UtiLisp のMC68000 への移植

和田 英一　　富岡 豊　　（東京大学工学部）

はじめに

UtiLisp[1)2)]は1980年の夏から1981年の春にかけ，当時東京大学工学部和田研究室の大学院学生であった近山が東京大学大型計算機センターのM200H 計算機のために開発したLispシステムで，リスト処理のプログラムだけでなく，システムプログラムも記述できるように工夫してある。概ねMacLisp 系であって，文字列，配列，配列要素などのデータも持ち，マクロや読込みマクロの機能も備えていた．このUtiLisp の上に，中島のProlog/KR[3)] をはじめ，いくつもの処理系やツールが作られたが，そのシステム記述への適性のほか，高速な実行性能の故に単に大型計算機センターだけでなく，各地の大学や研究所の計算センターへも貰われてゆき稼働している．

我々の研究室ではその後UtiLisp の保守や改良を行っていないが，1983年の春からモトローラMC68000 を中央処理装置とするSun Workstation が使えるようになったので，その上の作業用語としてUtiLisp を移植したいということになった．幸いM200H もMC68000 も1語32ビットのレジスタ16個を有し，24ビットのバイトアドレス空間を持っているから，全く異るアーキテクチャの計算機に移植するよりは楽であったけれども，命令体系は全然ちがうので移植はやはり一仕事であった．実際の移植作業では1983年の秋から冬にかけ富岡によりインタプリンタがまず移され[4)]，1984年春に富岡がガーベージコレクタを作った．1984年夏から中川がコンパイラを移植中である．Sun Workstation では1983年の夏から多田がUnixの移植作業をしていたが，その作業のほぼ終った時点でUtiLisp はUnixの下で動くようにした．コンパイラ，エディタを含めてSun Workstation への移植が完了したら，次は同じく研究室にあるSord M685 (Unos) へも移植したいと考えている．

以下移植について考えたこと，インタプリンタの移植，ガーベージコレクタ，インタプリタの性能評価の順に述べる．

移植に際して考えたこと．

ひとくちにUtiLisp を移植するというが，UtiLisp とは一体何だろうか．ユーザからみたUtiLisp は一応MacLisp やFranzLisp のごとくである．その特徴はあのIBM360のアーキテクチャに対して徹底的に行ったチューンアップにあるといえる．だとするとMC68000 へあのチューンアップは移植できるであろうか。
MC68000 では残念ながら同じ程度のチューンアップは期待できないことがわかった．しかし我々はできるだけ同じ処理をMC68000 上でも行うことにした．それはマニュアルにも書き切れなかった微妙な処理の順，エラーの発生のし方をできるだけM200H のそれに合わせたかったからだし，また同じ処理方式を移植した方が移植が早く確実にできると考えたからであった．しかし操作システムがちがい，

実記憶空間の大きさもちがうので，変更さぜるを得なかった点も少なくない．

第一に操作システム，入出力に直結している機能，関数は今のところMC68000では最低限のものしか用意していない．Sun Workstation では入出力はウインドウシステム，マウス等を使った新しい感覚のものを設計してつけるのがよいかと思われたからである．

第二にM200H のUtiLisp では入力では大文字小文字を区別せず，出力はデフォールトでは大文字になっている．また内部コードはEBCDICであった．それに対しMC68000 のUtiLisp では内部コードをASCII とし，大文字小文字を区別して扱うことにした．標準の関数名などには小文字を用いる．これはUnixの制御下で動くシステムとしてはFranzLisp もそうだけれども極めて当然の選択であろう．

第三にM200H では仮想記憶空間が広いので，ヒープ領域をふたつとる，いわゆる式年遷宮方式のガーベージコレクションを採用しているけれども，Sun Workstation では記憶装置用の仮想空間が左程広くはないから，式年遷宮方式を採用せず，Morrisの圧縮方式のガーベージコレクションを採用している．

MC68000 では浮動小数点はまだない．

インタプリタの移植

M200H のインタプリタはIBM360系のアセンブリ言語で記述してある．しかもかなりの部分は強力なマクロアセンブリの機能を縦横に駆使して記述してある．例えば基本関数car,cadrなどは図１のような具合いになっている．MC68000 のUtiLisp も68000 のアセンブリ言語で書くことにした．それはやはりUtiLisp というからにはできるだけチューンアップしたいからである．しかしここにふたつの問題があった．第一はM200H とMC68000 でアーキテクチャが多少はちがうことである．M200H は16のレジスタがほとんど対等であるがMC68000 ではアドレス用，データ用がそれぞれ8 ずつで，コンディションコードのセットの状態が異なる．またMC68000 の命令はレジスタの下の方のバイトを扱うには都合がよいが，レジスタの最上位バイトのポインタタグを扱うにはあまり適していない．第二は移植作業に使おうとしたMC68000 アセンブラにマクロの機能がないことである．

最初の問題はM200H のレジスタの使い方を調べ，それをどうMC68000 の上で模倣したらよいかをいろいろ考えて一応の解決をみた．しかしM200H のコーディングは360 系の命令を実に巧妙に利用しているということが改めてよくわかってきた．

