものとする．これはＴＲＳが合流性を持つための十分条件である[5]．

条件１（重なりがないこと）

ＴＲＳ中の規則$P_1 \triangleright Q_1$，$P_2 \triangleright Q_2$（同一の規則の場合も許す）に対してもP_1がP_2の変数でない部分項（P_2自身も許す）と単一化可能でないこと．

条件２（線形性）

ＴＲＳのすべての規則 P▷Q において，Ｐ中に同じ変数が２度以上出現しないこと．

以下本論文では，重なりがなくかつ線形なＴＲＳのみを扱い，単にＴＲＳと呼ぶ．ＴＲＳプログラム１は上記の条件を満たすＴＲＳの例である．

ＴＲＳプログラム１

f(0) ▷ s0	x+0 ▷ x
f(s(x)) ▷ f(x)*s(x)	x+s(y) ▷ s(x+y)

ＴＲＳプログラム１を用いたリダクション例

f(0+s0)	f(0+s0)
PO→f(s(0+0))	LI→f(s(0+0))
PO→f(0+0)*s(0+0)	LI→f(s0)
PO→f(0)*s0	LI→f(0)*s0
PO→s0*s0	LI→s0*s0
(a)並行最外ﾘﾀﾞｸｼｮﾝ	(b)最左最内ﾘﾀﾞｸｼｮﾝ

2．2 リデューサ（reducer）

書き換えを実際に行なうためのインタプリタ（ＴＲＳプログラムの処理系）をreducerという．並行最外リダクションを行なうためのreducerであるRpo（reduce parallel outermost）は以下にように定義される．

［定義］（並行最外ﾘﾀﾞｸｼｮﾝ戦略のreducer，Ｒｐｏ）

```
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)
```

定義中nf(t)はtが正規系か否かを判定する関数である．match(t)はすでに与えられているＴＲＳから，ｔ自身に適用可能な規則Ｐ▷Ｑを求め，Ｐθ≡ｔとなる置換θを返す関数である．本論文で扱っているＴＲＳは合流性を満たすためmatchが適用できる規則を探す順序を特に考慮する必要はない．ｔ自身に適用可能な規則がない時はfalseを評価結果として返す．rewrite(t)はmatch(t)によって求められた規則Ｐ▷Ｑ，置換θを用いＱθを返す関数である．

［定義］（最左最内ﾘﾀﾞｸｼｮﾝ戦略のreducer，Ｒｌｉ）

```
Rli(t) = if nf(t) then t
  elseif t∈Const then Rli-sub(a)
  else   Rli-sub(f(Rli(t1), ... ,Rli(tn))
                        where t≡f(t1, ... ,tn)

Rli-sub(t) = if nf(t) then t
  elseif match(t) then Rli(rewrite(t))
                  else t
```

定義中，t∈Const はｔが定数か否かを判定する関数である．Ｒｌｉの定義中，Rli-sub(f(Rli(t1), ... , Rli(tn)) において各Ｒｌｉが並列に動作すれば，これは並行最内戦略のreducerとなる．これを各々Ｒｐｉ（reduce parallel innermost），Ｒｐｉ－ｓｕｂと呼ぶ．

２．３　メッセージ交換を用いた最外ﾘﾀﾞｸｼｮﾝ[6]

２．２節で述べたreducerはRpo,Rpo-one,Rli,Rli-subといったプロセスが再帰を用いた制御でリダクションを実行している．これに対して我々はすでにＴＲＳ上の各関数記号に割り当てられたプロセス間でのメッセージの交換を用いて実行するメッセージ交換により，リダクションを行なうために各関数に割り当てるべきプロセスの外部構造，プロセス間のメッセージの交換の仕方（リダクションのためのプロトコル），及びプロセスの内部構造を明確に定め，その方法により正しく２．２節のreducerと同じ結果を得ることを示している[ ]．

２．２節で定義したＲｐｏでは，f(0+s0)の正規形はRpo(f(0+s0))で求められる．メッセージ交換を用いたリダクションでは，これをfに割り当てたプロセスproc(f)に対してメッセージ ss0,'出力先','po' を送ることによって求める（これをproc(...){0+s0,CRT,'pi'→}proc(f)のように記述する）．'po'はｾﾚｸﾀﾄｰｸﾝと呼ばれ，各プロセスがどの様な動きをすればよいかを外部から指示するために用いられる．'po'は並行最外戦略でリダクションすることを，'po-one'(Rpo-oneの相当する)は一度だけ最外戦略でリダクションすることを指示する．

