ＣＰＭ－６８Ｋ用Ｌｉｓｐ(konoLisp)のＬｉｓｐ言語による開発

角田 透　中村 輝雄
（日立ソフトウェアエンジニアリング㈱）

１．はじめに

konoLispは、1983年から日立ソフトウェアエンジニアリング㈱で開発しているLisp処理系である。まず、プロトタイプLispをVAX/VMS上でC言語を使って開発を行い、MC68000/CPM-68K用に移植も行なった。このプロトタイプLispの開発で発生した問題を解決するため、 MC68000のプロトタイプLisp上にシステム開発用Lispコンパイラを作成し、konoLispをLisp言語を使って開発した。(図1.1参照)

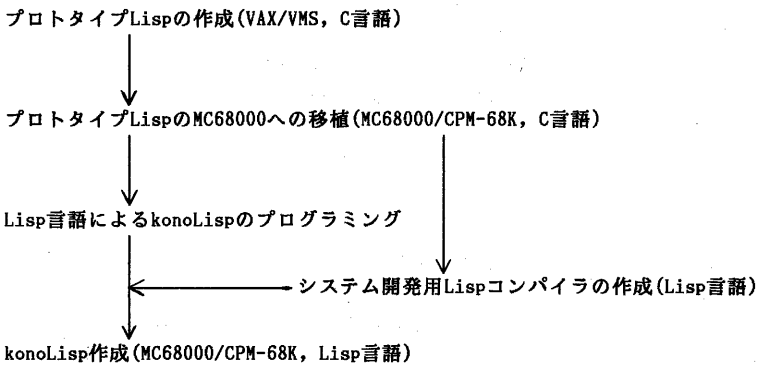

図1.1 konoLispの開発過程

現在のkonoLispの実行環境は次のとおりである。

CPU:	MC68000 (10MHz)
メモリ容量:	1Mバイト
OS:	CPM-68K
システム関数:	約150
セル数:	80Kセル　(1セル = 8バイト)

以下では、konoLisp開発の動機・目的、konoLispの概要、プロトタイプLispの開発において主にC言語を使って発生した問題点、システム開発用Lispコンパイラの概要、Lisp言語によるLisp開発の利点及び問題点について報告する。

２．konoLisp開発の動機・目的

Lispは、現在主に人工知能や知識工学分野のシステム開発、また数式処理のための言語として利用されている。しかし、Lispの特徴である記述性の良さや開発環境の良さは、プログラミング開発におけるツール類の作成やＯＡ用システムの開発にも十分な可能性があると考えられる。ところで、既存のLisp処理系を使ってこのようなシステムを開発した場合、以下の問題点が考えられる。

(1)日本語の使えるシステムを開発する場合に必要となる漢字の扱えるLisp処理系があまりない。

(2)各システムで、それに最も適した実行環境にLisp処理系をチューニングできない。
(3)文字列データの扱いやすいLisp処理系がない。
(4)入出力関数が限られているため、ファイルを使ったシステムの開発が困難である。
(5)マイコン上で実用に耐えるLisp処理系がないため、デスクトップなシステムを開発できない。
これらの問題を解決するため、Lisp処理系を開発することにした。

3．konoLispの概要

3．1　konoLispの特徴

konoLispは、2章であげた問題を解決する目的で開発したMC68000上のLisp処理系である。特徴として、次にあげるものが考えられる。

(1)日本語処理が行えるように、英数字と全く区別なく漢字が扱える。
(2)konoLispの大半がLisp言語で書かれているため、各システムに応じてkonoLispの機能を拡張したり、余分な機能を削除したりすることが容易に行える。
(3)文字列データを文字データのリストとして実現しており、文字列処理がリスト処理と同じように行える。
(4)Lispの不得手な分野(ファイル操作、画像処理 など)を他の言語で開発して、konoLispとリンクをとってシステムを開発できる。
(5)MC68000のリロケータブルな実行ファイルをkonoLispの中から実行できる。
(6)システム開発用とプログラム開発用(現在開発中)の2系統のコンパイラを持つ。システム開発用は、MC68000アセンブリコードを生成するコンパイラで、主にLispシステム自身を用途に合わせてチューニングするために利用するものである。一方、プログラム開発用は、いわゆるLispコンパイラであり、konoLisp上で各種アプリケーションプログラムの開発に利用するものである。

