Utilisp システムの開発†

近山 隆††*

近年，プログラム言語 Lisp は，その提供する高い会話的プログラム生産性のため，記号処理用言語としてのみならず，汎用のシステム記述言語として注目されている．筆者はシステム記述言語としての利用を目的として，Lisp の方言 Utilisp の処理系を Hitac M シリーズ VOS3 上に開発した．Utilisp はシステム記述用に必要な文字列や二進データの効率のよい処理機能や，操作系など外部との通信を行う機能をもつ．一方，Lisp 本来の特質を生かし，多様な利用形態に対応するため，高い柔軟性と拡張性を有している．本論文では，Utilisp システムの機能面での特徴を述べ，その必然性と，実現に用いた手法を解説する．

1. はじめに

Utilisp[1] (University of Tokyo Interactive LISt Processor) は，筆者が東京大学大型計算機センタの M 200 H/VOS 3 の TSS 上に開発した Lisp システムである．開発の目的は，ソフトウェアの生産性を非常に高めることのできる言語 Lisp を，いわゆる人工知能のようなリスト処理を主体としたプログラムのみでなく，プログラム言語処理系や種々のプログラミングツールなど，非数値処理を必要とするシステムの汎用記述言語として用いることにあった．

筆者らが HLISP[2] を用いて言語処理系の試作を行った際には[3]，Lisp の拡張性を生かして種々のツールを Lisp 自身で作成し用いることができたため，非常に短期間で試作を完了することができた．しかし，できあがった処理系は低効率で，試作としての成果はあがったものの，まったく実用に耐えないものとなってしまった．この低効率の原因の一つは，HLISP が Lisp の拡張性を十分に生かした使用法を予期していなかったため，効率のよい機能拡張が不可能であったことにあり，もう一つは，ファイルや文字列などの低レベルのデータを効率よく操作できる機能に欠けていた点にあった．これらは，これまでの Lisp システムの多くに共通する欠点であり，Lisp の使用目的を狭める主因となってきたものである．

実用に供するツールの記述言語として用いるためには，高い実行効率，わけても高効率の入出力や文字列処理機能が不可欠である．また，既存のプログラムを有効に利用するために，オペレーティングシステムのもつ諸機能の利用や，Lisp 以外の言語のプログラムの呼出し機能も欠かせない．一方，Lisp が本来もつ会話的ソフトウェア開発の高生産性や，言語の拡張性を十分に活用するためには，システムの諸機能をできるかぎり柔軟な形で開放することが望ましい．Utilisp の開発にあたっては，これらの点にとくに配慮した．

システムの柔軟性を高めることは，ややもすると実行効率を損ねる結果を招きかねない．Utilisp では，実行効率の低下を最小限に止めながら柔軟性が得られるように種々の工夫を施した．利用可能な論理空間には余裕があったので，やむをえない場合には空間効率をある程度犠牲にしても速度効率を損なわない方針をとった．また，適当な方法がない場合には，目的に応じて低効率ではあるが柔軟な機能と，単純ではあるが高効率な機能を使いわけられるようにした．

システムの柔軟性を高めるためには，通常の Lisp システムにない機能を付加する必要が生じる．このことによってシステムが複雑化し，初心者に使いにくいものになることは望ましくない．このため，付加的な機能に関する知識がなくても，表面上は従来の Lisp システム同様に使用できるようにした．また，可能な限りシステムを統一的で簡潔なものにし，初心者にも理解しやすいことを目標とした．

2. 処理系の概要

Utilisp は全体としては Lisp Machine Lisp[4] と共通点の多い，Maclisp[5] 系の Lisp の方言である．実行は解釈実行，翻訳実行の両方式が，個々の関数単位で混用できるようになっている．

データ型としては，リストを構成する二進セル，関数名や変数名などに用いる記号アトム，数値をあつか

† Implementation of the Utilisp System by TAKASHI CHIKAYAMA (Department of Mathematical Engineering and Instrumentation Physics, Faculty of Engineering, University of Tokyo).

