3N－1

バイトコードインタプリタを用いた Lisp コンパイラシステム

萩原知章, 岩井輝男, 中西正和
慶應義塾大学理工学部数理科学科

1. はじめに

世界的に行われている Lisp 言語の標準化の活動の 1 つとして、国際標準機構 ISO によるものがあり、ISLisp[1] [2] と呼ばれている。ISLisp の特徴は、Common Lisp の仕様から使用頻度の低い機能 を取り除いたものである。このため、Common Lisp に比べ処理系の作成が容易である。また、オブジェクト指向機能も兼ね備えている。

本研究では、ISLisp に準拠した Lisp の実装をバイトコードインタプリタにより行なった。この実装は 2 段階に分けられる。

- 第 1 段階（本システム）：コンパイラが Lisp のプログラムを後置記法に直し、中間コードに変換する。そして、このコードに最適化を施し、バイトコードで書かれたファイルに変換する（これ以降この作業をコンパイルという）。
- 第 2 段階：バイトコードインタプリタがバイトコードに変換されたプログラムを読み込み、解読し、スタック機械により実行する。

本稿では、第 1 段階のコンパイラの実装 [3] および、中間コードに最適化を施した際の実行効率について述べる。

2. 本システムの特徴

特徴としては、次の 2 つがあげられる。

- 機械に依存しないコードを生成するため他の機械に移植することが可能である。
- プログラムをコンパイルするため、プログラムの実行を通常のインタプリタよりも速く行なうことが可能である。

3. 本システムの構成

本システムは次の 3 つのフェーズから構成される。

1. 中間コード生成
2. コード最適化
3. コード生成

まず "1. 中間コード生成フェーズ" は、プログラムを M コード（最適化前のコード）に変換する。そして次の"2. コード最適化フェーズ" がこの M コードを P コード（最適化されたコード）に変換する。

この M コード・P コードは、Lisp 言語にほぼ 1 対 1 に対応するように定義した中間コードである。またここでは M コード・P コードを総称して LAP コードと呼ぶ。

最後の"3. バイトコード生成フェーズ" は P コードをバイトコードに変換する。バイトコードは、一般的なプロセッサの機械語と同様に数の列で構成されるものである。

4. 中間コード生成

LAP コードの構文は次の 2 つに分類される。

1. 式の実行
   ((EXECUTE TOP-LEVEL)(lap code)…(lap code))
2. 関数定義
   ((DEFUN 関数名 引数の数 &rest の有無)(lap code)…(lap code))

この構文の第 1 要素は、インタプリタが読み込むバイトコード列が即時実行するものであるか、または関数定義であるかを判断するための指標となる。

中間コード生成の手順は、まず前置記法で記述された Lisp の式を後置記法に変換する。次に、変換された式の要素を 1 つ 1 つ LAP コードに変換していく。

Lisp Compiler

Tomoaki HAGIWARA, Teruo IWAI, Masakazu NAKANISHI
Department of Computer Science, Keio University, 3-14-1 Hiyoshi, Kohoku-ku, Yokohama, Kanagawa Pref., 223, Japan

例えば、(cons 'a nil) を M コードに変換すると

```
(QUOTE A) (QUOTE NIL) (CONS)
```

となる。なお、cons のように頻繁に使われる Lisp の関数を効率化のため、LAP コードに割り当てている。

5. コード最適化

本システムでは、コード最適化の手法として覗き穴式最適化 [4] を用いている。これは M コードを局所的に（Lisp の式ごとに）眺めて、実行時間の短いコード列に置き換えるものである。覗き穴最適化は、そのすべての最適化を適用し、何度も繰り返し行なうことで、効果が得られる。覗き穴式最適化の例を以下に示す。

5.1 冗長な命令の削除

中間コード生成フェーズが Lisp の式を M コードに変換すると、冗長な命令列が出てくる。このような命令列はこの最適化で取り除くことが可能である。

例えば、2 つ連続する (NOT) の命令列は、削除することが可能である。

5.2 到達不能コードの削除

無条件分岐命令のすぐ次にある命令は、そこへ他から分岐してくることがない限り、実行されることはないので削除することが可能である。

5.3 制御の流れの最適化

無条件分岐命令の飛び先が再び無条件分岐命令であるとき、直接その飛び先へ分岐するように変更することが可能である。例を次にあげる。

```
(GOTO L1)        ->      (GOTO L2)
   ・ ・ ・                 ・ ・ ・
(LABEL L1)               (LABEL L1)
(GOTO L2)                (GOTO L2)
```

5.4 機械固有の機能の利用

本システムでは、特定の演算を効率よく実行するために、特別な命令を備えている。

例えば、普通関数呼び出しは、(CALL-GLOBAL label) によって行なわれるが、再帰呼び出しは、(CALL-LOCAL label) に変更することが可能である。後者の方が実行する関数の検索に時間がかからないため、実行速度が速い。

6. 実験結果

ここでは、ベンチマークプログラムをバイトコードに変換し、バイトコードインタプリタにより実行時間を計測した。ベンチマークプログラムとしては、**アッカーマン関数、フィボナッチ数列、tak 関数**を用いた。評価の方法としては M コードと P コードとの実行時間の計測を行ない、両者を比較する。実験結果を図 1 に示す。

	M コード (秒)	P コード (秒)
(ack 3 8)	29.79	18.57
(fib 30)	29.71	27.85
(tak 27 18 9)	146.90	87.16

図 1: 実行時間

以上の実験より最適化を施すことで、実行速度が向上することが確かめられた。

7. 今後の展望

今後は、さらに最適化について研究し、新たな最適化を加えていく予定である。

また、本システムで実装されていないオブジェクト指向の部分を順次実装していく予定である。

参考文献

[1] 伊藤貴康, 湯浅太一, 橋本ユキ子, 梅村恭司, 長坂篤, 岸田克己, ISlisp とその動向, 情報処理学会研究報告 95-SYM-78, 1995.

[2] Programming Language ISLISP Working Draft 11.4, ISO/IEC JTC 1/SC 22/WG 16., July 1994.

[3] 萩原知章. HIL コンパイラの実装, 慶応大学理工学部学士論文, 1995.

[4] Alfred V.Aho, Ravi Sethi, Jeffrey D.Ullman. Compilers Principles, Techniques, and Tools, Addison-Wesley, 1986.