MINI−LISP 上での Portable Prolog について*

北 原 栄 子

1. はじめに

中島 [1] で詳説されている Portable Prolog はごく小さい基本的な Prolog の処理系の1つである。筆者はこれを MINI-LISP の上にインプリメントしたので本稿ではそれについて述べる。まず，1981年に MINI-LISP を MELCOM-COSMO (700I, 700S) に移植した [2] 後，処理系の若干のバージョンアップを行った。また，1982年12月1日以降，計算センターの運用面で，MINI-LISP をライブラリとして登録したので，MINI-LISP の起動方法に変更があった。**2.** でそのことを述べる。**3.** では MINI-LISP で Portable Prolog の処理系を書き，それを初期データとして与え，実際に Portable Prolog を実行させた例を述べる。**3.** で述べた Portable Prolog は実行時間が非常に遅いので，それを短縮するために処理系を改良した。**4.** ではその改良と結果について述べる。以下では [2] の知識を前提としている。

2. MINI-LISP について

2.1 MINI-LISP のバージョンアップ

MINI-LISP に Portable Prolog の処理系をもたせるためと，MINI-LISP の使い勝手を良くするために以下の修正を行った。

(1) MINI-LISP に Portable Prolog の処理系をもたせるためにセル領域を

原稿受領日 1983年4月15日
* 本稿執筆にあたり小樽商科大学戸島熙教授より多くの有益な suggest をいただいた。また57年度戸島ゼミ生稲岡睦夫，奥井亮一両君の卒業論文 [4] [5] の結果の一部を利用させていただいた。記して感謝の意を表する。

拡大する必要があったので，FORTRAN 言語で書かれた，所謂「商大版」の領域を2倍の10000に増した。具体的にはFORTRAN ソースプログラムのMCAR (5000), MCDR (4994) を MCAR (10000), MCDR (9994) に，NILの値を5000から10000に変更した。

(2) 新たな組み込み関数として，表1に示したNUMBERP とSYMBOLPを加えた。

表 1

関 数 名	引数の数	対応する整数	説　　明	備考
NUMBERP (*x*)	1	131	*intp* と同じ	*
SYMBOLP (*x*)	1	132	*x* が記号アトムなら *T*, それ以外は *F*	*

(3) プログラム実行中ERRORが発生した場合，従来は */ を入力することでエラー状態の回復を行っていたが，ERROR の後は自動的にエラー状態を回復するように「商大版」のサブルーチン ERROR を以下のように変更した。

```
952.000       SUBROUTINE ERROR(N)
953.000       COMMON/LCCODE/NPAR,NTR,LINEPR,NCCLMN,NPCLMN,IOCODE(46)
954.000       COMMON /IOMODE/IMDIN,IMDGB,IMDTSS
955.000       COMMON INTLSW
956.000       IF (IMDTSS) 10,20,10
957.000    10 INTLSW=1
958.000    20 CALL LINE(-2)
959.000       WRITE(6,1) N
960.000     1 FORMAT(36X,5HERROR,I5)
961.000 C     CALL PRINT(7)
962.000 C   2 NCCLMN=0
963.000 C     CALL INCARD
964.000       INTLSW=1
965.000       CALL LISP
966.000       RETURN
967.000       END
```

これによって BATCH 処理の場合も，ERROR が生じた次の S 式から再び正しく評価されるようになった。

2.2 MINI-LISP の起動の仕方

計算センターのライブラリとして登録されたことにより，起動の仕方がBATCH 処理，TSS 処理それぞれ以下のように変った。ライブラリのアカウントは **PAP** である。

2.2.1 BATCH−プログラム入力がカードの場合

```
! JOB ␣ account, name [, password]
! XEQ* ␣ (FILE, MINILISP**, PAP)
  LISP プログラム
! FIN
```

* 700 S の場合は ! XEQ ␣ MINILISP. PAP

** ファイル名 MINILISP. PAP の内容

```
! RUN ␣ (LMN, MIMIMP, PAP)
! DATA
```

初期データの指定がみえないのは，AMP (Assign-Merge Processor) を用いてロードモジュール (この場合のロードモジュール名は MIMIMP) 中に DCB のパラメータの値を含むようにしたためである。即ち次のようにロードモジュールを作成した。

```
!SET F:55/INTL2;IN
!AMP
?MIMIMP
FOLLOWING LMN
MIMIMP
FOLLOWING DCBS
F:55
*** AMP TERMINATED : WITH NEW DCBS :
```

2.2.2 BATCH—プログラム入力がユーザファイルの場合

```
! JOB ␣ account, name [, password]
! SET ␣ F: 105*/ユーザファイル名; IN
! XEQ** ␣ (FILE, MINILISP, PAP)
! FIN
```

* 700 S の場合は ! SET ␣ F: 5/ユーザファイル名; IN

** 700 S の場合は ! XEQ ␣ MINILISP. PAP

2.2.3 TSS—端末よりプログラムを入力する場合

```
 MELCOM AT YOUR SERVICE -M700S
13:33 APR 07,'83   USER# 41    LINE# 12

LOGON PLEASE:                                          ……①

ON AT 13:33 APR 07,'83
CHECK DG/MAILBOX
 EXECUTION  F10
                  ***     OTARU UNIVERSITY OF COMMERCE     ***
                  *            COMPUTER CENTER               *
                  *    MELCOM COSMO-700S TSS SERVICE TIME    *
                  *        MONDAY        15:00 - 20:00       *
                  *        TUESDAY        9:15 - 20:00       *
                  *        WEDNESDAY      9:15 - 20:00       *
                  *        THURSDAY       9:15 - 20:00       *
                  *        FRIDAY         9:15 - 20:00       *
                  *        SATURDAY       9:15 - 17:00       *
                  *    THE FIRST DAY OF A MONTH AT SEVICE    *
                  *    TIME (HOKUDAI)                        *
                  *        MONDAY        15:00 - 20:00       *
                  *      (TUE - FRI)     11:30 - 20:00       *
                  *        SATURDAY      11:30 - 17:00       *
                  ********************************************
                  DAI 1 DAI 3 MONDAY IGAI WA 9:00 - 20:00 MADE
                  SHIYOO DEKIMASU
```

```
*STOP* 0
!MINILISPT.PAP                                    ……②
?
      ⎧ プログラム入力
      ⎩
?                                                 ……③
!
```