†† 東京大学工学部計数工学科
* 現在 (財)新世代コンピュータ技術開発機構研究所

うための整数，浮動小数点数に加えて，文字列，配列，配列要素，入出力流，機械語関数の計9種がある．

整数は24ビット長のポインタに埋め込むものだけを用意した．当面の利用目的にはこれで十分であり，演算結果の格納のために記憶割付けを行う必要がないため，演算速度・記憶効率の両面で有利である．これ以上の長さを必要とするものに対しては，むしろ多倍長整数を用意するのが得策であろう．

文字列はシステムの利用目的からしてもとくに効率のよい操作が必要である．ハードウェアのもつ文字列操作機能をシステムがそのまま利用できるように，文字列はベタ詰め表現をとった．このため，従来のLispシステムの多くが効率よくあつかえなかった大量のビット列に対する操作も，文字列に対する操作と共通する関数を用いて効率よくあつかえるようになった．

配列は一次元のものだけを用意した．多次元配列機能はマクロ（後述）を用いれば容易に実現できる．配列も，二進セルと同様，自由に割付け・解放が可能である．従来のLispではPascalのレコード型のようなデータ構造の表現に二進セルによるリストを用いるしかなく，二進セル間のポインタのための記憶場所や構造の大きさに比例するアクセスの手間を要したが，この自由度の高い配列を用いて表現すれば，構造中の各要素へのアクセス速度の点でも，記憶効率の点でも改善が見込まれる．

配列要素は，配列の一要素へのポインタである．これに対し変数と同様に値の参照や代入ができるようにしたことによって，参照渡し引数が統一的に記述できるようになった．また，配列用のマップ関数も自然な形で記述できる．

機械語関数は，組込み関数やコンパイル済関数の定義体である．プログラムから機械語関数の引数個数などを調べられるようにしたため，Lisp自身で虫とり支援ルーチンなどを作成するのが容易になっている．

Utilispの関数は，組込みの制御構造（CONDなど）を除き，EXPRやSUBRにあたる，きまった数の引数をすべて評価するものしかない．引数渡しを一元化することによって，システムの無用な複雑化を防ぎ，虫とり支援などのツールの開発を容易にすることができる．このために生じる機能の不足は，マクロ機能と省略可能引数の機能によって補える．

UtilispのマクロはMaclispのものと基本的には同じものである．マクロは引数リストを評価せずに受けとり，これを適当に加工してLispの式を作り出す．これをふたたび評価して，最終的な式の値を得る．すなわち，マクロは動的に呼び出されるLispのプリプロセッサであるともいえる．マクロは展開時に引数をまとめてリストにしてQUOTEをつけるようにすればFEXPRと同じ機能も果たせる．FEXPRなどがLisp関数の一種として特殊な存在であるのに比べ，マクロはメタプログラムであり，論理的にすっきりしているのが利点である．プリプロセッサの一種なのであるから，翻訳時には展開してから翻訳することになるので，翻訳実行時のオーバヘッドもない（**図1**）．

```
FEXPR による記述:
  (DEFUN FEXPR LOOP (L)
    (EVAL (APPEND '(PROG ( ) LOOP) L '(GO LOOP))))
マクロを用いた記述:
  (MACRO LOOP (L)
    (APPEND '(PROG ( ) LOOP) L '(GO LOOP)))
```

図1 マクロとFEXPRの用法
(DEFUN FEXPR...) はFEXPR関数を定義する．(MACRO...) はマクロを定義する．

Fig. 1 Usage of MACRO and FEXPR.

```
関数定義:
  (DEFUN FN (X Y (Z 'A) (W (CAR X)))
    ...)
呼出しと意味:
  (FN '(A B) 'C 'D)=(FN '(A B) 'C 'D 'A)
  (FN '(A B) 'C)    =(FN '(A B) 'C 'A 'A)
  (FN '(A B))..........誤り
  (FN) ..................誤り
```

図2 省略可能引数機能

Fig. 2 Default parameter mechanism.

