論　文

LISP のデータ表現*

——TOSBAC-5600 LISP を中心にして——

黒　川　利　明**

Abstract

LISP has several types of data, and that each LISP system has its own data representations. In the first part of this paper, those data representation of LISP 1.5, Stanford LISP 1.6, MACLISP and INTERLISP are surveyed. Next, how the data are represented in the late TOSBAC-5600 LISP system is described. Some remaining problems are also presented. At last, the author presents some hints to represent the data of the future LISP system.

1. まえがき

LISP は本来リストという型式のデータを扱うプログラム言語であるが，そのデータ型には，リスト以外の文字アトム (literal-atom)，数値アトム，アレイなどがある．各データ型に対する表現方法は，LISP の各方言で，それぞれの特徴をもっている．

この相違は，無論応用上の要求と計算機の制約とに由来するものではあるが，LISP 1.5，LISP 1.6，MACLISP，INTERLISP を展望すると，使用するメモリー量の減少と処理速度の向上が結果として追求されていることがわかるであろう．

そこで，TOSBAC-5600 LISP システムで，上の二目標をどのようにして達成しようとしたか，その結果はどうであったか，またその後の問題点はなにかについて述べる．

最後に，これから作成すべき LISP システムでのデータ表現について筆者の見通しを述べるとともに，LISP マシンの試みについて紹介する．

2. これまでの LISP でのデータ表現

——LISP 1.5, Stanford LISP 1.6, MACLISP, INTERLISP (BBN-LISP)——

Table 1 Data representation of LISP 1.5

データ型	ワード数	表　現
fixed-number	2	1 — fix-tag　value — full word space
floating-number	2	1 — floating-tag　value — full word space
literal atom	4~9(*)	PNAME　AAAA full word space
array	不定	pointer
list element	1	car　cdr

(*) これは literal-atom の PNAME が1ワードに納まる場合で，n ワード必要とする場合には，上の数字に $(n-1)$ を加えねばならない．9ワードというのは value (APVAL) と関数定義を Fig. 2 のようにもった場合．

2.1 LISP 1.5[6] (Table 1 参照)

LISP 1.5 のデータ表現の特徴は単純さにある．すなわち，ユーザが表面的に扱いうるデータはすべて free-list-area 内のポインタとして表わされている．したがって LISP 1.5 のインタプリンタの LISP による記述は単純かつ明快なものとなっている．この代償は，メモリー量についてはアトムの型表示や後述の a-list でリストを消費すること (**Fig. 1**(次頁参照))，処理速度については変数値のとりだしに a-list という変数-変数値の対を node とする tree (**Fig. 2**(次頁参照))を探索しなければならず遅くなるということである．

2.2 Stanford LISP 1.6 と MACLISP (Table 2

* Data Representation of LISP with an experience on TOSBAC-5600 LISP by Toshiaki KUROKAWA (Information Systems Laboratory, TOSHIBA Research and Development Center).

** 東京芝浦電気(株)総合研究所情報システム研究所

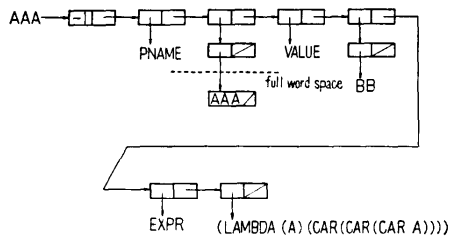

Fig. 1 Representing literal-atom in LISP 1.5

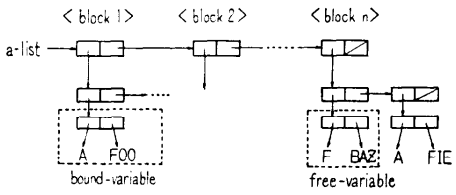

Fig. 2 Variable-binding by a-list

Table 2 Data representation of Stanford LISP 1.6

データ型	ワード数	表現
small-number	0	$\|n\| < 2^{15}$ (65536)
fixed-number	3	value / FIXNUM / full word space
floating-number	3	value / FLONUM / full word space
big-number	6～	POSNUM (NEGNUM) / value 0 / value 1 / full word space
string	4～	PNAME / full / "STRI / NG"\Λ\Λ word
literal atom	4～9	PNAME / VALUE / ATOM
array	不定	fixed-bits-integer, floating-number, pointer
list element	1	car | cdr

参照)

この二つは LISP 1.5 の単純さを部分的に失ったが速度と記憶容量に改良が施された例である.