① account, name [, password] を入力してログオンしセッションを開いて通信可能な状態にする。

② TSS 用ロードモジュール MINILISPT. PAP を起動させ入力促進記号 ▼?▼ (プロンプト) がでたらプログラム入力可能

③ LISP プログラムを終る時は ESC キーを2回押下する。

2. 2. 4 TSS−プログラム入力がユーザファイルの場合

```
 MELCOM AT YOUR SERVICE -M700S
13:59 APR 07,'83   USER# 27    LINE# 12

LOGON PLEASE:                                     ……①

ON AT 13:59 APR 07,'83
CHECK DC/MAILBOX
 EXECUTION  F1G
                ***     OTARU UNIVERSITY OF COMMERCE      ***
                *               COMPUTER CENTER             *
                *     MELCOM COSMO-700S TSS SERVICE TIME    *
                *        MONDAY        15:00 - 20:00        *
                *
                          ⎧
                          ⎩

!SET F:105/ ユーザファイル名 ;IN                  ……②
!MINILISPT.PAP                                    ……③
          ⎧
        ユーザファイル中の LISP プログラムの実行
          ⎩
*STOP*
!
```

①は **2. 2. 3** の①に同じ。

② LISP プログラムの入っているユーザファイルの指定

　＊ 700 S の場合

　! SET ␣ F: 5/ユーザファイル名; IN

③は **2. 3. 3** の②に同じ。

3. MINI-LISP 上での Portable Prolog

中島 [1] に示されている Portable Prolog は LISP で書かれた Prolog の処理系である。筆者はこれを MINI-LISP で書きかえ，MINI-LISP の関数の1つとして初期データに与え，MINI-LISP のプログラム実行中に関数 PROLOG を呼ぶことで Portable Prolog を使用出来るようにした。中島 [1] のものは

MACLISP によるものを書き直して基本的には LISP 1.5 の機能のみを使っている。これを MINI-LISP で書き直すためには，MINI-LISP の処理系の制約があり，いくつか変更を行う必要があった。以下それについて述べる。

3.1 変数とその値

Portable Prolog の変数は ▼*▼ で始まるアトムで示されていて (例 *A, *X)，変数のチェックにはアトムを分解する必要がある。しかし MINI-LISP にはそのための関数 (例えば EXPLODE) が組み込まれていないので，変数チェックのための関数 VAR? では関数 MEMQ を用いて次のように変数 *A〜*Z のチェックを行った。

```
DEFINE((
      (VAR? (LAMBDA (X)
            (MEMQ   X (QUOTE (*A *B *C *D *E *F *G *H *I *J
                              *K *L *M *N *O *P *Q *R *S *T
                              *U *V *W *X *Y *Z)) )  ))
      (MEMQ   (LAMBDA (X Y)
              (COND ((NULL Y) NIL)
                    ((EQ X (CAR Y)) T)
              (T (MEMQ   X (CDR Y))))))
```

だがこれでは変数になるものが制限され，*T1 とか *RESULT のような変数を扱えないことになる。これを避けるためには図 1 のようにすれば，(DECLEAR (*T1 *RESULT)) のように使用する変数をあらかじめ宣言することが出来るが，本稿ではその方法をとらなかった。

```
DEFINE(((DECLEAR (LAMBDA (X)
                 (SETQ VARLIST (APPEND X (QUOTE (*A *B *C *D *E *F *G
                                                 *H *I *J *K *L *M *N
                                                 *O *P *Q *R *S *T *U
                                                 *V *W *X *Y *Z))))))
       (APPEND (LAMBDA (X Y)
               (COND ((NULL X) Y)
                     ((ATOM X) (CONS X Y))
                     (T (CONS (CAR X) (APPEND (CDR X) Y))))))
       (VAR? (LAMBDA (X)
                     (MEMQ X VARLIST))) ))
```

図 1

3.2 関数の追加

MINI-LISP の組み込み関数は非常に少いため Portable Prolog 処理系で必要な関数を定義した。

① PROP [z ; y ; u]

MINI-LISP では 1 つの文字アトムに 1 つのセルが対応し，アトムの MCAR には印字名を文字値のリスト構造で表わした S-式のポインタ値が入り，

MCDR にはアトムの値が入れられる様になっている。関数 PROP は

```
DEFINE (( (PROP  (LAMBDA (Z Y U)
                 (SET  Z (CONS Y (CONS U NIL)))))
```

のように定義され，図2に示したように文字アトム z の値を，(u) の前に表示子 y をつけたものとしている。

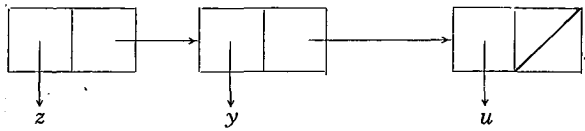

図 2

PROP は Portable Prolog 処理系の中では DEFINE-CLAUSE (初期データリスト（付表1）line no. 88～98) で使われ，Prolog のプログラムで定義された述語名（アトム）の値を，その述語名がプログラム中に初めて表れたときにはその定義の前に表示子 ▼PROLOG▼ をつけたリストとする。以下に示した例は Prolog プログラムで APPEND を述語として定義し実行したときに，関数 DEFINE-CLAUSE の PROP で APPEND の値がどう作られたかをトレースし，MINI-LISP のシステムの中でのアトム APPEND の値を出力させたものである。

```
?TRACE((PROP))
   APPLY  (TRACE ((PROP)))
     PN   1       23    321
   =====  NIL
     PN
?PROLOG()
   APPLY  (PROLOG NIL)
     PN   1          1
   PRINT  (**** PORTABLE PROLOG (IN MINILISP) ****)
     PN   1                          2          2     1
?+(APPEND () *X *X)=
1 *APPLY  (PROP (APPEND PROLOG (((APPEND NIL *X *X)))))
     PN   1      2              345                 54321
1     *=  (PROP (PROLOG (((APPEND NIL *X *X)))))
     PN   1      2        345                 54321
   PRINT  DEFINED
     PN
?-(APPEND (1(A B C D E F) *L)-(CALL PRINT *L)=
   PRINT  (A B C D E F)
     PN   1              1
   PRINT  *S*
     PN
?-(END)=
   =====  EPILOG
     PN
?EVAL(APPEND)
   APPLY  (EVAL (APPEND))
     PN   1      2      21
   =====  (PROLOG (((APPEND NIL *X *X))))
     PN   1        234                 4321
```

