記号処理 31－11
（1985. 3. 12）

マイクロコンピュータのためのＬｉｓｐシステムの作成

小川 貴英（津田塾大学 数学科）

1. はじめに

tiny LISPは、パーソナルコンピュータを使ってLISPの教育を行う目的で、開発したシステムである。言語の仕様は、ファイルの入出力など一部の例外を除き、Maclispのサブセットである。

津田塾大学では、PC9801F約30台がVAX11/780につながっており、単体のマイコンとして、あるいはVAXの端末として教育に使うことができる。これらを有効に使い、効果的なLispの教育を行うために、Lispシステムの開発を行った。

学生の実習は、VAXにあるLispシステムをTSSで使うことも考えられるが、以下のような問題点があり、入門教育にはマイクロコンピュータのほうが適していると思われる。

(1) 既存のLispシステムは、教育には大き過ぎる。
(2) 入門教育をTSSで行った場合、大勢が一斉に使うと、負荷が大き過ぎて、応答が悪くなる。

現在のマイクロコンピュータは、ハードウェアとしては、Lispの実習をするのに、十分な処理能力があると思われるが、ソフトウェアは、市販のLispシステムには、

(1) 安心して使用できるものが存在しない
(2) 要求する仕様のものが見当たらない
(3) 仕様に変更の必要が生じたときに対応できない

などの点で、問題がある。そこで、使い易さとポータビリティを重視したtiny LISPを開発した。

2. tiny LISPの設計方針

tiny LISP、は大学における教育のために開発したLISPシステムで、用途は、初心者教育に限らず、次のような教育目的で、使用することを考える。

(1) 入門教育
(2) 卒業研究
(3) Lispを利用した応用システムの教育

このため、使用者のレベルは多岐にわたる。入門教育では、初心者に抵抗感を与えないように、初心者に注意しすぎて、逆に、卒業研究のような、経験者が使いにくくならないように注意を払った。応用システムを書けるためには、セルが2万前後とれ、処理速度は、8−queenの92通りの解を表示するのに要する時間が1分前後を目安とした。tiny LISP開発にあたっては、以下の3項目を、列挙した順に重視し設計した。

(1) 使い易さ
(2) ポータビリティ
(3) 処理速度

2.1 言語の仕様

LISPには各種の方言があり、どのLISPを標準にするかが問題になる。tiny LISPは教育を主目的にしており、興味のもてる練習問題が沢山あることが重要である。WinstonのLISPのテキスト [1] には、内容的にも興味ある例題や問題が豊富にある。この本の例題がやり易いように、tiny LISPはMaclispのサブセットとした。したがって、配列を使っていない例題は、変更しなくてもそのままで実行できる。

最低限必要な関数の他には、(1) lispの特徴となる関数。例えば、map関係の関数、(2) 応用プログラムに必要な制御のための関数、*catch、*throw、*errsetを優先的に組み込んだ。

2.2 使い易さ

使い易さは、エラー表示のように、初心者から経験者まで共通に議論できるものもあるが、デバッガーの使いがってのように、利用者の習熟度により違うものもあり、単純に議論できない。さらに、実行の立ち上がりのはやさも重要である。tiny LISPはプログラム作製環境として、構造エディター、pretty printer、デバッガーをサポートすることを、前提に次の方針で設計した。

(1) 実行開始時のパラメータの設定

LISPを起動すれば、直ちに実行を開始し、特にパラメータの設定は、行なわない。アトム領域、セル領域、数値領域などの大きさの設定をする必要はない。

(2) モード

通常、初心者と経験者では都合のよい設定は異なり、標準の状態が固定されていると、毎回、設定の変更が必要になり、不便である。tiny LISPでは、いくつかの状態に対応して、モードをもうけ、各使用者ごとに、実行開始時の標準の状態を自由に設定できるようにした。例えば、入門教育の初心者に対しては、以下の状態が、標準でになる。これは、勿論変更が可能である。

1) trace backの表示

初心者が訳の分からない情報は表示しても無意味である。

2) デバッガーの呼び出し

訳の分からない状態に入ると、不安を招くだけであり、エラーが発生したときに、無条件にデバッガーへ入っても抜け出し方が分からないで、途方にくれるのだけである。デバッガーを、実際に、使いこなすには、かなりの予備知識が必要である。

