Generazione di Codice

Esercizio1 - Testo

(a) Si dia uno schema di traduzione ascendente per la generazione di codice a 3 indirizzi che traduca il comando while descritto dalla seguente grammatica:

W::= HC endw
HC::= HE do C
HE::= HI E
HI::= while

(a1) assumendo epressioni con generazione per Loc
(a2) assumendo espressioni con generazione per short circuit

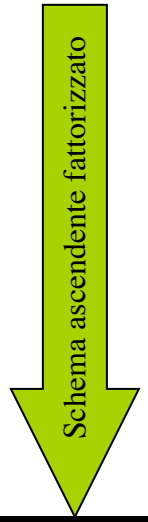
(b) Si mostri il codice generato dalla traduzione del seguente comando:

while x>5 do x:=x+7

mostrando prima il codice che si suppone sia generato da: x>5 e x:=x+7.

Esercizio 1- (a1)

[10]W::= while {init:= quad} E do
{w.next:= mk-L(quad);
emit('if' E.loc '= #false goto' --)} C {BK(C.next, init);
emit('goto' init)} endw



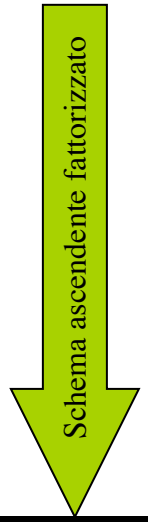
W::= HC endw
HC::= HE do C
HE::= HI E
HI::= while

Schema di traduzione

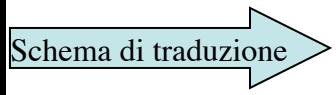
W::= HC endw
{emit('goto' HC.init);
W.next=HC.next;}
HC::= HE do C
{BK(C.next,HE.init);
HC.init = HE.init;
HC.next = HE.next;}
HE::= HI E
{HE.next = [quad];
emit('if' E.loc '= #false goto' --);
HE.init = HI.init;}
HI::= while
{HI.init = quad;}

Esercizio 1- (a2)

[10]W::= while {init:= quad} E do
{w.next:= E.false;
BK(E.true,quad);} C {BK(C.next, init);
emit('goto' init)} endw



W::= HC endw
HC::= HE do C
HE::= HI E
HI::= while



W::= HC endw
{emit('goto' HC.init);
W.next=HC.false;}
HC::= HE do C
{BK(C.next,HE.init);
HC.init = HE.init;
HC.false = HE.false;}
HE::= HI E
{HE.false = E.false;
BK(E.true,quad);
HE.init = HI.init;}
HI::= while
{HI.init = quad;}

Esercizio 1- (b)

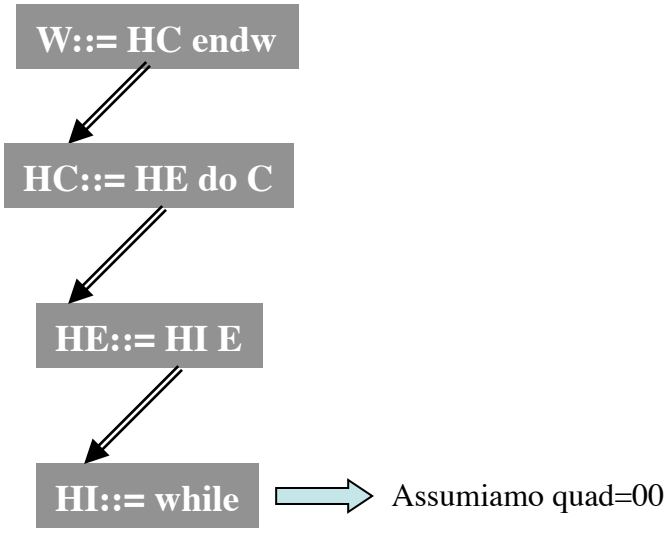
while x>5 do x:=x+7

Non possiamo usare Short-Circuit: Troppo complicato *emulare* operatori relazione:

x>5	loc1:=locx [>] #5	Assunto che: locx sia la locazione in symtab di x
x:=x+7	locx:=locx [+] #7	Assunto che: locx sia la locazione in symtab di x

