

LispマシンALPS/Iの性能評価

井田昌之, 小林茂男, 山方宏修 (青山学院大学)

1. はじめに

ALPS/Iはマイコンコンピュータ8080と64kワバブルメモリを中心とした専用マシンであり, その目標は実際的なLispプログラムの経済的走行と会話型の幸有処理形態の提供にある。51年1月のハードウェア及びLispプロセッサ第1版の誕生後, 度々の改良の結果現有システムは約100のsubr, fsubrをPROM中に組み込むようになり, また速度も第1版に比較して約1.5倍にあがっている。

ここではALPS/Iの機能の概略を紹介し, その実例として箱入娘パズルの解法を取り上げて報告する。

2. Lisp処理系について

ALPS/IのLispプロセッサは, Lisp1.5を基調としてEvalquote入力を解釈実行する約2kbyteのプログラムで, PROMに書き込まれ使用される。関数Evalquoteは文献1と同じ仕様で作られている。apply, evalは若干の高速化のための手直しに加え, 付加機能を実現するための手続きが入れられている。evconは再帰的ではなく, ループで実現されている。

LispシステムとしてのALPS/Iの特徴は, 次のようにまとめられる。

① Evalquoteインタプリタwithout A-list: 変数束縛はstack-bind, Funarg問題は束縛凍結により解決

② 次のデータタイプを持つ

A. 基本整数: アドレスそのものの数値

B. 数: 1つの数(整数及び浮動小数点数(現在組込中))

C. H-分子: H-領域上の属性, 値, 鍵をもつユニークオブジェクト

C-1. アトミックシンボル: pnameを鍵とする。属性はundef, subr, fsubr, expr, fexpr, apval, hexpr, harray

