島根大学教育学部紀要（自然科学）第10巻　9～13頁　昭和51年12月

FACOM 270-20 上への LISP の実現

林　　隆　一*

Ryuichi HAYASHI

Implementation of LISP for FACOM 270-20

Abstract: The implementation of LISP for FACOM 270-20 (main memory 16 k words, drum area 131 k words) is described.

LISP language is unfit for the computer with a small main memory because of requiring a large free storage space. In the present system, the specifications of LISP 1.5 were made alterations in order to save the consumption of the main memory and improve an execution time as follows:

(1) A push down stack instead of a-list is used for the binding mechanism of variables, and the virtualization of its stack is performed.

(2) The atom-headers are placed on the array which is token jointly with a hash table.

(3) A interger is represented by making use of the addresses other than the main memory.

1. はじめに

LISP は人工知能の研究に広く用いられている言語であり，関数言語，記号言語，リスト処理言語，再帰的言語および論理言語として特徴づけられる。LISP の基本的な考え方は McCarthy により展開されたもので，1960年に発表された論文「記号式の再帰的関数と計算機による処理」[1]に述べられている。続いて彼とその協力者達によって LISP 言語の文法が確立され，IBM 7090 に対し実現され利用できるようになった。この言語は LISP 1.5[2] と称され，理論的な言語として高く評価されている。その後，処理速度，記憶容量を始めとする実用的な要求を重視する方向から，LISP 1.6[3]，INTERLISP[4] 等が出現してきた。

一方，我が国でも，最近までほとんど見向きもされなかった LISP の処理システムがかなり報告[5]されるようになった。この原因の1つは，この言語の性格上ばく大な記憶容量を必要とするので実用としては不向きであると思われていたものが，大型計算機の出現にともない見直され始めたためであろう。

筆者は以前より自然言語処理に関心をもっている。自然言語処理に関しては，すでに多くのシステムが作成されている。それらのシステムの多くは LISP を host language として作られているので，情報の交換には LISP システムを作成することが必要である。また，最近のミニコンの普及にともなってミニコン用 LISP も作成され，使い方によれば十分利用価値があるという報告[6]もあり，小型の計算機上に LISP を実現することが何の意味もないことであるとは思われない。この様な動機のもとに，FACOM 270-20（主記憶容量 16K語，内臓ドラム容量 131K語）上にLISP を実現したので，その設計方針，システム等について述べる（以後 この LISP を LISP 270-20と呼ぶ）。

2. 設計方針

LISP には実用上の要求と計算機の制約から種々の方言があるが，LISP 1.5 は次の様な特徴をもっている。

データ構造は2進木を基本とする最も簡単な形であり，処理アルゴリズムは LISP 1.5 自身で書かれた簡潔な記述により明確に定義されている。このために，LISP 1.5 は類似の処理システムの研究の基礎として当分の間消滅することはないであろうし，現在でも LISP 1.5 で書かれたプログラムがかなり多く存在している。しかし，LISP 1.5 のメモリの消費は無視できるようなものではない。p-list は大きく，内部だけで利用することの多い a-list なども2進木のデータ構造をもつ。さらに悪いことには，データはすべてフリーリスト領域内のポインタで表わされるために仮想化が困難である。このままのデータ構造ではどのようなページングアルゴ

* 島根大学教育学部技術研究室

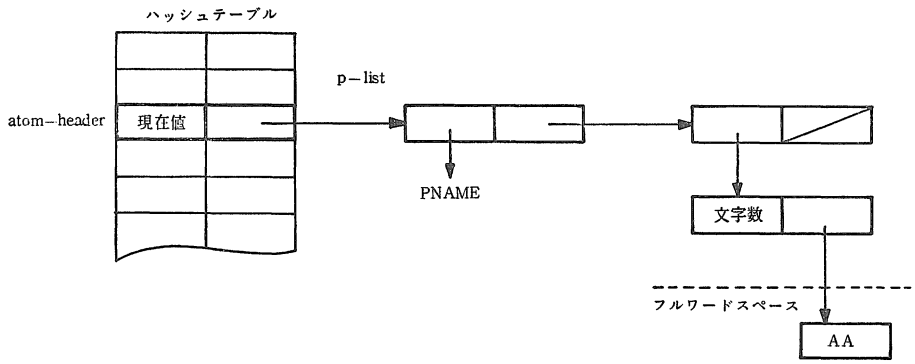

Fig. 1 Atom-header representation in LISP 270-20

リズムを採用しても効率のよいシステムを作ることは不可能であろう。

したがって，FACOM 270-20 のような小型の計算機上に LISP 1.5 をそのまま実現しても実用にはならないであろうと思われる。これを解決するには，p-list の形態をかえて小さくする，あるいは p-list をなくしてしまう等によってデータ構造そのものを変更することも考えられるが，LISP 1.5 の利点は失われることになる。そこで，本システムでは，できる限り LISP 1.5 の純粋性を保ちながら小型の計算機上に LISP を実現するという基本方針のもとで，次の様な点を変更している。

