LISP における変数の結合方式について

梶原　信樹*・戸田　学**・井上　克司***
高浪　五男***・谷口　弘***

A Note on Binding Strategies of the Variables in LISP

Nobuki KAJIHARA*, Manabu TODA**, Katsushi INOUE***
Itsuo TAKANAMI*** and Hiroshi TANIGUCHI***

Abstract

The purpose of this paper is to survey typical binding strategies of the λ-variables in LISP, and to improve several difficulties of some of them. We first survey "deep binding" and "shallow binding", which make use of a-list and p-list, respectively. We then survey "casual shallow binding" which makes use of the advantages of deep and shallow bindings. We finally point out several difficulties of casual shallow binding, and describe a way to improve them.

1. まえがき

LISPは，リスト処理，記号処理に適しているだけでなく，ゲーム，人工知能，言語理論，グラフ理論などの分野に現れる組合せ理論的探索問題を解くのに極めて適しており，また解くためのアルゴリズムを開発するのに役立つ言語である．このような有用性のために，既に種々のLISPシステムが開発され実用に供されている．[1)~5)]

LISPシステムを設計する際のひとつの問題として変数の結合方式がある．これは，LISPシステムの高速化に関係する重要な問題である．

従来よく用いられてきた変数の結合方式としては，a-リスト (association list) を用いるDeep-Bindingとp-リスト (property list) を用いるShallow-Bindingとがあり，それぞれ一長一短がある（本文参照）．最近，Deep-BindingとShallow-Bindingの長所を組合せて用いたCasual Shallow-Bindingと呼ばれる新しい変数の結合方式が提案された．[6)]

本稿の目的は，上記の各種の変数の結合方式を展望するとともに，新しく提案されたCasual Shallow-Bindingの問題点を指適し，その改善策などについて述べることである．

なお，本稿では，読者はLISPに関する基本的な知識を有しているものとして議論を進める．必要であれば，文献7～10）などを参照されたい．

* 大阪大学大学院
** 大学院　電子工学専攻
*** 電子工学科

2. a-リストを使う Deep-Binding

まず，a-リストを使うDeep-Bindingについて述べる．この方式では，変数とその値の結合状態（これを環境と呼ぶ）は，a-リストに対リストの形でたくわえられている．たとえば，次の様なa-リスト

$$((X\cdot A)(Y\cdot(ABC))(Z\cdot NIL)) \qquad \cdots\cdots(1)$$

は，変数xの値はA，変数yの値は(ABC)，変数zの値はNILであることを表わしている．したがって(1)式の環境で式

$$\text{car}\ [y] \qquad \cdots\cdots(2)$$

を評価すると値は

$$\text{car}\ [(ABC)]=A \qquad \cdots\cdots(3)$$

となる．

a-リストaは，一般的には次の様に書ける．

$$a=((x_1\cdot v_1)(x_2\cdot v_2)\cdots(x_n\cdot v_n)) \qquad \cdots\cdots(4)$$

x_i は変数名，v_i は環境 a における変数 x_i の値である．変数名 x と，その値 v のドット対

$$(x \cdot v) \qquad \cdots\cdots(5)$$

を変数 x の結合 (binding) と呼ぶことにする．環境 a における変数 x の結合を求めるアルゴリズム assoc は次のようになる．

```
assoc[x; a]=
    [
      null [a]→NIL;
      eq [x; caar [a]]→car [a];
      T→assoc [cdr [a]]
    ]                                    ……(6)
```

システム起動時では，a-リストは NIL であるが，ラムダ式の評価の際に，変数リストと引数リストの結合が行なわれると，a-リストは伸びる．例を以下に示す．次の式

```
λ[[x; y]; cons[car [x]; cdr [y]]]
      [(AB); (CDE)]                      ……(7)
```

を評価する場合を考える．最初，a-リストは，

$$(p_1 p_2 \cdots p_n) \qquad \cdots\cdots(8)$$

であったとする．$p_1, p_2, \cdots, p_n$ は，binding である．まず，変数 x には (AB) が，変数 y には (CDE) が結合されて a-リストは，次の様になる．

$$((X \cdot (AB))(Y \cdot (CDE))p_1 p_2 \cdots p_n) \qquad \cdots\cdots(9)$$

この環境で，次式

$$\text{cons [car } [x]; \text{cdr } [y]] \qquad \cdots\cdots(10)$$

が評価される．a-リストを見ると，変数 x の値は (AB) であるから (10) 式の car $[x]$ の値は A である．同様に変数 y の値は (CDE) であるから，cdr $[y]$ の値は (DE) である．したがって (10) 式の値は (ADE) となる．ラムダ式の評価が終ると，a-リストは元の状態に戻る．