C-2. ハッシュ化配列要素: 配列名aと添字θのドット対を鍵とする。aはアトミックシンボル, θは任意のS式でよい。

C-3. 連想計算連想子: HEXPR関数名と実引数のドット対を鍵とする。

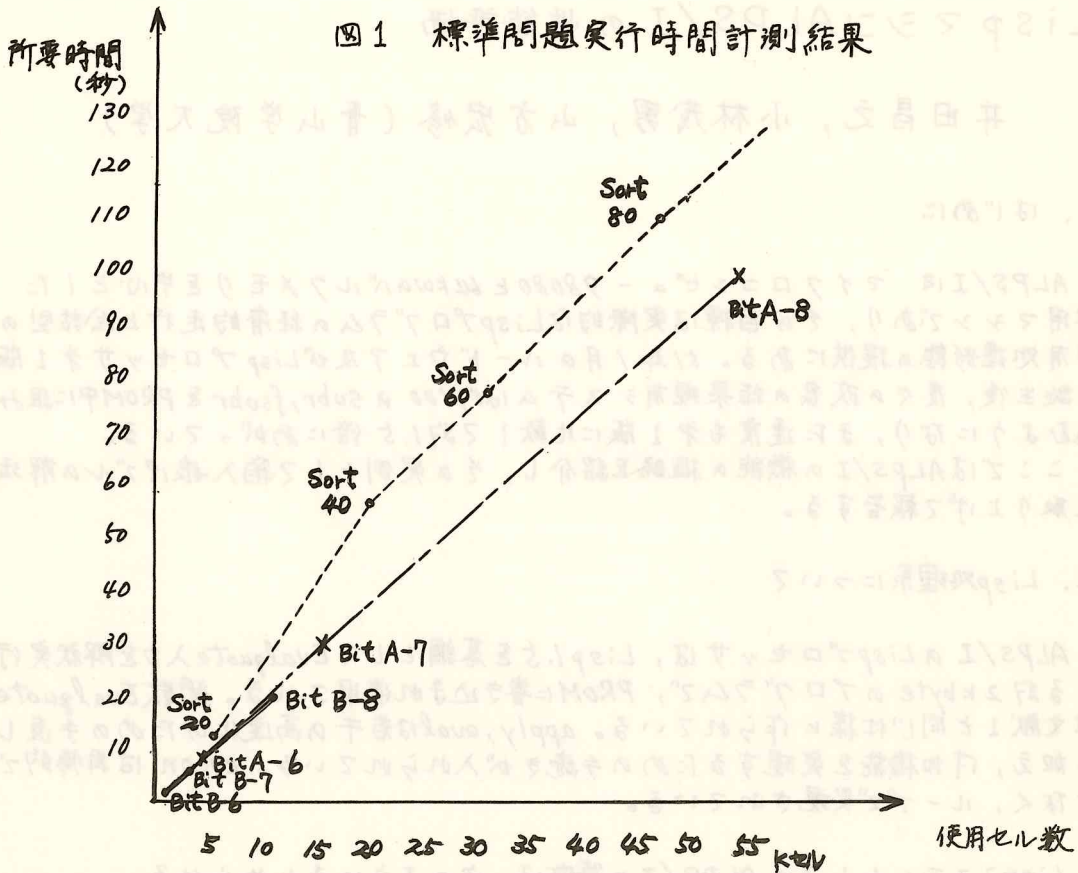
C-4. ラベル連想子: ラベル名とプログラム定義体のドット対を鍵とする。

D. L-分子: ドット対のリストセル

③ 約100の組込関数を持つTurn keyシステムである。

物理的な上限個数は次の通りである。

数=2048個(内1024個は基本整数), H-分子=2048個, L-分子=57,344個
74年夏のシンポジウムの標準問題の実行結果を図1に示す。Sort-100は137秒を要し, メモリは56kセル以上を使っている。実行時間の計測は9回平均してf(x)を実行させる汎関数をオブジェクトプログラムロード機能を用いてロードし, 実測したものである。(Sortは1回だけ)



④ 中間言語ドライバによるプログラマインタプリタ：最高10個までのプログラムを同時に保持できるエントリ-表と256語の中間言語ロード域（あふれは*PROGNに書き換えられフリーストレージへとCHAINされる）があり，プログラムはリスト表現からアレイ表現に書き換えられる。そして，アレイ上の各ステートメントを逐次解釈実行する形で処理が進められる。ラベルはH-領域上にラベル連想子を作るのに用いられ，中間言語域上は何も存在しない。

この経過について述べると次のようになる。高速化を計るにはコンパイラが本質的であり，インタプリタと比べて10倍くらい速くなるといわれている。しかしコンパイラの作成に着手することは現段階としては不要であったので，インタプリタの範囲内での簡易的な高速化を考えた。まず「オー」に目付いたのがプログラムである。COND式による定義やマクロ型の定義にはそれなりの簡潔さがあり，その処理もうなすけるものがあるがprog形式に至っては非常に原始的な方法が今もって行なわれているようである。プログラムインタプリタの定義には次のようなものがある。（ALPS/Iのプログラマインタプリタのオ1版はこれであった。）

ただし，*setq及び*setは必ず新たに束縛を生成するsetq及びsetとする。
massoc, evalはスタックバインドのためa-list引数に基づいたものとする。


```

prog(x) = prog((xx;val;v);
  mapc(car(x);λ((j);*seto(j;NIL)));
  xx:=cdr(x);
  map(xx;λ((j);(atom(car(j))→*set(car(j);cdr(j))))));
exec-loop:
  (null(xx)⇒ return(val);
  atom(car(xx))⇒ go(advance));
v:=car(xx);
xx:=cdr(xx);
eval(v);
go(exec-loop);
advance:
  xx:=cdr(xx);
  go(exec-loop))

go(x) = seto(xx;cdr(*assoc(x)))
return(x) = prog2(seto(xx;NIL);seto(val;eval(x)))

```

プログ変数の初期化を行ない、次にラベルをさがして各々 stack へ bind し go 文にそなえ、そののち順に評価を行なうものである。(この引数 x を 2 回走査する方法の他に 1 回の走査で済ませることも可能である。)

2 回の走査を仮定すると、1 回目のラベル探索時には実行文を読みとばしており、2 回目の逐次実行時にラベルを読みとばすという無駄が目につく。また go 文の処理は遅くかつ誤りを犯す危険がある。(束縛スタッフには変数の属性の区別がないので、未定義ラベルへの go 文があつてかつ外側で別に保持されている場合)