3．2　konoLisp実行時のメモリマップ

konoLispは起動時にメモリマップを決める。これにより、各アプリケーションプログラムに合わせて、スタックサイズ、セル数などを調整し最適な実行環境でプログラムを走らせることができる。標準的なメモリマップを図3.1に示す。

3．3　konoLispのデータ型

konoLispで扱えるデータ型を以下にあげる。

(1)シンボル
(2)文字
(3)文字列　(文字型データのリストと見なせる)
(4)整数　　(無限精度整数は現在開発中)
(5)実数
(6)ファイルポインタ　(C言語のファイルポインタと見なせる)
(7)配列　　(現在開発中)
(8)バイナリ(任意のアドレスをバイナリ型として扱える)

(9)リスト

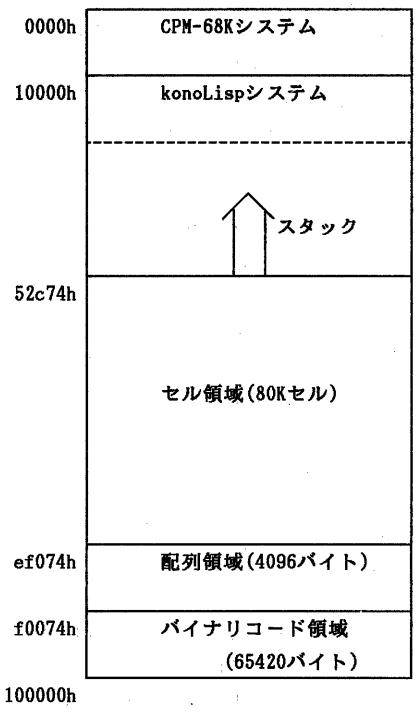

図3.1 konoLispの標準的なメモリマップ(メモリ容量: 1Mバイト)

4．C言語で開発したプロトタイプLispの問題点

プロトタイプLispの問題点は、主にC言語によって開発したことに起因する。

(1)スタックが一本化できない。

ガーベジコレクションのときのセルのマーク付けのために、Lispプログラム実行途中でスタックを参照する必要がある。このとき、正しくセルのマーク付けを行うためには、スタック上にあるマーク付けが行われるセルへのポインタとCプログラムで使われるデータとが区別されなければならない。このため、プロトタイプLispでは、セルへのポインタを積むスタックとCプログラムで使うデータを積むスタックを別にした。 また、バインディングのためのスタックと、catch-throw関数を実現するためのスタックとで、合わせて4つのスタックを使っている。この結果、メモリ効率は非常に悪くなった。

(2)実行速度が上がらない。

C言語では、当然ながらコンディションコードやレジスタを直接参照することができない。このため、アセンブリ言語による開発より実行速度はかなり劣ると考えられる。

(3)ソースプログラムが読みにくい。

C言語はリスト処理用の言語ではないため、たとえばreverse関数の定義を考えてもLispによる定義に比べて読みにくい。また、(1)で述べたようにシステムスタックが使えないために、関数呼出しが冗長になりさらにプログラムを読みにくくしている。(図4.1 参照)

```
x = append(a, b);          stack[pstack++] = a;
                           stack[pstack++] = b;
                           stack[pstack++] = 2;
                           append();
                           x = stack[--pstack];
```

図4.1 プロトタイプLispの関数呼出し

5．システム開発用Lispコンパイラの概要

Lisp言語でkonoLispを開発するために、プロトタイプLisp上でシステム開発用Lispコンパイラを開発した。このLispコンパイラは2パスのコンパイラである。つまり、Lispプログラムをまず仮想マシンのアセンブリコード(LAPコード)に変換して、次にこのLAPコードをMC68000のアセンブリコードに変換する。

reverse1を使ったコンパイル例を図5.1～5.3に示す。

```
(de  reverse1  (x y)
       (cond    ((null x) y)
                (t
                 (reverse1 (cdr x) (cons (car x) y)) ) ) )
```