(1) 変数の束縛機構には a-list のかわりにプッシュダウンスタックを利用し，これを仮想化する。

LISP 1.5 では，変数束縛に変数—変数値のポインタ対をリストにした a-list を用いるために多くのフリーリスト領域を消費する。実際にフリーリスト領域が不足する大部分の原因は a-list の増加である。a-list のようにデータをポインタを基礎にして表現するものは仮想化が困難であるので，フリーリスト領域が不足すればガーベッジコレクションを行わねばならないが，これは非常に時間を消費する。また，a-list では変数値を取り出す時に，a-list を左端から順番に探索するために時間がかかる。

a-list にはこの様な欠点があるが，プッシュダウンスタックを利用すれば次の様に改良できる。

(i) プッシュダウンスタックの “last in first out” の性質を利用すれば比較的効率よく仮想化が行なえる。

(ii) 変数値の取り出しには変数の現在値をもってくるだけなので，処理速度が速くなる。

(iii) 主にフリーリスト領域を消費する a-list がないので，ガーベッジコレクションの回数が大幅に減少する。

したがって，メモリ節減と処理速度向上という両方の点から，変数束縛機構には a-list のかわりにプッシュダウンスタックを採用する。

(2) atom-header を配列構造上にとり，この配列をハッシュテーブルと併用する。

LISP 1.5 ではアトムの型表示にフリーリスト領域内のポインタ対語の car 部に−1 を入れた atom-header を用いているが，LISP 270-20 では atom-header を固定した配列構造上にとり，アドレス空間によってアトムの型を表示する。Fig. 1 に示される様に，この配列はハッシュテーブルと併用され，アトム（変数）の現在値と p-list へのポインタが格納される。したがって，oblist は作らない。

この様な構造にするのは，変数の現在値の取り出しを速くするためとメモリ節減のためである。

(3) 整数表現に不要アドレスを利用する。

FACOM 270-20 の 1 語は16ビットであるが，メモリは16K語しか実装していないためにアドレッシングに不要部が生じる。この不要アドレスを利用して整数値データを表現し，メモリの節減と整数演算の高速化を実現する。取り扱える整数の範囲は−8192～8192とし，これらに24575のゲタをはかせたものをその内部表現とする。

3. LISP 270-20 システム

LISP 270-20 システムでは，記憶容量，記述表現の関係からコンパイラや binary program space は始めから考えに入れてなく，すべてインタープリタ形式をとっている。メインメモリは Fig. 2 の様に割り付けられている。

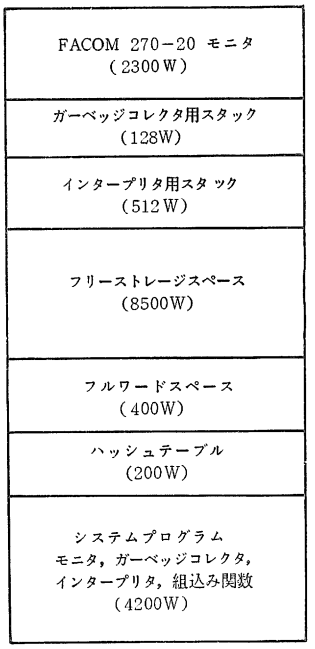

Fig. 2 Main Memory allocation in LISP 270-20 system

3.1 インタープリタ

LISP 270-20 インタープリタは，変数束縛に a-list のかわりにプッシュダウンスタックを採用しているという以外，LISP 1.5 インタープリタ[2]に準拠している。しかしながら，関数引数と FEXPR の定義式に対してプッシュダウンスタックをそのまま使ったのでは，a-list を使う LISP 1.5 と計算結果が異なる場合があるという問題がある。

LISP 270-20 では，HLISP が採用している方法[7),8)]でこの問題を解決している。FEXPR に関しては，通常の λ 変数名と FEXPR の定義の先頭の λ 変数名を同一変数名とした時に相違が生じるので，FEXPR の λ 変数名は特殊な変数名にして通常の λ 変数名と混乱しないようにする。関数引数では，関数実引数に自由変数が含まれ，かつその自由変数名が他の λ 表示中で λ 変数として使用されている時に相違が生じるので，関数実引数の評価時の値に束縛することを指定したい変数名を function の第2引数以下に指定する（個数は任意）。何も指定しなければ function を quote で代用する。この束縛指定を含む function の定義を示す。

```
function〔x ; y1 ; y2 ; ····· ; yn〕
  =〔null〔cdr〔x*〕〕⟶car〔x*〕 ;
    T⟶list〔FUNARG ; car〔x*〕 ; cdr〔x*〕 ;
          evlis〔cdr〔x*〕〕〕〕
```