例中，porc(KeyBoard)，proc(CRT)は各々，入力及び出力のためのプロセスである．

```
proc(KeyBoard){0+s0,CRT,'po'→}proc(f)
  => wait(proc(f)){0,s0,h,'po-one'→}proc(+)
  => wait(proc(f)){}active(proc(+))
  => wait(proc(f)){←s(0+0)}
  => active(proc(f))
  => active(proc(f)) proc(*)
  => {0+0,f(0+0),CRT,'po'→}proc(*)
          ...
  => {s0*s0→}proc(CRT)
```

以下にメッセージ交換による最外戦略のためのプロトコルを示す．

・match(f(g1, ... ,gn)) が FALSE以外の時

```
{g1, ... ,gn,ret_token,'po'→}proc(f)
=>{s1, ... ,sm,ret_token,'po'→}proc(g)
(where rewrite(f(g1, ... ,gn)≡g(s1, ... ,sm))
                                        (PO-1)
```

・match(f(g1, ... ,gn)) が FALSEの時

```
{g1, ... ,gn,ret_token,'po'→}proc(f)
      {s11, ... ,sm1,h1,'po-one'→} to proc(s1)
=>wait(proc(f))          ...
      {sn1, ... ,smn,hn,'po-one'→} to proc(sn)
```

(where $g_i \equiv s_i(s_1, \ldots, s_{mi})$)

(PO-2)

・proc(f)が待ち状態の時

{h_1→}

... wait(proc(f)) =>

{h_n→}

{$h_1, \ldots, h_n$,ret_token,'po'→}proc(f)

(PO-3)

・match(f($g_1, \ldots, g_n$)) が FALSE以外

{$g_1, \ldots, g_n$,ret_token,'po-one'→}proc(f)

=>{g($s_1, \ldots, s_m$)→}ret

(where rewrite(f($g_1, \ldots, g_n$)≡g($s_1, \ldots, s_m$))

(PO_ONE-1)

・match(f($g_1, \ldots, g_n$)) が FALSE

{$g_1, \ldots, g_n$,ret_token,'po-one'→}proc(f)

{$s_{11}, \ldots, s_{1m1}, h_1$,'po-one'→} to proc($s_1$)

=>wait(proc(f)) ...

{$s_{n1}, \ldots, s_{nmn}, h_n$,'po-one'→} to proc($s_n$)

(where $g_i \equiv s_i(s_1, \ldots, s_{mi})$)

(PO_ONE-2)

・proc(f)が待ち状態の時

{h_1→}

... wait(proc(f)) => {f($h_1, \ldots, h_n$)}ret

{h_n→}

(PO_ONE-3)

2．4 メッセージ交換を用いた最内ﾘﾀﾞｸｼｮﾝ[6]

2．3節同様，メッセージ交換を用いた最内リダクションでは，f(0+s0)の正規形はproc(f)に対してメッセージ 0+s0,結果の出力先,'pi' を送ることによって求める．下記に，f(0+s0)がf(s0)にリダクションされる様子を示す．'pi'がｾﾚｸﾀﾄｰｸﾝである．

proc(KeyBoard){0+s0,CRT,'pi'→}proc(f)

=> wait(proc(f)){0,s0,h,'pi'→}proc(+)

{'',h1,'pi'→}proc(0)

=> wait(proc(f)){}wait(proc(+))

{'',h1,'pi'→}proc(s)

{←0}

=> wait(proc(f)){}wait(proc(+))

{}wait(proc(s)){'',h,'pi'→}proc(0)

{←0}

=> wait(proc(f)){}wait(proc(+))

{}wait(proc(s)){←0}

{←0}

=> wait(proc(f)){}wait(proc(+))

{←s0}

=> wait(proc(f)){}

wait(proc(s)){0,0,h,'pi'→}proc(+)

...

=> wait(proc(f)){←s0}

...

