知的CAIのためのLispシステムの開発

最首和雄*・小沼雅樹**

* 工学部共通講座

** 松村製作所

(昭和60年8月30日受理)

Implementation of Lisp System for Intelligent CAI

Kazuo SAISHU*, Masaki KONUMA**

* *Department of Basic Engineering, Faculty of Engineering.*

** *Matumura Ltd.*

Abstraet

A LISP programming language is implemented in a micro-computer system for the application to the intelligent CAI, because the language makes easy to program of tutorial dialogues and the matching of numerical formulas. The CPU and OS of this system are MC 68000 and CP/M-68K respectively. The LISP implemented here is a subset of Utilisp from the viewpoint of external specification. So the users can make use of the system for general symbolic manipulation. The terminal of this system is PC-9801F/M personal computer. Its cost is low because the I/O routine is almost implemented in the terminal. This paper is written about the features of this system such that its construction, the excution time and the flow for developing application programs.

1. はじめに

数式解の照合や簡単な対話の機能をもった知的CAIの研究に携わってきて,下記の経過からそのためのシステムを開発する必要が生じた。まず文字列処理が中心となるので,言語としてはこのような分野で最もよく使用されているLispがよく,Utilispの使用経験から,それが使い易く,このような応用プログラムに適した言語であると考えてきた。ところでCAIシステムを開発する場合,次の点も考慮する必要があった。

(1)複数の学生を同時に指導することを前提とすると,1台当たりの価格が低いシステムであること。

(2)記号処理以外に,画面制御なども必要となるので,低レベルでの操作機能があること。

(3)ある程度の大きさのプログラムの処理が可能であり,処理速度でも満足できること。

このような点を考慮し,既存のものでこのような条件を満たすシステムがないので,十分な機能を備えており,手軽に利用でき,かつ安価なLispシステムの開発を計画した。

このために,セル数を十分大きく取れ,線形な記憶空間を使える点から,CPUにMC68000を用いたシステムを選び,OSはCP/M-68K[1]とした。またターミナルとしては既存のパーソナルコンピュータを利用することとした。このような条

件を満たすLisp処理系をマイクロコンピュータ上に開発した。今日までのプログラム実験により，このシステムで，所期の目的をある程度達成できたと判断した。ここではその概要を報告する。

このLispインタプリタの特徴としては，

(1) Maclisp系のLispの方言である。
(2) マイコン用Lispとしては高速である。
(3) 機種依存性が少ない。
(4) 非数値処理を必要とする分野一般に応用できる。

などがあげられる。

具体的には，外部仕様はUtilispシステム[2,3]のサブセットとし，文法的には可能な限りそのマニュアルに従った。ただし入出力に関する機能はCP/M-68Kに付属のC言語の実行時ライブラリを利用したため互換性がない。またコンパイラの機能はない。

処理の高速化および簡略化のためUtilispにみられる種々の柔軟な機能（リードマクロ機能，シンボルの登録方法など）は実現できなかったが，使い勝手をよくするために，デバッグ関係の機能や構造エディタなどはほとんど同じに動作するようにした。また高速化のためシステムのほとんどをアセンブリ言語により記述し，実行速度にあまり影響しない入出力などの部分をCで記述した。