(3) システムイメージの格納

toolは、一部LISPで記述してあり、毎回、初期設定のときに、プログラムを読み込んでいたのでは、効率が悪い。これを避けるため、メモリーイメージを格納し、開始時に自動的に、システムファイルから読み込むことができる。これにより、立ちあがりを速くすることができ、初期状態を自由に設定できる。

2.3 ポータビリティ

卒業研究に使う場合は、tiny LISPで入門教育を受けており、マイクロコンピュータの処理能力を越えた場合には、VAXなどで同じシステムが使用できると都合がよい。異なる計算機で利用するため、ポータビィリティを重視し、Cで記述した。16ビットと32ビットの間の互換性は、特に注意する必要がある。

tiny LISPの設計では、当然、効率は重視したが、分かり易さ、使い易さを優先し、実行効率は多少犠牲にした。応用プログラムを作って、教育に使う場合は、処理速度が問題になることがある。このような場合にも、Cで記述したシステムが手元にあることで、処理効率をあげるのに個別に対処できる。例えば、portable prologの場合であれば、unifyだけを、subr化すれば、かなり、処理速度を向上できる。

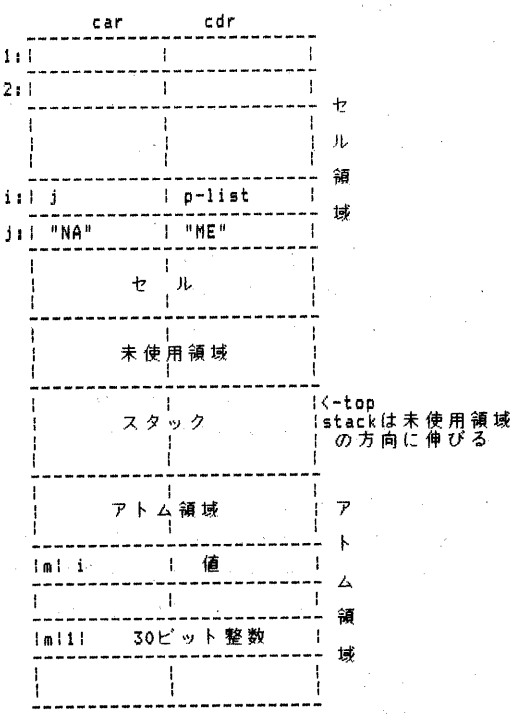
3. 主な特徴とimplementation

tiny LISPは、処理速度を重視し、shallow bindingを採用した。function argumentに対しては、特別な処理はしていない。以下で、主な特徴と、実際のimplementationについて、簡単に述べる。

3.1 メモリー管理

メモリの構造は単純にし、全データをcar部とcdr部の対からなるセルの中に持つ。consセルも、アトムも、実行時のスタックもすべて均一の構造をしたセルで表現する。個々のセルのcar部やcdr部には、

図 1. メモリマップ



mはマークビット

直接データかポインターが入る。直接データは、そこに値が直接はいり、ポインターは他のセルを指す。直接データは整数値か、subrの番号、p—nameの文字列のいずれかである。使用者は、メモリーの割り付けを意識する必要はない。メモリ管理で使用者が決定する、唯一のものは、アトムの数の設定だけである。事前に変更することを忘れても、入力中にアトム領域が不足したというエラーの発生した時点で、変更できる。初期設定を適当に決めておけば、初心者には、直接にはあまり関係ない。

ガーベッジコレクションは、スタックを使う都合や、メモリーイメージの格納のときに、データを詰めるために、リロケーションを行う。

3.2 基本となるデータ型

整数と記号の2種類が基本データとなる。配列は、教育目的には、なくてもかまわないし、メモリー管理を複雑にし、当初の設計方針に反する。文字列は、ファイル名などには必要であるが、tiny LISPでは、アトム名に任意の文字列を認めて、入出力で処理し、独立した型としては存在しない。通常のquoteのほかに、stringquoteを導入し、入出力の際の識別に使っている。関数としての機能は、quoteと全く同じでものである。

