22. LISP COMPILERの試作

東北大学 工学部

阿部芳彦・鈴木正幸・桂 重俊

1.はじめに

我々は、数式処理に関心を持ち、数年来この分野における研究を行なってきた。昨年本シンポジウムには「TSSによる数式処理システム`FORMAS'」を報告した。

`FORMAS'の場合は、FORTRANで書かれたシステムであり、独自のリスト処理機能により数式を取扱うが、こうした方式とは別にLispをホスト言語としたいくつかの数式処理システム(REDUCE, MACSYMA等)が開発されている。`FORMAS'はコンパクトなシステムを目指したために前者の方式を採用しているが、今後のシステムの開発、アルゴリズムの研究等に是非Lisp処理系が必要であると考え、本システムを作成した。

我々が最初に作成したLisp処理系は、NEAC 2200-500 TSSで作動するMLISP(Micro Lispの略,インタープリタ)であった。TSSのメモリーサイズが10Kwordであったために容量の面、そして速度の面で満足のいくものではなかった。

速度の向上をはかるために、Lispコンパイラを同じNEAC 2200-500 TSSで作成した。このコンパイラはS式を中間言語にコンパイルし、実行する方式をとった。しかし、システムの柔軟性、オブジェクト領域(コンパイルされた中間言語の格納領域)のくず(誤った関数のコンパイル結果)などの問題が起った。

今回報告するコンパイラは、以前のコンパイラの機能アップとともに、インタープリターコンパイラのシステムとし、前述の問題の解決をはかった。

本システムは、東北大学大型計算機センターACOS 700 TSSにインプリメントされている。

2.方針

本システムは、以下の方針のもとに作成した。

1. TSSを使用し、会話型で処理を行ない、できるだけ柔軟性をもたせ、デバッグ等を容易にする。
2. コンパイル、インタープリタの速度が多少遅くとも、コンパイルされたものの、実行速度をあげる。
3. コンパイラは計算機によらないものを作るために、中間言語(仮想的な計算機の命令)にコンパイルする。コンパイルするプログラムは2進木構造となったものとする。
4. ACOS 700 TSSのメモリサイズが30Kwであるので、コンパクトでなおかつ処理能力の強力なシステムを作成する。

3.コンパイラ

コンパイラは，COMPILEという関数でインタープリタによって呼びだされ，実行されるfsubrである。コンパイラはプログラムを中間言語にコンパイルするもので，仮想計算機によりシミュレートされる。

3.1 中間言語について

中間言語は、インタープリタの実行部分だけを取りだしたものであり，できるだけシンプルな形で表現した。用意したものを以下に示す。

中間言語の種類とその機能.

load 命令

LOADV	(name)	(name) で示される変数値をV-stackのtopにload
LOADC	(pointer)	(pointer) で示される値をV-stackのtopにload (QUOTE)
LOADS	(location)	STOREで値のしまってる変数値をV-stackのtopにload function の中の自由変数の load.

jump命令

JUMP	(location)	(location) へ jump
NILJP	(〃)	結果がNIL ならば(location) へ jump.
TJP	(〃)	結果がNIL以外ならば(location) へ jump
FUNC	(〃)	functionの処理. 定義番地をV-stackへloadして(location) へjump.

call 命令

SUBRi	(name)	組込み関数のcall. iによって引数の数を示す。
EXPRi	(〃)	定義された関数のcall.(push down and jump)
CEXPRi	(〃)	既コンパイル関数のcall.(push down and jump)
LOADF	(location)	funarg のcall (push down and call)

その他の命令

BIND	(引数の数)	lambda 変数のバインド
RTRN		1個の関数定義に対するreturn.
CNTN		no operation
STORE	(name)	(name) で示される変数値をstore.
POP		V-stackにおけるpop up
PUSH		V-stackにおけるpush down
MARK		V-stackに目印を付ける。

3.2 コンパイルについて.