2.2.1 Stanford LISP 1.6[12]

新しいデータ型として small-number と string が追加され，アレイの要素としてポインタだけではなく数値そのものが加えられた．small-number は free-list-area 外のポインタでありながら，LISP のデータとして扱われ，これによって数値アトムに対する必要メモリー量は飛躍的に減少した．

変数値は shallow-binding (付録(132 ページ)参照)

A Pointer object (list, atomic symbol, string, etc.)

segment number	ring numb	type bits	043 tag
word number	fields hardware requires to be zero		

A number (fixnum or flonum)

not used	not used	type bits	047 tag
machine number			

Fig. 3 MACLISP Data Representation on Multio (according to D. A. Moon[9])

Table 3 Data representation of INTERLISP

データ型	ワード数	表現
small-number	0	$-1536 \leqq n \leqq 1536$
fixed-number	1	
floating-number	1	
string	1	文字数 | 文字並 | full-word-space∧0 pointer (0 / 14 15 / 18 / 35)
literal atom	3	property-list / top-level-value; function-type / function-body; PNAME / function on files (reserved)
array	不定	36-bits-integer, floating-number, pointer
list-element	1	cdr | car
hash-array	不定	hash-item | hash-value

になっていて，機能としては a-list に劣る部分もあるが[10]，LISP 1.5 の a-list 方式に比べれば，変数に関する処理速度は大幅に向上している.

2.2.2 MACLISP[8]

MACLISP の PDP-10 上でのデータ表現は Stanford LISP 1.6 と同様なので省略する.

Multics 上ではポインタに2ワード必要とするため，list-element が4ワードになる．数値アトムは，ポインタ用の2ワードの内の1ワードを値そのものとすることにより，実質的な数値のためのセルが不要となる[9](**Fig. 3**).

2.3 INTERLISP[17] (**Table 3** 参照)

INTERLISP (以前の名は BBN-LISP[16]) は，これまでの LISP と違ってページング・ハードウェア上に作られている．LISP 1.5 のようなすべてのデータをポインタを基礎にして表現することは，ページ間の参照が増加するので避けねばならない.

INTERLISP では各ページごとにデータ型を定めることによって **Table 3** に見るとおり従来と全く異なったデータ表現に到達した．数値アトムの型表示がなくなり，literal-atom は連続したワード内に必要な情報

を格納している．しかも，これらの情報 (P-NAME, 関数型，関数の定義) は定められた位置にあるので，従来の LISP のような literal-atom の属性リスト (property-list) を探索するという余分なステップが不必要になる．

変数値については，スタックを用いた deep-binding 方式をとっており，LISP 1.5 の a-list 方式よりは速くなっている．

3. TOSBAC-5600 LISP[4] のデータ表現

前章での概観から導き出される結論は，「データ表現の統一性を失ってもよいから，各データごとに最適の表現を選ぶ事によって，メモリーも処理速度も得をすることができる．」というものである．これは，LISP が理論的な言語[5] から実用的言語に成長したことと無縁ではない．