現在ターミナルとしてはPC9801F/Mを用いており，そのOSはCP/M-86であり，エディタ等でそのOSのソフトウエアを利用している。

本文の構成は2－5章はLispシステムの概要，6章は性能評価，7章はUtilispとの比較，8章はユーティリティとプログラム実験からなっている。

2．内部データ構造

2.1 データの型

ここではこのLispで使われるデータの型について説明する。これらのデータをLispオブジェクトという。データ型は以下の9種類である。

＊CONS ： リストを構成する2進セルで，2つの要素CARとCDRからなる。これらの要素は任意のLispオブジェクトでよい。

以後CAR，CDRと呼ぶデータは，CONSの型のデータのこれらの要素を意味することとする。

＊整数 ： 24ビットの符号付き整数。
＊浮動小数点数 ： 内部表現56ビット（指数8ビット，仮数部48ビット）の浮動小数点数。
＊文字列 ： 文字の有限列で，各文字は8ビットのASCIIコードと解釈される。
＊シンボル ： シンボルは次の項目からなっている。
　値：シンボルが変数として使われた時の値。
　定義：シンボルが関数の名前と解釈された時の関数定義。
　属性：属性名と属性値の組のリストで，シンボルとあるLispオブジェクトを関連させる。
　印字名：入出力操作でシンボルの名前として使われる文字列。
＊ベクタ ： Lispオブジェクトの有限順序集合。各要素は任位のLispオブジェクトでよい。
＊リファレンス ： ベクタのある要素を指すポインタ。
＊ストリーム ： 入出力に関連したオブジェクト。すべての入出力はストリームに関して行なわれる。
＊コードピース ： 組み込み関数の本体を構成している機械語処理ルーチンへのポインタである。

2.2 データ表現

Lispオブジェクトの内部表現は，データを指すポインタとそのポインタが指すデータの型を示すタグからなっている（図1）。

ポインタは，MC 68000の全記憶空間16Mバイトを利用できるように24ビットとした。1記憶単位を32ビットとすると，残りの8ビットは，セル領域においてはガーベジコレクション時のマークとして1ビット，タグとして7ビット用いた（表1）。

整数に限り，番地部にはポインタではなく整数値そのものが入る。

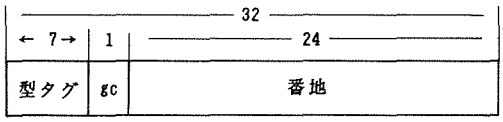

Fig. 1　Tagged pointer

Table 1　Tag values

型タグの値（2進）	型
1 0 0 0 0 0 0	CONSセル
0 1 0 0 0 0 0	整数
0 1 1 0 0 0 0	浮動小数点数
0 0 0 1 0 0 0	文字列
0 0 0 0 1 0 0	シンボル
0 0 0 0 0 1 1	ベクタ
0 0 0 0 0 1 0	リファレンス
0 0 0 0 0 0 1	ストリーム
0 0 0 1 0 0 1	コードピース

2.3　メモリレイアウト

主記憶が512Kバイトの場合のメモリレイアウトを図2に示す。これはメモリの実装量などにより変更される。現在のシステムはメモリが500Kと1Mバイトのものであり，その記憶容量は容易に増設出来るように構成されている。なお本システムにより，十分な性能を得るためには，500Kバイト以上の記憶領域が必要である。

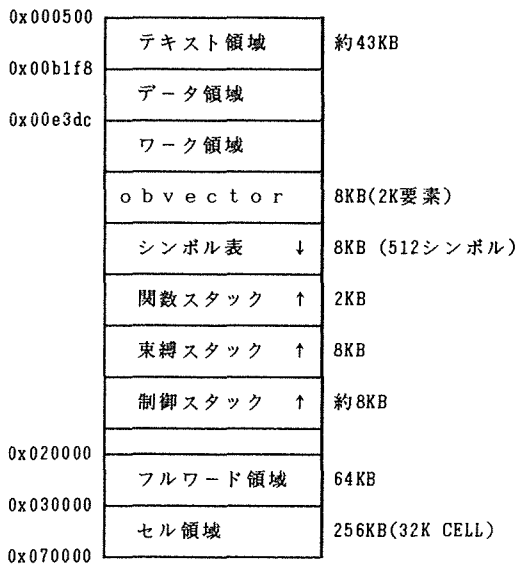

Fig. 2　Memory layout

2.3.1　obvector

obvectorとは，すべての有効なシンボルを登録するハッシュ表である。この表への登録をインターンという。最初から存在するシンボルと普通に読み込まれたシンボルはこの表に登録されている。リーダがシンボルと解釈される文字の列を読み込むと，obvectorに同じ印字名のシンボルが登録されていれば値としてそれを返し，なければ新しいシンボルを作りだし，obvectorに登録して返す。また，関数oblistはこの表から有効なシンボルを拾いだして，リストに組み立てる。

obvectorには，シンボルそのものが登録されるのではなく，シンボルを指すタグつきポインタが入る。

2.3.2　シンボル表

シンボルの実体はシンボル表領域の連続した16バイトに登録される。登録されたシンボルが削除されることはない。システムの起動時はこの表は空であり，最初から登録されているシンボル(389個）とその印字名はデータ領域に存在する。4バイトずつの束縛部，定義部，属性部と，印字名のポインター(3バイト)，関数型（1バイト）からなる。

