Zbornik radova Prirodno-matematičkog fakulteta-Univerzitet u Novom Sadu
knjiga 13 (1983)
Review of Research Faculty of Science-University of Novi Sad, Volume 13 (1983)

AN EXTENSION OF THE LISPKIT LISP-LANGUAGE VERSION ARL BY THE GENERALIZED FUNCTIONS OF SOME PRIMITIVE FUNCTIONS AND THEIR IMPLEMENTATION

Ljubomir Jerinić, Vojislav Stojković
Prirodno-matematički fakultet. Institut za matematiku
21000 Novi Sad, ul. dr Ilije Djuričića br. 4, Jugoslavija

ABSTRACT

The paper presents the implementation of an extension of the LISPKIT LISP-language version ARL by the following primitive generalized functions of the standard LISP-language:

(PLUS i1 ... ik) k>=1
(TIMES i1 ... ik)
(OR ℓ1 ... ℓk)
(AND ℓ1 ... ℓk)
(COND (ℓ1 e1) ... (ℓk ek)

i1,...,ik are integer well-formed expressions,
ℓ1,...,ℓk are logical well-formed expressions,
e1,...,ek are arbitrary well-formed expressions,

---

* *The same paper, but under the title:*
"An extension of LISPKIT LISP-language and its implementation" was reported at the Seventh Congress of Balkan Mathematicians in Athens, 1983.

AMS Mathematics subject classification (1980): 68A05
Key words and phrases: LISP-language, primitive generalized functions, well-formed expressions.

and their degenerated cases:

(PLUS)
(TIMES)
(OR)
(AND)
(COND)

in an indirect and direct way.

The implementation was performed in the LISPKIT LISP--language version ARL.

INTRODUCTION

P. Henderson [1] has described LISPKIT LISP-language and the principles of design of the LISPKIT LISP-system.

The Group for Functional Programming of the Institute of Mathematics, Faculty of Science University of Novi Sad [2], [3], [4], [5], [6] using the results of P. Henderson, and at the same time giving a number of its own solutions has constructed a LISPKIT LISP-system in FORTRAN-language and installed it in a DELTA 11/340 Computer System.

During work on solving various problems of symbolic data processing, for example:

- Symbolic differation of functions,
- Simulating of finite-state and pushdown automata,
- Match-functions etc.,

the need for the introduction of:

A) new primitive:
- Arithmetic functions: EXP
- Relational functions: LT, NE, GE, GT,

B) The generalized functions of the following primitive functions: ADD, MUL, DIS, CON, IF.

is appeared.

THE GENERALIZED FUNCTIONS

Let us generalize the following primitive functions of the LISPKIT LISP-language version ARL:

(ADD i1 i2)
(MUL i1 i2)
(DIS ℓ1 ℓ2)
(CON ℓ1 ℓ2
(IF ℓ e1 e2)

The generalization consists of taking an arbitrary number of arguments of function, k>=0.

For k = 0:
(ADD)
(MUL)
(DIS)
(CON)
(IIFF)

For k>= 1:

(ADD i1 ... ik)
(MUL i1 ... ik)
(DIS ℓ1 ... ℓk)
(CON ℓ1 ... ℓk)
(IIFF (ℓ1 e1) ... (ℓ1 ek))

For k = 0 the presented generalized functions are the following constants:

- 0
- 1
- F
- T
- Ω

Ω is an arbitable, usually NIL value. Ω serves for reserving the memory space for a pointer.

In the case of IF-function the meaning of the argument

was changed. Because of that, the new name of the function: IIFF was introduced.

The given generalized functions are not the functions in the usual mathematical sense. In mathematics it is usual that a function has a fixed number of arguments.

For k>= 1, the given generalized functions are sequentially identical with the following primitive generalized functions of the standard LISP-language:

(PLUS i1 ... ik)
(TIMES i1 ... ik)
(OR ℓ1 ... ℓk)
(AND ℓ1 ... ℓk)
(COND (ℓ1 e1) ... (ℓk ek))

The presented primitive generalized functions of the standard LISP-language can be defined in the following non-formal way by means of corresponding primitive functions of the LISPKIT LISP-language version ARL:

(PLUS i1 ... ik) = (ADD i1 (... (ADD ik (QUOTE 0)) ...))
(TIMES i1 ... ik) = (MUL i1 (... (MUL ik (QUOTE 1)) ...))
(OR ℓ1 ... ℓk) = (DIS ℓ1 (... (DIS ℓk (QUOTE F)) ...))
(AND ℓ1 ... ℓk) = (CON ℓ1 (... (CON ℓk (QUOTE T)) ...))
(COND (ℓ1 e1) ... (ℓk ek)) = (IF ℓ1 e1 (... (IF ℓk ek) ...))