処理速度でデータ表現に関係するものとしては，

① 型判別——アトムか？（ストリング，文字アトム，数値を包含する)，数値か？といった類別をも含む．
② 変数値のとり出し
③ 関数値のとり出し

の3点が問題になり，メモリー量については，

④ データの型をどのように表現するか？
⑤ 文字アトムのもつ情報の表現

の2点が問題になる．

3.1 どのような表現が実現されたか

以上5つの問題点に対してとられた解決法は，以下のとおりである．

(1) メモリー量を減らすために，データの存在する番地そのものに型情報を与える．すなわち LISP 内のアドレス空間を分割して，型ごとにデータを収納する．
(2) アドレス空間の配置を工夫して，型判別を最も少ないステップでできるようにする．この結果が，**Fig. 4** の配置である．
(3) small-number は，メモリーを消費しないので導入する．
(4) 変数値のとり出しには，shallow-binding を採用する．
(5) 文字アトムは4語の連続領域を確保し，各情報は定められた位置におく．したがって関数の定義がどこにあるかを探す必要はなくなる．

このようにして得られたデータ表現は，**Table 4** のようになっている．そして，この成果は **Table 5** の比較によって確められる．Table 5 の各項目は無作為に選ばれたものではなく，電子技術総合研究所の山口氏

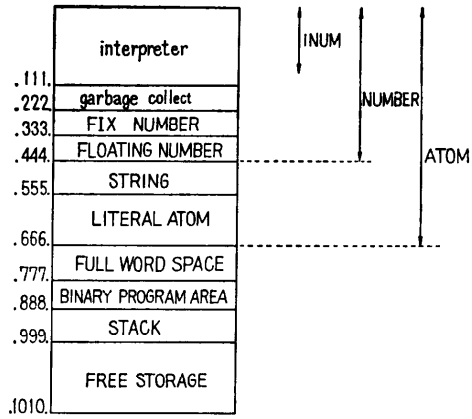

Fig. 4 memory allocation in TOSBAC-5600 LISP system for each datum

Table 4 Data representation of TOSBAC-5600 LISP system

データ型	ワード数	表現
small-number	0	アドレスそのもの (|n| < 8192) n ≧ 0 アドレス = n n < 0 アドレス = 16384 + n
fixed number	1	
floating number	1	
string	1	full wordへのpointer | 文字数
literal atom	4	value | function-body function-type-address | property-list string-pointer | revert flag | recursion-counter
array	不定	fixed-bits-integer, floating-number, pointer
list-element	1	car | cdr

Table 5 Comparison of the execution steps.

	数値アトムか？	アトムか？	SUBR 型関数か	変数値のとり出し
L 1.6	1	1	1	2
LISP 1.5	3(***)	2	5～(**)	20～(****)
Stanford LISP 1.6	7	2	5～(**)	8～(**)
INTERLISP	1(*)	1(*)	2	3～(****)

(*) ページシング機構の問題だが，一応1ステップでわかると考えた．
(**) property-list 上で，property の位置が問題になるので数字は property が property-list の最初にある場合のものである．
(***) number-tag-field の check が1ステップでできるものと考えた．
(****) deep-binding なので，数字は一番浅い変数の場合である．

のデータ[19]によって確認されているように，LISP の万能評価関数 (interpreter) での使用頻度の高い基本操作である．

3.2 TOSBAC-5600 LISP のデータ表現での問題点

上記のようにして得られたデータ表現にいくつかの問題点があった．以下それについて述べる．

1) 文字アトムの構造について

① 3ワードの連続領域で良かったのではないか？

変数値，関数型，関数値，属性リスト，p-name は不可欠だが，revert や depth-counter は特別な場合にだけ使われるので，属性リストに貯えてもよい．そうすれば3ワードですむ．しかし，将来の拡張を考えれば，4ワード確保している方が，都合がよいという意見もある．

② 変数値の shallow-binding はよかったのか？

付録でも述べる様に，shallow-binding が一番速いかどうかには異論がある．shallow-binding でなければ，アトムの領域が2ワードですむ．

③ 変数間の value-sharing* は必要なかったか？

実用上の要求によるであろうが，一変数の値を変更することによって，いくつかの他の変数の値まで同様に変更されるというのは面白い．また他の利点としては，あらゆる値を変数がもちうる事（現行では，「未定義変数」という値はとれない）が挙げられる．欠点は，リストを1語余分に必要とすることと，変数値のとり出しに1ステップ CAR をとるという操作が必要になるということである．