2.3.3　フルワード領域

文字列の入る領域をフルワード領域としたが，この領域はいわゆるセル構造のフルワードとは異なり，連続な記憶領域として使用している。

ここには文字列とベクタが入るが，これらのオブジェクトは可変長であり，かつ，全ビットが有効であるから，普通のマークスイープ法によるガーベジコレクションは困難である。ガーベジコレクションの際には，フルワード領域に限りコピー型を採用した。このため，実際に使用できる領域は半分である。

2.3.4　スタック

3本のスタックの用途はそれぞれ以下のとおり。

制御スタック　：サブルーチンの戻り番地，通常の関数の引数渡しなどに使用する。
束縛スタック　：変数束縛および，Lispオブジェクトの一時退避用。このスタック上のオブジェクトはガーベジコレクション時にルートとなる。
関数スタック　：Lisp関数の履歴を保存する。

3．評価過程[2,3]

Lispのフォーム（式）の評価過程は以下のとおりである。

フォームがシンボルでもCONSでもなければ，すなわち整数，浮動小数点数，文字列，コードビース，ベクタ，リファレンス，あるいはストリームならば，その評価結果は単にフォームそれ自身である。

フォームがシンボルならば結果はそのシンボルが束縛されている値である。シンボルが未束縛ならばエラーが起きる。

フォームがCONSならば，それは関数かラムダ式の引数への適用である（ここでは「関数」は特殊フォーム，マクロ等を含む広義の関数の意味で使う）。

3.1 関数の型

本システムでは関数を表2のように分類している。

Table 2 Types of function

型	引数の数	引数評価	引数の数の検査	引数渡し
EXPR		有	有	束縛変数
FEXPR	0～	無	無	1変数にフォームのCDRを束縛
LEXPR	0～	有	無	同上
MACRO	0～	無	無	同上
SUBR0	0	-	無	― ― ―
SUBR1	1	有	無	制御スタック
SUBR2	2	有	無	同上
SUBR3	3	有	無	同上
CSUBR1	1	有	有	同上
CSUBR2	2	有	有	同上
CSUBR3	3	有	有	同上
FSUBR1	1	無	無	同上
FSUBR2	2	無	無	同上
FSUBR3	3	無	無	同上
FSUBR	0～	無	無	制御スタック上に引数のリスト
LSUBR	0～	有	無	同上
CLSUBR	1～	有	有	同上
SLSUBR	0～	有	無	束縛スタック
RSUBR1	1	有	有	d0
RSUBR2	2	有	有	d1：第1引数、d2：第2引数

d0、d1、d2：データ・レジスタ

EXPR, FEXPR, LEXPRおよびMACROがそれぞれ，関数defun, df, dl, macroにより定義されるユーザ定義関数で，その他はシステム組み込みである。関数defmacroにより定義されるマクロはMACROの特殊な場合として扱う。

あるシンボルが組み込み関数と定義されているならば，そのシンボルの定義部は，コードピースである。

表2の中で，引数の評価と数の検査の有無は，それぞれの関数型の前処理の段階で行なうかどうかを示しているのであって，各関数の処理ルーチンに制御が渡ってから，引数の評価や数の検査を行なう場合もある。