図5.1 reverse1の関数定義

(注)reverseは (reverse1 x nil)と定義できる。

このシステム開発用LispコンパイラはkonoLispの開発に必要であったが、Lisp言語によって他のシステム(LOGOインタプリタ、Prologインタプリタ など)を開発する場合にも有用である。

6．konoLispをLisp言語で開発した利点

konoLispは、システム起動時に実行される初期値設定ルーチンおよびガーベジコレクタを除き、すべてLisp言語で書いている。このため、ソースプログラムサイズが約4.4K行 (konoLisp本体であるLisp言語による記述は約2.1K行)と小さいため、機能拡張及び変更などの保守が容易に行える。図6.1にkonoLispのモジュール構成を示す。

```
  (lap reverse1 subr)
  (init 'reverse1 '(x y) 'A3 'subr)
  (move 3 'A4)
  (move 2 'A5)
Rreverse1
  (a__nulljumpf 'A4 'g0001)
  (move 'A5 'A0)
  (return)
  (jump 'g0000)
g0001
  (a_cdr 'A4 'A0)
  (push 'A0)
  (a_car 'A4 'A0)
  (a_cons 'A0 'A5 'A0 'A1)
  (push 'A0)
  (pop 'A5)
  (pop 'A4)
  (move 'A6 'sp)
  (jump 'Rreverse1)
g0000
  (return)
  (end 2)
```

図5.2 reverse1のLAPコード

```
.globl reverse1
.globl _Freverse1
.text
reverse1:
cmpi.l #$66000002, D0
beq f0002
move.l _Freverse1, -(sp)
jsr _eargnum
f0002:
link A6, #0
movea.l 12(A6), A4
movea.l 8(A6), A5
Rreverse1:
cmp.l A4, D7
bne g0001
movea.l A5, A0
unlk A6
rts
g0001:
cmpa.l #$00ffffff, A4
bls c0003
move.l _Fcdr, -(sp)
jsr _etype
c0003:
movea.l 4(A4), A0
move.l A0, -(sp)
cmpa.l #$00ffffff, A4
bls c0004
move.l _Fcar, -(sp)
jsr _etype
c0004:
movea.l (A4), A0
cmp.l D7, D5
bne c0005
move.l #$66000131, -(sp)
jsr _garbage
adda.l #4, sp
c0005:
movea.l D5, A3
move.l 4(A3), D5
move.l A0, (A3)
move.l A5, 4(A3)
move.l A3, A0
move.l A0, -(sp)
movea.l (sp)+, A5
movea.l (sp)+, A4
movea.l A6, sp
jmp Rreverse1
g0000:
unlk A6
rts
```

