

(USEFUL FUNCTIONS

A DEFINITION OF THE INVERSE QUOTE FUNCTION : @

Joachim LAUBSCH

Institut für Informatik
Universität Stuttgart
Azenbergstr. 12
7000 Stuttgart 1

It frequently happens that a LISP programmer wants a function to produce a data-structure or function containing constant and variable substructures. The usual solution is to program a form containing a lot of data-structure composing functions (like LIST, CONS and APPEND). The resulting expression is hard to decipher for humans unless more mnemonic constructor-functions are defined. A simple way out is to write the resulting structure with its variable substructures especially marked.

For example, return a lambda-expression which will evaluate any form in an environment where X and Y are bound to successive elements of L, and F receives the result :

```
(list 'lambda '(form)
      (append (list (list 'lambda '(X Y)
                          (list F '(eval form)))
                L)))
```

Compare this with the inverse quote version :

```
@(lambda (form) ; @ is inverse quote macro
  ((lambda (X Y)
    (=F (eval form))) ; = means eval the following
    ; element
    ,L)) ; , is as = but uses all
        ; elements of the value as
        ; elements in the current list
```

As a second example, consider how WOODS could have returned structures without using BUILDQ

```
(S =TYPE =SUBJ (TNS= TNS) (VP (V= V)))
```

The following is the MACLISP code for @ (with the ew*1 borrowed and improved from a trace package of the MIT-AI-Lab). The function RPLACO replaces a CONS. Feel free to include other macros using READMAC.

```
(READMAC MACRO
(NLAMBDA (F)
  ; TURNS MACRO-CHARS ON WHILE DOING THING;
  ; (READMAC <A-LIST> <THING-TO-DO>);
  ; <A-LIST> HAS PAIRS OF CHAR AND (QUOTED) FUNCTION;
  (COND
    ((CADR F)
     (LET
      (CHAR (CAADDR F) FN (CDAADR F))
      (RPLACO
       F
       'LET
       @
       ((SYNTAX (STATUS SYNTAX =CHAR)
        FNTYP
        (FNTYP '=CHAR)
        OLD
        (GET '=CHAR FNTYP))
        (PROG2 (SSTATUS MACRO =CHAR =FN)
         (READMAC = (CADR F) , (CDDR F))
         (FUNCALL 'SSTATUS 'SYNTAX '=CHAR SYNTAX)
         (AND FNTYP (PUTPROP '=CHAR OLD FNTYP))
         ; SET STATUS AND RESET AFTERWARDS;))))
      ; DO THING WITH READ OR READLIST;
      ((RPLACO F (CAADDR F) (CDADDR F))))))
```

```
(LET MACRO
(NLAMBDA (F)
  (COND ((CADR F) (RPLACA F 'LET1))
        ((CDDR F) (RPLACO F 'PROGN (CDDR F)))
        ((RPLACO F (CAADDR F) (CDADDR F))))))
```

```
(LET1 MACRO
(NLAMBDA (F)
  ((LAMBDA (V)
   (COND
    ((NULL (CDDR V))
     (RPLACO F
      (CONS 'LAMBDA (CONS (LIST (CAR V)) (CDDR F))
            (LIST (CADR V))))
     (V (RPLACO
      F
      (CONS 'LAMBDA
        (CONS (LIST (CAR V))
              (LIST
                (CONS 'LET1 (CONS (CDDR V) (CDDR F))))
              (LIST (CADR V))))))
      (CADR F))))))
```

```
(QU* MACRO
(NLAMBDA (X)
; LISTS WITH EV OR EV* ARE EVALUATED;
; AND THEIR RESULTS WILL BE CONSED;
; OR APPENDED RESPECTIVELY;
((LAMBDA (Y) (RFLACO X (CAR Y) (CDR Y))) (QU*1 (CADR X))))))
```

```
(QU*1 EXPR
(LAMBDA (X)
(COND
((NULL X) NIL)
((ATOM X) (LIST 'QUOTE X))
((EQ (CAR X) 'EV) (CADR X))
(OPTIM
(COND
((ATOM (CAR X))
(LIST 'CONS (LIST 'QUOTE (CAR X)) (QU*1 (CDR X))))
((EQ (CAAR X) 'EV*)
(LIST 'APPEND (CADR X) (QU*1 (CDR X))))
((LIST 'CONS (QU*1 (CAR X)) (QU*1 (CDR X))))))))))
```

```
(OPTIM EXPR
(LAMBDA (X)
; ELIMINATES UNNECESSARY FN-CALLS;
(SELECTQ (CAR X)
(CONS
; (CONS X (LIST ---)) => (LIST X ---);
(COND
((CADR X)
(AND (EQ (CAADDR X) 'LIST)
(SETQ X (CONS 'LIST (CONS (CADR X) (CAADDR X))))))
((SETQ X (LIST 'LIST (CADR X))))))
(APPEND
; (APPEND X (APPEND ---)) => (APPEND X ---);
(COND
((CADR X)
(AND (EQ (CAADDR X) 'APPEND)
(SETQ X (CONS 'APPEND (CONS (CADR X) (CAADDR X))))))
((SETQ X (CADR X))))
NIL)
(AND (CATCH (MAPC '(LAMBDA (ARG)
(COND ((ATOM ARG) (THROW NIL))
((EQ (CAR ARG) 'QUOTE))
((THROW NIL))))
(CDR X)))
(SETQ X (LIST 'QUOTE (EVAL X)))
; F IS-IN (APPEND CONS LIST);
; (F 'A 'B ---) => 'VALUE;
; WHERE VALUE = (EVAL (F 'A 'B ---));)
X))
```

SEND MORE USEFUL FUNCTIONS)