各記号は、p—name、property listと値を持つ。記号の値は、関数と変数を区別しない。したがって、関数名と変数名は同じ名前を使えないが、それほど支障はないと思われる。教育的には、同じ名前を使わない方が望ましい。さらに、記号用のセル領域の節約にもなる。

記号名の構文規則は単純化し、2種の括弧（parenthesesとbracket）と、ピリオド、'"' 以外の文字は、英字と同じレベルで記号の名前に使用できる。'"' でくくれば、括弧とピリオドも記号名に使ってかまわない。エスケープによる表記は見にくいのでとらない。'"' にいれたときは、大文字、小文字を識別する。

数値は30ビット整数まで扱うことができる。tiny LISPの主たる目的は、記号処理の教育あり、計算は目的ではない。したがって、実数は扱かわない。整数は、tiny LISPでの数値表現としては、30ビットあれば十分である。16ビットの機械で30ビットの整数を効率よく処理できないので、14ビット整数を導入し、処理速度の向上を計った。14ビットの整数でも、繰り返しの計数には十分である。整数の内部表現について、利用者は意識する必要はない。

3.3 メモリーイメージの格納

メモリーイメージの格納は、単にイメージだけでなく、各種のフラグも同時に退避する。したがって、使用者ごとの設定は、メモリーイメージを一度つくれば、各自が都合のよい状態から、直ちに開始できる。格納した、イメージには、自動開始関数を指定できる。例えば、図2に示すように、(prolog)を、自動開始関数として格納すれば、直ちにprologのプログラムを開始することができるようになっている。起動のとき、パラメータを指定しなければ、LISP.SYSからデータを読み込む。

図 2. メモリーイメージの格納と自動開始の例

```
A>lisp

  MICRO LISP V1.6          (c) T.Ogawa 12/01/84


Library V1.7 (c) T.Ogawa                    #1222
>
>(comment "ここでprologのプログラムを作る")
>
>(store "PROLOG" '(prolog))
==(PROLOG)
>(exit)

A>lisp prolog

  MICRO LISP V1.6          (c) T.Ogawa 12/01/84

(PORTABLE PROLOG ON LISP)
PRL>end
>(gc t)
total cell size=10000
( used=3384 free=6316 stack=0 atom=287/300 )
==1
>
```

3.4 エラーメッセージ

エラー表示は、分かり易いものにする。LISPでは、例えば、(plus 'x 1)を実行したときに、plusの引数が数値でないというメッセージだけでは、そのエラーが一体何処で起きたのか分からない。どの関数の、どの式でplusを実行したかも分かるようにする。

図 3. エラーメッセジの例

```
>(de test (x) (plus x x))
==TEST
>(test 'a)

*** Error: non-numeric argument. ***  PLUS
expression: (PLUS X X)
function: TEST
```

3.5 入出力

入出力は、単純化して入出力用のstreamの切り換えで行う。streamは入力用、出力用に各1本あり、番号で指定する。これにより、任意のファイルのアクセスができる。出力streamを、プリンターに切り換えるだけで、簡単に、実行経過のハードコピーがとれるようになる。streamの切り換えを使い、LISPの関数、データを格納したり、読み込ムプログラムは、LISPで簡単に記述できる。LISPで記述した標準関数として、save、loadが準備されている。

入力を簡単にするために、超括弧が使用できる。また入力中の、余計な閉じ括弧は、無視し、英字は大文字に変換する。quote、string quoteの処理も入力で行い、"string."のように、'"'でくくることで、アトム名の中に、任意の特殊文字を入れることができる。

図 4. loadの定義と使い方

```
(DE LOAD (F) (OPEN F 0) (SELECTINPUT 0) F)

(save "ファイル名" 関数、変数のリスト "注釈")
(load "ファイル名")
```

3.6 構造エディター

LISPのプログラムを容易にするために、構造エディターをサポートする。エディタは、LISPで記述し、簡単に変更することができる。メモリーの効率はよくないが、初心者が安心して使用できるように、copy方式をとる。したがって、放棄すれば元に戻る。

編集機能としては、括弧の挿入、削除、S一式の挿入と削除、削除したデータを任意の場所に戻す機能等がある。プログラムを小さくするために、命令数は、必要最小限にとどめるてある。pretty printerの機能もエディタが持っている。