Applichiamo lo schema di traduzione

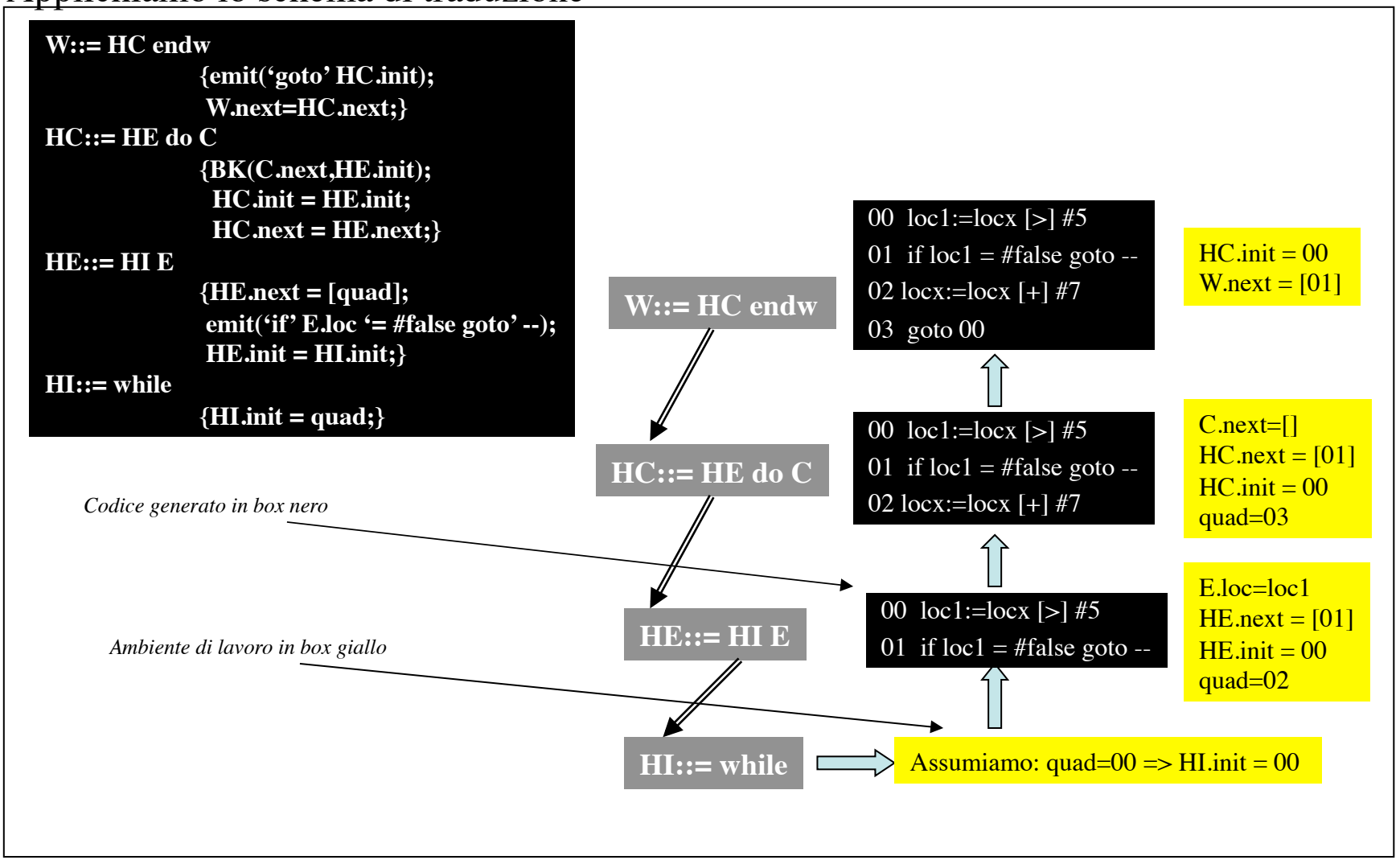
```
W::= HC endw
                {emit('goto' HC.init);
                 W.next=HC.next;}
HC::= HE do C
                {BK(C.next,HE.init);
                  HC.init = HE.init;
                  HC.next = HE.next;}
HE::= HI E
                {HE.next = [quad];
               emit('if' E.loc '= #false goto' --);
                  HE.init = HI.init;}
HI::= while
                {HI.init = quad;}
```



Esercizio 1- (b)

while x>5 do x:=x+7

Applichiamo lo schema di traduzione



Esercizio 2

Esercizio2 - Testo

(a) Si dia uno schema di traduzione discendente per la generazione di codice a 3 indirizzi che traduca il comando for del C descritto dalla seguente grammatica:

FR::= for (E_1 ; E_2 ; E_3) C

(a1) si dia lo schema di generazione di codice per espressioni per loc;

(a2) assumendo epressioni con generazione per Loc;

(a3) si dia lo schema di generazione di codice per E_2 con short circuit;

(a3) assumendo espressione E_2 con generazione per short circuit;

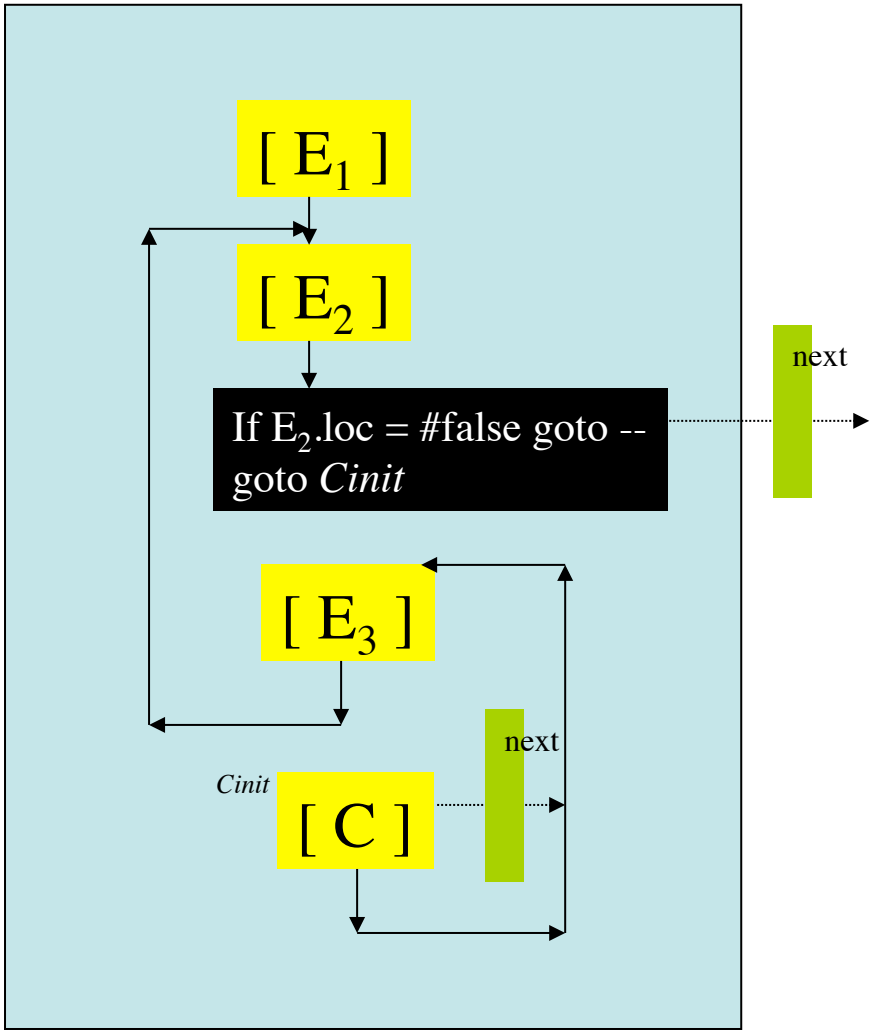
(b) Si mostri il codice generato dalla traduzione del seguente comando:

for (;true;)x:=y+7

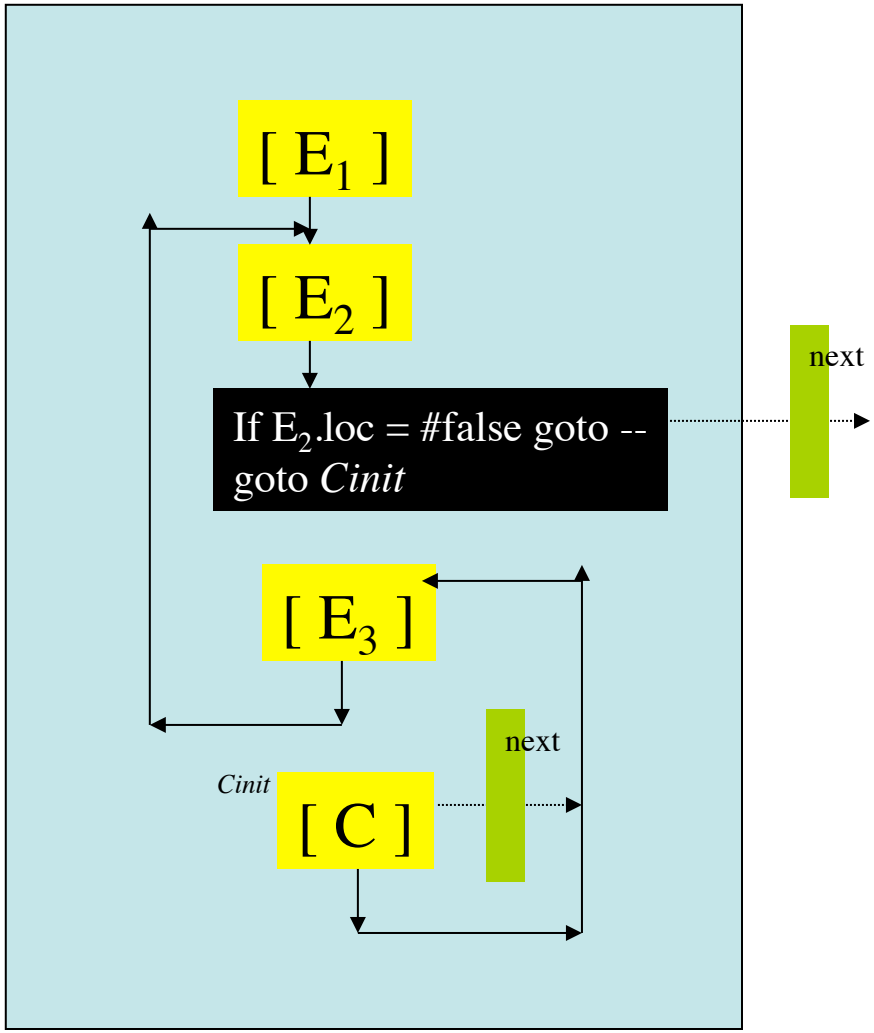
mostrando prima il codice che si suppone sia generato dai componenti.

Esercizio2 - (a1)

FR::= for (E_1 ; E_2 ; E_3) C



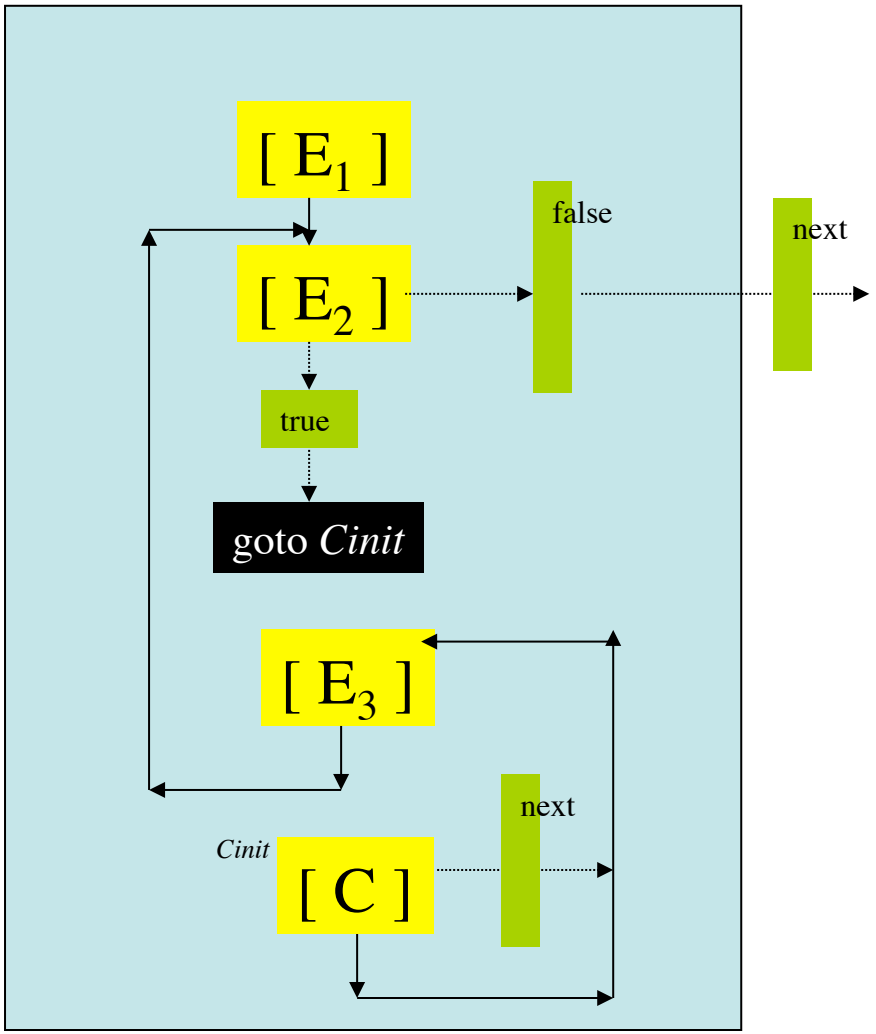
Esercizio2 - (a2)