② GET $[x; y]$

x は文字アトム，y は表示子である。関数 GET は

```
DEFINE ((
  (GET (LAMBDA (X Y)
       (COND ((NULL X) NIL)
       ((ATOM (SETQ EX (EVAL X))) NIL)
       ((EQ    (CAR EX) Y)(CADR EX))
       (T (GET (CDDR EX) Y))) ))
```

のように定義されている。即ちアトム x の値のリストを (SETQ EX (EVAL X)) で求めて，求まったリストを順に探し，表示子 y と等しいものを見つける。もし見つかれば表示子 y の後のリストをその値とする。もし見つからなかったり，アトム x の値がアトムであれば GET の返す値を NIL とする。GET は関数 DEFINE-CLAUSE, REFUTES に使われている。①の PROP の例で作成した APPEND を使って GET の使用例を示す。

```
?GET(APPEND PROLOG)
  APPLY  (GET (APPEND PROLOG))
    PN   1    2              21
  =====  (((APPEND NIL *X *X)))
    PN   123                 321
```

③ NCONC [x; y]

x, y ともにリスト。リスト x そのものにリスト y を追加してリスト x を修正する。即ちリスト x の末尾のセルの cdr 部をかきかえて y に接続する。従って新しいセルを作り出さずにリスト x とリスト y をつけることになる。

④ MAPCAR [x; f]

x の値であるリスト (x_1, x_2...x_n) の各要素に関数 f を作用させたもの ($f[x_1]$ $f[x_2]$...$f[x_n]$) を値として返す関数である。

⑤ AND [x_1, x_2..., x_n]

x_1 から順に評価し，NIL が現れたら値は NIL となり，その後の式は評価しない。最後まで評価して NIL が現れなければこの値は T となる。この関数の定義には MINI-LISP の non-spread λ を用いている。

3.3 トレース

Portable Prolog は引数 NEW-SUBST や OLD-SUBST で循環リストを作るために，多くの LISP システムではトレース出来る Portable Prolog の関数が制約されると思われるが，MINI-LISP は循環リストであれば ▼0L▼ というアトムを出力することになっている [3] ので全ての関数のトレースが可能であり，Portable Prolog の実行状況をみることが出来る。しかし

```
+(APPEND(  )*X *X):
+(APPEND(*A. *X) *Y(*A. *Z))-(APPEND *X *Y *Z):
-(APPEND(A B)(C D) *L)-(CALL PRINT * L):
-(END):
```

のプログラムをトレースするとLP用紙40枚の出力となる。

3.4 そ の 他

MINI-LISPのシステム組み込み関数名をPrologの述語名として使用すると，その関数が再定義されることになり，関数PROLOGを-(END):でぬけてもその定義は変えられない。Prologのプログラムを書くときは，初期データリストのline no. 1～10に示されているシステム組み込み関数名をPrologの述語名として使ってはいけない。

3.5 実 行 例

このようにしてMINI-LISP上に作成したPortable Prologを用いてPrologのプログラムを実行した例を以下に示す。

① BATCH処理実行例

4-QUEENを述語として定義したプログラムのファイルをユーザファイル(ファイル名: FOUR-QUEEN)として読み込み実行した例(700Sによる)。

—入力カードデック—

```
                                !JOB  account , name (,Password)
                                !LIMIT (TIME,30)
                                !SET F:5/FOUR-QUEEN
                                !XEQ MINILISP.PAP
                                !FIN
```

—出力結果—

```
                        INPUT=DEFINE(((EQUAL (LAMBDA(X Y)
                        INPUT=                (COND ((ATOM X)(COND ((ATOM Y )(EQ X Y)) (T F) ))
                        INPUT=                      ((EQUAL (CAR X)(CAR Y))(EQUAL (CDR X)(CDR Y)))
                        INPUT=                      (T F))) )))
APPLY  (DEFINE (((EQUAL (LAMBDA (X Y) (COND ((ATOM X) (COND ((ATOM Y) (EQ X Y)) (T,F))) ((EQUAL (CAR X) (CAR Y)) (EQU
  PN   1      234     5      6  6 6     78     8 8     9A      A A      A9 9   987 78      9     9 9     98 8
       AL (CDR X) (CDR Y))) (T F)))))))
  PN       9     9 9     987 7   7654321
*****  NIL
  PN
                        INPUT=PROLOG()
APPLY  (PROLOG NIL)
  PN   1      1
PRINT  (**** PORTABLE PROLOG (IN MINILISP) ****)
  PN   1                    2          2     1
                          INPUT=+(FQUEEN *Q)=(QUEENS (1 2 3 4)   NIL *Q):
PRINT  DEFINED
  PN
                        INPUT=+(QUEENS NIL *Y *Y):
PRINT  DEFINED
  PN
```

```
                                INPUT=+(QUELNS *X *Y *Z)-(SELECT *U *X *V)-(SAFE *U *Y 1)
                                INPUT=-(QUEENS *V (*U.*Y) *Z)$
  PRINT  DEFINED
    PN
                                INPUT=+(SELECT *X (*X.*Y) *Y)$
  PRINT  DEFINED
    PN
                                INPUT=+(SELECT *U (*X.*Y) (*X.*V))-(SELECT *U *Y *V)$
  PRINT  DEFINED
    PN
                                INPUT=+(SAFE *U NIL *W)$
  PRINT  DEFINED
    PN
                                INPUT=+(SAFE *U (*P.*Q) *N)-(NODIAG *U *P *N)-(EVAL (ADD1 *N ) *M)
                                INPUT=-(SAFE *U *Q *M)$
  PRINT  DEFINED
    PN
                                INPUT=+(NODIAG *U *P *N)-(EVAL (A+ *P *N) *T)-(EVAL (A- *P *N) *S)
                                INPUT=-(EVAL (EQUAL *U *T)   F)-(EVAL  (EQUAL *U *S)  F)$
  PRINT  DEFINED
    PN
                                INPUT=-(FQUEEN *L)-(CALL PRINT *L)$
  PRINT  (S 1 4 2)
    PN   1       1
  PRINT  *S*
    PN
                                INPUT=-(END)$
  *****  EPILOG
    PN
*STOP*
```