3.7 デバッガー

tiny LISPではsubrも含めTRACEができるが、TRACEは、デバッグにはあまり役立たない。より効果的な、デバッグをするために、デバッガーをサポートしている。デバッガーが呼び出されるのは、次のいずれかの場合である。

(1) エラーがあった
(2) ESC鍵で割り込んだ
(3) breakを呼び出した

図 5. デバッガーの使用例

```
>(de test (x) (+ x y))
==TEST
>(pp test)
(LAMBDA (X) (+ X Y))
==NIL
>(test)

*** Error: number of argument. ***  TEST
>(test 10)

*** Error: unbound variable. ***  Y
expression: (+ X Y)
function: TEST
Debugger V1.0
?0
Frame position: 0
lambda frame: TEST
X: 10
?(resultis 100)
==110
>
```

デバッガーがアクティブでなければ、実行を中断し、トップレベルに戻る。デバッガーに入ると、実行時スタックの中を調べ局所変数の値を見る、値を変更する、関数を編集し、再実行する、trace backすることなどができる。breakしたとき、あるいは、未定義関数や変数を評価したときなどは、値を返して、継続することも可能である。

4. システムの大きさと処理能力

tiny LISPの大きさは、Cのソースコードで約1500行ある。CP/M-86用でobjectコードが約32KBあり、1万セルのtiny LISPが、メモリー128KBのマイクロコンピュータで動いている。

関数は、subr, fsubrが137個、editorなどLISPで書いたものが、25個ある。

表1は、LISPコンテストの問題[4]の実行結果である。入門教育には、処理能力はこれで十分である。卒業研究でも、簡単な実験程度なら十分処理できることを示している。あえて言えば1万セルでは、少し不足気味である。メモリーがもう少し大きくて、2万セルもあれば、卒業研究も含めて、かなりの処理がマイクロコンピュータで、できると思われる。

表2は、8-queenプログラム（図6）を異なる機種で実行した結果である。VAXでは10万セルとってある。実行はすべて、経過時間で、ストップウォッチで計った。これをみると、VAXで実行しても、実時間はたいして改善されないことが分かる。教育には、マイクロコンピュータが沢山あった方が、効率的であることを示している。

表3は、8-queenのプログラムの実行時間を、異なるバージョンで計ったもので、ハンドコンパイルの効果を示す。時間は、8088（8MHZ）で計測した。V1.1には、属性リスト、デバッガーがない。さらに、V1.1の整数は14ビットのみである。V1.7は、30ビットの整数を扱い、実行時にはデバッガーのための情報をスタックにいれている。V1.7Aは、効率を上げるため、一部（ソースで約100行）を手続単位でハンドコンパイルしたものである。これを見ると、V1.7になってデバッガー等のオーバーヘッドで、約15%遅くなったが、ハンドコンパイルにより20%以上処理速度があがり、V1.7AはV1.1よりもさらに速くなっていることが分かる。オブジェクトコードの大きさも、1KB以上改善された。

表1. LISPコンテストの問題の実行結果
括弧内はガーベッジコレクションの回数

	時間(ms)
[1-1:] Tarai-4	44140
[1-2:] Tarai-5	1201120
[1-4:] Tak-18-12-6	246720
[2-1:] List-Tarai-4	86600(6)
[2-2:] Srev-5	2416
[2-3:] Srev-6	9660
[2-4:] Qsort-50	2608
[2-5:] Nrev-30	1946
[2-6:] Rev-30	2.6
[2-7:] Drev-30	2.2
[2-8:] Rev-10	1.94(1)
[2-9:] Rev-100	15.8
[2-10:] Rev-1000	202.
[2-12:] Drev-10	1.02
[2-13:] Drev-100	6.0
[2-14:] Drev-1000	55.2
[6-1:] Seq-100	1880
[7-1:] BITA-5	542
[7-2:] BITA-6	2006
[7-3:] BITB-5	196
[7-4:] BITB-6	542
[9-1:] TPU-1	28940(4)
[9-2:] TPU-2	127580(16)
[9-3:] TPU-3	47100(5)
[9-4:] TPU-4	63340(8)
[9-5:] TPU-5	7440(0)
[9-6:] TPU-6	262640(32)
[9-7:] TPU-7	47920(5)
[9-8:] TPU-8	43360(3)
[9-9:] TPU-9	29350(2)
[10-1:] Prl-Rev-15	29800(1)
[10-2:] Prl-sort-20	45540(2)
[11-1:] Diff-3	506
[11-2:] Diff-5	45520(1)

