울산대학교 연구논문집 제18권(자연과학·공학편) 제 1 호 pp. 21~24. 1987.
UOU Report Vol. 18. (Natural Science and Engineering) No. 1. pp. 21~24. 1987.
University of Ulsan.

A Study on Lisp Compiler Implementation and Some Extensions to It

Bae, Jae-Hak
Dept. of Computer Science
(Received September 30, 1986)

〈Abstract〉

A Lisp compiler is implemented in muLISP on IBM PC AT. Its original form is Henderson's LISPKIT complier. It is extended to include delayed evaluation primitives, nondeterministic primitives, and the ability to compile elseless IF expressions and three logical expressions.

With nondeterministic primitives we can avoid the explicit programming of backtracking. Not only functional equivalents of the notion of a coroutine but also functional programs which process infinite structures are expressed with delayed evaluation primitives.

리스프 편역기의 구현과 그것의 확장에 관한 연구

배 재 학
전 자 계 산 학 과
(1986. 9. 30 접수)

〈요 약〉

헨더슨의 리스프키트 편역기를 뮤리스프로 IBM PC AT에 구현하였다. 그리고 이 편역기를 확장시켜 비결정적 식과 지연성 평가식을 편역할 수 있게 하였으며 else 부분이 없는 IF식과 3가지의 논리식을 편역할 수 있게 하였다. 비결정적 식을 사용할 수 있게 함으로써 명시적으로 되물림을 표현할 때 생기는 여러가지 부자연스러움을 극복할 수 있게 되었으며 지연성 평가식을 이용하여 병렬처리 과정을 리스프로 표현할 수 있게 되었다. 또한 else 부분이 없는 IF식과 논리식을 편역하는 기능을 추가시킴으로써 보다 간결한 리스프 프로그램을 얻을 수 있게 하였다.

Ⅰ. Introduction

In Henderson's book [1980] he explains the functional style of programming, the structure of functional languages, their application and implementation, and such advanced concepts as delayed evaluation, nondeterministic programs and high-order functions. He notes that with delayed evaluation we can take a functional approach to parallelism. He also touches on that in more complex problems, where the nondeterminism is introduced in many places, the direct formulation in terms of nondeterministic primitives will have a substantial advantage over the explicit programming of backtracking.

Following his implementation description a prototype LISP compiler is implemented in muLISP on IBM PC AT. Its original from is

Henderson's LISPKIT compiler. The LISPKIT compiler is augmented with delayed evaluation primitives, nondeterministic primitives, and the ability to compile elseless IF expressions and 3 logical expressions.

Ⅱ. Lisp Compiler Implementation and the SECD machine

Henderson's LISPKIT compiler translates LISP programs into object programs which run on the SECD machine. This abstract machine was invented by Ladin [1964].

The complete set of instructions for the SECD machine with their mnemonics is as follows.

LD	load
LDC	load constant
LDF	load function
AP	apply function
RTN	return
DUM	create dummy environment
RAP	recursive apply
SEL	select subcontrol
JOIN	rejoin main control
CAR	take car of item on top of stack
CDR	take cdr of item on top of stack
ATOM	apply atom predicate to top stack item
CONS	form cons of top two stack items
EQ	apply eq predicate to top two stack items
ADD SUB MUL DIV REM	apply arithmetic operation to top two stack items
LEQ	apply "less than or equal to" predicate to top two stack items
STOP	stop

Henderson describes the object code which is generated for each well-formed expression as follows. If *e* is a well-formed expression and *n* is a namelist, then it is denoted by e^*n the code which is generated when *e* is compiled with respect to the name list *n*.