この方式では，expr 関数が多重の再帰呼び出しを行なって，かつ大域変数の参照を行なう様な場合には，大域変数を参照するために，再帰呼び出しの多重度と変数リストの要素の数の積に比例して，a-リストの検索を行なわなければならない．例えば次の様な関数 fn

```
fn[x; y]=
  λ[
    [x; y];
    [
      null [x]→cons [z; y];
      T→fn [cdr [x]; y]
    ]
  ]                                      ……(11)
```

を考える．次式

$$fn[(AB); C] \qquad \cdots\cdots(12)$$

を環境 a_0

$$a_0 = ((Z \cdot D)) \qquad \cdots\cdots(13)$$

の下に評価すると，a-リストは関数 fn が呼ばれるたびに次の様に大きくなる．

$$a_0 = ((Z \cdot D))$$

$$a_1 = ((X \cdot (AB))(Y \cdot C)(Z \cdot D))$$

$$a_2 = ((X \cdot (B))(Y \cdot C)(X \cdot (AB))(Y \cdot C)(Z \cdot D))$$

$$a_3 = ((X \cdot NIL)(Y \cdot C)(X \cdot (B))(Y \cdot C)$$
$$(X \cdot (AB))(Y \cdot C)(Z \cdot D)) \qquad \cdots\cdots(14)$$

fn が3回目に呼ばれて，変数の結合が行なわれたときの a-リストが a_3 である．このとき，null $[x] = T$ であるから，cons $[z, y]$ がこの問題の値となる．しかし，変数 z の値を知るには，a-リスト a_3 から要素 $(Z \cdot D)$ を見つけるまでに，6個の要素を調べる必要がある．

3. p-リストを使う Shallow-Binding

次に，p-リストを使う Shallow-Binding について述べる．この方式は，各アトムの p-リストに valuecell という属性を作り，その値を現在の環境におけるアトム (変数) の binding とするものである (Fig. 1)．

変数の値が代入されていないときは，valuecell の値は NIL である．変数の binding が必要になったときは，変数の valuecell の値を見つけるだけでよい．

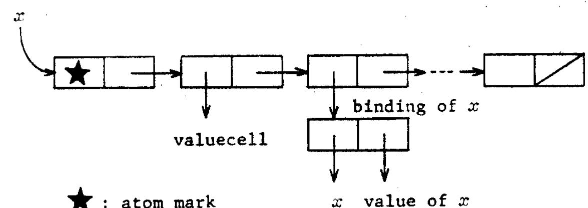


Fig. 1 Atom structure with valuecell.

この方式を更に詳しく説明する．この方式では，環境を

（1） environment tree（環境木）

（2） current environment pointer

（3） 各アトムについた valuecell

によって保持する．(1) の環境木は，Fig. 2 の構造を持つ．a-リストの構造とほとんで同じである．Fig. 2 において e', e として示されている節 (node) は，環境節 (environment node) と呼ばれる．

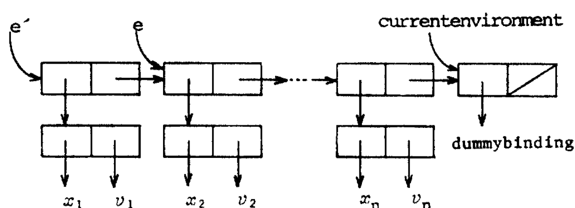


Fig. 2 Environment tree.

環境節 e' は次の様に解釈される．環境 e' と環境 e とで変数値が異なるのは変数 x_1 だけで環境 e' での変数 x_1 の値は v_1 である．これは，a-リストの解釈のしかたと同一である．

環境木と a-リストの異なる点は，Fig. 2 を例にとると，現在の環境が，a-リストでは，e' であったのに対し，環境木では，currentenvironment であるということである．これをルート (root) と呼ぶ．このときの各変数の binding は，各変数の valuecell の中に入っている．そして環境 e' からルートまでの各環境節の car 部には，環境 e' と現在の環境とで値の結合の仕方が異なっている変数の環境 e' での binding が入っている．