中間言語域とラベル連想子を用意し、1 回目の走査では各々の登録を行ない、2 回目にはアレイの各要素の逐次評価をすることにより無駄なリスト探索を省き、かつ正しく処理することができるとする。この結果 ALPS/IDE の prog インタプリタの概略は次のようになっている。

```

prog(x) = prog((pform;val;xx);
  xy:=ppoint;
  (car(x)⇒ progn(ppoint:=addl(ppoint);progtbl(ppoint):=cons(LOCAL;car(x))));
  pform:=x;
  (null(cdr(x))⇒ return(NIL));
  mapc(cdr(x);λ((j);(atom(j)⇒ hstore(j;pform;ppoint);
    T⇒ progn(ppoint:=addl(ppoint);progtbl(ppoint):=j))));
exec-loop:
  xx:=addl(xx);
  eval(progtbl(xx));
  (eq(xx;ppoint)⇒ return(val));
  go(exec-loop);
)

go(x) = (atom(x*)∧v:=hdotp(x*;pform)⇒ progn(xy:=v;progtbl(xx):=cons(**GO;v));
  T⇒ ppoint:=hdotp(eval(x);pform))
**go(x) = xx:=x
return(x) = progn(setq(xx;ppoint);setq(val;eval(x)))

```

progtbl はプログ中間言語を表わす。ppoint は progtbl 用のポインターとする。
 x^* は x 自身をさす。hdotp[$x; y$] は (x, y) を鍵部にもつ H-分子の値部
を値とする関数とする。対応する分子がなければ NIL。
hstore[$a; 0; v$] は $(a, 0)$ を鍵部に、 v を値部にもつ H-分子を生成し、値
 v とする関数とする。

go文は L1.6型であり、アトム引数の go文に対しては中間言語上で無条件分岐
(木60)への書き換えが行われる。この結果ラベルの処理に関しては $O(n)$
から $O(1)$ に高速化され(これはその時までにつまれたスタックの要素数)、実行
ループに関してはリストのたぐりがなくなったためメモリ読み出しの回数を半
分に減らすことができた。標準問題である sequence は プログ形式が主ループで
あるのでこの改良が大きく影響しており、この問題に関しては同程度のメモリ容
量をもつ ミニコン Lisp より 1.2 倍 ~ 1.8 倍速い実行結果が出ている。

⑤ 即時回収コレクタ: 後藤英一教授の示唆 (bit 誌 Lisp 入門) に基づき, ALPS/I
には即時回収コレクタが含まれている。このコレクタの目的は次のようなもので
ある。全セルのマーキングによりガベジコレクションを行なう通常のガベジコレ
クタは所要時間が大きく、その間 Lisp の処理の進行を止めしてしまうので (1) ガベ
ジコレクションの機会を減らすこと (2) ガベジコレクションのアルゴリズムを
良くすることが必要である。ガベジコレクションタイムがインタプリタ処理時間
を越えては、ちり集めシステムの様を呈してしまふ。

このため即時回収コレクタはインタプリタ内で消費するセルを、不要になった
時点で即時回収する。eval 中の evalis は引数のリストを作成するために呼ば
出されるもので、実行後は不要になる。evalis の定義を次のように変えること
によりオーバーヘッドを余り増すことなしにこの操作を行なうことができる。

```
evalis(m) =
  (null(m) → setq(*last; NIL);
   null(cdr(m)) → setq(*last; list(eval(car(m))));
   T → cons(eval(car(m)); evalis(cdr(m))))
```

*last にはこの結果は evalis の値のリストの最後のドット打が与えられる。この
*last のアドレスの cdr 部は必ず NIL であり、この部分を現在の Freelist の
トップを示すポインタで rplacdすればよい。このポインタを freetop, evalis[m]
の結果を m' としてこの過程は次のように書ける。

```
progn(rplacd(*last; freetop); setq(freetop; m'))
```

即時回収コレクタにより例えば, sequence の問題では約 20% のセルが回収され
ている (sequence-20, sequence-40 の結果のグラフより確認)。sequence
-100 においてもガベジコレクタの作動は不要であり、物理的には 56K セルである
リスト領域が即時回収コレクタにより意味的に拡張され、70K セル程度の大きさ
を持つとも考えられる。

M 式による定義を直線的にコーディングすると、手続きの見やすさが上がる半
面、束縛動作の多いプログラムの場合にリストの消費が多くなり、スペース使用
効率が悪くなるが、この点もこのコレクタが補っている。(再帰コールは 8080 の
call 命令と pushpop によりそのまま記述できる。)

⑥ハッシュ化配列: ALPS/I の設計時に次の3つの点があげられ, その解として HLisp を参考にして考え出されたのがハッシュ化配列である。

- 1) アレイ機構の組み込みの必要性への対応
- 2) User-property の組み込みの必要性和アトム・plist の廃止の対策
- 3) ユニフォームオブジェクトの統一的な扱い