第二の問題はマクロ処理系を作って前処理をすることで解決することにした．このマクロは当然のことながらLispで書くことにし，昔のLisp Assembly Program LAP にならってlap68 と名づけた．lap68 はUtiLisp で書いてもよかったが，今回はSun Workstation との仕事のやり易さから大型計算機センターのVax/11-780上のFranzLisp を使うことにした．それは我々のSun Workstaionの作業の大部分がこのVax のUnix上で開発してはダウンロードする方式によっていたからである．従ってMC68000 のUtiLisp の開発はまずlap68 言語でインタプリタ

のプログラムを書き，lap68 でアセンブラ言語に展開する．次にVax 上の MC68000 用クロスアセンブラでオブジェクトコードにし，それをダウンロードするという手順になる．

MC68000 にはいろいろなアドレッシングモードがある．そのモードはアドレス部の書き方による．しかしアセンブラ流のあまりにも奔放な構文規則はS 式としてなじまないので，まずS 式用アドレッシングモードを次のようにきめた．

Rn	→	Rn
an@	→	an@
an@+	→	an@+
an@-	→	an@-
an@(d)	→	(an @ d)
an@(d,Ri:W)	→	(an @ d Ri W)
normal	→	normal
#immediate	→	(quote immediate)

アドレッシングモードがきまれはあとは命令の書き方だけだが，ラベルはアトム記号で書く；命令，擬命令，マクロの呼出しはリストで書く；コメントはLisp のコメントを書くということにした．lap はアトム記号をみれば改行してアトム記号を出力，そのあとにコロンをつけてアセンブラ言語のラベルにする．リストをみればそのcar のアトムがマクロと定義してあるかどうかを見，マクロならそのマクロ定義に従ってマクロ展開する．マクロでなければ普通の機械命令か制御用の疑命令であるので，アドレッシングモードをアセンブラ言語のそれに戻しながら展開する．マクロで展開した命令列にもマクロが含まれている可能性があるのでこれを再びlap68 に通す．Lispのマクロは展開結果をもう一度評価するので (lap 命令列) のようなリストを作り出せばよく非常に簡単であった．例えば，iflistのマクロはlap68 では

```
(dm iflist (an addr)
    (lapt `((movl ,an do)
            (jmi ,addr)))))
```

と書く．dmは結局macro を呼んでiflistのmacro 定義をするが，その前にiflist のp-lit に目印をおいたりする作業も行うマクロである．とにかくこれでiflist はマクロになった．UtiLisp にもFranzLisp にもマクロが用意してありその働きは殆ど一致しているので将来この移植をUtiLisp で行うように変更するのは容易である．また両Lispともバッククォートの機能を有しており，上の例に見るようにマクロの定義は極めて簡単である．

この定義はanに入っているポインタがリストを指すものなら先頭バイトが80になっているので，この内容が負かどうかしらべればよい．しかしanではコンディションコードがたたないのでd0へコピーし，その結果のコンディションコードmiでaddrに分岐するというコードになっている．

(iflist a5 error) というマクロ呼出しがあればこれは

```
(movl a5 d0)
(jmi error)
```

というlap68 命令列になり，ついで

```
movl a5,d0
jmi error
```

というアセンブラ言語の命令列になる．

図２にいくつかのマクロ定義の例，図３にevalの付近のlap68 によるコーディング，図４に同コーディングの展開結果を示す．

M200H のUtiLisp はcsect を使って出力領域を切り換えるがlap68 ではcsect に見合うだけファイルを用意し，csect マクロで出力ストリームを切り換えて出力領域を制御した，lap 終了後，出力ストリームをcat し，a68 でアセンブリする．

ガーベージコレクタ

はじめにも書いたように，MC68000 UtiLisp では式年遷宮方式を採用しなかった．普通のマークアンドスウィープのGCを行う．マークするにはスタックを使って辿るのが一般的だけれども，記憶容量がきついということと，タグ領域に空きビットが充分残っていることから，多少遅くなるかと思われたが逆転ポインタの方式を採用した．（マークビットが一番きびしいのは文字列内部のワードで，ここは仕方がないからAscii コードのb8ビットが0 なのを利用してそこにマークをつけることにした．）一番面倒なのは配列要素を扱っている場合で，この時は配列の先頭を探す．この配列にマークがついていればそれまでだが，マークなしならまずマークし，配列の要素のポインタの先のマークにゆき，最後に飛び込んできた配列要素から戻る．この戻り番地は配列の各要素のポインタの先を次々とマークしているあいだ中，配列の中で順送りに記憶されている．

マークのあとはコンパクションを行う．文字列とか配列とかいろいろなサイズのオブジェクトをヒープ領域内にとるので，コンパクションしないとクラブメンテーションがひどくなるからである。コンパクションにはMorrisとかJonkers とかの方法があるが,MC68000 UtiLispではMorris流を使っている．ただし配列，文字列を先頭から走査してくるときは大きさがわかるけれども，しっぽの方から走査してくると大きさがわからないので，順方向走査のときにゴミとゴミでない部分を分けて，逆に辿れるようリンクを作ってゆく．