ただし，x* は引数のS式である。

これらの解決法はいずれも利用者に責任をもたせることになるが，LISP 1.6[3)] の BCP (Binding Context Pointer) を用いる方法よりはインタープリタが単純で，かつ処理速度が速いという利点がある。

3.2 プッシュダウンスタックの仮想化

スタックの底部はすぐに実行されることはないので，スタックを上部と底部に2分割し，オーバ・プッシュダウンの時はスタックの底部を2次記憶装置であるドラムに掃き出し，かつ上部を底部に平行移動する。オーバ・ポップアップの時はスタックの底部にドラムに記憶されているスタックを読み込む。

スタックを上部と底部に分割したのは，プッシュダウンとポップアップはランダムにおこるので，ドラムへの read/write 回数をなるべく少なくするということを考慮したためである。

次に，スタックの深さはどの程度が最適かという問題がある。浅くすればメモリの節約になるが，ドラムへの read/write の回数が増え速度に影響することになる。現在のところメモリ節減という方向から，スタックの深さを512語（1回の操作で2語使用）にとり256語づつに分割している。

3.3 ガーベッジコレクタ

LISP プログラムを実行させると，結果が得られるまでにガーベッジコレクタが数回呼ばれるのが通例である。特に，小型計算機ではガーベッジコレクションの回数はかなり大きくなると考えられるので，ガーベッジコレクタの速度はシステム全体の速度に多大の影響を及ぼすことになる。

ガーベッジコレクタの目印付けアルゴリズムとしては，プッシュダウンスタックを利用する方法とメモリを消費しない逆転ポインタ法があるが，LISP 270-20 では速度の点から専用のスタック（深さ128語）を設けて Knuth のアルゴリズムB[9)] を採用した。仮想化するプッシュダウンスタックを共用しないでガーベッジコレクタ専用のスタックを設けたのは，リストの深さが128を越えることはほとんど考えられないので，この程度のメモリ節減よりは速度の向上を重視したためである。

3.4 組込み関数

組込み関数はメモリ節約のため最小限にし，現在のところ Table 1 に示される34種類である。これらの組込み関数は LISP 1.5 の関数とほぼ同じ機能をもち，相違は表中に明記してある（もちろん，LISP 270-20 は a-list を用いないのでインタープリタ関数に a-list を指定する必要はない）。また，処理速度の関係からすべて SUBR 型あるいは FSUBR 型で作られている。

Table 1 Functions in LISP 270-20 system

(1) 基本関数
- car〔x〕 : SUBR
- cdr〔x〕 : SUBR
- cons〔x ; y〕 : SUBR
- atom〔x〕 : SUBR
  数はアトムとみなさない。
- eq〔x ; y〕 : SUBR
  数の内部表現は一意である。

(2) 準基本関数
- null〔x〕 : SUBR
- caar〔x〕 : SUBR
- cadr〔x〕 : SUBR
- cdar〔x〕 : SUBR
- cddr〔x〕 : SUBR

(3) インタープリタ関数
- evalquote〔fn ; args〕 : SUBR
- apply〔fn ; args〕 : SUBR
- eval〔form〕 : SUBR
- evlis〔x〕 : SUBR
- cond〔x_1 ; x_2 ; … ; x_n〕 : FSUBR
- quote〔x〕 : FSUBR
- function〔x; y_1; … ;y_n〕: FSUBR
  y_i は自由変数で，指定しなければ quote で代用する。

(4) 関数定義および p-list の操作に有効な関数
- define〔x〕 : SUBR
- deflist〔x ; ind〕 : SUBR
- get〔x ; y〕 : SUBR

(5) 入出力関数
- read〔 〕 : SUBR
- print〔y〕 : SUBR

(6) デバッグ用関数
- trace〔x〕 : SUBR
- untrace〔x〕 : SUBR

(7) プログラム形式
- prog〔x ; y_1 ; … ; y_n〕 : FSUBR
- set〔x ; y〕 : SUBR
- setq〔x ; y〕 : FSUBR
- go〔x〕 : FSUBR
- return〔x〕 : SUBR

(8) 算術関数
- numberp〔x〕 : SUBR
- minus〔x〕 : SUBR
- minusp〔x〕 : SUBR
- plus〔x_1 ; x_2 ; … ; x_n〕 : FSUBR
- times〔x_1 ; x_2; … ; x_n〕: FSUBR
  取り扱える数値は整数だけで，その範囲は −8192〜8192 である。