ハッシュ化配列の各要素の形式は属性部に「ハッシュ化配列要素」, 鍵部に「ハッシュ化配列名」と添字になるS式のドット対をもちことALPS/I型H-分子に統一されている。

ハッシュ化配列に対しては array, dearray, seta, delete の4つの関数がある。

array : fsubr	配列宣言文
dearray : subr	引数で指定したハッシュ化配列全要素の消去
seta : fsubr	seta[array-name; subscript; value] 要素への値のセット。array-name 以外は評価される。
delete : fsubr	delete[array-name; subscript] 要素の消去, subscript は評価される。 array-name[subscript] で占有されていたH-領域中の スロットは自由化される。

箱入帳の解法においては盤面用の配列, 差集合をとるためのマーキング, 盤面番号の変換表, 次に動きうる手互を定める手続きの格納などに使われている。

⑦試行用汎関数 Try

ALPS/I は次のような汎関数 Try をもっている。

```
try(x;fn) = prog((xx);
  xx:=x;
  loop:(null(xx) → return(NIL);
  fn(car(xx)) → return(car(xx)));
  xx:=cdr(xx);
  go(loop) )
```

リストxのcarエレメント中で関数fnに対してNIL以外の値を持つものがあればそれを値として返す。なければ"NIL"である。これは集合の中から条件に合うものを一つだけ抜き出すような場合に便利であり, 又速い。(この操作はInterLispでの catch[mapc[x;lambda[j];[fn[j] → throw[j]]]]) でも可能である。)

⑧連想計算

ALPS/I では連想計算を行なう1引数の関数定義体に対して hexpr という属性が設けられ, その処理が各々 apply, eval 中に組み込まれている。hexpr として定義された関数は例えば apply においては次の手順により処理される。(引数を評価することを加えれば eval 内でも同様である。)

```
eq(get-attrib(fn);HEXPR) →
  (hdotp(fn;car(args)) → hassoc(fn;car(args));
  T → progn(v:=apply(value(fn);args);hstore(fn;car(args);v)))
```

ALPS/I の H-分子は HLisp 型 H-分子と異なり, ドット対そのものではなく, 三つ組であり, Listセルの monocopy ははいていない。このためリストを引数とする連想計算においては, リストの一意性が保証されていないので注意を要する。

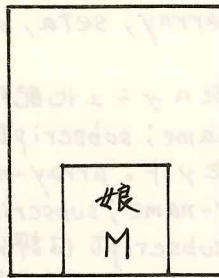
3. 箱入娘パズルについて

「箱入娘パズル程度の問題が解けないようでは、実際に意味のある記号処理の問題を処理するシステムとして全く迫力がない。」という後藤先生の御意見に基づき、性能試験用に箱入娘パズルの解法を試みたものである。

箱入娘パズルは、初期配置 C_0 から娘を門の所へ連れ出した最終配置 C_f に至る最短経路を見い出す問題である。

父 P1	娘 M	母 P2
下男 P3	番頭 B 子供 C1 子供 C2	下女 P4
子供 C3		子供 C4

初期配置 C_0



最終配置 C_f

(M以外の物の位置は任意)

初期配置 C_0 から n 手で 2^n 通りうる配置の集合を S_n , S_n に属する配置から1手で移り得る配置の集合を $smove[S_n]$ とする。この時 S_{n+1} は次式で与えられる。

$$S_{n+1} = smove[S_n] - S_n - S_{n-1},$$

$$S_0 = \{C_0\}, S_{-1} = NIL$$

S_n の中に C_f が含まれたら次式に従って、この目的配置に到達する手順として各々 S_n より d_n を1つだけ取り出す。

$$d_0 = C_0, d_n = C_f$$

$$d_n = try[*smove[d_{n+1}]; \lambda[l[j]; member[j; S_n]]]$$

計算機によってこのパズルは既に解かれており、 $n=8/2^n$ である。Lisp系の言語を用いた解としては、LispQによる19分31秒というものが76年記号処理委員会報告集に報告されている。また、その手続きの概略はbit誌Lisp入門No. 13(1974)に完全に記述されており、我々もこの手続きをもとに作成した。

```
nextconf(n;snl;sn) = prog((dn; print(list(n;length(sn)));
                          (null(sn) → return(NIL);
                          seto(dn;choosel(final(sn))) → go(z);
                          seto(dn;nextconf(addl(n);sn;advance(snl;sn))) → go(z);
                          T → return(NIL));
                          z
                          printconf(n;dn);
                          return(choosel(intersection(snl;smove(list(dn))))))
```