THE IMPLEMENTATION OF GENERALIZED FUNCTIONS

The generalized functions cannot be implemented as the user functions. The definitional expression:

(LAMBDA(x1 ... xk)e)

of the LISPKIT LISP-language requires, known beforehand, the fixed number of arguments of functions which is to be defined.

The generalized functions can be implemented exclusively as the primitive functions, that is, they must be built-in into the system. Each building-in of functions into the system

requires the modification of the system.

The implementation of the generalized functions requires only the modification of the translator, that is, the part of the LISPKIT LISP-system which executes the translation of the program from the LISPKIT LISP-language in the machine language of the SECD-machine. The modification of the program simulator of the SECD-machine is not necessary.

There are two methods of implementation of the generalized functions:

- indirect and
- direct.

In the case of indirect implementation the generalized function is translated in the composition of the corresponding primitive function and this composition is further translated into machine language of the SECD-machine by means of the same translator.

In the case of direct implementation, the generalized function is translated straight into the machine language of the SECD-machine.

The indirect implementation does not require a knowledge of:

- the machine language of the SECD-machine and
- the code of translation.

Indirect implementation is:

- more simple and
- slower than

direct implementation.

The indirect translation of the generalized functions can also be realized in the form of the special pre-translator.

THE IMPLEMENTATION OF PLUS-FUNCTION

The part of the translator which serves for translation of the ADD-function is:

```
(1)...  (IF (EQ (CAR E)
                (QUOTE ADD)
            )
            (COMP (CAR (CDR E))
                  N
                  (COMP (CAR (CDR (CDR E)))
                        N
                        (CONS (QUOTE ADD)
                              C
                        )
                  )
             ) ...
```

A) Indirect implementation

The degenerated PLUS-function is translated in:

(PLUS) ⟶ (QUOTE 0)

The non-degenerated PLUS-function is translated in the composition of ADD-functions:

(PLUS i1 ... ik) → (ADD i1(... (ADD ik(QUOTE 0))...))

The obtained translations are further translated in the machine language of the SECD-machine by means of the same translator.

The translator is extended with:

```
(2)...  (IF (EQ (CAR E)
                (QUOTE PLUS)
            )
            (COMP (PLUSFUN (CDR E))
                  N
                  C
            )...
```

where is:

```
(3)...  (PLUSFUN LAMBDA (E)
          (IF (EQ E
                  (QUOTE NIL)
              )
              (QUOTE (QUOTE 0))
              (SPOJ3 (QUOTE ADD)
                     (CAR E)
                     (PLUSFUN (CDR E))
              )
          )
        )...

(4)...  (SPOJ3 LAMBDA (A B C)
          (CONS A
                (CONS B
                      (CONS C
                            (QUOTE NIL)
                      )
                )
          )
        )...
```

It is necessary that the translator still contains (1).

B) Direct implementation

The degenerated PLUS-functions is translated in:

(PLUS) → (LDC 0.c)

The non-degenerated PLUS-function is translated in:

(PLUS i1 ... ik) → (LD i1 ... LD ik LOC O ADD ... ADD.c)

The translator is extended with:

```
(2')... (IF (EQ (CAR E)
                (QUOTE PLUS)
                )
                (PLUSFUN (CDR E)
                         N
                         C
                )...
```

where is:

```
(3')...  (PLUSFUN LAMBDA (E N C)
           (IF (EQ E
                   (QUOTE NIL)
               )
               (CONS (QUOTE LOC)
                     (CONS (QUOTE 0)
                           C
                     )
               )
               (COMP (CAR E)
                     N
                     (PLUSFUN (CDR E)
                              N
                              (CONS (QUOTE ADD)
                                    C))) ))...
```

The translator can, but need not, contain (1), depending on whether one wishes or not to keep ADD-function.

The implementation of TIMES, OR and AND function.

The given functions are completely implemented in a similar way as the PLUS-function.