表 2. 8-queenのすべての解を求めるのに要した時間
tiny LISP Version 1.1を使用し、時間は、ストップウォッチで計測した。プログラムは図6に示す。

OS	CPU	使用言語	時間(秒)	備考
CP/M-80	Z80(4MHZ)	BDS C	200.5	
CP/M-86	I8086(8MHZ)	DeSmet C	59.8	
CP/M-86	I8088(8MHZ)	DeSmet C	84.6	
VMS	VAX-11/780	C/VMS	45.5	32ビット

表 3. ハンドコンパイルの効果

Version	時間(秒)	
V1.1	84.6	14ビット整数のみ 属性リストなし
V1.7	97.98	30ビット整数、属性リスト
V1.7A	72.94	一部、ハンドコンパイル

図 6. 8-queenのプログラム

```
(DEFUN QUEEN (SIZE)
  (PROG (N)
    (SETQ N 0)
    (CLOCK)
    (COL 1 (MAKELIST SIZE) NIL NIL NIL)
    (CLOCK)))

(DEFUN CHECK NIL
  (PRIN1 (SETQ N (ADD1 N))) (PRINT STATE))

(DEFUN COL (X C X+Y X-Y STATE)
  (COND ((NULL C) (CHECK)) (T (MAP 'PLACE C))))

(DEFUN PLACE (Y)
  (SETQ Y (CAR Y))
  (COND
    ((MEMBER (+ X Y) X+Y) NIL)
    ((MEMBER (- X Y) X-Y) NIL)
    (T
      (COL
        (ADD1 X)
        (CDR C)
        (CONS (+ X Y) X+Y)
        (CONS (- X Y) X-Y)
        (CONS Y STATE))))
  (SETQ C (APPEND (CDR C) (LIST (CAR C)))))

(DEFUN MAKELIST (N)
  (COND ((< N 1) NIL)
        (T (CONS N (MAKELIST (SUB1 N))))))
```

5. 教育効果

主に入門教育にtiny LISPを使用した結果、tiny LISPは、教育目的に十分な処理能力を備えていることがわかった。

特に目立った初心者の誤りとしては、括弧の閉じ忘れ、quoteとstringquoteの混乱、再帰呼び出しの暴走などがある。quoteとstringquoteは一方は閉じる必要がなく、他方は閉じる必要があるための混乱のようである。括弧が閉じていないのに閉じた、あるいはquoteを閉じるつもりで使う結果、入力待ちなのに実行していると感違いし、いつまでも待っている例が散見された。これらの誤りに対しては、入力待ちのときには、促進記号を表示することで解決した。初心者には、促進記号は不可欠である。

再帰の終了条件を忘れる、間違えることは、頻繁に起こるので、実行時のスタックのあふれのチェックも絶対に入れる必要がある。

提供した道具もよく使いこなしていたようである。エディターは他の言語の場合よりは、うまく使っていたようである。LISPの場合は、これなしではどうにもならないといった面もある。エディタの機能も、入門教育の小さいプログラムなら十分であったが、もう少し大きくなると、機能の追加、あるいは他のテキストエディター[7]を使用することが必要になる。

pretty printerは、よく使われた道具である。教育の面から見ると、これは単にきれいに表示する目的以上に、使用者に大きな影響を与えたように思われる。いつも、これを使い表示しているうちに、段付の重要性を認識し、入力の際には、似たような段付をして、プログラムを入れるようになる。超括弧を使うよりは、段付で括弧の対応を確認している場合の方が多かったようである。教育においては、標準的なよい道具を与えて使わせることが、いかに重要かを示すよい例である。

デバッガーは、教育開始時に間に合わなかったこともあり、十分に検討できなかったが、余り多くの機能を入れるよりは、エラーメッセージを丁寧に出す方が、はるかに重要であると思われる。デバッグに実際に使うよりは、実行時のスタックの状況などを調べることにより、再帰呼び出しの実行の仕方などを理解させる教育目的に、役立つように思われる。