分かり易くするために Fig. 2 の代わりに Fig. 3 を考える．Fig. 3 は，現在の環境 e_0 では，変数 x の値は b であり，環境 e_1 においては，変数 x の値は a であるということを表わしている．環境 e_1 が現在の環境となったときの環境木とアトム x は，Fig. 4 の様になる．

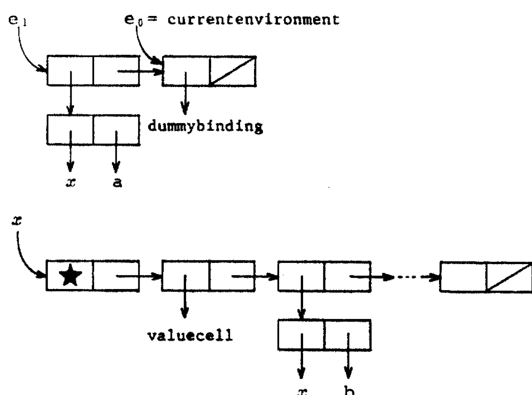


Fig. 3 Environment e_0.

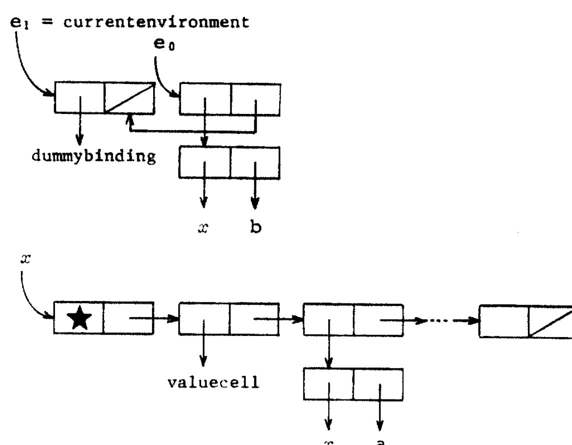


Fig. 4 Environment e_1.

Fig. 2 において，環境 e' を現在の環境とするには，Fig. 3 から Fig. 4 への手順を逐次繰返せばよい．

このアルゴリズムは，reroot として次の様に書ける．a は，これからルートにしようとする環境節である．

```
reroot [a]=
   [
     null [cdr [a]]→a;
     T→onestep [a; reroot [cdr [a]]]
   ]                                        ……(15)

onestep [new; old]=
  prog[
     [tmp; vc];
     rplacd [new; NIL];
     rplacd [old; new];
     tmp:=car [old];
     rplaca [old; car [new]];
     rplaca [new; tmp];
     vc:=prop [caar [old]; VALUECELL; error];
     tmp:=car [vc];
     rplaca [vc; car [old]];
     rplaca [old; tmp];
     return [new]
  ]                                         ……(16)
```

(16) 式の onestep は，Fig. 3 から Fig. 4 への手順を表わしている．Fig. 3, Fig. 4 の e_1 を new, e_0 を old として見ると onestep の動作は容易に理解できる．(15) 式の reroot は，onestep の動作を逐次繰返す．

4. Shallow-Binding を用いた assoc, apply

次に，3章で述べた，Shallow-Binding を用いた assoc のアルゴリズムと，小規模な apply の例を示す．

```
assoc [v; a] = vcell [v]                      ……(17)

   vcell [v] = get [v; VALUECELL] ……(18)

apply [f; x; a] =
    [
     atom [f] → [
                 eq [f; CAR] → caar [x];
                 eq [f; [CDR] → cdar [x];
                 eq [f; CONS] → cons [car
                            [x]; cadr [x]];
                 eq [f; ATOM] → atom [car
                                  [x]];
                 eq [f; EQ] → eq [car
                            [x]; cadr [x]];
                 T → apply [eval [f; a]; x; a]
                ];
     eq [car [f]; LAMBDA]
      → prog 1 [
                 eval [
                       caddr [f];
                       reroot [pairlis [cadr
                                [f]; x; a]]
                      ];
                 reroot [a]
                ];
     eq [car [f]; FUNARG]
      → prog 1 [
                apply [cadr [f]; x; reroot
                            [caddr [f]]];
                reroot [a]
               ]
    ]                                           ……(19)
```

```
prog i [e1; e2; …; en] =
     evaluates e1, e2, …, en in order
     and returns the value of ei            ……(20)
```

(17) 式は，assoc のアルゴリズムである．Shallow-Binding を用いているので a-リストの検索をする必要はなく変数 v の p-リストの valuecell という属性の値を求めるだけでよい．