The analogous expressions of (1), (2), (3), (2'), (3') are the following:

```
(1")... (IF (EQ (CAR E)
                (QUOTE MUL)
            )
            (COMP (CAR (CDR E))
                  N
                  (COMP (CAR (CDR (CDR E)))
                        N
                        (CONS (QUOTE MUL)
                              C
                        )
                  )
            )
```

```
      (IF (EQ (CAR E)
              (QUOTE DIS)
          )
          (COMP (CAR (CDR E))
                N
                (COMP (CAR (CDR (CDR E)))
                      N
                      (CONS (QUOTE DIS)
                            C
                      )
                )
          )
      (IF (EQ (CAR E)
              (QUOTE CON)
          )
          (COMP (CAR (CDR E))
                N
                (COMP (CAR (CDR (CDR E)))
                      N
                      (CONS (QUOTE CON)
                            C
                      )
                )
          )...
(2")... (IF (EQ (CAR E)(QUOTE TIMES))
            (COMP (TIMESFUN (CDR E)) N C)
        (IF (EQ (CAR E)(QUOTE OR))
            (COMP (ORFUN (CDR E)) N C)
        (IF (EQ (CAR E)(QUOTE AND))
            (COMP (ANDFUN (CDR E) N C)...
(3")... (TIMESFUN LAMBDA (E)
          (IF (EQ E (QUOTE NIL))
              (QUOTE (QUOTE 1))
              (SPOJ3 (QUOTE MUL)(CAR E)(TIMESFUN (CDR E))) ))
          (ORFUN LAMBDA (E)
            (IF (EQ E (QUOTE NIL))
```

```
                (QUOTE (QUOTE F))
                (SPOJ3 (QUOTE OR)(CAR E)(ORFUN (CDR E))) ))
          (ANDFUN LAMBDA (E)
            (IF (EQ E (QUOTE NIL))
                (QUOTE (QUOTE T))
                (SPOJ3 (QUOTE AND)(CAR E)(ANDFUN (CDR E))) ))...

(2'")...  (IF (EQ (CAR E)(QUOTE TIMES))
              (TIMESFUN (CDR E) N C)
          (IF (EQ (CAR E)(QUOTE OR))
              (ORFUN (CDR E) N C)
          (IF (EQ (CAR E)(QUOTE AND))
              (ANDFUN (CDR E) N C)...

(3'")...  (TIMESFUN LAMBDA (E N C)
            (IF (EQ E (QUOTE NIL))
                (CONS (QUOTE LDC)(CONS (QUOTE 1) C ))
                (COMP (CAR E) N (TIMESFUN (CDR E)
                                          N
                                          (CONS (QUOTE MUL)
                                                C
                                          )
                                )
                )
            )
          )
          (ORFUN LAMBDA (E N C)
            (IF (EQ E (QUOTE NIL))
                (CONS (QUOTE LDC)(CONS (QUOTE F) C ))
                (COMP (CAR E) N (ORFUN (CDR E)
                                       N
                                       (CONS (QUOTE OR)
                                             C
                                       )
                                )
                )
             )
           )
```

```
(ANDFUN LAMBDA (E N C)
  (IF (EQ E (QUOTE NIL))
      (CONS (QUOTE LDC)(CONS (QUOTE T) C ))
      (COMP (CAR E) N (ANDFUN (CDR E)
                              N
                              (CONS (QUOTE AND)
                                    C
                              )
                      )
      )
  )...
```

Implementation of COND-function

The part of the translator which serves for the translation of the IF-function is:

```
(1)...   (IF (EQ (CAR E)
             (QUOTE IF)
             )
             (COMP (CAR (CDR E))
                   N
                   (CONS (QUOTE SEL)
                         (CONS (COMP (CAR (CDR (CDR E)))
                                     N
                                     (QUOTE (JOIN))
                                     )
                                     (CONS (COMP (CAR (CDR (CDR (CDR E))))
                                                 N
                                                 (QUOTE (JOIN))
                                           )
                                           C
                                     )
                         )
                   )
             )...
```

A) Indirect implementation

The degenerated COND-function is translated in:

(COND) → (QUOTE Ω)

The non-degenerated COND-function is translated in the composition of the IF-function:

(COND (ℓ1 e1) ...(ℓk ek)) → (IF ℓ1 e1(...(IF ℓk ek Ω) ...))

The obtained translations are further translated in the machine language of the SECD-machine by means the same translator. The translator is extended with:

```
(2)...  (IF (EQ (CAR E)
                (QUOTE COND)
            )
            (COMP (CONDFUN (CDR E))
                  N
                  C
            )...
```

where is:

```
(3)...  (CONDFUN LAMBDA (E)
          (IF (EQ E
                  (QUOTE NIL)
              )
              (QUOTE (QUOTE Ω))
              (SPOJ4 (QUOTE IF))
                     (CAR (CAR E))
                     (CAR (CAR (CDR E)))
                     (CONDFUN (CDR E))
              )
          )
        )...
(4)...  (SPOJ4 LAMBDA (A B C D)
           (CONS A
                 (CONS B
                       (CONS C
                             (CONS D
```

```
                                    (QUOTE NIL)
                                )
                            )
                        )
                    )...
```

It is necessary that the translator still contains (1).