x*n: (LD i) where i=location(x, n)
(QUOTE s)*n: (LDC s)
(ADD e1 e2): e1*n/e2*n/(ADD)
where x/y means append(x, y)
(SUB e1 e2)*n: e1*n/e2*n/(SUB)
(MUL e1 e2)*n: e1*n/e2*n/(MUL)
(DIV e1 e2)*n: e1*n/e2*n/(DIV)
(REM e1 e2)*n: e1*n/e2*n/(REM)
(EQ e1 e2)*n: e1*n/e2*n/(EQ)
(LEQ e1 e2)*n: e1*n/e2*n/(LEQ)
(CAR e)*n: e*n/(CAR)
(CDR e)*n: e*n/(CDR)
(CONS e1 e2)*n: e2*n/e1*n/(CONS)
(ATOM e)*n: e*n/(ATOM)
(IF e1 e2 e3)*n: e1*n/(SEL e2*n/(JOIN) e3*n/(JOIN))
(LAMBDA (x1…xk) e)*n: (LDF e*((x1…xk).n)/(RTN))
(e e1…ek)*n: (LDC NIL)/ek*n/(CONS)/…/ e1*n/(CONS)/e*n/(AP)
(LET e(x1, e1) …(xk.ek))*n: (LDC NIL)/ek*n/(CONS)/ …/e1*n/(CONS)/ (LDF e*m/(RTN) AP) where m=((x1… xk).n)
(LETREC e (x1, e1) · (xk.ek))*n: (DUM LDC NIL)/ek*m/(CONS)/ /e1*m/ (CONS)/(LDF e*m/(RTN) RAP) where m ((x1 xk).n)

Ⅲ. Implementation Technique of the Extensions

1. Delayed Evaluation Primitives

To give the LISPKIT compiler the ability to compile delayed evaluation primitives we need 3 new machine instructions. They are LDE(load expression), UPD(return and update), and APO (apply parameterless function). Their machine codes are 22, 23, and 24 respectively.

The well-formed expressions (DELAY *e*) and (FORCE *e*) are complied as follows.

(DELAY e)*n: (LDE e*n/(UPD))
(FORCE e)*n e*n/(APO)

In the augmented LISPKIT compiler this compilation is represented as shown below.

```
(IF(EQ(CAR E)'DELAY)
  (CONS'22(CONS(COMP(CAR(CDR E))N'(23))
              C
          )
  )
(IF(EQ(CAR E)'FORCE)
  (COMP(CAR(CDR E))N(CONS '24 C))
```

2. Nondeterministic Primitives

To compile nondeterministic primitives it is required for the LISPKIT compiler to include 2 new machine instructions. SOR and NON are they. Their machine codes are 25 and 26 respectively.

The well-formed expressions (NONDET *e*1 *e*2) and (NONE) are compiled as written below.

```
(NONDET e1 e2)*n: (SOR e1*n/(JOIN) e2*n/
                      (JOIN))
(NONE)*n: (NON)
```

This compilation is represented in the augmented LISPKIT compiler as follows.

```
(IF(EQ(CAR E)'NONDET)
  (LET((EXPR1(COMP(CAR(CDRE))N'(9)))
       (EXPR2(COMP(CAR(CDR(CDR E)))
         N'(9)))
      )
    (CONS'25(CONS EXPR1(CONS EXPR2
      C)))
  )
(IF(EQ(CAR E)'NONE)
  (CONS'26 C)
```

3. Logical Expressions and Elseless IF Expressions

We adopt a textual transformation technique to compile logical expressions and elseless IF expressions. After transformation the compiler compiles the transfermed expression. The transformation technique is described as follows.

```
(AND e1 e2) : (IF(EQ e1(QUOTE T))
                 (IF(EQ e2(QUOTE T))
                    (QUOTE T)
                    (QUOTE F))
                 (QUOTE F)
              )
(OR e1 e2) : (IF(EQ e1(QUOTE T))
                (QUOTE T)
                (IF(EQ e2(QUOTE T))
                   (QUOTE T)
                   (QUOTE F)
                )
             )
(NOT e) : (IF(EQ e(QUOTE T))
             (QUOTE F)
             (QUOTE T)
          )
(IF e1 e2) : (IF e1 e2(QUOTE NIL))
```

Ⅳ. Conclusion and Further Research Topics

Our LISP compiler is written in LISP, so that it can be extended to include various primitives which are required in some application area. Viewed in this light our approach acts as a guide to include application-dependent primitives in a LISP compiler.

There remain many interesting research topics for us to study further. Some of them are listed below.

a. To include primitives for supporting high-order function definitions and evaluation
b. To make a simulator of a LISP machine in LISP
c. To generate target codes in assembly language of a real machine
d. To generate optimized codes
e. To include Error Recovery ability
f. To include I/O primitives

References

[1] Henderson, P. (1980), Functional Programming Application and Implementation, Prentice-Hall International, Inc., London.

[2] Ladin, P.J. (1963), The Mechanical Evaluation of Expressions, Computer Journal, 6(4), 308–20.