ALPS/I2の解はnextconf, ある1つの配置confから一手で動きうる配置の集合を求める*smove, 配置のパック・アンパックを行なう*fn2, 駒位置の連続を調べるconnectなどよりなる。各ルーチンの説明は後述する。

さて、箱入娘パズルによる性能試験の項目としては、次のようなものがあると考えられる。

- ① すべての手続きをその言語で書き表わすことができる。
- ② 手続き構成を支援するツール、関数が備わっている。
- ③ その手続きを動かすのに十分な記憶容量を持つ。
- ④ 実用上十分な高速性がある。

①を満たすことは明らかなので②, ③, ④が問題になると思われる。ALPS/I2のもつ機能のうちハッシュ化配列, 関数Tryなどは②に対するサポートになっている。中間言語方式progインタプリタ, オブジェクトロード機能は④に対する

サポートになっている。しかし記憶装置は64Kワードしかなく(カセットMTはソビエトファイルに使用),残念ながら81手解の初期配置から解を求めることができない。(各配置は1語にパックし,数として記憶するのが一番簡潔かつ高速であるが数の保持は1024しかできない。配置を $(M \times B, P1 \times P2, P3 \times P4, S)$ の型でリストにして保持するにしても1配置に4語,さらに S_n にまとめるために1配置につき1語を要するので dn を求めるために覚えられる約1万2千の配置を保持するのに60K語リストセルが必要である。)

次に各手続きを説明する。

- ① *fn2 1つの配置を表わす数 $\Leftrightarrow M, B, P1 \sim P4, S, X, Y$ の間の変換を行なう。このルーチンは高速化のためにオブジェクトロード機能を用いて8080機械語により作成されているが, Lispの仕様をまげることとはされていない。
例: *fn2[0;dn]: ある配置dnの内容を各変数に分離する
- ② nextconf ハッシュ化配列TAGが S_{n+1} を作るためのマーカー γ (S_n, S_{n-1} 及びその時点までの S_{n+1} の各配置に対して)に用いられている。不要になったTAGの要素はdelete関数により消去される。H-領域は2Kエントリ-を持つので充分に使用できる。また*smoveをassoccomp指差することにより,帰路における再計算を不要にすることができる。

FUNCTION EVALQUOTE HAS BEEN ENTERED, ARGUMENTS...

```

DEFINE((
(NEXTCONF (LAMBDA ( )
(PROG ( )
(CSETQ STACK (CONS SN (QUOTE (NIL))))
(SETA TAG (CAR SN) 1)
Y (PRINT N)
(COND ((NULL SN)(RETURN NIL))
((TRY SN(QUOTE (LAMBDA (J))(PROGN (*FN2 0 J)(EQ M 13))))(GO X)))
(CSETQ VALUE NIL)
(MAPC SN (QUOTE (LAMBDA (J)(MAPC (*SMOVE J)
(QUOTE (LAMBDA(K)(COND ((NOT (EQ (ERRSET (TAG K) T) 1))
(PROGN (SETA TAG K 1)(CSETQ VALUE (CONS K VALUE))))))))))
(MAPC SN1 (QUOTE (LAMBDA (J)(DELETE TAG J))))
(CSETQ N (ADD1 N))
(CSETQ SN1 SN)
(CSETQ SN VALUE)
(CSETQ STACK (CONS SN STACK))
(PRINT SN)
(GO Y)
X (CSETQ STACK (CDDR STACK))(PRINT (QUOTE $$$ YATTAZO $))
(PRINT N)(PRIN1 (QUOTE M=))(PRIN1 M)(PRIN1 (QUOTE B=))(PRIN1 B)
(PRIN1 (QUOTE P1=))(PRIN1 P1) (PRIN1 (QUOTE P2=))(PRIN1 P2)
(PRIN1 (QUOTE P3=))(PRIN1 P3)(PRIN1 (QUOTE P4=))(PRIN1 P4)
(PRIN1 (QUOTE S=))(PRINT S)
(CSETQ DN (TRY SN1 (QUOTE (LAMBDA (K)(TRY (*SMOVE K)(QUOTE (LAMBDA(J)
(PROGN (*FN2 0 J)(EQ M 13)))))) ))
Z (CSETQ N (SUB1 N))(CSETQ SN SN1)(CSETQ SN1 (POPUP NIL))
(PRINT N)
(*FN2 0 DN)
(PRIN1 (QUOTE M=))(PRIN1 M)(PRIN1 (QUOTE B=))(PRIN1 B)
(PRIN1 (QUOTE P1=))(PRIN1 P1)(PRIN1 (QUOTE P2=))(PRIN1 P2)
(PRIN1 (QUOTE P3=))(PRIN1 P3)(PRIN1 (QUOTE P4=))(PRIN1 P4)
(PRIN1 (QUOTE S=))(PRINT S)
(MAPC SN (QUOTE (LAMBDA (J)(DELETE TAG J))))
(MAPC SN1 (QUOTE (LAMBDA (J)(SETA TAG J 2))))
(CSETQ DN (TRY (*SMOVE DN)(QUOTE (LAMBDA (J)(EQ (ERRSET (TAG J) T) 2))))
(COND ((NULL DN)(RETURN NIL))
(GO Z))))))

