

```

1
2 datatype s_espressione = NUM of int
3   | STRINGA of string
4   | T | F | NIL
5   | DOT of s_espressione * s_espressione ;
6
7 datatype Sexpr = Var of string
8   | Quote of s_espressione
9   | Op of string * Sexpr list
10  | If of Sexpr * Sexpr * Sexpr
11  | Lambda of string list * Sexpr
12  | Call of Sexpr * Sexpr list
13  | Let of Sexpr * string list * Sexpr list
14  | Letrec of Sexpr * string list * Sexpr list;
15
16 datatype secdexpr = ADD | SUB | MUL | DIV | REM | EQ | LEQ | CAR | CDR | CONS
17   | ATOM | JOIN | RTN | AP | DUM | RAP | STOP
18   | LD of int*int
19   | LDC of s_espressione
20   | SEL of secdexpr list * secdexpr list
21   | LDF of secdexpr list ;
22
23
24
25 exception NonTrovato ;
26
27 fun find(nil,y)= raise NonTrovato |
28   find(xl::L,y) = if xl=y then 0 else 1+find(L,y);
29
30 fun location(x,i,nil) = raise NonTrovato |
31   location(x,i,nl::L) =
32     let
33       val temp = find(nl,x) handle NonTrovato => ~1
34     in
35       if temp = ~1
36       then location(x,i+1,L) handle NonTrovato => (~1,~1)
37       else (i,temp)
38     end;
39
40 (* COMP *)
41 fun COMP(e: Sexpr, n: string list list, c: secdexpr list): secdexpr list=
42 case e of
43   Var x => LD(location(x,0,n))::c |
44
45   Quote k => LDC(k)::c |
46
47   Op(operator,sl) =>
48     (case operator of
49       (* hd -> restituisce la testa della lista
50        tl -> restituisce il resto della lista
51        hd(tl()) -> restituisce la testa del resto della lista
52        *)
53       (* OPERATORI BINARI *)
54       "ADD" => COMP(hd(sl),n,COMP(hd(tl(sl)),n,(ADD::c))) |
55       "SUB" => COMP(hd(sl),n,COMP(hd(tl(sl)),n,(SUB::c))) |
56       "MUL" => COMP(hd(sl),n,COMP(hd(tl(sl)),n,(MUL::c))) |
57       "DIV" => COMP(hd(sl),n,COMP(hd(tl(sl)),n,(DIV::c))) |
58       "REM" => COMP(hd(sl),n,COMP(hd(tl(sl)),n,(REM::c))) |
59       "EQ" => COMP(hd(sl),n,COMP(hd(tl(sl)),n,(EQ::c))) |
60       "LEQ" => COMP(hd(sl),n,COMP(hd(tl(sl)),n,(LEQ::c))) |
61       (* OPERATORI UNARI *)
62       "CAR" => COMP(hd(sl),n,(CAR::c)) |
63       "CDR" => COMP(hd(sl),n,(CDR::c)) |
64       "ATOM" => COMP(hd(sl),n,(ATOM::c)) | (* non ne sono sicuro !!!
65       *)
66       "CONS" => COMP(hd(tl(sl)),n,COMP(hd(sl),n,(CONS::c)))
67     ) |
68
69   If(cond,e1,e2) =>
70     let
71       val thenpt = COMP(e1,n,[JOIN])
72       val elsept = COMP(e2,n,[JOIN])
73     in
74       COMP(cond,n,SEL(thenpt,elsept)::c)
75     end |
76
77   Lambda(varLegate,sl) => [ LDF ( COMP(sl,varLegate::n,[RTN]) ) ]
78 |
79   Call(exp,lista_var) =>
80     let
81       fun Funzione(nil: Sexpr list , n: string list list): secdexpr

```

```

80         list= nil
           | Funzione(Testa::Coda: Sexpr list , n: string list list):
           secdexpr list= Funzione(Coda,n)@COMP(Testa,n,[])@[CONS]
81     in
82         LDC(NIL)::Funzione(lista_var,n)@COMP(exp,n,[AP])@c
83     end
84 | Let(e,v,lista) =>
85     let
86         fun Funzione(nil: Sexpr list , n: string list list): secdexpr
           list=nil□
87         | Funzione(Testa::Coda: Sexpr list , n: string list list):
           secdexpr list=Funzione(Coda,n)@COMP(Testa,n,[])@[CONS]
88     in
89         LDC(NIL)::Funzione(lista,n)@ LDF( COMP(e,v::n,[RTN]))::[AP]@c
90     end
91
92 | Letrec(e,v,lista) =>
93     let
94         fun Funzione(nil: Sexpr list , n: string list list): secdexpr
           list=nil□
95         | Funzione(Testa::Coda: Sexpr list , n: string list list):
           secdexpr list=Funzione(Coda,n)@COMP(Testa,n,[])@[CONS]
96     in
97         DUM::LDC(NIL)::Funzione(lista,v::n)@LDF(
           COMP(e,v::n,[RTN]))::[RAP]@c
98     end ;
99
100

```