(19) 式は，apply のアルゴリズムである．この式の 12～18行は，ラムダ式の評価を行なう部分である．この部分を再び以下に書く．

```
prog 1 [
        eval [                                    …①
              caddr [f];
              reroot [pairlis [caddr [f]; x; a]]
             ];
        reroot [a]                                …②
       ]                                       ……(21)
```

(21) 式①では，pairlis でラムダ式の変数と引数を対リストにして a につなぐ．そして reroot を行なって現在の環境を pairlis [cadr[f]; x; a] に切替え，その環境でラムダ式の本体 (caddr[f]) を評価している．(21) 式②では，環境を元の状態に戻している．

(19) 式の 20～23 行は，関数引数の評価を行なう部分である．この部分を再び以下に書く．

```
prog 1 [
        apply [cadr [f]; x; reroot [caddr [f]]];
                                                  …①
        reroot [a]                                …②
       ]                                       ……(22)
```

(22) 式①では，reroot [caddr[f]] で FUNARG の第2引数を現在の環境とし，その環境の下で FUNARG の第1引数，cadr[f] の関数に引数 x を与えて評価している．そして (22) 式②で環境を再び元の状態に戻している．

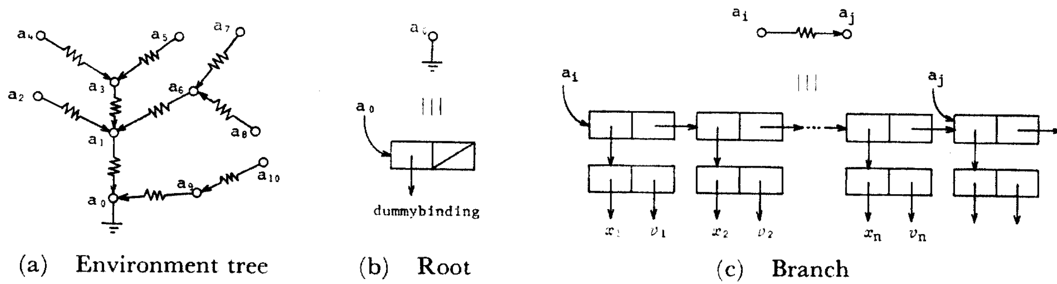
5. Casual Shollow-Binding

2章で述べた Deep-Binding と3，4章で述べた Shallow-Binding の特徴をまとめると Table 1 の様になる．

Table 1 で variable-access time というのは，変数が与えられたときに，その値を求めるのに必要な時間である．Shallow-Binding では，これは一定であるが，Deep-Binding では，2章で述べたように a-リストの

Table 1 Comparison of Shallow-Binding and Deep-Binding

	variable-access time	context-switching time
Shallow-Binding	constant	unbounded
Deep-Binding	unbounded	constant



(a) Environment tree　(b) Root　(c) Branch

Fig. 5 Environment tree.

長さに比例して長くなる．context-switching time というのは，ある環境を他の環境へ切替えるのに必要な時間である．Deep-Binding では，a-リストへのポインタを書き替えるだけでよいので一定であるが，Shallow-Binding では，ある環境と，他の環境とで変数の値の異なっている数に比例して reroot に時間がかかる．

この様に Shallow-Binding と Deep-Binding には一長一短がある．しかし context-switching よりも variable-access の方がより一般的に生じるので，variable-access の効率の良い，Shallow-Binding の方が効率のよい変数結合方式だともいえる．

以下に述べる Casual Shallow-Binding は，Shallow-Binding と Deep-Binding を併用して，全体の効率を高めようとする方式である．

Shallow-Binding の環境木は，インタープリターの処理について Fig. 5 (a) に示したような木 (tree) になる．Fig. 5 (a) の内部の詳細を同図 (b), (c) に示す．

Fig. 5 (a) の状態では，ルートは a_0 である．もしこの状態から，例えばルートを a_1 に変えようとすると，reroot [a_1] を実行する必要がある．しかし (17) 式で述べた assoc のアルゴリズムを少し工夫すれば reroot [a_1] を行なわなくても環境 a_1 での変数の binding を求めることができる．そのアルゴリズムを (23) 式に示す．

```
assoc [v; a] =
    [
      null [cdr [a]] → vcell [v];        …①
      eq [v; caar [a]] → car [a];        …②
      T → assoc [v; cdr [a]]             …③
    ]                                    ……(23)
```