```

END OF EVALQUOTE, VALUE IS..
(NEXTCONF)

③ *SMOVE Conf から移りうるすべての配置の集合を値とする関数。M, B, P1~P4 についての移動可能配置は現在位置をもとにハッシュ化配列を引用し、手続きを取り出し実行させる。これは演算ルーチンの速度に比べ配列参照の方が速いこと、このために必要とされる M, B, P に対する手続きのセル数はヨク程度で済むことなどによる。子供 C1~C4 の移動に関しては、すきま(x, y) との位置計算により行なう方が速いのでハッシュ化配列は使用しない。

```
FUNCTION EVALQUOTE HAS BEEN ENTERED. ARGUMENTS...
DEFINE((
  (*SMOVE (LAMBDA (CONF)
    (*PROGN
```

```
      (CSETQ VAL NIL)
      (*FN2 0 CONF)
      (CSETQ WW ((MM M) NIL))
      (COND (WW (CSETQ VAL (CONS WW VAL))))
      (CSETQ WW ((BB B) NIL))
      (CSETQ VAL (NCONC WW VAL))
        (CSETQ NOWP 1)
      (CSETQ WW ((PP P1) NIL))
      (CSETQ VAL (NCONC WW VAL))
      (CSETQ NOWP 2)
      (CSETQ WW ((PP P2) NIL))
      (CSETQ VAL (NCONC WW VAL))
      (CSETQ NOWP 3)
      (CSETQ WW ((PP P3) NIL))
      (CSETQ VAL (NCONC WW VAL))
      (CSETQ NOWP 4)
      (CSETQ WW ((PP P4) NIL))
      (CSETQ VAL (NCONC WW VAL))
      (CSETQ NM (OLD M))
```

現在の M, B, P の位置に対応してハッシュ化配列 MM, BB, PP の要素の手続きにより各々 WW に移動可能配置を set し VAL に結合させる。

```
      (SETA MARK NM 1)
      (SETA MARK (ADD1 NM) 1)
      (SETA MARK (SUM NM 5) 1)
      (SETA MARK (SUM NM 6) 1)
      (CSETQ NX (OLD X))
      (SETA MARK NX 1)
      (CSETQ NY (OLD Y))
      (SETA MARK NY 1)
      (CSETQ NB (OLDB B))
      (SETA MARK NB 1)
      (SETA MARK (ADD1 NB) 1)
      (CSETQ OP1 (OLD P1))
      (CSETQ OP2 (OLD P2))
      (CSETQ OP3 (OLD P3))
      (CSETQ OP4 (OLD P4))
      (SETA MARK OP1 1)
      (SETA MARK (SUM OP1 5) 1)
      (SETA MARK OP2 1)
      (SETA MARK (SUM OP2 5) 1)
      (SETA MARK OP3 1)
      (SETA MARK (SUM OP3 5) 1)
      (SETA MARK OP4 1)
      (SETA MARK (SUM OP4 5) 1)
      (CSETQ L 23)
      (CSETQ CLIST NIL)
```