実行速度

この原稿を書いている時点では時間計測関係の関数やloop類があまりできていないので,Lisp コンテストの例を正確にはかるのは困難であった．しかしtarai 5 については130 秒程度で結果が得られた．

参考文献

1)近山　隆：" UtiLisp システムの開発" 情報処理学会論文誌 vol.24, No.5 (Sept.1983) pp.599-604

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

3)中島秀之：" 知識表現用語としてのProlog/KR " 情報処理学会論文誌 vol.25, No.2 (Mar.1984) pp.180-186

4)富岡　豊：" プログラム記述用マクロの定義とプログラム言語処理系の移植" 東京大学工学部計数工学科卒業論文(1984 年3 月)

```
CR        C¥R ,
※
CAR       C¥R ,
CDR       C¥R ,
※
CAAR      C¥R ,
CADR      C¥R ,


          MACRO
&NAME     C¥R
&NAME     SUBR  1,1
          L     A,LOCAL1
          LCLA  &L
          LCLC  &N
&N        SETC  '&NAME'
&L        SETA  K'&N-2
          AIF   (&L EQ 0).EXIT
.LOOP     AIF   ('&N'(&L+1,1) EQ 'A').CAR
          CDRA ,
          AGO   .NEXT
.CAR      ANOP
          CARA ,
.NEXT     ANOP
&L        SETA  &L-1
          AIF   (&L NE 0).LOOP
.EXIT     ANOP
          CODEND RET
          MEND
```

図 1

```
(setq *out_port* t)
(setq *main* (outfile 'demo2.a68))
(lapt
      '(        (csect *main*)
                (\.text)
        evandret
                (pea return)
        eval
                (iflist A evrec)
                (cmpl N A)
                (jle evret)
                (valuea)
        evret
                (rts)
        evrec
                (funcent)
        evl
                (cdrcar A)
                (movl A CB)
        evl1
                (iflist A evfn1)
        evfna
                (ifnotsy A evfnnsy)
                (movl (A@ 12) A)
                (cmpl '0xbfffffff A)
                (jls evfna1)
                (jmp A@)
        evfna1
                (comptag A '0x6000)
                (jne evl1)
                (movl A CB)
        evsubr
                (iflist D evsubr3)
                (clrl NA)
                (movl (CB@ 16) WW)
                (jmp WW@)
        evsubr1
                (ifnotsy A evsubr2)
                (valuea)
        evsubr2
                (push A)
                (ifatom D evsubr6)
        evsubr3

            ; [中 略]
```

図 3

```
(dm push (rn)
                (lapt `((movl ,rn sp@-))))
(dm pop (rn)
                (lapt `((movl sp@+ ,rn))))
(dm cdrcar (an)
                (lapt `((movl ,(concat an "@+") D)
                        (movl ,(concat an "@") A))))
(dm valuea ()
                (lapt `((movb A@ DWW)
                        (cmpb '0xc0 DWW)
                        (jeq ubverr)
                        (movl A@ A)))
)
(dm comptag (an tag)
                (lapt `((movl ,an DWW)
                        (swap DWW)
                        (andw '0xff00 DWW)
                        (cmpw ,tag DWW))))
(dm funcent ()
                (lapt '((link a6 '0)
                        (push CB)       ;(movl CB sp@-)
                        )))

(dm iflist (an addr)
                (lapt `((movl ,an DWW)
                        (jmi ,addr))))
(dm ifnotsy (an addr)
                (lapt `((cmpl N ,an)
                        (jlt ,addr))))
(dm ifatom (an addr)
                (lapt `((movl ,an DWW)
                        (jpl ,addr))))


(dm csect (sym)
      (setq *out_port* (eval sym)))
```

図 2

```
        .text
evandret:
        pea     return
eval:
        movl    a5,d0
        jmi     evrec
        cmpl    d7,a5
        jle     evret
        movb    a5@,d0
        cmpb    #0xc0,d0
        jeq     ubverr
        movl    a5@,a5
evret:
        rts
evrec:
        link    a6,#0
        movl    a3,sp@-
evl:
        movl    a5@+,a4
        movl    a5@,a5
        movl    a5,a3
evl1:
        movl    a5,d0
        jmi     evfnl
evfna:
        cmpl    d7,a5
        jlt     evfnnsy
        movl    a5@(12),a5
        cmpl    #0xbfffffff,a5
        jls     evfna1
        jmp     a5@
evfna1:
        movl    a5,d0
        swap    d0
        andw    #0xff00,d0
        cmpw    #0x6000,d0
        jne     evl1
        movl    a5,a3
evsubr:
        movl    a4,d0
        jmi     evsubr3
        clrl    d3
        movl    a3@(16),a0
        jmp     a0@
evsubr1:
        cmpl    d7,a5
        jlt     evsubr2
        movb    a5@,d0
        cmpb    #0xc0,d0
        jeq     ubverr
        movl    a5@,a5
evsubr2:
        movl    a5,sp@-
        movl    a4,d0
        jpl     evsubr6
evsubr3:
```

図 4