3.2 関数の評価

ここでは，関数の評価過程について述べる。

評価するフォームがCONSならば，評価ルーチン（以後evalと呼ぶ）はまず，3個のスタックポンインタのオーバフローを検査する。オーバフローが起きていれば，それぞれのエラー処理ルーチンへ飛ぶ。端末からの割込みについては，evalを一定回数実行する毎に検査する。そして，フォームのCARのシンボルの関数型を取り出し，その値に応じた処理アドレスに飛ぶ。このとき，2つのレジスターに評価するフォームのアドレスと関数名であるシンボルのアドレスが入っている。

3.2.1 EXPR

EXPRはユーザ定義の関数で，あるシンボルがEXPRの時その定義は以下のようになっている。

（LAMBDA-LIST. BODY）

BODYは本体ともよばれる。この定義は，関数getdが返す関数の定義から，lambdaを省略した形である。このラムダ式（省略形）をシンボルの定義部が指している。

EXPRの評価はまず，与えられた実引数の評価から始まる。引数を左から右に順に評価し，束縛スタックに積んでいく。引数の終わりはフォームの終わりであるから，リストの終わり，すなわちCDRがアトムであるかどうかで判断する。ここで評価は，関数evalを再帰的に呼び出すことで行なわれる。また評価の結果，すなわち引数の値を束縛スタックに積むのはGC（ガーベジコレクション）によりガーベジとなるのを防ぐためでもある。

次に，関数定義のCAR，すなわちラムダリストを取り出し，引数の値を順にスタックから取り出しラムダリスト中の変数に束縛する。このとき，引数の数がラムダリストの長さより多いとエラーが起きる。少ないときは，ラムダリストの残りの要素が（変数 . 省略値）の形であればその省略値を評価しその変数に束縛する。そうでない場合，すなわち，残りの要素が（変数 . 省略値）の形でないならばエラーが起きる。

束縛が済むと，本体と呼ばれるラムダ式のCDRの要素を左から右に順に評価し，最後の評価の結果がこのEXPRの評価の最終的な値となる。この評価手順をprognで評価すると呼ぶこととする（Lisp関数のprognと同様な手続きという意味で用いる）。続いて，ラムダリスト中の変数を元の値に戻し，EXPRの評価を終わり，evalの呼出し元に戻る。

3.2.2 FEXPR

FEXPRは，不定個の評価されない引数をもつユーザ定義関数である。シンボルの定義部には，EXPRと同様にラムダ式が入っているが，このラムダ式中のラムダリストはただひとつの変数からなるリストであるとみなされる。

まず，ただひとつの仮引数を引数のリスト（評価するフォームのCDR）に，評価せずに束縛する。次に，EXPRと同様にラムダ式のBODY（本体ともいう）をprognにより評価し，変数の仮引数の束縛を元に戻す。このprognが返す値がFEXPRの評価の結果となる。

3.2.3 LFXPR

LEXPRの評価過程はFEXPRとほとんど同じであるが，引数を評価してから束縛する。この評価は引数リストの各要素を評価し，結果のリストを帰す関数evlisによる。

3.2.4 MACRO

MACROの評価過程の前半はFEXPRとほとんど同じである。すなわち仮引数に評価するフォームのCDRを束縛し，BODYをprognにより評価し，束縛を元に戻す。この後，prognが返した結果を引数としてevalを呼び出す。この結果がMACROの評価結果となる。定義のラムダ式のラムダリストの要素が，シンボルではなくてCONSの時，このMACROはdefmacroにより定義されたと解釈し，ラムダリストのCAR中のシンボ

ルを，引数リストの対応する部分に束縛する。束縛の後は普通のマクロ同じである。

3.2.5 SUBRO

SUBROは引数を取らない組み込み関数である。実引数が与えられたとしても無視する。評価過程は単純で，単に定義部のコードピースを呼出すだけである。

3.2.6 SUBR 1～SUBR 3

それぞれ，1，2，3個の評価された引数を取る組み込み関数である。

処理ルーチンは，引数を決まった数だけ右から左に順に評価した後に（引数が足りなければ“0”を，多ければ無視して）制御スタックに逆順にpushして，定義部のコードピースを呼び出す。