② TSS 処理

リストの接続を行う述語 APPEND の実行例(700S による)

```
 MELCOM AT YOUR SERVICE -M700S
14:22 MAR 31,'83   USER# 1E    LINE# 12

LOGON PLEASE:

ON AT 14:22 MAR 31,'83
CHECK DC/MAILBOX
 EXECUTION  F10
                ###     OTARU UNIVERSITY OF COMMERCE     ###
                #            COMPUTER CENTER              #
                #    MELCOM COSMO-700S TSS SERVICE TIME   #
                #       MONDAY        15:00 - 20:00       #
                #       TUESDAY        9:15 - 20:00       #
                #       WEDNESDAY      9:15 - 20:00       #
                #       THURSDAY       9:15 - 20:00       #
                #       FRIDAY         9:15 - 20:00       #
                #       SATURDAY       9:15 - 17:00       #
                #    THE FIRST DAY OF A MONTH AT SEVICE   #
                #    TIME (HOKUDAI)                       #
                #        MONDAY        15:00 - 20:00       #
                #       (TUE - FRI)    11:30 - 20:00       #
                #        SATURDAY      11:30 - 17:00       #
                ##########################################
                DAI 1 DAI 3 MONDAY IGAI WA 9:00 - 20:00 MADE
                SHIYOO DEKIMASU

 *STOP* 0

 !MINILISPT.PAP
 ?MDIN(0)
                                    INPUT=MDIN(0)
    APPLY  (MDIN (0))
      PN   1     2 21
    =====  0
      PN
 ?PROLOG()
    APPLY  (PROLOG NIL)
      PN   1          1
    PRINT  (**** PORTABLE PROLOG (IN MINILISP) ****)
      PN   1                    2          2     1
 ?+(APPEND () *X *X):
    PRINT  DEFINED
      PN
 ?+(APPEND (*A.*X) *Y (*A.*Z))-(APPEND *X *Y *Z):
    PRINT  DEFINED
      PN
 ?-(APPEND (A B C)(D E F) *L)-(CALL PRINT *L):
    PRINT  (A B C D E F)
      PN   1          1
```

```
   PRINT  *S*
     PN
?-(APPEND (THIS IS) *X (THIS IS PORTABLE PROLOG))-(CALL PRINT *X):
   PRINT  (PORTABLE PROLOG)
     PN   1              1
   PRINT  *S*
     PN
?-(APPEND *X (A GIRL)(SHE IS A GIRL))-(CALL PRINT *X):
   PRINT  (SHE IS)
     PN   1      1
   PRINT  *S*
     PN
?-(END):
   =====  EPILOG
     PN
?
```

4. Portable Prolog の改良

3. で述べた Portable Prolog の処理系では 4−QUEEN（**3.5** の BATCH 処理例）の最初の解が求まるまで14分44秒93かかった。これは他の LISP 言語でインプリメントされた Portable Prolog の処理系に比べて非常に遅い。そこで **3.** で作成した Portable Prolog の実行時間を短縮させるためにいくつかの改良を行った。その第1は，MINI-LISP では PROG はシステム組み込み関数とされておらず LISP で書かれた1つの関数として提供されているため，Portable Prolog の処理系で PROG を使用するとその度にこの PROG 関数が使用されることになるので **3.** の処理系を PROG を使用せずに書き直してみた。第2に Prolog の処理系をトレースしてみると変数チェックを頻繁に行っていることがわかるので，中島 [1] の示唆に従って Prolog の変数をリストとして表わす方法をとってみた（例えば *A, *X を (VAR A), (VAR X) とする)。更に Portable Prolog 中の関数の数を出来るだけ少くすることも試みた。

4. 1 PROG を使わず Portable Prolog を書き直した結果と効果

初期データリストのline no. 11～37 までが MINI-LISP で提供されている PROG 関数の定義である。LISP プログラム中に PROG が使われると，その都度 PROG から *GO までの各関数を解釈実行することになる。Portable Prolog の処理系では PROG が PROLOG, READ-EXEC, READ-REST, READ-SIGN, DEFINE-CLAUSE, REFUTE, TRY-SYS, FETCH, FETCH-VALUE, NCONC の各関数で使われているのでそれらを初期入力データに示

したように PROG を使わずに書きかえた。その処理系を使ってプログラムの実行時間を測った結果が表4の (2) である。

4.2 変数をリストとして表現した結果と効果

Prolog の表記上の問題から Portable Prolog の変数をアトムとすると，MINI-LISP にはアトムを分解するための関数が組み込まれていないので変数チェックは **3.1** に示したように関数 MEMQ を使って行うことになる。しかしこれでは頻繁に使われる変数チェックの関数 VAR? の効率が極めて悪いと思われたので，筆者は中島 [1] の示唆に従って変数をリストとして表現し，その car 部をとって ▼VAR▼ と等しいかどうかを判定することで変数チェックを行う方法をとった。そのために関数 VAR? と ASSOC を以下のように変えた。

```
(VAR? (LAMBDA(X)
      (COND ((ATOM X) NIL)
            ((EQ (QUOTE VAR)(CAR X)) T)
            (T NIL))))
(ASSOC  (LAMBDA (X L)
        (COND ((ATOM L) NIL)
              ((EQUAL X (CAAR L)) (CAR L))
              (T (ASSOC X (CDR L))))))
```

この処理系を使ってプログラムの実行時間を測った結果が表4の (3) である。この結果は **3.** の処理系の約4倍早い処理系が出来たことを示している。しかし変数をリストとして表わす方法は Prolog のプログラムの表記が煩雑になるばかりでなく，処理系全体の領域が大きくなるなど問題も多いと思われる。これは MINI-LISP にアトムを分解するための関数を組み込むことで解決されるので，MINI-LISP の今後の課題となるであろう。

4.3 新しい Portable Prolog の処理系

4.1, **4.2** で述べたように Portable Prolog を書き直し，さらに関数 ASSIGNED?, FIRST, SECOND, THIRD を関数として定義しないようにした新しい Portable Prolog の処理系が初期入力データの line no. 51～246 に示したものである。これを用いてプログラムの実行時間を測ったものが表4の (4) である。