省略可能引数もマクロとして実現することが可能であるが，組込み関数の多くが省略可能引数をもつことが望ましかったこともあり，ラムダ結合の際の基本機能として組み込んだ．ラムダ引数並びの要素がリストの形をしているときには，第1要素が引数名，第2要素が省略時の値を表す．関数の呼出し側で省略された引数に対し，関数側に省略時値の指定があれば，その式を評価してあたかも引数として渡されたかのように用いる（**図2**）．省略可能引数機能のないシステムでは，不定個の引数をとれる関数（LEXPRなど）をその目的に利用せざるをえないが，読解性の面でも，引数個数の誤り検出の面でも不利である．

3. 柔軟性を高める機能

利用者がシステムのもつ機能を容易に変更・拡張できるように，システム組込みの機能はできる限りパラメタ化する方針をとった．

```
(DEFUN MY-TOPLEVEL ( )
  (LOOP (SETQ LAST-INPUT (READ))          ; 読み込んだS式を変数に格納
        (SETQ + (PRINT (EVAL LAST-INPUT))); 評価結果を打ち出し，同時に "+" に入れる
        (SETQ - LAST-INPUT)))             ; 読み込んであった式を "-" に入れる
(SETQ TOPLEVEL MY-TOPLEVEL)               ; トップレベル定義を変更
(TOPLEVEL)                                ; トップレベルへ戻る
```

前回読み込んだS式を "-"，その評価結果を "+" という．変数の値としてアクセスできるようにした，トップレベル定義．

図 3 トップレベル定義の例

Fig. 3 User-definable toplevel.

S式の入出力には，Maclisp に見られる表駆動方式を採用した．特定の変数の値である配列を表として用いるので，この変数に利用者が用意した表を値として設定すれば，読込み時の文字の意味（括弧，註釈開始など）を自由に変更することができ，わずかに文法の異なる他のシステムからの移行などに便利である．また「'A」などの略記を一般化した読込みマクロ機能を提供し，利用者が適当な略記法を定義することもできるようにした．

名前表への記号アトムの登録のしかたも利用者定義可能とした．このため，Lisp とは異なる名前空間構造をもつような言語の処理系を作成する場合でも，語句解析と名前表探索程度の仕事は Lisp の読込みルーチンを利用して効率よく行うことができる．また，Lisp 自身についても，木構造の複数名前空間は有用であり，これを管理するためのツールも Lisp で記述することができる．

誤り処理ルーチンや端末からの割込みを処理するルーチンも，それぞれ特定の変数の値を用いるようになっているので，利用者定義が可能である．誤り処理ルーチンは誤りの生じた環境で誤り原因情報を引数として呼び出す．そこに至るまでの履歴情報を Lisp データの形で入手するための関数も用意したので，誤り診断情報の加工も Lisp 自身で記述できる．また，誤り時の例外処理も，多重飛出し機能 (CATCH-THROW) と組み合わせることにより，通例の ERRSET を用いる方法よりも緻密に行うことができる．

トップレベルの評価関数は一応 EVAL であるが，トップレベルのドライバ自身も利用者定義が可能である(**図3**)．このような機能がなくとも，あたかもトップレベルであるかのように振舞う関数を実行すればよさそうだが，それでは誤り時の処理などがうまくいかない場合がある．

ここに述べてきたような諸機能は，いずれもなんらかの実行速度上のオーバヘッドを伴うものである．しかし，入出力に関するもの以外は例外的な場合に対する処理に関するもので，全体の効率にはほとんど影響を与えない．また，入出力はもともと操作システムとの通信などにかなりの手間を要するものであり，それに比べればここに述べたような機能によるオーバヘッドはごく小さいものである．大量のデータの入出力を高速に行う必要がある場合には，S式表現でデータを格納するのはそもそも効率的でない．文字列や二進データ用には別に効率のよい入出力手段を用意したので，これを用いるのが適当である．

4. 高速化のための技法

Utilisp では実用ツールの記述言語として十分な効率を得るために，高速化のための種々の工夫を行っている．

データの型に関する情報は，Cambridge Lisp[6] に習い，そのデータを指すポインタの側にタグとして格納する方式をとった(**図4**)．この方式をとれば，データ本体内に型情報を置く方式に比して，型の判別に必要な記憶参照回数が1回少なくて済む．ことにリストの要素の型を順次調べるような場合，リストを構成する二進セルの線形化* が行われていれば，参照する記憶領域を局所化でき，キャッシュ記憶と主記憶，さらには二次記憶とのデータのやりとりを少なくできる．