=> {s0*s0→}proc(CRT)

メッセージ交換を用いて並行最内戦略でのリダクションを行なうためには以下のプロトコルを用いる．

・aが定数の時

・match(a) が FALSE以外の時

{'',ret_token,'pi'→}proc(a)

=>{$s_1, \ldots, s_m$,ret,'pi'→}proc(g)

(where rewrite(f($g_1, \ldots, g_n$)≡g($s_1, \ldots, s_m$))

(PI-1)

・match(a) が FALSEの時

{'',ret_token,'pi'→}proc(a) => {NF(a)→}ret

(PI-2)

・fが定数でないとき

・isNF($g_1, \ldots, g_n$) が TRUE

かつmatch(f($g_1, \ldots, g_n$)) が FALSE以外の時

{$g_1, \ldots, g_n$,ret_token,'pi'→}proc(f)

=>{$s_1, \ldots, s_m$,ret,'pi'→}proc(g)

(where rewrite(a)≡g($s_1, \ldots, s_m$))

(PI-3)

・isNF($g_1, \ldots, g_n$) が TRUE,

かつmatch(f($g_1, \ldots, g_n$)) が FALSE

{$g_1, \ldots, g_n$,ret_token,'pi'→}proc(f)

$$\Rightarrow \{NF(f(g_1, \ldots, g_n)\rightarrow\}ret \quad \text{(PI-4)}$$

・isNF($g_1, \ldots, g_n$) が FALSE

$\{g_1, \ldots, g_n, ret\_token, 'pi'\rightarrow\}proc(f)$

$$\{s_{11}, \ldots, s_{1m1}, h_1, 'pi'\rightarrow\} \text{ to } proc(s_1)$$
$$\Rightarrow wait(proc(f)) \quad \ldots$$
$$\{s_{n1}, \ldots, s_{nmn}, h_n, 'pi'\rightarrow\} \text{ to } proc(s_n)$$
$$(\text{where } g_i \equiv s_i(s_1, \ldots, s_{mi})) \quad \text{(PI-5)}$$

・proc(f)が待ち状態の時

$\{h_1\rightarrow\}$

$\ldots$ wait(proc(f)) => $\{f(h_1, \ldots, h_n)\}ret\_token$

$\{h_n\rightarrow\}$

(PI-6)

以上２．２節，２．３節で述べたreducerと２．２節で述べた従来のreducerとの間では以下の性質が成立する[ ]．これは，メッセージ交換によるreducerが正しくリダクションを行なうことを示している．

性質１[6]

（１） $Rpo(f(t_1, \ldots, t_n)) \Rightarrow \ldots \Rightarrow s$

iff

$\{t_1, \ldots, t_n, R, 'po'\rightarrow\}proc(f) \Rightarrow$

$\ldots \Rightarrow \{NF(s)\}proc(R)$

（２） $Rpi(f(t_1, \ldots, t_n)) \Rightarrow \ldots \Rightarrow s$

iff

$\{t_1, \ldots, t_n, R, 'pi'\rightarrow\}proc(f) \Rightarrow$

$\ldots \Rightarrow \{NF(s)\}proc(R)$

(NF(s)はsが正規項であることを示す)

3 AgentCellを用いた視覚化

3．1 AgentCell

AgentCellとは，ＴＲＳにおけるリダクションを視覚的，対話的におこなうために，２章で述べたproc(f)（AgentCellではとくにreductionAgentと呼ぶ）に対して図形的外観であるview，ユーザーやviewの制御行なうviewAgentを加えたものである．すなわち，関数記号fに割り当てられる，AgentCell(f)は以下のようになる．

AgentCell(f)

=<reductionAgent(f) view(f) viewAgent(f)>

AgentCellの各要素はそれぞれ，オブジェクト指向言語smallTalk80のＭＶＣモデルのモデル，view，コントローラーに対応する．リダクションは各reductionAgent(f)間の通信で，２．３節，２．４節で述べたプロトコルを用いて実行される．reductionAgent(f)は自分自身がリダクションされた等の状態をview(f)およびviewAgent(f)と通信をすることを除いて，２．３節，２．４節で述べたproc(f)と同様である．

view(f)は関数記号fの図形的外観であり，現在のシステムでは図３．１に示す外観と，通常の文字で表示する２つの外観をもたせている．この２つのviewはユーザーが選択できるようになっている（図４．３の文字列化に対応）．ユーザーは各関数記号に割り当てられたviewを直接操作することにより，対話的にリダクションを行なうことができる．