初めにインプリメントした Portable Prolog の処理系のおよそ8倍早く実行することの出来る処理系を得ることが出来たが，筆者は充分に満足できる結果

表4　各処理系の実行時間

プログラム	結果	(1)	(2)	(3)	(4)
+(APPEND () *X *X)‡ +(APPEND (*A.*X) *Y (*A.*Z))-(APPEND *X *Y *Z)‡ -(APPEND (A B)(C D) *L)-(CALL PRINT *L)‡ -(END)‡	PRINT (A B C D) PN 1 1 PRINT *S* PN	秒 28 07	秒 24 71	秒 6 83	秒 3 58
+(REVERSE *L *R)-(REV1 *L () *R)‡ +(REV1 () *R *R)‡ +(REV1 (*X.*T) *W *R)-(REV1 *T (*X.*W) *R)‡ -(REVERSE (1 2 3 4 5 6 7 8 9) *L)-(CALL PRINT *L)‡ -(END)‡	PRINT (9 8 7 6 5 4 3 2 1) PN 1 1 PRINT *S* PN	分 秒 1 44 78	分 秒 1 34 34	秒 24 82	秒 12 83
+(FACT 0 1)‡ +(FACT *N *X)-(EVAL (SUB1 *N) *M)-(FACT *M *O) -(EVAL (A* *N *O) *X)‡ -(FACT 5 *L)-(CALL PRINT *L)‡ -(END)‡	PRINT 120 PN PRINT *S* PN	秒 56 96	秒 47 77	秒 15 86	秒 7 24
+(SENT *B *Y)-(NP *B *Z)-(VP *Z *Y)‡ +(NP *B *Y)-(DET *B *Z)-(NOUN *Z *Y)‡ +(DET (A.*X) *X)‡ +(DET (THE.*X) *X)‡ +(NOUN (ELEPHANT.*X) *X)‡ +(NOUN (UNICORN.*X) *X)‡ +(VP *B *Y)-(V1 *B *Y)‡ +(VP *B *Y)-(VT *B *Z)-(NP *Z *Y)‡ +(V1 (DREAMS.*A) *X)‡ +(VT (LIKES.*X) *X)‡ -(SENT *X ())-(CALL PRINT *X)-(REPEAT)‡ -(END)‡	PRINT (A ELEPHANT DREAMS) PN 1 1 PRINT (A ELEPHANT LIKES A ELEPHANT) PN 1 1 PRINT (A ELEPHANT LIKES A UNICORN) PN 1 1 ⋮ PRINT (THE UNICORN LIKES THE UNICORN) PN 1 1 PRINT NIL PN	分 秒 5 28 90	分 秒 4 41 59	分 秒 1 32 86	秒 41 47
+(FQUEEN *L)-(QUEENS (1 2 3 4) NIL *Q)‡ +(QUEENS NIL *Y *Y)‡ +(QUEENS *X *Y *Z)-(SELECT *U *X *V)-(SAFE *U *Y 1) -(QUEENS *V (*U.*Y) *Z)‡ +(SELECT *X (*X.*Y) *Y)‡ +(SELECT *U (*X.*Y) (*X.*V))-(SELECT *U *Y *V)‡ +(SAFE *U NIL *M)‡ +(SAFE *U (*P.*Q) *N)-(NODIAG *U *P *N)-(EVAL (ADD1 *N) *M) -(SAFE *U *Q *M)‡ +(NODIAG *U *P *N)-(EVAL (A* *P *N) *T)-(EVAL (A- *P *N) *S) -(EVAL (EQUAL *U *T) F)-(EVAL (EQUAL *U *S) F)‡ -(FQUEEN *L)-(CALL PRINT *L)‡ -(END)‡	PRINT (3 1 4 2) PN 1 1 PRINT *S* PN	分 秒 14 44 93	分 秒 13 9 00	分 秒 3 51 83	分 秒 2 6 89
-(FQUEEN *L)-(CALL PRINT *L)-(REPEAT)‡	PRINT (3 1 4 2) PN 1 1 PRINT (2 4 1 3) PN 1 1 PRINT NIL PN	分 秒 33 47 00	分 秒 29 40 00	分 秒 8 44 97	分 秒 4 48 93

表 4

注) (1) **3.** でインプリメント

(2) (1) を PROG を使わずに書き直した処理系

(3) (1) を変数をリストとして扱うように書き直した処理系

(4) (2) と (3) を併せた処理系

(3), (4) の処理系で使われた Prolog のプログラムは *A を (VAR A), *X を (VAR X) と書きかえたものである。例えば APPEND についてみると以下のようなプログラムとなる。

```
+(APPEND () (VAR X) (VAR X))!
+(APPEND((VAR A).(VAR X))(VAR Y)((VAR A).(VAR Z)))
-(APPEND(VAR X)(VAR Y)(VAR Z))!
-(APPEND (A B)(C D) (VAR L))-(CALL PRINT (VAR L))!
-(END)!
```

時間の測定は BATCH で各プログラムを実行し，その CPU TIME から空のプログラム[2]の CPU TIME を引いたものを Prolog プログラムの実行時間とした。これは MINI-LISP の初期データを評価するための時間を除くためである。

とは思っていない。しかし MINI-LISP が FORTRAN 言語で書かれたインタプリタであり，更に Portable Prolog がこの MINI-LISP で書かれていることを考えれば，この遅さは止むを得ないことかも知れない。Prolog 処理系が実用的であるためには Assembly 言語で書かれた PROLOG インタプリタが必要である[1]が，この MINI-LISP で書かれた Potable Prolog も初歩的なプログラミングの練習用として使用することが出来ると考えられる。

<参 考 文 献>

1) 中島秀之「Prolog 入門」1～3,「bit」vol. 14. No. 5～7, 1982.

2) 北原栄子「MINI-LISP の移植」「商学討究」, 第 32 巻第 2 号, 1981, pp. 113～131.

3) 後藤英一・戸島 熙・石畑 清「記号処理の基礎と応用」(情報処理叢書 8), 情報処理学会, 1981.