↓ 以下の説明：一語にパックされた conf は子供の位置ではなくすきまの位置関数 S を保持しているの、各駒とすきまの位置に目印をつけることにより C1~C4 の位置を探し、それを CLIST にまとめめる。このために配列 MARK が用いられている。MARK には初期値ゼロが実行に先立って与えられている。配列 OLD, OLDB は C の位置を求めするための統一化された駒位置の番号への変換を行なう変数テーブルである。

基本位置番号
(左下は B, 他は右上)

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
16	17	18	19

↓ OLD OLDB

0	1	2	3
5	6	7	8
10	11	12	13
15	16	17	18
20	21	22	23

```
FINDC
  (COND ((NOT (OR
    (EQ (MARK L) 1)
    (EQ L 4) (EQ L 9) (EQ L 14) (EQ L 19)
    ))
    (CSETQ CLIST (CONS L CLIST))))
  (SETA MARK L 0)
  (COND ((ZEROP L) (GO FINDX)))
  (CSETQ L (SUB1 L))
  (GO FINDC)
```

例 OLD(4) = 5
OLDB(9) = 15

C の位置のための
盤面位置番号

次ページへ続く


```

FINDX
  (CSETQ PREDXY (CONNECT NX NY))
  (MAPC CLIST
    (QUOTE (LAMBDA (C)
      (PROGN
        (CSETQ OC (NEWC C))
        (CSETQ PREDCX (CONNECT C NX))
        (CSETQ PREDCY (CONNECT C NY))
        (COND ((AND PREDCX PREDXY)
          (CSETQ VAL (CONS (*FN2 6 OC Y)(CONS (*FN2 6 OC X) VAL))))
          (PREDCX (CSETQ VAL (CONS (*FN2 6 OC Y) VAL)))
          ((AND PREDCY PREDXY)
            (CSETQ VAL (CONS (*FN2 6 OC X)(CONS (*FN2 6 OC Y) VAL))))
          (PREDCY (CSETQ VAL (CONS (*FN2 6 OC X) VAL)))
        ))))
      (RETURN VAL)
    ))))
END OF EVALQUOTE, VALUE IS..
(*SMOVE)

```

connect[x; y]はxとyが隣接していればT, そうでなければNILを値とする関数である。 $connect[x; y] = |x-y|=1 \vee |x-y|=5$

④ ハッシュ化配列によるM, B, Pの移動可能配置の並べあげと現在位置による分割実行に先立ち配列MM, BB, PPの要素をセットしておく。

次に示すのはBBのセットの一部で, 例えば"Bが盤面の左上にいる時(B=0:基本位置0と1を占める場合)に(BB B)の評価により呼び出される値を与えている。もしすきまが下側に2つ並んでいたら(S=71), そこへ移動させた配置を生成する。もしすきまの一方が右横にある場合(X=2)には一つ右へ移動した配置を生成し, さらに他方のすきまもその右に隣接している場合にはそこにまで移動した配置の生成も付け加えられる。

FUNCTION EVALQUOTE HAS BEEN ENTERED, ARGUMENTS..!

```

PROGN (
  (SETA BB 0
    (QUOTE (LAMBDA ( ) (PROGN (CSETQ WK NIL)
      (COND ((EQ S 71) (CSETQ WK (CONS (*FN2 2 3 1) WK)))
        ((EQ X 2) (PROG2
          (COND ((EQ Y 3) (CSETQ WK (CONS (*FN2 2 2 1
            ) WK))))
          (CSETQ WK (CONS (*FN2 3 1 0 Y) WK))
        ))))
    ))))

```

⋮ (以下省略)

以上のように5箱入娘パズルの解法のためにH-分子212個, リストセル5200個が使用される。

⑤ 実行例 次ページに付録として添付する。

4. まとめ

PROM中に書き込まれたLisp処理系はこれを機会に再点検され, デバッグされた。現在の青山学院におけるLispプログラムの処理という観点から見れば記述力としては問題ないが, やはり大型の問題を解くには記憶容量を増やす方策を立てる必要がある。(速度はミニコン程度という感じである。付録参照) 現在例えば, 次の

いくつかの検討をすすめており、別の機会に報告を行ないたい。

① フロッピーディスクを接続し仮想化を行なう。ないしはDISK入出力命令をいれる。

② バルクメモリの配置とデータタイプの分類を変える。フリースレーゾ32K, H-領域32Kとし、数をH-分子としてH-領域中にとる。(スタッフ等はRAMへ持って行くが別のメモリをつくる。)