図5.3 reverse1のMC68000アセンブリコード

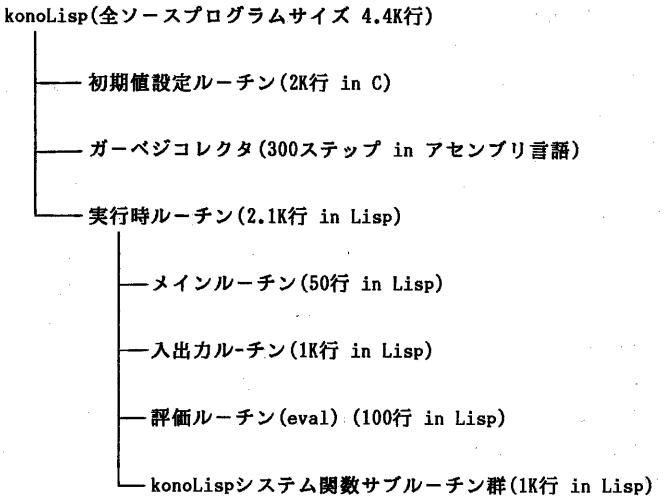

図6.1 konoLispのモジュール構成

実行時ルーチンをすべてLisp言語で書きLispコンパイラを使って開発したため、4章で述べたC言語を使って開発したプロトタイプLispの問題点が解決した。

(1)スタックが一本化できたため、メモリ効率が良くなった。

(2)Lispコンパイラを使ってMC68000を有効に使えるようになった。

(ⅰ)freelist, t, nilへのポインタをデータレジスタD5, D6, D7にそれぞれ割り付けた。

(ⅱ)データレジスタとアドレスレジスタの使い分けができた。

(ⅲ)関数に渡す引数の数をD0に入れることで、効率よく引数の数のチェックができるようになった。

(3)ソースプログラムサイズが約4.4K行と小さくなり、読みやすく保守が容易になった。

(4)LispコンパイラのMC68000コードジェネレータの変更で容易に他のマシンへ移植できると考えている。(現在　8086へ移植中)

7．konoLispをLisp言語で開発したときの問題点

Lisp言語でkonoLispの実行時ルーチンを書くために、特別なLisp関数を導入している。これらの特別なLisp関数は、Lispコンパイラによってインラインに展開される大きさの関数である。(アセンブリコードで1ステップから30ステップ程度) この特別な関数を必要としたケースとして以下にあげる場合があった。

(1)評価ルーチン(eval)でのバインディング機構

konoLispでは、シャローバインディングを使っているが、バインディングの際の仮引数の値をスタックに退避する関数を必要とした。

(2)評価ルーチン(eval)でのLispシステム関数の呼出し機構

(3)catch-throw関数の実現

(4)入出力ルーチンでの周辺機器インターフェイス

入出力ルーチンでの周辺機器インターフェイスは大いにOSに依存する部分である。konoLispでは、CPM-68Kのbios,bdosコールを用いている。このbios,bdosコールを行うためのLisp関数を必要とした。尚、C言語でのopen,close,creatおよびputc,getcに対応する関数は、このbios,bdosコールを行う関数を使ってLisp言語で実現している。

8．ベンチマーク

図8.1にベンチマークで用いたプログラムを、表8.1に測定結果を示す。

```
(de tarai (x y z)
      (cond  ((greaterp x y)
              (tarai (tarai (sub1 x) y z)
                     (tarai (sub1 y) z x)
                     (tarai (sub1 z) x y) ) )
             (t y) ) )
(de srev (x)
      (cond  ((null x) nil)
             (t
              (sapp (srev (cdr x)) (cons (car x) nil)) ) ) )
(de sapp (x y)
      (cond  ((null x) y)
             (t
              (sappend (srev (cdr (srev x)))
                       (cons (car (srev x)) y) ) ) ) )
```

図8.1 tarai および slow-reverse の関数定義

表8.1 tarai および slow-reverse の実行時間

(単位：秒)

	インタプリタ	コンパイラ
(tarai 8 4 0)	13.5	1.7
(tarai 10 5 0)	362.2	48.9
(srev '(1 2 3 4 5 6 7))	12.2	0.7
(srev '(1 2 3 4 5 6 7 8))	48.4	2.4

9．おわりに

Lispコンパイラを使ってLisp言語によるkonoLispの開発を行い、6章で述べたようにソースプログラムサイズが小さく保守性の良いLisp処理系が得られた。 また、アセンブリコードを生成するLispコンパイラを開発したことにより、konoLisp自身を変更することもできるようになった。今後

は、konoLispの機能拡張を含めて、OAなどの優れた対話性を求められる一般事務分野のシステム開発にkonoLispを使い、konoLispの評価を行っていく予定である。

１０．参考文献

1) Allen, J.: Anatomy of LISP: McGRAW-HILL BOOK COMPANY(1978)

2) CP/M-68K Operating System User's Guide: DIGITAL RESEARCH(1983)

3) Kernighan, B. W., Richie, D. M., 石田晴久訳: プログラミング言語C: 共立出版 (1981)