4) 稲岡睦夫「MINI-LISP における PORTABLE PROLOG の変更—PROG 機能を使わずに定義—」。

5) 奥井亮一「MINI-LISP SYSTEM の改良—PROLOG の改良—」。

---

1) 近くそのようなProlog インタプリタを学内的にのみ試用に供することを計画している。

2) 空のプログラムの内容は

```
PROLOG (  )
-(END):
```

である。

付表1 初期データリスト

```
  1 -     1.000  301 (NIL F T LAMBDA LABEL TRFLAG COND QUOTE SETQ PROG2 AND OR F1 F2 F3 PROG)
  2 -     2.000   108 (((READ 1) (READ1 2) (RSERV 3) (PRINT 101) (PRINT1 102) (QUOTE 103) (EVAL
  3 -     3.000    104) (ATOM 105) (CAR 106) (CDR 107) (DEFINE 108) (INTP 109) (MDIN 110)
  4 -     4.000  (ADD1 111) (SUB1 112) (ZEROP 113) (MINUS 114) (ONEP 115) (LENGTH 116)
  5 -     5.000  (MDGB 117)(CAAR 118) (CADR 119)( CDAR 120)(CDDR 121) (CAAAR 122) (CAADR 123)
  6 -     6.000  (CADAR 124) (CDAAR 125) (CADDR 126) (CDDAR 127) (CDADR 128) (CDDDR 129)
  7 -     7.000  (MDTSS 130) (NUMBERP 131) (SYMBOLP 132)
  8 -     8.000  (CONS 201) (EQ 202) (APPLY 203) (RPLACA 204) (RPLACD 205) (PROG2 206)
  9 -     9.000  (SETQ 207) (SET 208) (GTP 209) (LTP 210) (A+ 211) (A- 211)
 10 -    10.000  (A* 213) (A/ 214) (EXPT 215) (DIVIDE 216)   (LIST 301) ))
 11 -    11.000 DEFINE((
 12 -    12.000 (PROG (LAMBDA *U ((LAMBDA (*V) (COND ((ATOM (SETQ *W (CAR *U))) (*PROG))
 13 -    13.000       (T (*PV1)))) *U)))
 14 -    14.000 (*PV1 (LAMBDA () (APPLY (LIST (QUOTE LAMBDA) (LIST (CAR *W))
 15 -    15.000       (QUOTE (COND ((SETQ *W (CDR *W)) (*PV1)) (T (*PROG))))) (LIST NIL))))
 16 -    16.000 (*PROG (LAMBDA () (COND
 17 -    17.000       ((ATOM (SETQ *V (CDR *V))) NIL)
 18 -    18.000       ((ATOM (SETQ *W (CAR *V))) (*PROG))
 19 -    19.000       (T (*PEVAL)))))
 20 -    20.000 (*PEVAL (LAMBDA () (COND
 21 -    21.000       ((EQ (SETQ *CARW (CAR *W)) (QUOTE COND)) (*PCOND))
 22 -    22.000       ((EQ *CARW (QUOTE RETURN)) (COND ((ATOM (SETQ *W (CDR *W))) NIL)
 23 -    23.000            (T (EVAL (CAR *W))))))
 24 -    24.000       ((EQ *CARW (QUOTE GO)) (PROG2 (LIST (SETQ *W (CAR (CDR *W)))
 25 -    25.000            (SETQ *V *U)) (*GO)))
 26 -    26.000       ((LIST (EVAL *W)) (*PROG)))))
 27 -    27.000 (*PCOND (LAMBDA () (COND
 28 -    28.000       ((ATOM (SETQ *W (CDR *W))) (*PROG))
 29 -    29.000       ((ATOM (SETQ *CARW (CAR *W))) (PRINT (QUOTE (ERROR IN *PCOND))))
 30 -    30.000       (((LAMBDA (*W) (EVAL (CAR *CARW))) *W) (COND
 31 -    31.000            ((ATOM (SETQ *W (CAR (CDR (CAR *W))))) (*PROG)) (T (*PEVAL))))
 32 -    32.000       (T (*PCOND)))))
 33 -    33.000 (*GO (LAMBDA () (COND
 34 -    34.000       ((ATOM (SETQ *V (CDR *V))) NIL)
 35 -    35.000       ((EQ *W (CAR *V)) (*PROG))
 36 -    36.000       (T (*GO)))))
 37 -    37.000 ))
 38 -    38.000 DEFINE ((
 39 -    39.000 (TRACE (LAMBDA (TLIST) (COND
 40 -    40.000       ((ATOM TLIST)  NIL)
 41 -    41.000       (T (PROG2 (SET (CAR TLIST)  (CONS TRFLAG (104  (CAR TLIST))))
 42 -    42.000                 (      TRACE (CDR TLIST))))
 43 -    43.000 )))
 44 -    44.000 (UNTRACE (LAMBDA (TLIST) (COND
 45 -    45.000       ((ATOM TLIST)  NIL)
 46 -    46.000       (T (PROG2 (SET (CAR TLIST)  (CDR (104  (CAR TLIST))))
 47 -    47.000                 (      UNTRACE (CDR TLIST))))
 48 -    48.000 )))
 49 -    49.000       (NULL (LAMBDA (X) (EQ X NIL)))
 50 -    50.000 ))
 51 -    51.000 DEFINE((
 52 -    52.000 (PROLOG (LAMBDA NIL
 53 -    53.000                 (PROG2 (PRINT (QUOTE (***PORTABLE PROLOG (IN MINILISP VER.02)***)
 54 -    54.000                 ))
 55 -    55.000                  (PROG2 (SETQ EPILOG NIL)
 56 -    56.000                         (PROLOG1 RESULT FETCHED-SUBST EPILOG)))))
 57 -    57.000        (PROLOG1 (LAMBDA (RESULT FETCHED-SUBST EPILOG)
 58 -    58.000                 (PROG2 (SETQ RESULT (READ-EXEC SIGN))
 59 -    59.000                        (COND ((104 EPILOG)(QUOTE EPILOG))
 60 -    60.000                              ((EQ RESULT (QUOTE SYNTAX-ERROR))
 61 -    61.000                               (PROLOG1 RESULT FETCHED-SUBST EPILOG))
 62 -    62.000                              (T (PROG2  (PRINT RESULT)
 63 -    63.000                                         (PROLOG1  RESULT FETCHED-SUBST EPILOG)))
 64 -    64.000        ))))
 65 -    65.000         (READ-EXEC (LAMBDA (SIGN)
 66 -    66.000                (PROG2 (SETQ SIGN (READ))
 67 -    67.000                       (COND ((EQ SIGN (QUOTE +))(DEFINE-CLAUSE (READ-REST)))
 68 -    68.000                             ((EQ SIGN (QUOTE -))(REFUTE-CLAUSE (READ-REST)))
 69 -    69.000                             (T (READ-ERROR SIGN))) )))
 70 -    70.000         (READ-REST (LAMBDA NIL
 71 -    71.000            (READ-REST1  FORM REST)))
 72 -    72.000         (READ-REST1 (LAMBDA (FORM REST)
 73 -    73.000                (PROG2 (SETQ FORM (READ))
 74 -    74.000                       (COND ((ATOM FORM)(READ-ERROR FORM))