なお，筆者の LISP 作成の目的の1つは自然言語処理にあるので，数値処理能力は低く，小数点数及び浮動小数点数は扱えず，算術関数もほとんど組み込まれていない。

4. おわりに

本報では，小型計算機上に LISP を実現する際のメモリ節減と処理速度向上を考慮した1つの構成を示した。

本システムの完全な評価は未だ終ってなく，どの程度の仕事が可能かは今後の課題に残されているが，現在までのところ次の様な所見をもっている。

(1) LISP が小型計算機に不向きであることは確かであるが，LISP 270-20 システムは LISP の教育，普及，簡単なゲームのプログラム，プログラムのデバッグ程度なら十分役立つ。

(2) LISP 270-20 は LISP 1.5 のデータ構造の統一性という特徴を重視した構成としている。しかし，LISP 1.5 からのはずれ方が中途半端であることは否定できない。したがって，LISP 1.5 の完全なサブセットにすることが困難であるならば，小型計算機での LISP の第一の要求はなるべく大きなフリーリスト領域をとり有効に利用したいことであるので，データ構造の統一性は失われても実用的データ構造にする方がいいのではないか。

(3) その他検討，改良を必要とする主な問題としては，インタープリタにおけるフリーリスト領域の消費の抑制，メインメモリでの最適なスペースの割り付け，デバッグ用関数の開発等がある。

なお，LISP 270-20 システムによるプログラムの実行例を付録にあげておく。

参考文献

1) J. McCarthy : Recursive Functions of Symbolic Expressions and Their Computation by Machine, Comm. of ACM, Vol. 3, pp. 184〜195 (1960).
2) J. McCarthy et al. : LISP 1.5 Programmer's Manual, MIT Press (1962).
3) L. H. Quam, W. Diffie : Stanford LISP 1.6 Manual, Stanford Artificial Intelligence Laboratory Operating Note, No. 28.4 (1970).
4) W. Teitelman : INTERLISP Reference Manual, Xerox (1974).
5) 記号処理シンポジウム報告集，情報処理学会プログラミング・シンポジウム委員会 (Jul. 1974).
6) 渡辺 勝，鈴木則久：ミニコンにおける LISP，生産研究，Vol. 26, No. 1，東京大学生産技術研究所，pp. 21〜24 (1974)。
7) 後藤英一：LISP 入門，bit, Vol. 6, No. 7，共立出版，pp. 11〜12 (1974)。
8) 後藤英一：LISP 入門，bit, Vol. 6, No. 8，共立出版，pp. 6〜10 (1974)。
9) D. E. Knuth : The Art of Computer Programming Vol. 1, Addison-Wesley Publishing Company, pp. 414〜415 (1968).

付録　LISP 270-20 シス テムによるプログラムの実行例

```
C   THE DEFINITION OF DELETE AND ITS TEST
  (DEFINE ((DELETE(LAMBDA (X Y) (COND
          ((NULL Y)  NIL)
          ((EQ X (CAR Y)) (DELETE X (CDR Y)))
          (T (CONS (CAR Y) (DELETE X (CDR Y))) ) )))))
  (TRACE (EQ))
  (DELETE E (D E L E T E))
  STOP))
FUNCTION EVAL@UOTE HAS BEEN EXTEND,ARGUMENTS
DEFINE
(((DELETE (LAMBDA (X Y) (COND ((NULL Y) NIL) ((EQ X (CAR Y)) (DELETE X
 (CDR Y))) (T (CONS (CAR Y) (DELETE X (CDR Y)))))))))
END OF EVAL@UOTE, VALUE IS
(DELETE)
FUNCTION EVAL@UOTE HAS BEEN EXTEND,ARGUMENTS
TRACE
((EQ))
END OF EVAL@UOTE, VALUE IS
NIL
FUNCTION EVAL@UOTE HAS BEEN EXTEND,ARGUMENTS
DELETE
(E (D E L E T E))
 ARGUMENTS OF EQ
(E D)
 VALUE OF EQ
NIL
 ARGUMENTS OF EQ
(E E)
 VALUE OF EQ
T
 ARGUMENTS OF EQ
(E L)
 VALUE OF EQ
NIL
 ARGUMENTS OF EQ
(E E)
 VALUE OF EQ
T
 ARGUMENTS OF EQ
(E T)
 VALUE OF EQ
NIL
 ARGUMENTS OF EQ
(E E)
 VALUE OF EQ
T
END OF EVAL@UOTE, VALUE IS
(D L T)
```