3.2.7 CSUBR 1～FSUBR 3

それぞれ，SUBR 1，SUBR 2，SUBR 3と同じであるが引数が足りないときはエラーとする。

3.2.8 FSUBR 1～FSUBR 3

それぞれ，SUBR 1，SUBR 2，SUBR 3と同じだが引数を評価しない。

3.2.9 FSUBR

不定個の評価されない引数を取る組み込み関数である。

引数のリスト（評価するフォームのCDR）を制御スタックにpushして，定義部のコードピースを呼び出す。

3.2.10 LSUBR

不定個の評価された引数を取る組み込み関数である。

引数リストをevlisで評価し，この結果を制御スタックにpushして，定義部のコードピースを呼ぶ。

3.2.11 CLSUBR

LSUBRと同じであるが，少なくとも1個の引数が与えられないとエラーとする。

3.2.12 SLSUBR

LSUBRと同じであるが，引数はリストとしてではなく束縛スタックにpushされて渡される。

3.2.13 RSUBR 1，RSUBR 2

それぞれ，CSUBR 1，CSUBR 2と同じであるが，引数はレジスタにより渡される。

4． 変数束縛

このシステムでは，変数束縛は“浅い束縛”[4]（shallow binding）である。

シンボルはその値に注目されている時は変数と呼ばれる。この変数に対する操作を以下に記す。

変数に対する操作は，変数値の取り出しと変数値の変更に分けて考えられる。変数値の取り出しは，単にその変数の束縛（値ともいう）部を取り出すだけである。この時，その変数が未束縛（値を持っていない）ならば，エラーとする。これは，シンボルの生成時に，未束縛を意味する特定の値を束縛部に書き込んでおくことで実現されている。変数値の変更には，代入とラムダ束縛がある。代入はその変数の束縛部に値を書き込むことであり，以前の値は捨てられる。ラムダ束縛は代入と同様にある値を書き込む操作であるが，以前の値を保存しておき後で復元する。

ある変数Aに値Vをラムダ束縛することを考える。

ブロックの入り口：変数Aの現在の値を取り出し，束縛スタックにpushし，変数Aの束縛部のアドレス，すなわち，後で値が書き戻されるべきアドレスをpushする。そして，新しい値VをAの束縛部に書き込む。

ブロックの出口：束縛スタックから書き戻すべ

きアドレスをpopし，このアドレスに値をpopする。

以上は一変数に注目したものであるが，実際には不定個の変数がまとめてラムダ束縛される。このため，束縛スタック上のどの部分が，変数の古い値とその戻されるアドレスの組なのかを示すために，Lispオブジェクトではない特定の値をpushしている。

5．ガーベジコレクション（GC）[5,6,7]

このシステムでは，ガーベジコレクションの対象となるのは，セル領域とフルワード領域である。セル領域については，普通のマークスイープ法を，フルワード領域については移動法（コピー型）を採用している。

ガーベジコレクタは2つ用意されている。ひとつは，セル領域が足りなくなった時に起動されるもので，これはフルワードについてはなにも行なわない。もうひとつは，フルワード領域が足りなくなった時，すなわち文字列とベクタ用の空間が足りなくなった時，および関数gcが呼ばれた時に起動されるもので，セル領域とフルワード領域の両方のガーベジを回収する。

5.1 印付けフェーズ

ガーベジコレクションのルートとなるのは，シンボルの値，定義，属性，および印字名，束縛スタック上のLispオブジェクト，そして大域脱出のためのCatch-listである。

これらのルートから始めて，それぞれの型に応じた印付けを行なう。このフェーズで，必要ならばフルワード領域のコピーも同時に行なう。

(1)CONS　：　GCビットを1にし，CAR，CDRについて印付けをする。
(2)浮動小数点数：　GCビットを1にする。
(3)文字列：　該当する文字列が既に移動済みであれば，そのアドレスに書き変える。そうでなくて，まだ移動していないならば，新しい空間に文字列をコピーして，コピー先のアドレスを記録する。
(4)ベクタ：　文字列と同様の処理を行なった後，各要素について印付けを行なう。
(5)リファレンス：　ベクタの先頭を見つけてから，ベクタと同様な処理を行ない，必要ならばアドレスを書き変える。