型タグの値は，頻繁に行われる型の判定がとくに高速になるように割り当てた(**図5**)．ことに，リストの終端に達したかどうかの判定による繰返し構造は非常に多用されるので，ハードウェア組込みの索表用命令を用いて，判定と分岐が一命令で可能になるようにした．

変数の結合にはスタックによる浅い結合方式を用い

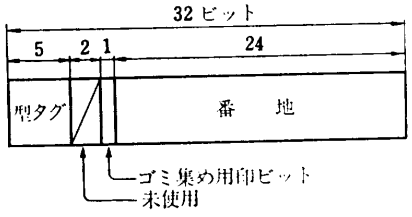

図 4 型タグつきポインタ

Fig. 4 Tagged pointer.

* データセル内のポインタが指す対象ができるだけ隣接する番地にくるようにセルを並べ直し，データの局所性を改善すること．

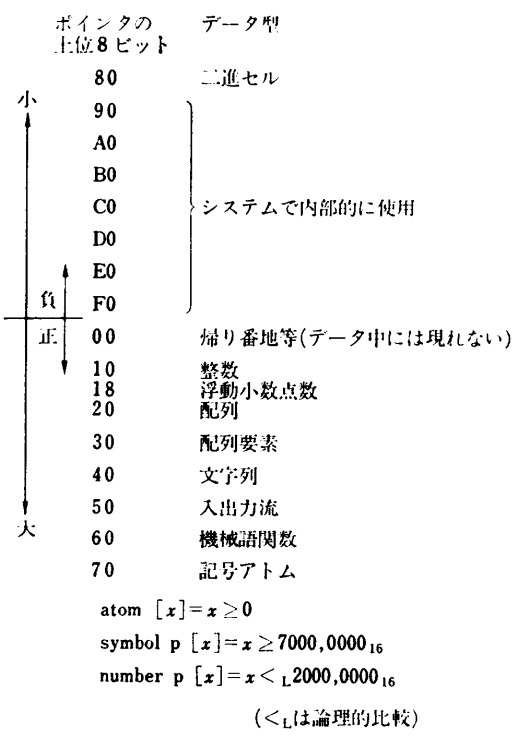

図 5 タグ値の割当てと型の判定

Fig. 5 Tag values and type tests.

た．スタック管理のオーバヘッドを減らすため，スタックは1本で制御用，結合用，作業場所用を兼ねる．スタック中のデータにもすべてタグを付与するので，ゴミ集めにも支障はない．

文字列にはベタ詰めの表現を用いたが，文字列処理を中心とするようなプログラムを効率よく実行するためには，可変長文字列のための記憶割付けが効率よく行えることが不可欠である．従来の Lisp 処理系の多くが文字列にベタ詰め表現を用いず，数文字からなるブロックのリストなどの形式で表現していたのは，可変長データに対する記憶管理が困難であったことが最大の原因である．Utilisp では，複写方式のゴミ集め法[7),8)]を採用することによって，空領域をすべて連続したひとつの領域にまとめることができたため，可変長領域の割付けも容易に行うことができるようになった．複写方式では，ゴミ集めにともなって自然な形で使用中データの線形化を行うので，ページングによるオーバヘッドも減少させることができる．複写方式の最大の欠点は一時に利用可能な論理記憶空間が実質上半減してしまうことにあるが，現在のところは利用可能な論理空間に余裕があるので，問題はない．

翻訳系は Lisp 自身で記述されており，

```
関数定義（S式表現）:
  (DEFUN FN (X Y (Z 'A) (W (CAR X)))...)
対応する機械語表現:
  引数0個→ENTRY0 B ERROR
  1個→ENTRY1 B ERROR   } 誤り処理ルーチンへ
  2個→ENTRY2 B LAB2
  3個→ENTRY3 B LAB3
  4個→ENTRY4 B LAB4
      LAB2  EQU *
        {'A の評価}
      LAB3  EQU *
        {(CAR X) の評価}
      LAB4  EQU *
        {本体}
```

図 6 省略可能引数の実現法

Fig. 6 Implementation of default parameters.