 75 -    75.000                             ((EQ (SETQ REST (READ-SIGN))(QUOTE SYNTAX-ERROR))
 76 -    76.000                             (QUOTE SYNTAX-ERROR))
 77 -    77.000                             (T (CONS FORM REST)))) ))
 78 -    78.000         (READ-SIGN  (LAMBDA NIL
 79 -    79.000                (PROG2
 80 -    80.000                (SETQ SIGN (READ))
 81 -    81.000                (COND ((EQ SIGN (QUOTE -))(READ-REST))
 82 -    82.000                      ((EQ SIGN (QUOTE +)) NIL)
 83 -    83.000                      (T (READ-ERROR SIGN))))))
 84 -    84.000         (READ-ERROR  (LAMBDA (OBJ)
 85 -    85.000                (PROG2
 86 -    86.000                (PRINT (LIST (QUOTE SYNTAX)(QUOTE ERROR) OBJ))
```

```
 87 -  87.000              (QUOTE SYNTAX-ERROR))))
 88 -  88.000       (DEFINE-CLAUSE  (LAMBDA (CLAUSE)
 89 -  89.000       (COND((EQ CLAUSE (QUOTE SYNTAX-ERROR))(QUOTE SYNTAX-ERROR))
 90 -  90.000             (T (PROG2
 91 -  91.000                  (PROG2
 92 -  92.000                  (SETQ DEFINITION (GET (CAAR CLAUSE) (QUOTE PROLOG)))
 93 -  93.000                  (COND((NULL DEFINITION)
 94 -  94.000                        (PROP (CAAR CLAUSE)
 95 -  95.000                              (QUOTE PROLOG)
 96 -  96.000                              (CONS CLAUSE NIL)))
 97 -  97.000                       (T (ADDQ DEFINITION (CONS CLAUSE NIL)))))
 98 -  98.000                  (QUOTE DEFINED))))))
 99 -  99.000       (REFUTE-CLAUSE  (LAMBDA (CLAUSE)
100 - 100.000       (COND((EQ CLAUSE (QUOTE SYNTAX-ERROR))(QUOTE SYNTAX-ERROR))
101 - 101.000             (T (REFUTES CLAUSE
102 - 102.000                         (CONS NIL NIL)
103 - 103.000                         (CONS NIL NIL)
104 - 104.000                         NIL)))))
105 - 105.000       (REFUTES  (LAMBDA (CLAUSE NEW-SUBST OLD-SUBST CUE)
106 - 106.000       (COND ((NULL CLAUSE)
107 - 107.000              (COND ((NULL CUE) (QUOTE *S*))
108 - 108.000                    (T (REFUTES (CAAR CUE)
109 - 109.000                                (CONS NIL NIL)
110 - 110.000                                (CDAR CUE)
111 - 111.000                                (CDR CUE)))))
112 - 112.000             (T (REFUTE CLAUSE
113 - 113.000                        (GET (CAAR CLAUSE) (QUOTE PROLOG)))))))
114 - 114.000       (REFUTE  (LAMBDA (CLAUSE DEFINITIONS)
115 - 115.000              (REFUTE1 CLAUSE DEFINITIONS UNDO-LIST)))
116 - 116.000       (REFUTE1 (LAMBDA (CLAUSE DEFINITIONS UNDO-LIST)
117 - 117.000         (PROG2
118 - 118.000         (SETQ UNDO-LIST NIL)
119 - 119.000         (COND ((NULL DEFINITIONS)
120 - 120.000                (COND ((AND (TRY-SYS (CAR CLAUSE) OLD-SUBST)
121 - 121.000                            (REFUTES (CDR CLAUSE)
122 - 122.000                                     (CONS NIL NIL)
123 - 123.000                                     OLD-SUBST
124 - 124.000                                     CUE))
125 - 125.000                       (QUOTE *S*))
126 - 126.000                      (T (UNDO UNDO-LIST))))
127 - 127.000              (T (RESOLVE DEFINITIONS))))))
128 - 128.000       (RESOLVE  (LAMBDA (DEFINITIONS)
129 - 129.000        (COND ((NULL DEFINITIONS)NIL)
130 - 130.000              ((AND (UNIFY (CAR CLAUSE)
131 - 131.000                           OLD-SUBST
132 - 132.000                           (CAAR DEFINITIONS)
133 - 133.000                           NEW-SUBST)
134 - 134.000                    (REFUTES (CDAR DEFINITIONS)
135 - 135.000                             (CONS NIL NIL)
136 - 136.000                             NEW-SUBST
137 - 137.000                             (CONS (CONS (CDR CLAUSE) OLD-SUBST) CUE)))
138 - 138.000               (QUOTE *S*))
139 - 139.000              (T (PROG2
140 - 140.000                 (UNDO UNDO-LIST)
141 - 141.000                 (RESOLVE (CDR DEFINITIONS)))))))
142 - 142.000       (UNDO  (LAMBDA (U)
143 - 143.000         (COND ((NULL U)(SETQ UNDO-LIST NIL))
144 - 144.000               (T (PROG2
145 - 145.000                  (RPLACD (CAR U) (CDDAR U))
146 - 146.000                  (UNDO (CDR U)))))))
147 - 147.000       (TRY-SYS  (LAMBDA (FORM SUBST)
148 - 148.000         (COND ((EQ (CAR FORM) (QUOTE CALL))
149 - 149.000                (PROG2
150 - 150.000                (APPLY (FETCH-VALUE (CADR FORM) SUBST)
151 - 151.000                       (MAPCAR (CDDR FORM)
152 - 152.000                               (QUOTE (LAMBDA (X)
153 - 153.000                                      (FETCH-VALUE X SUBST)))))
154 - 154.000                ))
155 - 155.000               ((EQ (CAR FORM) (QUOTE EVAL))
156 - 156.000                (COND((UNIFY (APPLY (FETCH-VALUE (CAADR FORM) SUBST)