B) Direct implementation

The generated COND-function is translated in:

(COND) → (LDC Ω.c)

The non-degenerated COND-function is translated in:

(COND (ℓ1 e1) ...(ℓk ek)) →

(LD ℓ1 SEL (LD e1 JOIN)(...(LD ℓk SEL(LD ek JOIN)(LDC Ω JOIN)JOIN)...))

The translator is extended with:

```
(2')    (IF (EQ (CAR E)
                (QUOTE COND)
            )
            (CONDFUN (CDR E)
                     N
                     C
            )...
```

where is:

```
(3')... (CONDFUN LAMBDA (E N C)
           (IF (EQ E
                   (QUOTE NIL)
               (CONS (QUOTE LDC)
                     (CONS (QUOTE Ω)
                           C
                     )
               )
            (IF (EQ (CDR E)
                    (QUOTE NIL)
                )
```

```
(COMP (CAR (CDR (CAR E)))
      N
      C
)
(COMP (CAR (CAR E))
      N
      (CONS (QUOTE SEL)
            (CONS (COMP (CAR (CDR (CAR E)))
                        N
                        (QUOTE (JOIN))
                  )
                  (CONS (CONDFUN (CDR E)
                                 N
                                 (QUOTE (JOIN))
                        )
                        C
                  )
            )
      )
)
)...
```

REFERENCES

[1] P. Henderson, *Functional programming, Prentice Hall, 1980.*

[2] V. Stojković i dr., *Implementacija NP-tehnike obrade S-izraza u FORTRAN-jeziku, 5.Medjunarodni simpozij "Kompjuter na Sveučilištu", Cavtat, 1983.*

[3] V. Stojković i dr., *Dinamičko upravljanje memorijom sa gledišta korišćenja i implementacije programskih jezika, 5.Medjunarodni simpozij "Kompjuter na Sveučilištu", Cavtat, 1983.*

[4] V. Stojković i dr., *Programska implementacija simulatora SECD-mašine, XXVII Jugoslovenska Konferencija ETAN-a, Struga, 1983.*

[5] I. Stojmenović i dr., *O implementaciji prevodioca LISPKIT LISP-jezika na jezik SECD-mašine izvršenoj na FORTRAN-jeziku, Informatica, 1984.*

[6] V. Stojković i dr., *LISPKIT LISP-jezik verzija ARL, Bilten SAP Vojvodine za nauku i informatiku, 1984.*

[7] J.Mc Carthy, *A basis for a mathematical theory of computation, studies in logic, Computer programming and formal systems, North-Holland, 1967.*

Received by the editors December 10, 1983.

REZIME

O PROŠIRENJU LISPKIT LISP-JEZIKA VERZIJE ARL POMOĆU UOPŠTENIH FUNKCIJA NEKIH PRIMITIVNIH FUNKCIJA I NJIHOVA IMPLEMENTACIJA

U radu je izvršeno uopštavanje sledećih primitivnih funkcija LISPKIT LISP-jezika verzije ARL:

(ADD i1 i2)
(MUL i1 i2)
(DIS ℓ1 ℓ2)
(CON ℓ1 ℓ2)
(IF ℓ e1 e2)

-i1,i2 su celobrojni valjani izrazi,
-ℓ,ℓ1,ℓ2 su logički valjani izrazi i
-e1,e2 su proizvoljni valjani izrazi.

Uopštavanje se sastoji u tome da se dozvoli proizvoljan broj k,k>=0, argumenta funkcije:

za k=0:	za k>=1
(ADD)	(ADD i1 ... ik)
(MUL)	(MUL i1 ... ik)
(DIS)	(DIS ℓ1 ... ℓk)
(CON)	(CON ℓ1 ... ℓk)
(IIFF)	(IIFF (ℓ1 e1) ... (ℓk...ek))

-i1,...,ik su celobrojni valjani izrazi
-ℓ1,...,ℓk su logički valjani izrazi i
-e1,...,ek su proizvoljni valjani izrazi.

Za k>=1, navedene uopštene funkcije su redom identične sa sledećim primitivnim uopštenim funkcijama standardnog LISP-jezika:

(PLUS i1 ... ik)
(TIMES i1 ... ik)
(OR ℓ1 ... ℓk)
(AND ℓ1 ... ℓk)
(COND (ℓ1 e1) ... (ℓk ek)).

Navedene primitivne uopštene funkcije standardnog LISP-jezika, zajedno sa njihovim degenerisanim slučajem (PLUS), (TIMES), (OR), (AND) i (COND), implementirane su na indirektan i direktan način na LISPKIT LISP-jeziku verzija ARL.