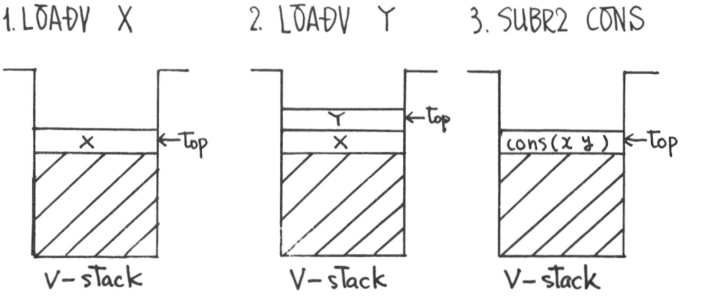
インタープリタは、実行においてプログラムの文脈解析をスタックを使って行ないながら処理をすすめるが、この文脈解析の部分だけを独立に行ない，プログラムを中間言語に変換する事がコンパイルである。このコンパイルを行なう関数をM式で表わせば、ほとんどLispの万能関数と同じように書け，したがってコンパイラは、インタープリタとほぼ同じ大きさのプログラムとなる。

3.3 実行について.

コンパイルされた関数の実行は、コンパイラによって作られた中間言語をデータとして、エクゼキュータ(中間言語の動作がかかれているルーチン)が実行される事で行なわれる。実行の際の変数の処理はV-stack(value stack)で行なわれる。また戻り番地用に別のスタック(push down stack)が使われる。実行の様子を簡単なプログラムで示す。

中間言語例

```
0001   LOADV      X
0002   LOADV      Y
0003   SUBR2   CONS
```



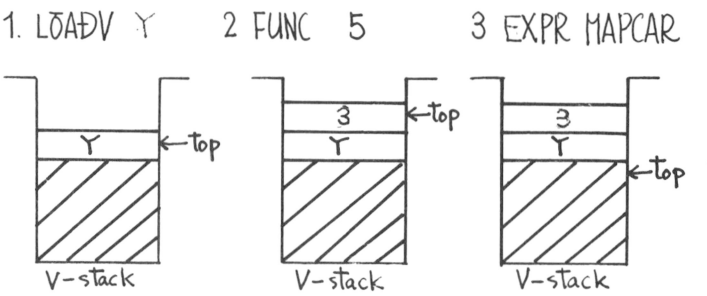
上図の例は(CONS X Y)のコンパイル結果と実行の様子である。

1.2はそれぞれX,YがV-stackのtopにloadされる。

3で、X,YがCONSの引数となり、計算結果がV-stackのtopにloadされる。関数の部分がEXPRの場合には、現在の番地をpush downして定義番地、あるいは関数がコンパイルされていない場合はインタープリタに実行を移す。

中間言語例

```
0001   LOADV      Y
0002   FUNC     0005
0003   SUBR1     CAR
0004   RTRN
0005   EXPR  MAPCAR
```



上図の例は(MAPCAR Y (FUNCTION CAR))のコンパイル結果と実行の様子である。1はYをload, 2はfunctionの引数の定義番地(3)をV-stackにloadし(5)へjump. 3はMAPCARのcallである。Yと定義番地が引渡たされ、MAPCARの中で、この定義番地がLOADFでcallされる。そしてRTRNできたMAPCARへ戻る。

以上、実行について簡単な例で説明したが、他の中間言語も同様にシンプルな働きであり、エクゼキュータはかなりコンパクトに書くことができた。

3.4 コンパイラとインタープリタ

本システムは前にも述べたとうり、インタープリタ・コンパイラの処理系である。コンパイラは普通、関数のパッケージを作るために利用される。そのためインタープリタは、プログラムの作成、デバッグ等を主として行なうようになる。そうい意味で、コンパイル時、インタープリタの実行時には細かなチェックを行なうよう

コンパイラの処理には、大きくわけると次の3点があげられる。

1. SUBR, EXPR の処理（後の引数を評価してから関数が呼ばれる）
2. FSUBR, FEXPRの処理（1.とは引数の評価のしかたが異なる）
3. LABEL, LAMBDAの処理

1の場合が最も基本的で、prefix notationをpostfixに変換する処理である。

```
(CONS (CAR X) Y)
```

```
0001  LOADV    X      } (CAR X)
0002  SUBR1    CAR    }
0003  LOADV    Y
0004  SUBR2    CONS
```

2の場合は、FEXPRは引数をQUOTEして関数に引渡し、FSUBRは展開形式でコンパイルする。

```
(COND ((ATOM X) X)
      (T Y))
```

```
0001  LOADV    X      } P1 (ATOM X)
0002  SUBR1    ATOM   }
0003  NILJP    0006
0004  LOADV    X      } e1 (X)
0005  JUMP     0007   }
0006  LOADV    Y      } e2 (Y)
0007  CNTN
```