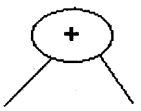

図３．１ view(＋)

3．2 AgentCellの通信

AgentCellは他のAgentCellと通信を行なうことにより，２章で述べた方法によるリダクションを行なうと同時に，自分自身のviewを表示してゆく．したがって，AgentCell間通信のプロトコルを定めるためには，２章で定めた各規則に対して，その規則が適用されたときに，view，viewAgentの間でのメッセージ交換のプロトコルを定めればよい．例えば，AgentCell(f)がメッセージ 0+s0,CRT,'po'を受け取ったとき行なうべきことは，

（１）reductionAgent(f)に対してメッセージ0+s0,CRT,'po'を送ると同時に，自分自身のview(f)を表示する．

（２）AgentCell(+)に対して0，s0,を引数として用いて表示するようにメッセージdrawを送る．

（３）AgentCell(+)がメッセージdrawを受け取ったときは，自分自身のviewを表示し，さらにメッセージの引数に対してviewを表示するようにメッセージを送る．

また，rewriteによって書き換えた後は自分自身のview(f)を消去するとともに，agentCell(+)に対してもそのviewを消去メッセージを送る必要がある．

このプロトコルは，４章で実現されているような，部分項に対してpo等のメッセージを送る場合でもそのまま適用できる．ここでは(PO-1)に対してのみ示したが，同様にして，２章で述べた各リダクションのためのプロトコルの規則にviewを表示するためのプロトコルを付加することができる．

4　実現手法

われわれは３章で述べた方法に基づき対話的，視覚的なリダクションシステムを試作した．構成を図４．１に示す．システムの構築するにあたりわれわれは，グラフィカルユーザーインターフェースの構成法の一つであるシーハイムモデル[2]を採用した．試作したシステムはシーハイムモデルの各対応する部分にAgentCellのview，viewAgent，reductionAgentを分散的に配置したものである．また，試作にあたって，シーハイムモデルのアプリケーションインターフェースに対応する部分をＨＬＰ（後述）を介して通信を行なうネットワークによって結合した．view及viewAgentはApple社のMacintoshⅡ上のHyperCard[1]及びそのスクリプト記述言語であるHyperTalk[1]を用いて作成されている．

ＨＬＰ(HyperText Linking Protcol)は通信回線を用いてHyperCardの各種動作を制御するための通信プロトコルであり，HyperCard側ではMacintoshⅡのシリアルポートを制御するHyperTalkの外部コマンド(XCMD)として作成した．また，reductionAgent側ではsun3上のScheme[7]を用いて作成した．ＨＬＰ間は学内ネットワークであるＴＡＩＮＳを用いている．

各reductionAgentはsun3上にSchemeを用いて作成されている．図４．３，図４．４，図４．５に本システムの実行例を示す．図４．５に示すように，システムの実現にあたっては，ルートの関数記号（図４．５中のｆ）のみでなく，各AgentCellがPO,PO-ONE,PIの各メッセージを受け取れるように拡張してある．したがって，ユーザーが対話的に任意の部分項を任意の戦略でリダクションする事が可能である．また，得られた結果はＨｙｐｅｒＣａｒｄ上に書き込まれるため（すな

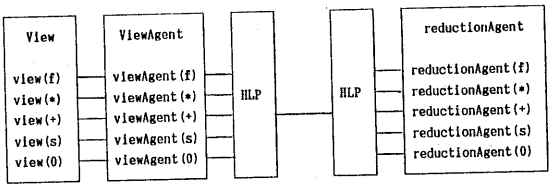

図４．１　試作システムの構成

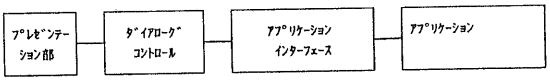

図４．２　シーハイムモデル[2]