表 4. tiny LISPの関数

&P
*
*CAR
*CATCH
*CDR
*ERRSET
FEXPR
*THROW
*UNPACKN
+
-
/
<
>
A<
A>
ABASE
ABORT
ABS
ADD1
AND
APPEND
APPLY
ASSOC
ATOM
BOUNDP
BREAK
CAAAR
CAADR
CAAR
CADAR
CADDR
CADR
CAR
CDAAR
CDADR
CDAR
CDDAR
CDDDR
CDDR
CDR
CHARP
CHR
CLOCK
CLOSE
COMMENT
COND
CONS
CUTNTH
DE
DEBUG
DEFINE
DEFUN
DELETE
DELETENTH
DF
DIFFERENCE
DIV
ED
EQ
EQUAL
ERROR
ERROR-NUMBER
EVAL
EXIT
EXPLODEN
FEXPRP
FIRSTN
GC
GENSYM
GET
GETCHARN
GO
GREATERP
HARDCOPY
LAST
LENGTH
LESSP
LINEFEED
LIST
LISTP
LOAD
LOOP
LPRINC
LPRINT
MAKUNBOUND
MAP
MAPC
MAPCAN
MAPCAR
MAPCON
MAPLIST
MAX
MEMBER
MEMQ
MIN
MINUS
MINUSP
NCONC
NCONS
NEQ
NOT
NREVERSE
NTHCDR
NULL
NUMBERP
OBLIST
ODDP
OPEN
OR
ORD
PACK
PAGE
PAIRP
PLENGTH
PLIST
PLUS
PLUSP
POP
PP
PRIN1
PRINC
PRINT
PROG
PROG1
PROG2
PROGN
PUSH
PUTPROP
QUOTE
QUOTIENT
RAND
READ
READCH
REMAINDER
REMOB
REMPROP
REPLACENTH
RESULTIS
RETURN
REVERSE
RPLACA
RPLACD
SAVE
SELECTINPUT
SELECTOUTPUT
SET
SETPLIST
SETQ
STORE
SUB1
SUBRP
SUBST
SYMBOLP
TERPRI
TIMES
TRACE
TYPE
UNPACK
UNTRACE
VERBOSE
ZEROP

教育にマイクロコンピュータを使ったことは、非常に有効であったと考える。マイクロコンピュータが都合がよいのは、計算機の知識がほとんどなくても、気楽に使える点にある。LISPの場合であれば、電源の入れ方さえ分かれば、後はLISPのことだけ知っていればよい。また仮に、途中でトラブルが発生したとしても、最後は、フロッピーを抜き取り、電源をきればよい。こんなことも、TSSに比べ気楽にさせるようである。また、今回はCP/M-86を使用したが、tiny LISPで作業をする限りでは、ファイル名にほとんど制限がなく、ファイル名の構文等、余計なことを気にする必要がない。こういった気楽さが、入門教育には不可欠である。

6 おわりに

tiny LISPを使い、TQAS[9]の拡張、portable prologをのせる、mooのプログラムを書いてゲームをやってみる等、いくつかのプログラムをはしらせてみたが、ほぼ当初の目的を達成している。

各種のリソースを有効に使うためには、

(1) グラフィック機能
(2) 端末としての機能

等の機能追加について、今後、検討する必要がある。

参考文献

[1] P. H. Winston, B. K. P. Horn: LISP、Addiso-Wesley、1981、
白井、安部訳：LISP、培風館.

[2] 後藤、戸島、石畑：記号処理の基礎と応用、情報処理学会、1981.

[3] K. M. Pitman: The Revised Maclisp Manual、MIT/LCS/TR-295、1983.

[4] 奥乃：第三回LISPコンテストおよび第一回PROLOGコンテストの課題案、記号処理28-4、情報処理学会、1984.

[5] 松永：マイクロコンピュータ用LISPを比較する、情報処理学会第26回全国大会4D-2、1983.

[6] 小川：Note on tiny LISP、津田塾大学数学科、1984.

[7] 小川：SSEユーザーマニュアル、津田塾大学数学科、1984

[8] 中島：Prolog、産業図書、1983.

[9] 安西、佐伯、難波：LISPで学ぶ認知心理学 1、2、3、東京大学出版会.