(23) 式①では，まず cdr [a] が NIL かどうか調べる．もし，cdr [a] = NIL であれば a はルートであるから，vcell [v] を実行する．もし cdr [a] ≠ NIL であれば a はルートではない．このとき環境 a における変数 v の binding は，a からそのときのルートまでの対リストの中にある可能性がある．したがって (23) 式③では，caar [a] と v を比較し等しければ car [a] が変数 v の binding であるからそれを値とする．(23) 式③で assoc を再帰的に呼び出し，以上の手順を繰返す．a からルートまでの対リストの中に変数 v が見つからなかったときには，v の valuecell の値が v の binding となる．

Casual Shallow-Binding では，(23) 式の assoc が使用される。Casual Shallow-Binding では，通常は a-リストによる Deep-Binding を行なう．そして，リスプのプログラムの中で関数 shallow が実行されると，その時の環境が reroot される．したがってそれ以後は，その環境における変数へのアクセスは，Shallow-Binding になる．関数 shallow は次の様に書ける．

Shallow [ ] = prog 2 [reroot [a]; T] ……(24)

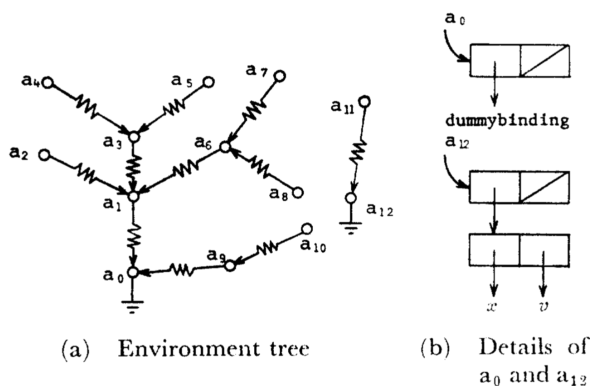
a は，関数 shallow が呼ばれたときの a-リストである．

Casual Shallow-Binding では，リスプのプログラマが，プログラムの性質を考慮しながら関数 shallow を適当に使うことによりプログラムの実行を効率的にすることができる．

6. Casual Shallow-Binding の改良

本章では，5 章で述べた Casual Shallow-Binding の問題点とその解決策について述べる．

まず，Casual Shallow-Binding の問題を述べる．Casual Shallow-Binding では Fig. 5 の様に環境木 (a-リスト) は，木状に伸びてゆく．そして環境木が木のときにのみ (15) 式で表わした reroot, (23) 式で表わした assoc のアルゴリズムは，有効に動作する．環境木がラムダ式の評価のときにのみ伸びるのであれば，環境木は必ず木になる．しかし a-リストを引数とする関数 (apply, eval, etc) に a-リストとして，環境木



(a) Environment tree (b) Details of a_0 and a_{12}

Fig. 6 Forest-like environment tree.

の中に含まれている環境節以外が与えられたとすると，環境木は，Fig. 6 の様に林 (forest) になる．

Fig. 6 は，Fig. 5 の環境で例えば次式

$$\text{eval}\ [x\,;a_{11}] \quad \cdots\cdots(25)$$

を評価しようとしたときの状態を表わしている．いま

$$a_{11}=((x_1\cdot v_1)(x_2\cdot v_2)\cdots(x_n\cdot v_n))$$

$$x\neq x_1,\ x_2,\ldots,\ x_n$$

$$x\neq \text{apval constant} \quad \cdots\cdots(26)$$

であったとする．(25) 式は環境 a_{11} で変数 x の値を求めようとするものである．したがって eval の中では，式

$$\text{assoc}\ [x\,;a_{11}] \quad \cdots\cdots(27)$$

が評価される．しかし a_{11} には変数 x は含まれていないので，(27) 式の値は *NIL* とならなければならない．しかし (23) 式の assoc に引数として x, a_{11} を与えて実行すると，n 回 assoc が再帰的に呼ばれた後に (23) 式①が実行され

$$\text{assoc}\ [x\,;a_{11}]=\text{vcell}\ [x] \quad \cdots\cdots(28)$$

となる．もし，そのとき変数 x の valuecell の値が *NIL* であれば，これは正しい結果である．しかし valuecell の値が *NIL* でないときには，期待される効果と異なることになる．