2) 数値について

① small-number の領域 (Fig. 4 のINUM) は，もっと広げて，large-integer の手前 (Fig. 4 の .222.) までにするべきであった．これは設計上の不手際である．

② LISP による数式処理システム（例としては，MACLISP による MACSYMA[22]）のことを考えれば，big-number（無限長整数），multi-precision-floating-number（無限精度実数），複素数，倍精度実数などといった数値型の存在が有用となる．特に何語長でも必要なだけ取った数値というのは，LISP でこそ実現できるものであろう．

3) ストリングについて

Table 3 の INTERLISP のように，先頭の文字位置をデータに含めるべきであった．これによってサブストリングを変更すると全体のストリングが変更されるという効果や，不必要に fullword-space を消費しないという効果が生じるからである．

4) データ領域のサイズ宣言について

Fig. 4 のようなアドレス空間分割による欠点は，実行前に各データ領域の大きさを宣言せねばならないことと，実行中にどれか1つのデータ領域が一杯になっても実行を中止せねばならないことである．しかし，TOSBAC-5600 LISP システムは，TSS-interactive-system なので，ユーザが適当なデータ領域宣言を得るのは，そう困難な事ではなく，最初の設計時に心配されたほどの障害ではない．

4. これからの LISP システムでのデータ表現

これからの LISP システムは大別すると二つに分れる．一つは既存の計算機方式のうえで作成する場合であり，一つは LISP マシンを実現する場合である．

4.1 既存の計算機方式に依存した場合

現在注目すべき動きは，東京大学後藤研究室の H-LISP の association を含んだデータ[3]とか，電子技術総合研究所での AQV-triplet を使用し得る LISP システム[20]とかのように，リスト以外で表現されるデータを LISP の中に取り込もうとする動きである．これは，これまでの LISP データ表現の発展の延長線にあるものと考えられる．究極的には ALGOL-N などに見られるようなデータ定義機能を LISP に取り込むことになるであろう．

必要なことは，LISP においてリスト表現が，どこで本質的に働いているのかという議論である．将来は，その本質的な部分以外のデータ表現では，ハードウェア，OS を考慮して最も効率的な表現を，リスト形式を離れて実現することになるであろう．

一般的なデータ表現としては，作成時の労力を考えなければ INTERLISP-TOSBAC-5600 LISP 方式のデータ表現が処理速度とメモリーの二点で優れている

* Stanford LISP 1.6 や MACLISP の shallow-binding では複数個の変数が（番地として）同一の値を共有することができるが，TOSBAC-5600 LISP では，Table 4 に見るとおり文字アトムの領域内に値を取り込んでいるために，変数値の共有は実現できない．

であろう．LISP 使用上の一般的傾向として，データ領域に 100 K 以上のポインタを使用する様な大規模なシステム作成が行われつつあるので，バイト・マシンの場合には 16 ビットではなく，24～32 ビットのポインタが必要になる．この場合には実際に用いるアドレスには 20 ビット位で十分なので，余ったビットを利用して，後述の LISP マシンのように，情報を蓄えることも可能である．問題は，これらのビットの取り扱いがどの程度のステップ数で可能かという点にあろう．

変数値の扱いについては，shallow-binding か，スタックを用いた deep-binding かの二者択一になる．どちらが有利であるかについては，計算機方式，スタックの構成，動かされるプログラム（自由変数が多ければ shallow-binding が有利）によって結論が変わり得る．

4.2 LISP マシンの場合

LISP マシンの試みは，たとえマイクロプログラムによるものであれ，ソフトウェアの要請によってハードウェア（ファームウェア）を構成するという大きな意味をもつ．既存の演算体系では多くのステップを必要とした操作が 1 ステップでできるので，LISP の弱点であった処理速度と記憶容量の二点は大幅に改善される見通しがある．