組込み関数呼出しの展開
引数の評価順の変更
不要な型検査の除去
再帰呼出しの繰返しへの変更

などの最適化を行うが，必ずしも徹底したものではない．

翻訳系の生成するオブジェクトは完全に再配置可能で，翻訳後も関数呼出しは（展開してしまうものを除き）すべて関数名のアトムを経由して行うので，解釈実行関数とたがいに呼び合うことができる．

翻訳によるものも含め，機械語関数における省略可能引数の機能は，引数の個数ごとに別個の入口点を設けることによって実現した(**図6**)．入口点の選択は引数個数によるインデクシング* で行う．最小個数未満の引数個数に対応する入口点には，引数個数誤りの処理ルーチンへの分岐命令を格納しておけば，最小引数個数の検査も同時に行うことができる．最大個数の検査は別途に行わざるをえない．しかし，引数の省略を許さない場合においても最低限1回は引数個数の検査が必要なのであるから，省略可能引数機能の導入による実行速度上のオーバヘッドは皆無ということになる．記憶領域上のオーバヘッドは，引数個数×分岐命令語長程度かかるが，かなり大規模なプログラムでも関数の総数は1,000のオーダであり，引数も通常数個程度であるから，余分に必要な記憶領域はデータ用に必要な領域に比して無視できる程度のものである．

5. ユーティリティ

現在 Utilisp ではプリティプリンタ，構造エディタなどのツールが利用できる．

* 実際には内部ではハードウェアのアドレシング機構に合わせて常に引数個数は4倍してあつかい，これでインデクシングを行う．

```
(DEFMACRO INCR (X N) (LIST 'SETQ X (LIST. 'PLUS X N)))
```

① 逆 QUOTE を用いないとき

```
(DEFMACRO INCR (X N) `(SETQ ,X (PLUS ,X ,N)))
```

② 逆 QUOTE を用いたとき

図 7 逆 QUOTE 機能
Fig. 7 Backquote facility.

表 1 Lisp コンテストの問題の実行時間
Table 1 Execution time for the problems of the Lisp contest.

	HLISP	OLISP	LISP 1.9	Utilisp
BITA-7	108	176	278	12
BITA-8	377	622	953	44
BITB-8	105	91	126	10
BITB-9	356	307	712	35
SORT-100	135	423	605	28
TARAI-5	8,905	18,934	43,609	604
TPU-1	1,029	658	1,591	68
TPU-2	2,871	2,064	6,764	244
TPU-3	1,227	850	2,978	96
TPU-4	1,707	1,161	3,250	133
TPU-5	181	132	555	15
TPU-6	4,893	3,617	27,049	453
TPU-7	1,053	776	3,189	82
TPU-8	724	596	1,163	76
TPU-9	545	424	964	52
Hardware	HITAC 8800	ACOS 77/800-2	TOSBAC 5600/160	HITAC M 200 H

注) 単位はミリ秒，ゴミ集を除く CPU 時間．
データは Utilisp 以外は第 2 回 Lisp コンテストのもの．Utilisp については筆者の計測による．

プリティプリンタは括弧のネスティングを見やすくするために段付けを行ってS式の書出しを行うのが基本機能である．Lisp のS式はネスト関係が明白であるから，Algol 風の構文をもつ言語と比較して万人の納得する段付け容易である．また，Lisp の普及が進まない大きな原因が括弧の対応づけの煩わしさにあることを考えると，プリティプリンタは Lisp システムに不可欠のツールであるといえる．Utilisp のプリティプリンタは，段付けに加えて QUOTE の "'" による略記や，構造体の記述を容易にする逆 QUOTE 記法を用いる機能* をもつ (**図 7**)．

Utilisp の構造エディタ USE は，基本的には Interlisp[10] のエディタと同系のものである．

このほかにも，虫とり支援ツールやマクロ定義支援ツールなどのツールもあるが，まだ個々のツールに機能の不足が見られ，今後の充実が必要である．核となる処理系はこうしたツールの Lisp 自身による記述を想定した柔軟性をもつので，ツールの開発は順調に進むものと予想される．

* 逆 QUOTE 記法では，"`" の後にくるS式は QUOTE されるが，そのなかでコンマの後の部分だけは評価される．この機能は，読込み時に適当な LIST, CONS, QUOTE などに展開することによって実現される．