以上が Casual Shallow-Binding の第 1 の問題点である．

もう一つの問題は，Fig. 6 の状態で，

$$\text{reroot}\ [a_{11}] \quad \cdots\cdots(29)$$

を実行したときに生じる．(29) 式を実行すると，実際は，現在のルートは a_0 であるが，a_{12} が現在のルートと見なされてしまう．その場合は (29) 式によって，変数 $x_1, x_2,\ldots, x_{n-1}$ の valuecell には正しく binding $(x_1\cdot v_1), (x_2\cdot v_2),\ldots, (x_{n-1}\cdot v_{n-1})$ が代入されるが，それ以外の変数については，valuecell に何が入っているかわからないという状況が生じる．(29) 式の実行によって期待する結果は変数 $x_1, x_2,\ldots, x_n$ の valuecell には binding $(x_1\cdot v_1), (x_2\cdot v_2),\ldots, (x_n\cdot v_n)$ が代入され，その他の変数の valuecell には，*NIL* が代入されることである．

これらの，2 つの問題点は，topenvironment という定数と currentenvironment という変数を持ち込み assoc, reroot を少し変更することにより解決できる．

定数 topenvironment は，すべての変数の binding が *NIL* となっている環境を保持している．そして，変数 currentenvironment は 3 章で述べた様に現在の環境（ルート）を保持している．この 2 つの定数，変数を使って，assoc, reroot を書きかえると次の様になる．

```
assoc [v; a]=
  [
    null [cdr [a]]
      →[
          eq [a; currentenvironment]→
                          vcell [v];
          T→NIL
        ];
    eq [v; caar [a]]→caar [a];
    T→assoc [v; cdr [a]]
  ]                                        ……(30)
```

```
reroot [a]=
  prog [
       [ ];
       [
         null [cdr [a]]
          →[
              eq [a; currentenvironment]→a;
              T→prog 2 [
                     rplacd [a;
                        topenvironment];
                        reroot [a]
                     ]
            ],
         T→onestep [a; reroot [cdr [a]]]
       ];
       currentenvironment:=a;
       return [a]
      ]                                    ……(31)
```

(30) 式では，cdr [a] が *NIL* のときは，a が cur-

rentenvironment かどうか調べる．もし a が currentenvironment に一致すれば変数 v の binding は v の valuecell の中にある．しかし a が currentenvironment でなければ，変数 v には値は代入されていないので NIL を値とする．8行目の $eq[v; caar[a]]\rightarrow\cdots$ 以降は（23）式と同じである．

（31）式では，a が Fig. 6 の a_{11} の様なときには，a_{12} の cdr 部が topenvironment に書きかえられ，root[a] が実行される．これにより林状の環境木は木になり，そして reroot[a] の結果も期待通りになる．

7. むすび

本稿では，従来の各種の変数の結合方式について展望し，更に，文献6）で新しく提案された，Casual Shallow-Binding の問題点の解決策を示した．文献6）で述べられた方法は，a-リストの持つ，‘環境を保持するという機能’を，p-リストを使った Shallow-Binding で実現しようとしたものであるが，本稿で示した様に，環境の保存の方法に不完全さが残されていた．

本稿で示した，改良された Casual Shallow-Binding では a-リストの持っている環境保存の機能を，完全に実現させることができた．

謝辞　LISP に関し御教示いただいた本学工業短期大学部の石原好宏先生に感謝致します．

参考文献

1) 東出正裕，安部憲広，小西所二，辻　三郎：Bフレーム・Mフレームを使用したミニコンリスプ FLISP，情報処理学会論文誌，**20**, 8-16 (1979)
2) 黒川利明：LISP 1.9 プログラミングシステム，情報処理，**17**, 1056-1063 (1976)
3) 長尾　真，中村和雄：高速補助記憶装置を使用したミニコン用 LISP 1.6 システム，情報処理，**17**, 720-728 (1976)
4) HLISP 説明書，日立製作所中央研究所 (1968)
5) 中西正和：KLISP の拡張機能とその応用，情報処理，**11**, 619-623 (1970)
6) H. G. Baker, Jr.: Shallow Binding in LISP 1.5, CACM, **21**, 565-569 (1978)
7) J. McCarthy 他: LISP 1.5 Programmer's Manual, The M.I.T. Press (1962)
8) L. H. Quam, W. Diffie: Stanford LISP 1.6 Manual, Stanford Artificial Intelligence Laboratory Operating Note, No. 28.4 (1968)
9) 中西正和：LISP 入門，近代科学社 (1977)
10) 後藤英一：LISP 入門，bit, **6**, No. 1～**7**, No. 2 (1974, 1975)

（昭和56年4月6日　受理）