他のオブジェクト，すなわち，シンボル，整数，コードピース，そしてストリームについては何もしない。

5.2 回収フェーズ

フルワード領域についてのガーベジコレクションは既に印付けフェーズで終了しているから，回収の対象となるのはセル領域だけである。

まず，フリーリストをnilにして，セル領域の全体について，印の付いていないセルはフリーリストにLisp関数のpushの意味でのプッシュをして，印の付いているセルは印を消す。

6．高速化技法と性能評価

高速化のため，空間的効率より時間的効率を優先して工夫を施した。これまで述べた点も含め，それらについてまとめておく。

(1)浅い束縛を採用した。[4]
(2)シンボルに関数型を示すタグを付し，関数の型を細分した。
(3)ガーベジの発生を少なくするため，可能なかぎり関数の引き数渡しは，スタックあるいはレジスタによるようにした。
(4)obvectorと呼ばれるハッシュ表を用いた。[2,3]
(5)再帰呼び出しで定義されている関数も，可能なら普通のループに改めた。
(6)型タグは型判定が高速に出来るように選んだ。ただし，MC 68000の命令セットはレジスタの最上位バイトを扱うのに適さないことと，アドレスレジスタの操作でコンデションコードが設定されないなどの問題点がある。

これらの技法で(1)，(4)は一般に良く用いられる技法で，その他はこのインタプリタで行なった工夫である。それらの性能評価のため，文献(8)(9)のLisp 68K（CP/M-68K上で動く）と同じテストプログラムを実行した。プログラムを図3に示す。結果は表3のとおりである(CPUクロック8MHz，

Lisp 68Kは2 wait)。このように本システムは他　　られた。
のシステムと比較しても遜色がないことが確かめ

```
(defun nqueen (m) (queen m nil m) )
(defun queen (n b l)
  (cond ((zerop n) nil)
        ((or (memq n b) (qp l b)) (queen (subl n) b l))
        (t (nconc (cond ((eq (length b) (subl l))
                         (princ "*") (list (cons n b)))
                        (t (queen l (cons n b) l)))
                  (queen (subl n) b l)))))
(defun qp (k m)
  (cond ((null m) nil)
        ((eq (abs (difference n (car m))) k) t)
        (t (qp (addl k) (cdr m)))))
```

Fig.3.1 N-QUEEN[8]

```
(defun tarai (x y z)
  (cond ((greaterp x y)
         (tarai (tarai (subl x) y z)
                (tarai (subl y) x z)
                (tarai (subl z) x y)))
        (t y)))
```

Fig.3.2 TARAI[10]

Table 3 Excution time for the test programs (単位：秒)

	Lisp＊	Lisp68K(8)	Lisp68K(9)
(nqueen 6)	1.6	3.0	
(nqueen 8)	25.5	52.6	39.585
(tarai 8 4 0)	6.4	11.6	8.623
(tarai 10 5 0)	167.6		234.778

Lisp＊　本システム
Lisp68K(8)　文献(8)のシステム
Lisp68K(9)　文献(9)のシステム

Lispシステムの性能は実行速度だけで判断されるべきではない。むしろ，使い勝手のよさとか，全体のわかり易さなどの方が重要であろう。使い易いシステムであるためには，プログラムの入力およびデバッグの効率が十分に高くなければならない。このため，このシステムにも構造エディタやトレーサなどの機能を用意してあり，これらの機能がこのシステムを使い易いものにしている。

7．Utilispとの比較

ここではUtilispと比較し，このシステムにない機能をまとめ，CAIシステムを構成する場合の問題点について述べる。

＊リードマクロと1文字オブジェクトについて

Utilispでは柔軟性を高めるため，入出力の表駆動でマクロテーブル上にマクロ文字を定義することにより，他のシステムからの移行を容易にしている。このシステムはリードテーブルのみによる駆動である。