FR::= for (E_1;{init:= quad}
E_2 {ifnext=[quad];
emit('if' E_2.loc '= #false goto' --);
Cinit=[quad];
emit('goto' --);
Einit=quad;}
E_3 {emit('goto' init);
BK(Cinit,quad);}
C {FR.next=ifnext;
BK(C.next,Einit);
emit('goto' Einit)}
endw

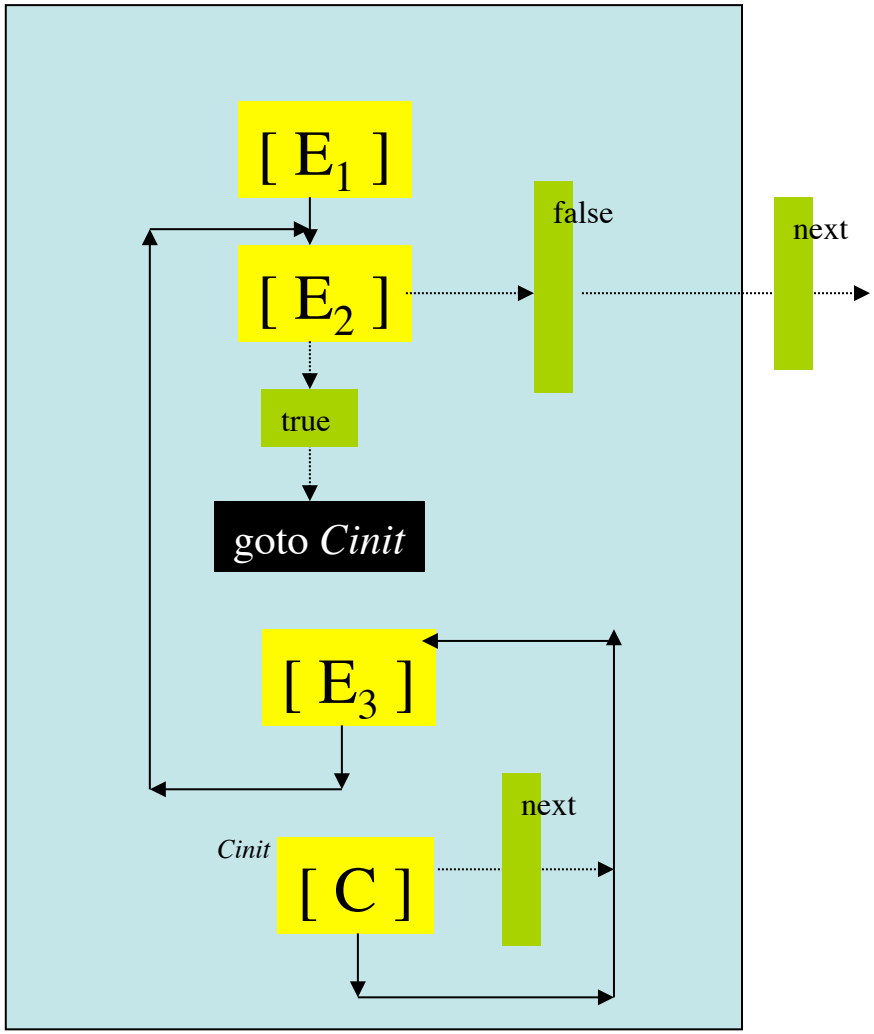
Esercizio2 - (a3)

FR::= for (E_1 ; E_2 ; E_3) C



Esercizio2 - (a4)

FR::= for (E_1 ; E_2 ; E_3) C



Esercizio2 - (b)

```
for (;true;)x:=y+7
```

Esteniamo la grammatica data in modo tale da considereare la mancanza di espressioni (ricordiamo che il C ha l'assegnamento come forma di espressione).

Esercizio 3

Esercizio3 - Testo

(a) Si dia uno schema di traduzione discendente per la generazione di codice a 3 indirizzi che traduca il comando for del C descritto dalla seguente grammatica:

CaSe::= case E of PairList Default
PairList::= E EList : C; PairList
PairList::= ε
EList::=, E EList
Default::= C

(a1) si dia lo schema di generazione di codice per espressioni per loc;
(a2) assumendo epressioni con generazione per Loc;

(b) Si mostri il codice generato dalla traduzione del seguente comando:

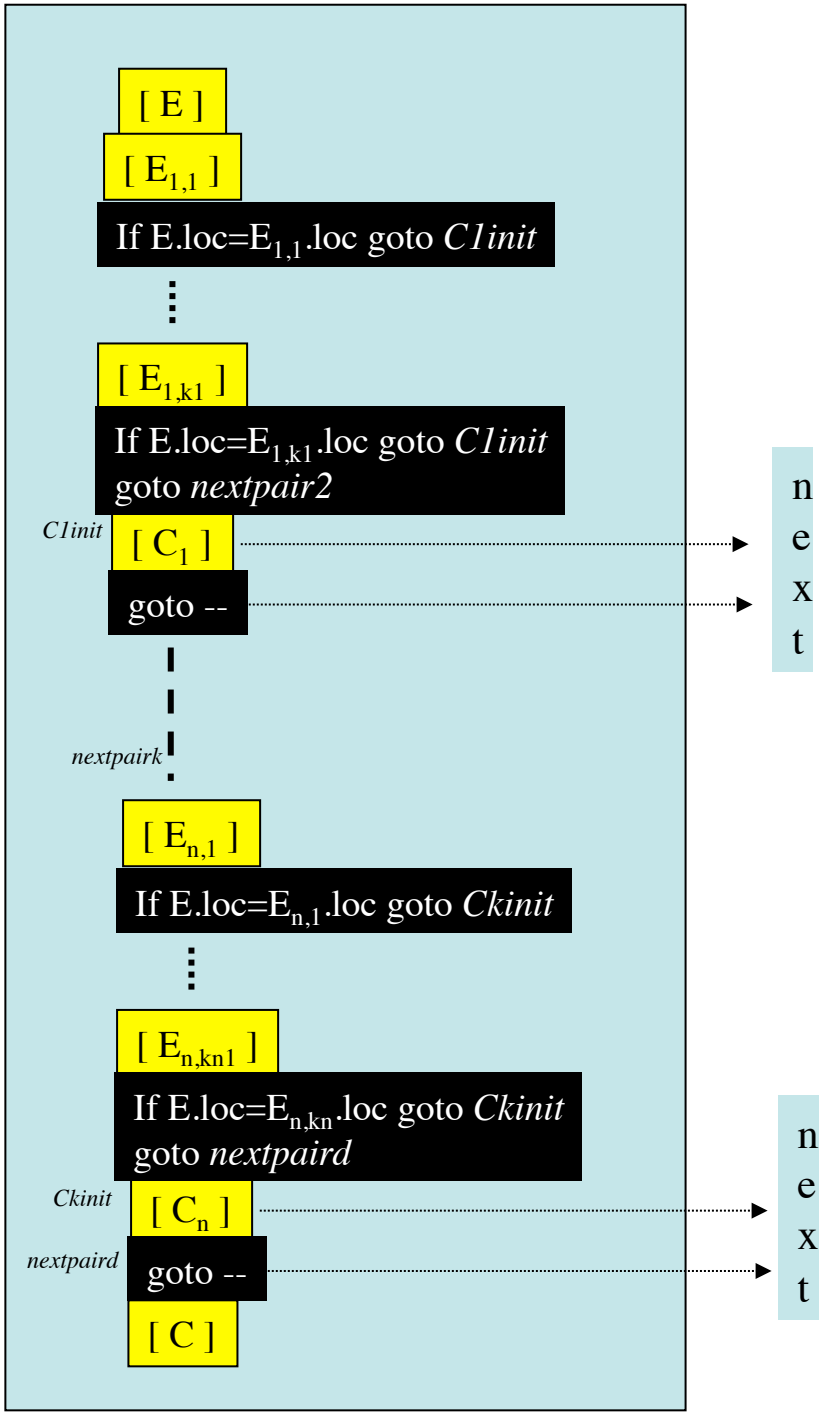
```
case x+y of x,x+2:x=y;
            5:x=y-x;
            x=3
```

mostrando prima il codice che si suppone sia generato dai componenti.

Esercizio 3 - (a1)

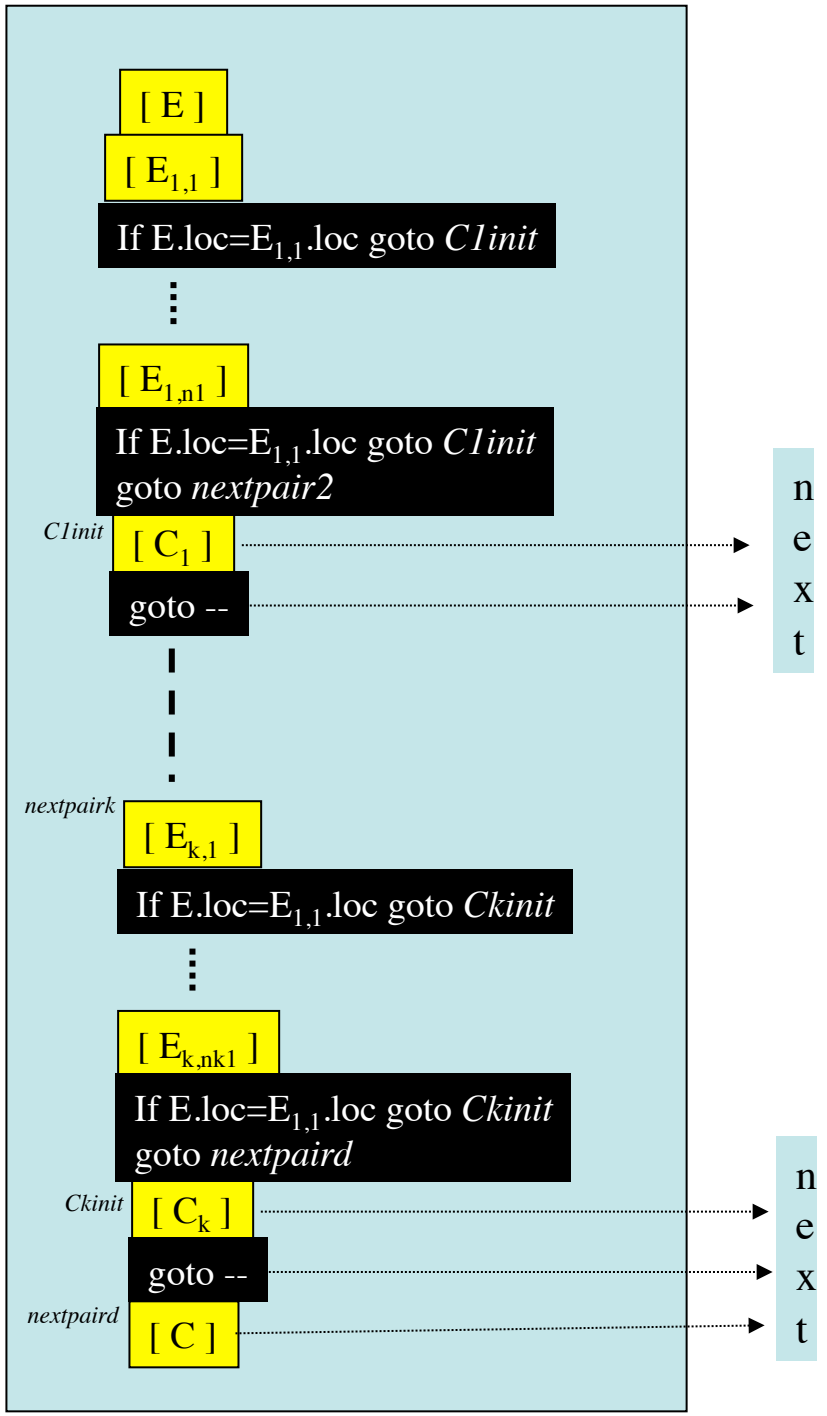
CS::= case E of PL D
PL::= E EL : C; PL
PL::= ε
EL::=, E EL
EL::= ε
D::= C