現在 LISP マシンは，日本では電子技術総合研究所[14]，米国では MIT[11]，Xerox[2] などで試みられている．

これらの試みの特徴としては以下に挙げるものがある．

（1） データ，特に literal-atom の持つ情報をビットの形式で貯え，しかも高速で処理する．

（2） コントロールに Bobrow のモデル[1]（いわゆる spaghetti-stack）を使用し，環境をデータとして扱う．

（3） ポインタのアドレシングを 24～32 ビット取って大容量（4 M ワード以上）のデータを扱う．

（4） リストというデータ自体をコンパクトな表現にする．

（5） コンパイルされたコードを考えたデータ構造と演算体系をとる．

LISP マシンでのポインタは，一般的には 32 ビットで表現される．この 32 ビットはアドレス領域と情報領域に分かたれる．情報領域のビットは，ガーベッジコレクション，データ型についての情報をこのポインタについて運ぶ．literal-atom の場合にはさらに，変数値の有無，関数型などを，リスト要素の場合には cdr が次のワードであるとか，NIL であるとかいう情報を運べる．しかもこれらの情報は従来の符号ビットのようにロードされるだけで入手できるように演算体系が組まれる．これによってコンパイルコードも簡単なものになる．

上記のポインタ表現で問題となるのは，リスト要素の cdr に関するビットで，ガーベッジコレクションの扱いが問題になる．

一方 spaghetti-stack によるコントロールは，プログラム言語のデータとして「環境」が明白な地位を占めるという点で大きな意味をもっている．これは，CONNIVER[7]，QA 4[13] など LISP で書かれた高級言語による成果を，LISP マシンというレベルに取り込むものである．これによって応用面では，モデルの作成や仮想世界の扱いが容易となって，人工知能システムや自然言語を扱うシステムの作成が容易となるに違いない．

LISP マシンの試みは，まだ開始されたばかりではっきりとした成果は生じていない．しかし，LISP マシンが成功すれば，これまで TSS を備えた大型機でのみ実現されてきた強力な LISP システムが安価に提供されることになり，プログラム言語はもとより，ハードウェアや計算機複合体や人工知能システムの設計にまで大きな影響をもたらすであろう．

5. おわりに

LISP という一言語のデータ表現についても，その実現型式は一通りではなく，多くの変化がある．どのような選択をするかは，設計者の自由であるとともに責任である．ユーザーにとってより望ましい LISP システムがソフトウェアのみならずハードウェアの改良によって実現されることを切望する．

TOSBAC-5600 LISP システムは，電子技術総合研究所よりパターン大型プロジェクトの一環として発注されたもので，設計にあたって島内，筧，米沢氏の御指導をいただいた．本文の内容に関して電子技術総合研究所推論機構研究室の横井氏より有益な批判をいただいた．

参考文献

1) D. G. Bobrow & B. Wegbreit: A Model and Stack Implementation of Multiple Environments, Comm. ACM, Vol. 16, No. 10, pp. 591