＊ガーベジコレクションを除く記憶領域管理機能

この機能により，GCの条件を設定，その他記憶領域の管理が柔軟にできる。500Kバイトのシステムでは，GC時間は非常に短く（1回で数秒以内），今後ある程度規模が大きくなっても，GCの条件設定は必要ないと考えており，この機能の必要性も少ない。ただしその他の記憶領域の管理の柔軟性では本システムには十分な機能はない。

＊その他

(1)記号アトムの登録方法の設定

(2)外部プログラムの呼び出し機能およびOSの

利用
(3)コンパイラとそれに関連した機能
(4)バッククォート機能
(5)その他の1部の関数

(1)の機能については応用プログラムに有効に利用する方法は考えてない。(3)については，CAIシステムの完成時に，処理速度で問題が生じた時，その機能が有効になる。その他についてはあれば便利であるという程度であろう。

このように比較すると，Utilispより劣る点もあるが，現在までのCAIシステムの開発状況からそれらの機能の必要性は少ないと感じている。当面はこのシステムで応用プログラムを開発すること，その結果から必要性の高い機能があれば，このシステムを改良することとした。

8．ユーティリティとプログラム実験

8.1 ユーティリティと環境について

このLispシステムにはプリティプリンタ，構造エディタなどの機能を組み込んである。構造エディタは，普通のテキストエディタと違い，List構造そのものを編集するためのエディタである。これらの機能は必ず使用するものであることと，システムの安全性のために，必要なとき読み込んで利用するのでなく，組み込みとした。

CP/M-86またはCP/M-68Kのファイルを直接プリティプリントしたファイルを生成する機能も持たせた。更にLispから直接CP/M-86を呼べるようになっている。エディタとしては，Lispの構造エディタと，CP/M-86上のスクリーンエディタを利用する方法の2種類の利用方法によっている。

またこのシステムにより，知的CAIシステムの構築を一つの具体的目標としているので，このLispでは日本語をも扱えるようになっている。このような応用プログラムの開発では日本語ワードプロセッサーにより作成したファイルをCP/Mファイルに変換して利用している。このような応用プログラム開発時の流れを図4に示す。

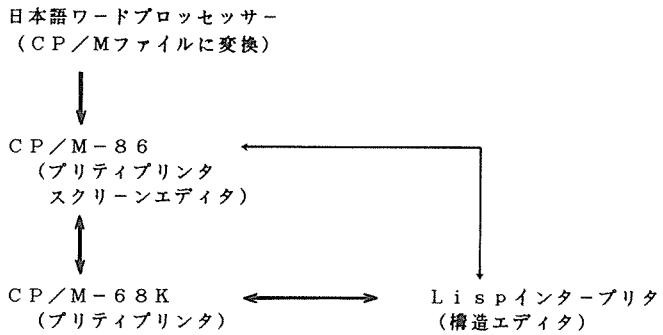

Fig. 4　The flow for developing application programs

8.2 プログラム実験

このシステムのバグを除き，使い易さを確認するために，ある程度のLispプログラムを動作させた。その主なものは文献(12)，(13)の大部分のプログラム，および文献(14)のPortable Prologである。これらの実験から記号処理を中心とした応用プログラムの研究に十分使える見通しを得た。確率統計用CAIへの適用例を図5に示す。これはある事象の確率をそれらの事象からなる積事象の確率の和で表わす式を導く例と，それを質問応答形式で用いた例である。

```
>(pex)
->a ∩~b ∩~c ∩~d
p(a) -p(a∩b) -p(a∩c) -p(a∩d) +p(a∩b∩c)
+p(a∩b∩d) +p(a∩c∩d) -p(a∩b∩c∩d)
->(a ∩~b) ∪~c
1 -p(c) +p(a∩c) -p(a∩b∩c)
->
nil
>(pcai)
q->p(a∪b)
a->p(a) +p(b)
Bad!!!
a->p(a) +p(b) -p(a∩b)
ok
end
nil
>(pcai)
q->p(a∪~b)
a->p(a) -p(b)
Bad!!!
a-> 1 -p(b) +p(a∩b)
ok
end
nil
```