Case E of $E_{1,1}$ … $E_{1,k1}$: C_1
…
$E_{n,1}$ … $E_{n,kn}$: C_n
C



Esercizio 3 - (a2)

CS::= case E of {PL.inL=E.loc;PL.inR=[]}
PL {BK(PL.nextpair,quad);}
D {CS.next=PL.next+D.next;}
PL_1::= {BK(PL_1.inR,quad);
E {Cinit=[quad]; EL.inL=PL_1.inL;
emit('if'PL_1.inL=E.loc 'goto' --);}
EL : {BK(Cinit+EL.Cinit,quad);}
C ; {Tnext=C.next+[quad];
emit('goto' --); PL_2 inL=PL_1.inL;
PL_2.inR=EL.nextpair;}
PL_2 {PL_1.next= PL_2.next +Tnext;
PL_1.nextpair=PL_2. nextpair}
PL::= ε {PL.nextpair=PL.inR; PL.next=[];}
EL_1::=, **E** {Cinit=[quad]; EL_2.inL=EL_1.inL;
emit('if'EL_1.inL=E.loc 'goto' --);}
EL_2 {EL_1.Cinit=Cinit+ EL_2.Cinit;
EL_1.nextpair=EL_2.nextpair;
EL::= ε {EL.nextpair=[quad]; EL.Cinit=[];
emit('goto' --);}
D::= C {D.next=C.next;}



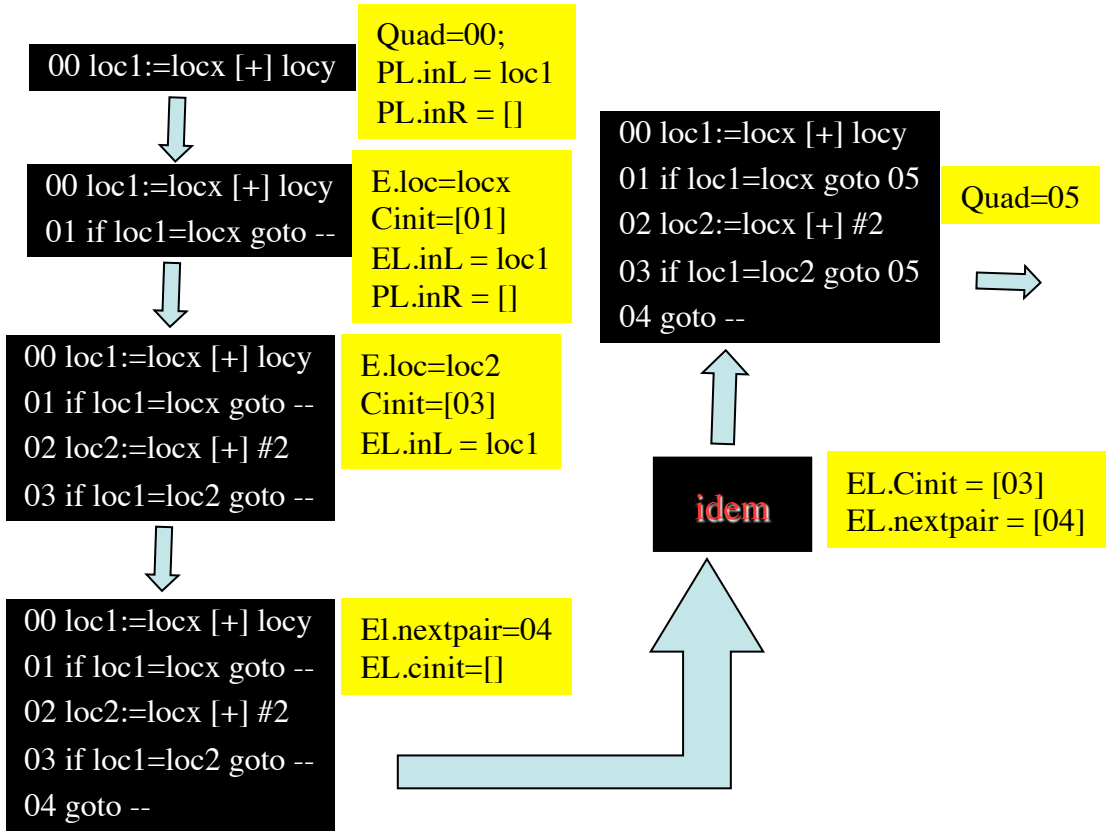
Esercizio 3 - (b)

```
CS::= case E of {PL.inL=E.loc;PL.inR=[]}
            PL {BK(PL.nextpair,quad);}
            D {CS.next=PL.next+D.next;}
PL1::= {BK(PL1.inR,quad);
        E {Cinit=[quad]; EL.inL=PL1.inL;
           emit('if'PL1.inL=E.loc 'goto' --);}
        EL : {BK(Cinit+EL.Cinit,quad);}
        C ; {Tnext=C.next+[quad];
             emit('goto' --); PL2 inL=PL1.inL;
             PL2.inR=EL.nextpair;}
        PL2 {PL1.next= PL2.next +Tnext;
             PL1.nextpair=PL2. nextpair}
PL::= ε  {PL.nextpair=PL.inR; PL.next=[];}
EL1::=, E {Cinit=[quad]; EL2.inL=EL1.inL;
           emit('if'EL1.inL=E.loc 'goto' --);}
        EL2 {EL1.Cinit=Cinit+ EL2.Cinit;
             EL1.nextpair=EL2.nextpair;
EL::= ε {EL.nextpair=[quad]; EL.Cinit=[]
         emit('goto' --);}
D::= C {D.next=C.next;}
```

```
case x+y of x,x+2:x=y;
            5:x=y-x;
            x=3
```