3の場合、LAMBDAはバインド命令を作る処理である。
fibonacci の関数のコンパイル例を下図に示す。

```
(LAMBDA (N)
  (COND ((LTP N 2) 1)
        (T (A+ (FB (A- N 1))
               (FB (A- N 2))
        ))))
```

```
0001  BIND     1      } LAMBDA処理
0002           N      }
0003  LOADV    N
0004  LOADC    2
0005  SUBR2    LTP
0006  NILJP    0009
0007  LOADC    1
0008  JUMP     0018
0009  LOADV    N
0010  LOADC    1
0011  SUBR2    A-
0012  EXPR     FB
0013  LOADV    N
0014  LOADC    2
0015  SUBR2    A-
0016  EXPR     FB
0017  SUBR2    A+
0018  RTRN
```

にして、速度の問題はコンパイルされた関数で解決する。また、デバッグ,柔軟性という事で、インタープリタとエクゼキュータはどちらからも呼び合えるようにした。

4.終りに.

最後に、本システムの検討をおこなう。

1)速度.

いろいろな方式のコンパイラが考えられる思うが、我々がコンパイラの作成により、速度の向上がはかれると考えた点を挙げる。

① V-stackを設け、連想リストを作る事を省く。

②連想リストを作らない事でガーベッジコレクションの回数が減る。

③プログラムの実行と文脈解析を分割してしまうので、実行時のpush downが大幅に減る。

④FSUBRは引数の形がプログラムの段階でわかっているので、展開形式で表現する。(FEXPRは別)

以上の4点である。現在、インタープリタの5倍程度の速度が得られている。今後は、現在のバインド方式を改良し、V-stackだけでバインドする方法を考えていて、さらに速度があげられるであろう。また、SUBR, LSUBRを中間言語としてしまうと、インタープリタとのルーチンの共有化が難しいために、SUBRiという形で代表しているが、少なくとも基本的なものについては中間言語として持ったほうがよかった。

2)柔軟性.

インタープリタがプログラムの作成,コンパイラによって速度を上げるという方針で作成したので、プログラム,実行のデバッグ等が統一的にできた。ただ、エディタを持たなかったために,使いにくい点が生じているので、是非作成しようと考えている。

3)コンパイルとコンパイラ

S式をコンパイルする方式では、jump命令が増す、コンパイラの書きにくさ,プログラムが大きくなる、などの欠点があり2進木構造となったものをコンパイルの対象としている。またインタープリタとコンパイラでLispプログラムの支援はいっさいなくしている。Lispのプログラミングは関数定義の積み重ねであるからコンパイラによって関数をパッケージ化していく方法は使いやすいものだろう。

以上が、簡単であるが本システムに対する我々の評価である。こうしたコンパイラは、書きやすく,ゆりとコンパクトであるので,その効用は大きいと思う。
一般に遅いといわれるLisp処理系にはコンパイラが是非必要だと思われる。

いくつかのプログラムの実行時間とコンパイル時間の表を示す。

Program		Interpreter	Compiler	Compile
WANG	A	40 (ms)	10 (ms)	300(程度)
	B	200 (〃)	44 〃	
BITA	5	638 (ms)	102 (ms)	100 (〃)
	6	2441 (〃)	314 (〃)	
	7	7562 (〃)	940 (〃)	
BITB	5	223 (ms)	48 (〃)	100 (〃)
	6	752 (〃)	137 (〃)	
	7	2307 (〃)	427 (〃)	
SORT	20	2420 (ms)	512 (ms)	150 (〃)
	40	7812 (〃)	1232 〃	

GBC時間を含んでいる。(コンパイラはいずれもGBCはおこなわれなかった。)

参考文献

1) J. McCarthy et al : 'Lisp 1.5 Programmer's Mannual" MIT Press (1962)

2) D.G. Bobrow, B. Wegbreit : 'A model and stack implementation of multiple environmens' CACM, vol. 16, no. 10, oct. 1973

3) 後藤英一 : Lisp入門①～⑭, bit, 1975

4) "記号処理シンポジウム報告集", 情報処理学会プログラミングシンポジウム委員会 (1974)