Fig. 5 An Example of Application to CAI System

キーボードからの入力行の意味

（1）->の行：展開すべき確率事象

（2）q->の行：質問の確率式

（3）a->の行：応答の展開した確率式

（4）> の行： Lisp関数の入力

（注）~は否定を表わす。

(pex) ：展開した確率式を求める関数

(pcai) ：確率式について質問応答を行なう関数

9. おわりに

このLispシステムでは，主に知的CAIシステムへの利用を目的として，大きいLispプログラムをマイクロコンピュータ上で高速で，手軽に処理できるようにする，という所期の目標をある程度実現できた。Utilispの大部分の機能があるので，単に特定の目的だけでなく，一般の記号処理問題にも十分使えるといえる。知的CAI用としては，Lispインタプリタと OSを合わせたシステムとしての活用が出来るということも特徴である。

また本システムは最近良く使用されているパーソナルコンピュータをターミナルとして，MC 68000システムをホストとしており，I/O関係の機能はターミナルに任せる構成なので，非常に安価にシステムを構成できるのも特徴である。

今後はこのシステムにより応用プログラムの研究，開発を行なうとともに，ターミナルプログラムを改良して，Lispによるグラフィック処理，テキスト画面制御等の機能を加え，システムの改良に当たる予定である。

ソース／アセンブルリストと外部仕様の詳細について文献(11)を参照してほしい。

CP/M-86, CP/M-68Kは Digital Research社の登録商標である。

MC 68000は Motorola 社の商標である。

Utilispは東京大学で近山隆氏により開発され，大型計算機システムで利用できる言語である。

謝辞 本システムの開発にあたり，プログラム実験やハードウエアの改良に当たってくれた研究室の皆様に感謝する。

参考文献

1） CP/M-68K Operating System Mauual, Digital Research (1983)

2） T.Chikayama : Utilisp Manual, Math. Eng. Tech. Rep, 81－6. Dept. of Meth. Eng, Univ. of Tokyo (1981)

3） 近山：情報処理学会論文誌 **24**, 599.(1983)

4） 黒川：情報処理学会論文誌 **20**, 524(1979)

5） L.P.Deutsch, and D.G.Bobrow: Comm. ACM, **19**, 522 (1976)

6） E.W.Dijkstra, et al.: Lecture Note in Computer Science, vol.46 (Springer-

Verlag, New York 1976) P.43
7) Kung, H.T. and Song, S.W.: Proc.18th Annual Symposium on Foudation of Computer Science, (1977) P.120
8) 小林, 山本, 青木: 情報処理学会記号処理研資料, 29－9, (1894)
9) 山本, 戸島, 村田: bit, 85－3, (1985) P.340
10) 奥乃: 情報処理学会記号処理研資料, 28－4
11) 小沼: 昭和59年度卒業論文
12) P.H.Winston and B.K.P.Horn: LISP (白井, 安部訳) (培風館, 1982)
13) 佐伯監修: LISPで学ぶ認知心理学2, 3巻 (東京大学出版会, 1982, 83)
14) 中島秀行: PROLOG (産業図書, 1983)

付　録

Utilispとの主な相違点

＊ 大文字小文字を区分し, 標準の関数名などは小文字とした。
＊ ストリームを扱う関数の仕様。
＊ 浮動小数点数の精度。Utilispは64ビット, 本システムでは56ビット。
＊ 記憶領域の管理方法。
＊ 他のLispシステムとの互換性のため, 関数型FEXPRとLEXPRを用意した。
＊ Utilispにある以下の機能はない。
- コンパイラおよびコンパラに関連した機能
- リードマクロおよび一文字オブジェクト
- 記号アトムの登録方法の設定
- バッククォート機能
- ガーベジコレクタを除く記憶領域管理機能
- 外部プログラムの呼出しおよびTSSに関する機能
- TIME, DATE-TIMEなどの時間を扱う関数
- ARCSIN, ARCCOS, ARCTAN