Assunto che: locx sia la locazione in symtab di x

x+y	loc1:=locx [+] locy
x+2	loc2:=locx [+] #2
x=y	locx:= locy
x=y+x	loc3:=locy [-] locx locx:= locy
x=3	locx:=#3



Esercizio 3 - (b-2)

CS::= case E of {PL.inL=E.loc;PL.inR=[]}
PL {BK(PL.nextpair,quad);}
D {CS.next=PL.next+D.next;}
PL_1::= {BK(PL_1.inR,quad);
E {Cinit=[quad]; EL.inL=PL_1.inL;
emit('if'PL_1.inL=E.loc 'goto' --);}
EL : {BK(Cinit+EL.Cinit,quad);}
C ; {Tnext=C.next+[quad];
emit('goto' --); PL_2 inL=PL_1.inL;
PL_2.inR=EL.nextpair;}
PL_2 {PL_1.next= PL_2.next +Tnext;
PL_1.nextpair=PL_2. nextpair}
PL::= ε {PL.nextpair=PL.inR; PL.next=[];}
EL_1::=, E {Cinit=[quad]; EL_2.inL=EL_1.inL;
emit('if'EL_1.inL=E.loc 'goto' --);}
EL_2 {EL_1.Cinit=Cinit+ EL_2.Cinit;
EL_1.nextpair=EL_2.nextpair;
EL::= ε {EL.nextpair=[quad]; EL.Cinit=[];
emit('goto' --);}
D::= C {D.next=C.next;}

```
case x+y of x,x+2:x=y;
            5:x=y-x;
            x=3
```

Assunto che: locx sia la locazione in symtab di x

x+y	loc1:=locx [+] locy
x+2	loc2:=locx [+] #2
x=y	locx:= locy
x=y+x	loc3:=locy [-] locx locx:= locy
x=3	locx:=#3

```
00 loc1:=locx [+] locy
01 if loc1=locx goto 05
02 loc2:=locx [+] #2
03 if loc1=loc2 goto 05
04 goto --
```

E.loc=locx
Cinit=[01]
EL.inL = loc1
PL.inL=loc1
PL.inR = []
EL.Cinit = [03]
EL.nextpair =[04]
Quad=05

```
00 loc1:=locx [+] locy
01 if loc1=locx goto 05
02 loc2:=locx [+] #2
03 if loc1=loc2 goto 05
04 goto --
05 locx:=locy
06 goto --
```

c.next=[]
Tnext=[06]
PL_2.inL=loc1
PL_2.inR=[04]

```
00 loc1:=locx [+] locy
01 if loc1=locx goto 05
02 loc2:=locx [+] #2
03 if loc1=loc2 goto 05
04 goto 07
05 locx:=locy
06 goto --
07 if loc1=#5 goto 09
08 goto 12
00 loc3:=locy [-] locx
10 locx:= locy
11 goto --
12 locx:=#3
```

```
00 loc1:=locx [+] locy
01 if loc1=locx goto 05
02 loc2:=locx [+] #2
03 if loc1=loc2 goto 05
04 goto 07
05 locx:=locy
06 goto --
```

quad=07

Esercizio 4

Esercizio3 - Testo

(a) Si cosiderino espressioni con sola somma di interi, identificatori di variabile, literals. Si dia uno schema di traduzione discendente che generi una traduzione source-to-source che fornisca un'espressione equivalente in cui sono stati calcolati tutti i termini che possono essere valutati a compile time. L'espressione risultante deve contenere al più un literal. Ad esempio:

3+z+y+2 è trasformata in z+y+5

(b) Si applichi lo schema dato alla trasformazione dell'espressione dell'esempio

(c) Lo si saprebbe estendere a espressioni con somme e prodotti, identificatori, literals, grouping e precedenza di prodotto?

(d) Si saprebbe generare uno schema di traduzione che generi il codice a 3 indirizzi che risolve a compile-time i calcoli che possono essere risolti staticamente.