6. 性能評価

表 1 に第 2 回 Lisp コンテスト[9] の各問題に対する Utilisp とコンテスト参加の代表的な処理系の所要実行時間をまとめた．速度の差は顕著であり，ハードウェアの速度差を考慮しても，Utilisp の効率の高さは明白である．これらの問題はリスト処理に関するものであり，Utilisp が本領とする文字列処理について比較すれば，この差はさらに広がるものと予想できる．

速度以上に重要な Lisp 処理系の性能として，使い勝手のよさがある．これは数字に表しにくいもので，使い慣れた処理系が使いやすいという面もあるため，客観的評価は困難である．また，Lisp 処理系単独の問題ではなく，Lisp 上のユーティリティ群や，Lisp をとりまくオペレーティングシステムなどの環境，さらには，キーボードの操作性，端末機との通信速度などのハードウェアの要因にも大きく左右される．Utilisp はまだ完成後日も浅く，使い勝手に関する評価は今後の使用経験を通じて行われていくべきものと考える．

7. 開発と使用の経緯

開発の着手は 1980 年 7 月で，同年末までに解釈実行系が，翌年 6 月には翻訳実行系が完成した．これに並行して種々のツールの開発や移植を行った．

現在，東京大学大型計算機センタで公開されているのをはじめ，Facom M シリーズ用に手直ししたものを含めて，各地の大学や研究所で稼動を開始している．その用途は，当初の目途のとおり，人工知能的なものに限らず，言語処理系やツールの記述用にもひろがっている．とくに，大量のデータのコード変換やフォーマット変換など，単純ではあるがある程度の効率を要求され，しかも一過性のものであるためアセンブリ語などでは記述が面倒，という処理の記述にはたいへんに便利に用いられている．これは，Lisp が本来もつ会話的プログラム開発の高生産性と，Utilisp の特徴である大量の二進データの高効率な処理機能との結合の生んだ結果である．

8. おわりに

Utilisp システムでは，柔軟性と効率の両立という所期の目標をある程度達成しえた．これは，データ表現や評価の算法の工夫により，ハードウェアのもつ機能を十分に活かしつつ，柔軟性を取り入れた設計方針の成果である．

現在のシステムはまだ使用経験が浅く，十分なユーティリティ機能が備わっているとはいいがたい．今後の利用者の拡大とともにユーティリティ・プログラムを蓄積し，使いやすいシステムに仕上げていくことが必要である．

謝辞 最後に，種々のコメントをいただいた和田英一教授や Utilisp の利用者諸氏と，Facom M シリーズへの移植の労をとられた稲田信幸氏，多くのコメントをいただいた査読者の方に感謝の意を表したい．

参考文献

1) Chikayama, T.: Utilisp Manual, *Math. Eng. Tech. Rep.*, 81-6, Univ. of Tokyo (1981).
2) Kanada, Y.: *HLISP and Supplementary HLISP-REDUCE Manual*, Univ. of Tokyo (1979).
3) 近山：プログラミング言語 ADA 処理系の試作，第21回プログラミング・シンポジウム報告集，pp. 137-142 (1980).
4) Weinreb, D. and Moon, D.: *Lisp Machine Manual*, Symbolics Inc., Cambridge, Massachusetts (1981).
5) Moon, D.: *Maclisp Reference Manual*, Project MAC, MIT, Cambridge, Massachusetts (1974).
6) Fitch, J.P. and Norman, A.C.: Implementing LISP in a High-Level Language, *Software*, Vol. 7, No. 6, pp. 713-725 (1977).
7) Fenichel, R.R. and Yochelson, J.C.: A LISP Garbage-Collector for Virtual-Memory Computer Systems, *CACM*, Vol. 12, No. 11, pp. 611-612 (1969).
8) Morris, F.L.: A Time- and Space-Efficient Garbage Compaction Algorithm, *CACM*, Vol. 21, No. 8, pp. 662-665 (1978).
9) 竹内：第2回 LISP コンテスト，情報処理，Vol. 20, No. 3, pp. 192-199 (1979).
10) Titelman, W.: *INTERLISP REFERENCE MANUAL*, XEROX Palo Alto Research Center, Palo Alto, California (1978).

(昭和57年2月 1 日受付)
(昭和58年3月11日採録)