～603 (1963).
2) L. P. Deutsch: A LISP Machine with Very Compact Programs, Proc. 3rd IJCAI, pp. 697～793, (1973).
3) E. Goto: Monocopy and Associative Algorithms in an Extended Lisp, 記号処理シンポジウム, (Jul. 1974).
4) 黒川, 八木: TOSBAC-5600 LISP 1.6, 東芝レビュー, Vol. 29, No. 10, pp. 876～880, (1974).
5) J. McCarthy: Recursive Functions of Expressions and their Computation by Machine, Comm. ACM, Vol. 3, pp. 184～195, (1960).
6) J. McCarthy et al: LISP 1.5 Programmer's Manual, MIT Press, (1966).
7) D. V. McDermott, G. J. Sussman: The Conniver Reference Manual, MIT AI memo 259 a, (Jan. 1974).
8) D. A. Moon: MACLISP Reference Manual, Project MAC, MIT, (Apr. 1974).
9) D. A. Moon: Private Communications, (Nov. 1974).
10) J. Moses: Functions of function in LISP ——Why the Funarg Problems called Environmetal Problems——, MIT AI Memo 199, (Jun. 1970).
11) R. Greenblatt: The LISP Machine, MIT, AI Lab. Working Paper 79, (Nov. 1974). Also private communications with J. Moses.
12) L. H. Quam, W. Diffie: Stanford LISP 1.6 Manual, SAILON 28.4, (1970).
13) J. F. Rulifson: QA 4 Programming Concept, SRI AIG Tech. Note 60, (Aug. 1971).
14) 島田, 山口, 坂村: LISP マシンとその評価, 情報処理学会計算機アーキテクチャ研究会資料 74-7, (Nov. 1974).
15) 島田: Private Communications, (Jan. 1975).
16) W. Teitelman, D. G. Bobrow et al.: BBN-LISP TENEX Reference Manual, BBN, (Feb. 1972).
17) W. Teitelman: INTERLISP Reference Manual, Xerox, (Feb. 1974).
18) W. Teitelman: Private Communications, (Mnr. 1975).
19) 山口: LISP における万能評価関数の動的特性の測定, ETL 内部資料, (Jan. 1975).
20) 横井, 松田: EXPLAIN, ETL 内部資料, (Dec. 1974).
21) LISP 1.6 User's Manual, EPICS-5-ON, 電子技術総合研究所, (Mar. 1974).
22) MACSYMA Reference Manual, 第7版, The MATHLAB Group, Project MAC-MIT, (Sep. 1974).

(昭和50年5月2日受付)
(昭和50年7月7日再受付)

付録 LISP の変数名変数値の処理方式

LISP の変数名—変数値の処理には **Table 6** にみられる種類がある.

deep-binding は, 変数名-変数値の対を LIFO スタックまたは tree に貯えるものである. 変数値を得るためには, 変数名が一致する対を求めて探索せねばならない. 特に自由変数についてはずっと奥 (deep) まで探さないといけないのでこの名があるらしい.

shallow-binding は, 変数値を変数の property-list または特定のセルに貯える. 変数の現在の値は変数からすぐ求められる. 特に自由変数についてはほとんど探索をせずに値を求めることができる. しかし, 束縛変数については, 古い値を貯えることだけでなく, 将来その値を回復するという操作が必要になる. したがってスタックによる deep-binding の方が, 全体としては, shallow-binding より速いという説[15]もあるが, 筆者はまだ参考となるデータを入手していない. もちろん両者はプログラムによって速度が異なる. 自由変数の使用量が多ければ shallow-binding が速くなり, 少なければ逆に遅くなるからである.

a-list 方式が非能率なのは明白だが, Moses が指摘している[10]ように,「環境」を保存するという機能を含んでいる. この機能は shallow-binding ではもちろん, スタックを用いた deep-binding でも実現は困難である. この機能を積極的に実現することにより,「環境」の保存のみならず活用を計ろうとしたのが Bobrow の提案した[1] spaghetti-stack である. spaghetti-stack は強力な機能を備えているが問題となるのは, これまでの「環境」を意識しないで書かれたプログラムの処理速度が極端に落ちるのではないかという心配である. アルゴリズムの詳細はわからないが, 工夫すれば1割以下のスピードダウンで従来のプログラムを処理できるという報告があり[18] LISP マシン上では, それ程大きな負担にならないであろうと考えられている.

Table 6 Binding mechanisms in LISP

処理方式	確保領域	例となる LISP システム
deep binding	a-list	LISP 1.5
	stack	INTERLISP HLISP
shallow binding	p-list	Stanford LISP 1.6, MACLISP
	atom-header	TOSBAC-5600 LISP
Bobrow model	spaghetti-stack	LISP マシン (Deutsch, Greenblatt, 島田)