Prof. Dr. H. Stoyan
Universität Erlangen-Nürnberg
Institut für Mathematische Maschinen
und Datenverarbeitung (Informatik VIII)
Am Weichselgarten 9
8520 Erlangen

COMPILER

Compile(L,S)=Print(comp2(quint(L),S) is the compiler function which compiles a LISP defined function into a SAP program. The functions comp2 and quint are discussed below along with their auxiliary functions. Lap is discussed in another memo.

Comp2(L,S) is a function which writes a program for the list L and has as value, a list which contains this program. The instructions in this program are SAP instructions. S is used for two purposes. If car(L) is lambda or label, the symbol S will be used as the symbolic location of the closed subroutine of the program for L which will leave its answer in the accumulator. Otherwise S will serve as the symbolic location where the program for L will leave its answer. At the beginning of a compilation, the programmer may choose S to be any name that he wishes to call the function which he is compiling.

Quint(L) is an abbreviation for quintessence (L) which is a function that analyzes all subexpressions of L and has as value a list containing all distinct subexpressions of L without repetition. By distinct is meant that two lists L1 and L2 are distinct if eq2 (L1,L2)=0. Quint(L) also changes the original list L so that two or more occurrences of nondistinct subexpressions are made identical. An occurrence of a subexpression is any 15-bit quantity which points at that subexpression.

Compl(L,R) is an auxiliary function used by comp2(L,S) to administer temporary storage and other symbolic locations. This function writes a program for L and adds it to the list R on which the program is being accumulated. If car(L)=label or lambda, it generates a symbol W for the symbolic location of this program and another symbol T where is located a TXLtoW. If car(L) is an object, the symbolic location T of this object may be found on its property list. If no symbol is there then T is generated and put on the property list. If car(L) is none of these cases, it generates a symbol T and writes a program for L which stores its results in T. In all cases the value of compl(L,R) is T. Also, if car(L) was not an object, then it is made into an object and the symbol T is put onto its property list.

Saving for recursive functions is administrated by constants A,B,C,D,E. A is used as a flag to signal that a label has been

previously met and hence this is a recursive situation. B is a list of symbolic locations of objects which have been created since the last recursion. C is the list of objects that need to be saved. D is the symbolic location, plus one, of the last object to be saved and E is the number to be saved. A is originally set to 1. When the first recursion is met, A is set to 0. The test which tells what must be saved is as follows. An object must be saved if between the time it was created and used, a recursion has occurred. Hence, the test is merely to search the B list for the object. If it is not there, then we must save it i.e. it is put on the C list.

```
Comp2(L,S)
     R1=0
1--> R1=move(make move check the B-list and put things on C-list)
          (maplis(cdr(L),λ(J,compl(car(J),R1))),ARG)
     T1=search(car(L),λ(K,car(K)=funct),λ(K,cadr(K)),λ(J,0))
     If T1=0, go to a
   b R1=  attrib(R1,list(list(0,TSX,T1,4),move(ac,S))))
     Return R1
   a T1=getsym(car(L))
     IfA=0, go to c
   d B=0
     go to b
   c R1=attrib(R1,list(list(0,TSX,SAVE,4),list(0,0,D,0,E)))
     A=1
     go to d
car(L)=label-> A=0,B=0,C=0,D=gensym,E=gensym
     attrib(cadr(L),list(symb,S))
     M=caddr(L)
     I=gensym
     R2=attrib(cons(list(S,BSS),0),move(IR4,I))
     R2=attrib(R2,move(ARG,maplis(cadr(M),λ(J,compl(J,0))))
     R2=attrib(R2,comp2(caddr(M),ac))
     If A=0, go to a
     R2=attrib(R2,list(list(0,TSX,UNSAVE,4),
          list(0,0,D,0,E)))
   a R2=attrib(R2,attrib(move(I,IR4),list(0,TRA,1,4)))
     If A=0, go to b
     C=distrb (C)
```

```
      C=attrib(cons(I,C),D)
      N=0
      R2=attrib(R2,maplis(C,λ(J,list(I+N,SYN,car(J)))))
                                        N=N+1
      R2=attrib(R2,list(E,EQU,N-1,))
   b  Return R2
car(L)=lambda→I=gensym
               R3=attrib(cons(list(S,BSS),0),move(IR4,I))
               R3=attrib(R3,move(ARG,maplis(cadr(L),λ(J,compl(J,0)))
               R3=attrib(R3,comp2(caddr(L),ac)
               R3=attrib(R3,attrib(move(I,TR4),list(0,TRA,1,4)))
               Return R3
car(L)=cond→K=gensym
               R4=0
               mapcon(cdr(L),λ(J,1P=gensym Return Condfu(J,P,K,R4)))
               attrib(R4,list(K,BSS))
               Return R4
condfu(J,P,K,R4)=list(move(compl(caar(J),R4),ac),list(0,
               TZE,P),move(compl(cadar(J),R4),S),list
               (0,TRA,K),list(P,BSS))
compl(L,R)=    If car(L)=-1,go to b
               T=gettemp(L)
               If T=0, go to a
               search(B,λ(J,car(J)=T),λ(J,0),λ(J,attrib(C,cons(T,0))))
               return T
            a  T=get symb(L)
               If T=0,go to f
               return T
            f  T=gensym
            c  B=attrib(cons(T,0),B)
               attrib(L,list(temp,T))
               return T
            b  T=gensym
               If car(L)=label, go to g
               If car(L)=lambda, go to d
               attrib(R,comp 2(L,T))
            e  replaca(L,-1)
               replacd(L,0)
               go to c
```

```
        d       W=gensym
        h       attrib(R,list(T,TXL,W))
                attrib(R,comp2(L,W))
                go to e
        g       W=print name of cadr(L)
                go to h
```

```
Distel(L)=mapcon(L,λ(J,caar(J)=-1→0,1→search(cdr(J),λ(K,equ2(car(K),
                                car(J),0),λ(K,prog(replaca(K,car(J)),
                                   return0)),λ(0,cons(car(J),0)))))
mapconI(L)=mapcon(L,λ(J,caar(J)=-1→0,1→mapcon(car(J),λ(K,caar(K)
     =-1→0,1→cons(car(K),0)))))
Union(L,D)=attrib(mapcon(L,λ(J,search(D,λ(K,equ2(car(K),car(J),0)),λ(K,
     replaca(K,car(J)),λ(0,cons(car(J),0))))),D)
     return 0
Quintessence(L)=(L=0→0,car(L)=cond→
          T=0
          mapcon(cdr(L),λ(J,          T=attrib(quint(cdar(J)),T
                                      T=union(quint(cons(caar(J),0)),T)
          Return T                    return 0
        1→Union(Distel(L),quint(mapcon I(distel(L)))))
Distob(L)=mapcon(L,λ(J,search(cdr(J),λ(K,equ(car(K),car(J))λ(K,0),
          λ(0,cons(car(J),0)))))
Move(L1,L2)=(L1VL2=0→0,1→append((car(L1)=car(L2)→0,etc),move
          (cdr(L1),cdr(L2))
```