謝辞 日頃御指導頂く経営工学科間野若太郎教授に感謝いたします。

FUNCTION EVALQUOTE HAS BEEN ENTERED, ARGUMENTS...
ASSOCCOMP>(*SMOVE))

END OF EVALQUOTE, VALUE IS..
T

FUNCTION EVALQUOTE HAS BEEN ENTERED, ARGUMENTS...
NEXTCONF NIL

```

0
(018E023H 018BBC23H 018BBD23H)
1
(018BBB23H 018BB923H 018BBA23H 018E9523H 018E9623H 018E9E23H 018E9D23H)
2
(018E4127H 018E3A2BH 018E4227H 018E3263H 019E8023H 018E29A3H 018E3163H 018E9223H
01CB7D23H)
3
(41CB0923H 05CB1A23H 01CB8823H 01CB7E23H 028E17A3H 018E3AA2H 819E0423H 419E0C23H
019EA623H 019E8423H 018E2E63H 018E293BH 018E3D27H)
4
(028E173BH 018E673BH 018E2D3BH 018E2667H 019E8523H 019EB923H 019EA723H 419E1023H
809E1723H 018E74A2H 018E3EA2H 128E04A3H 01CB2B63H 01CB7F23H 05CB1C23H 01CB2063H
01CB9223H 01CB8923H 45CB0123H 41CB0A23H)
5
(01CB3C27H 01CB8023H 01CB2E63H 01CB3D27H 05CB1463H 05CB1D23H 41CB0B23H 01CB3B27H
41CB0263H 018E42A2H 018E9EA2H 018E78A2H 809E2913H 019CB023H 019CBE23H 0190BD23H
0198A223H 419E1123H 018E313BH 018E953BH 018E6B3BH 128E043BH)
6
(019E803BH 018E9E3AH 4190BE23H 01984627H 0198B823H 0198A123H 0190B623H 0190BB23H
0190BE23H 019DBB23H 019CB823H 019CB723H 019CB623H 809E3A12H 018E95B2H 41CB0327H
01CB2667H 01CB8323H 01CB8723H 01CB8823H 01CB8423H 05CB1527H)
7
(01D8A623H 01CB8E23H 019E80B2H 018E29BAH 018E31B6H 809E7412H 809E3E12H 019CA023H
019CA123H 019C9E23H 019CA223H 019EB923H 019DA923H 019DB523H 0190B823H 0190BC23H
0190A923H 0190BA23H 0190A023H 0190B523H 0190B723H 0198A023H 0198BE23H 0198B723H
01984227H 01984527H 4190B823H 4190B723H 4190BC23H 4190BD23H 018E3ABAH 018E427AH
819E043BH 419E0C3BH 019EA63BH 019E843BH)
8
YATTAZO
8
M=13B=8P1=0P2=1P3=2P4=3S=166
7
M=12B=8P1=0P2=1P3=2P4=3S=179
6
M=12B=8P1=0P2=1P3=2P4=3S=176
5
M=12B=11P1=0P2=1P3=2P4=3S=146
4
M=12B=11P1=0P2=1P3=2P4=3S=136
3
M=12B=11P1=0P2=1P3=2P4=3S=125
2
M=8B=11P1=0P2=1P3=2P4=3S=185
1
M=8B=11P1=0P2=1P3=2P4=3S=188
0
M=8B=11P1=0P2=1P3=2P4=3S=190
END OF EVALQUOTE, VALUE IS...
NIL

```

付録 実行例

実行にはいる前に $S_n = (C_0)$,
 $S_{n-1} = NIL$, $N = 0$ がセット
される。

C_0 の実行時間は印刷時間を含
めて約3分20秒である。(内、往
路3分、帰路は完全に印刷待ち
2分20秒) 所要メモリはリスト
セルが44K, H-分子が527エント
リである。*SMOVEの連想計
算を指差しないと帰路が約1分
を要し、4分の所要時間、リスト
セル55K, H-分子334エントリーに
なる。連想計算はL-分子の消費を60~70%にお
さえている。また、この時置えられた配置の総数が
133, 所要時間3分(往路)より8手解の場合は約
300分程度と推測される。

P1	P2	P3	P4
M		C1	C2
		B	
C3	C4		

初期配置 C_0

111秒: $PDP-11 = 10 =$
55

UNIX 7-8
手解3分は11