157 - 157.000                                    (MAPCAR (CDADR FORM)
158 - 158.000                                            (QUOTE (LAMBDA (X)
159 - 159.000                                                   (FETCH-VALUE X SUBST)))))
160 - 160.000                             SUBST
161 - 161.000                             (CADDR FORM)
162 - 162.000                             SUBST) T)
163 - 163.000                      (T NIL)))
164 - 164.000               ((EQ (CAR FORM) (QUOTE END))
165 - 165.000                (SETQ EPILOG T))
166 - 166.000               (T NIL))))
167 - 167.000       (UNIFY  (LAMBDA (X X-SUBST Y Y-SUBST)
168 - 168.000       (COND ((VAR? X)
169 - 169.000              (COND ((ASSOC X (CDR X-SUBST))
170 - 170.000                     (UNIFY (FETCH X X-SUBST)
171 - 171.000                            FETCHED-SUBST
172 - 172.000                            Y
173 - 173.000                            Y-SUBST))
174 - 174.000                    (T (LINK X X-SUBST Y Y-SUBST))))
175 - 175.000          ((VAR? Y) (UNIFY Y Y-SUBST X X-SUBST))
176 - 176.000          ((ATOM X) (EQ X Y)))
```

```
177 - 177.000            ((ATOM Y) NIL)
178 - 178.000            ((UNIFY (CAR X) X-SUBST (CAR Y) Y-SUBST)
179 - 179.000             (UNIFY (CDR X) X-SUBST (CDR Y) Y-SUBST))
180 - 180.000            (T NIL))))
181 - 181.000      (VAR? (LAMBDA(X)
182 - 182.000            (COND ((ATOM X) NIL)
183 - 183.000                  ((EQ (QUOTE VAR)(CAR X)) T)
184 - 184.000                  (T NIL))))
185 - 185.000       (FETCH  (LAMBDA (X SUBST)
186 - 186.000            (PROG2
187 - 187.000            (SETQ FETCHED-SUBST SUBST)
188 - 188.000            (COND ((VAR? X)
189 - 189.000                   (PROG2
190 - 190.000                   (SETQ V (ASSOC X (CDR SUBST)))
191 - 191.000                   (COND ((NULL V) X)
192 - 192.000                         (T (PROG2
193 - 193.000                            (SETQ FETCHED-SUBST (CDDR V))
194 - 194.000                            (FETCH (CADR V) (CDDR V)))))))
195 - 195.000                  (T X)))))
196 - 196.000       (FETCH-VALUE  (LAMBDA (X SUBST)
197 - 197.000            (COND ((VAR? X)
198 - 198.000                   (PROG2
199 - 199.000                   (SETQ V (ASSOC X (CDR SUBST)))
200 - 200.000                   (COND ((NULL V) X)
201 - 201.000                         (T (FETCH-VALUE (CADR V) (CDDR V))))))
202 - 202.000                  ((ATOM X) X)
203 - 203.000                  (T (CONS (FETCH-VALUE (CAR X) SUBST)
204 - 204.000                           (FETCH-VALUE (CDR X) SUBST))))))
205 - 205.000       (LINK  (LAMBDA (X X-SUBST Y Y-SUBST)
206 - 206.000        (COND ((AND (EQ X Y) (EQ X-SUBST Y-SUBST)) T)
207 - 207.000              (T (SETQ UNDO-LIST
208 - 208.000                       (CONS
209 - 209.000                       (RPLACD X-SUBST (CONS
210 - 210.000                        (CONS X (CONS (FETCH Y Y-SUBST)
211 - 211.000                                      FETCHED-SUBST))
212 - 212.000                                       (CDR X-SUBST)))
213 - 213.000                       UNDO-LIST))))))
214 - 214.000       (ASSOC  (LAMBDA (X L)
215 - 215.000             (COND ((ATOM L) NIL)
216 - 216.000                   ((EQUAL X (CAAR L)) (CAR L))
217 - 217.000                   (T (ASSOC X (CDR L))))))
218 - 218.000       (PROP  (LAMBDA (Z Y U)
219 - 219.000        (SET Z (CONS Y (CONS U NIL)))))
220 - 220.000       (GET  (LAMBDA (X Y)
221 - 221.000         (COND ((NULL X) NIL)
222 - 222.000               ((ATOM (SETQ EX (EVAL X))) NIL)
223 - 223.000               ((EQ (CAR EX) Y) (CADR EX))
224 - 224.000               (T (GET (CDDR EX) Y)))))
225 - 225.000       (NCONC  (LAMBDA (X Y)
226 - 226.000          (COND ((NULL X) Y)
227 - 227.000                (T (NCONC2 (RPLACD (NCONC1 X) Y) X)))))
228 - 228.000       (NCONC1  (LAMBDA (X)
229 - 229.000          (COND ((NULL (CDR X)) X)
230 - 230.000                (T (NCONC1 (CDR X))))))
231 - 231.000       (NCONC2 (LAMBDA (X Y) Y))
232 - 232.000       (MAPCAR  (LAMBDA (LST FUN)
233 - 233.000             (COND ((NULL LST)NIL)
234 - 234.000                   (T (CONS (FUN (CAR LST))
235 - 235.000                            (MAPCAR (CDR LST) FUN))))))
236 - 236.000       (AND (LAMBDA L (AND* L)))
237 - 237.000       (AND*  (LAMBDA (L)
238 - 238.000         (COND ((NULL L)T)
239 - 239.000               ((EVAL (CAR L)) (AND* (CDR L)))
240 - 240.000               (T NIL))))
241 - 241.000       (EQUAL (LAMBDA (X Y)
242 - 242.000             (COND ((ATOM X)(EQ X Y))
243 - 243.000                   ((ATOM Y) F)
244 - 244.000                   ((EQUAL (CAR X)(CAR Y))(EQUAL (CDR X)(CDR Y)))
245 - 245.000                   (T F))))
246 - 246.000  ))
```