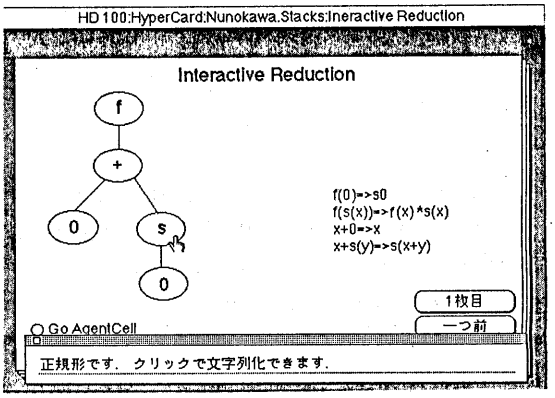

図４．３　実行例（１）

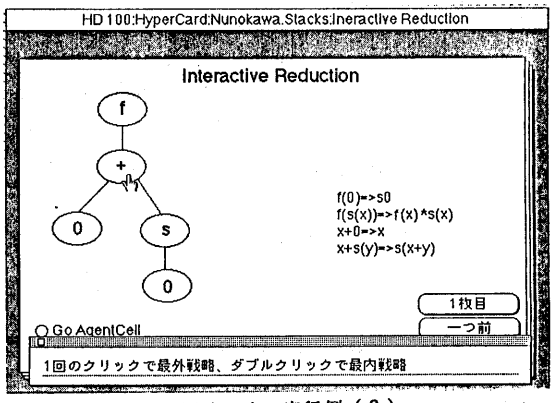

図４．４　実行例（２）

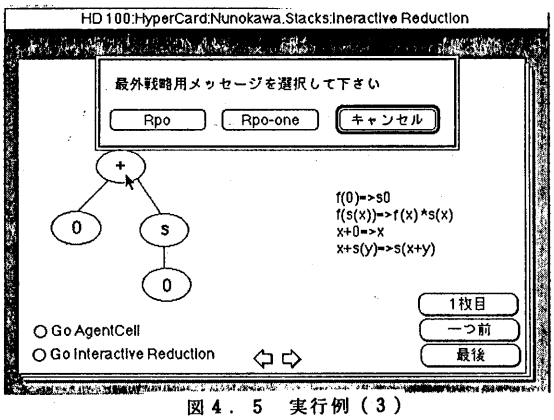

図４．５　実行例（３）

わち，各対話における結果であるAgentCellの構造がカードに保存されるため），その結果をＨｙｐｅｒＣａｒｄの各機能で利用することができる．特にrecent機能を用いれば，リダクションの任意の時点にさかのぼり，その時点からまた，別の戦略で実行を再開することが可能である．

5　まとめ

本稿ではメッセージ交換を用いてリダクションを行なう方法の一つの応用として，リダクションを視覚的かつ対話的に行なうシステムを示した．本稿ではそのためにAgentCellをいう考え方を導入し，その間の通信によって本稿の目的とするシステムの構築を行なった．また，AgentCellの各機能を学内ネットワークを介して分散的に配置することによりプロトタイプを作成し，その実例を示した．

本システムは，現在，単にリダクションを行なうシステムであり，それを用いた証明システムを構築するには至っていない．本システムを本格的な証明，検証システムへ応用する際に必要となる機能をAgentCellに埋め込み，AgentCell自身の機能を高めることは今後の課題である．また，本システムで用いたグラフィカルユーザーインターフェースの構成法，特に図４．１に示す構成法は，本稿の目的としたリダクションシステムのみでなく，ネットワークを介して利用する多くのシステムの汎用的なユーザーインターフェースとして利用することができる．現在我々は，この考え方に基づき，大型計算機上のＬｉｓｐシステムをこのユーザーインターフェースを用いて利用するシステムも構築中である．このように図４．１を汎用のシステムをして用いるときのAgentCellのあり方，機能を明確にすることは非常に興味深い課題である．

参考文献

[1]HyperCard User's Guide,Apple Computer Inc(1987)

[2]Lowgre,J:History,State and Feature of User Interface Management System,SIGCHI Bulletin,Vol.20,No.1(1988),pp.32-44

[3]Hoffmann,C.M and O'Donnel,M.J :Programming with Equations. ACM Trans.on Prog. Lang. Syst.

[4]稲垣,坂部:抽象データ型の仕様記述法の基礎（１）－多ソート代数と等式論理－，情報処理，Vol.25,No.1(1984)，pp.47-53

[5]O'Donnell,M. :Computing in Systems Described by Equations. Lecture Notes in Comput. Sci. No.58, Springer(1977)

[6]布川，野口：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,pp29-38

[7]MIT Scheme Reference Manual(1989),MIT