Esercizio 4 – (a)

attributi & operatori

Grammatica LR(1) utilizzata

E::= S
S::= S+T
S::= T
T::= ide | num

Usiamo due attributi sintetizzati:
-exp di tipo stringa per E,S,T
-val di tipo int per S,T
Usiamo operatori:
-||: stringxstring->string
-string: int->string
-int: lessema->string

Grammatica attributata

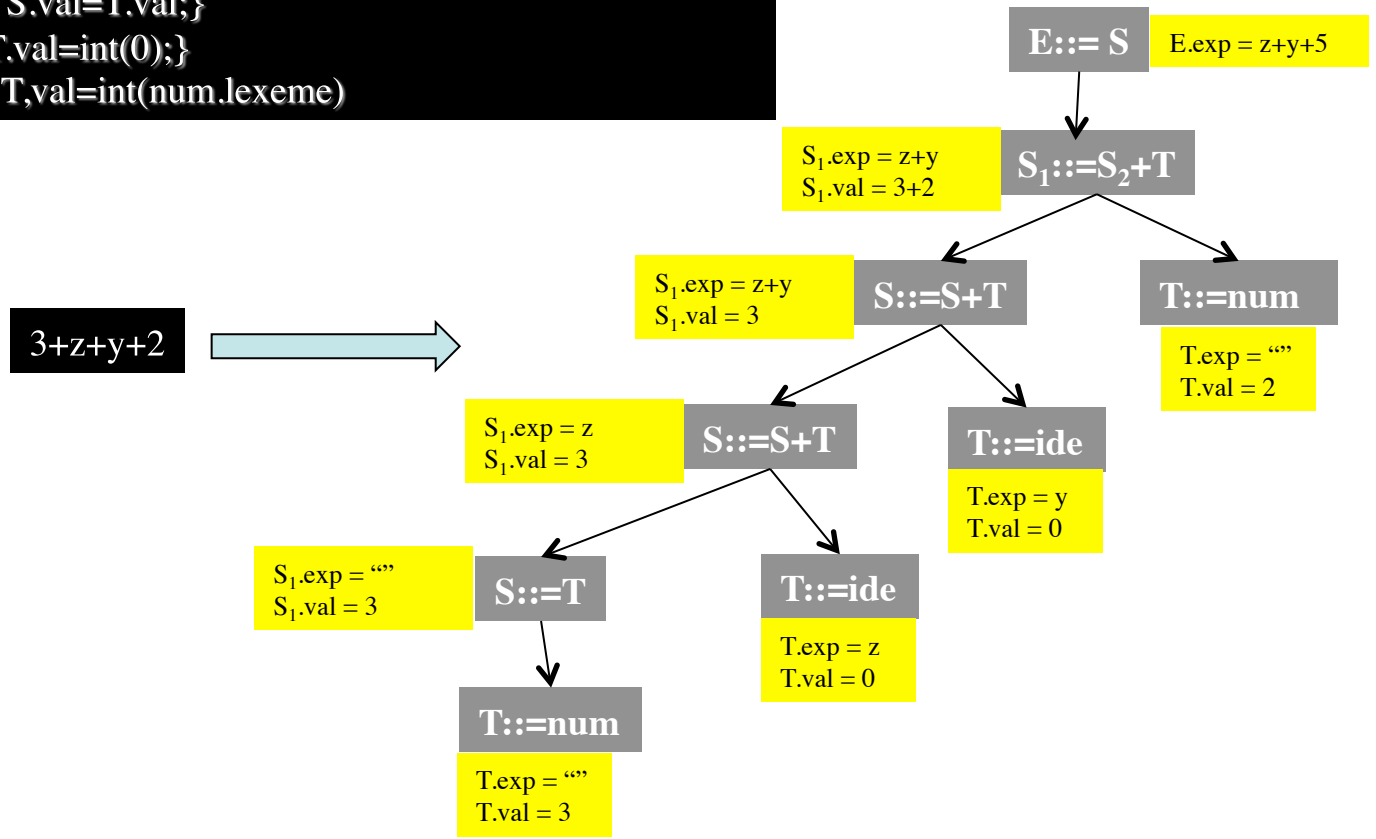
E::= S {if (S.val=0) then E.exp=S.exp
else E,exp = S.exp ||" +"||string(S.val);}
S_1::= S_2+T {if ((T.exp≠"")&(S_2.exp≠0)) then S_1.exp = S_2.exp||"+"||T.exp
else if ((S_2.exp = T.exp)or(T.exp="") then S_1.exp = S_2.exp
else S_1.exp = T.exp;
S_1.val = S_2.val + T.val;}
S::= T {S.exp = T.exp; S.val = T.val;}
T::= ide {T.exp=ide; T.val=int(0);}
| num {T.exp=""; T,val=int(num.lexeme)

Esercizio 4 – (b)

Grammatica attributata

E::= S {if (S.val=0) then E.exp=S.exp
else E,exp = S.exp ||" +"||string(S.val);}
S_1::= S_2+T {if ((T.exp≠"")&(S_2.exp≠"")) then S_1.exp = S_2.exp||"+"||T.exp
else if ((S_2.exp=T.exp)or(T.exp="") then S_1.exp = S_2.exp
else S_1.exp=T.exp;
S_1.val = S_2.val + T.val;}
S::= T {S.exp = T.exp; S.val=T.val;}
T::= ide {T.exp=ide; T.val=int(0);}
| num {T.exp=""; T,val=int(num.lexeme)

3+z+y+2



Esercizio 4 – (c)

Grammatica attributata: source-to-sorce translation

```
E::= S {if (S.val=0) then E.exp=S.exp
          else E,exp = S.exp ||" +"||string(S.val);}
S1::= S2+T {if ((T.exp≠"")&(S2.exp≠"")) then S1.exp = S2.exp||"+"||T.exp
                else if ((S2.exp=T.exp)or(T.exp="") then S1.exp = S2.exp
                        else S1.exp=T.exp;
                S1.val = S2.val + T.val;}
S::= T {S.exp = T.exp; S.val=T.val;}
T::= ide  {T.exp=ide; T.val=int(0);}
     | num {T.exp=""; T,val=int(num.lexeme)
```

Grammatica attributata: 3-address code generation

```
E::= S {if (S.val=0) then E.loc=S.loc
          else {t=newtemp; emit(t":="S.loc"[+] #"S.val;};}
S1::= S2+T {if ((T.exp≠"")&(S2.exp≠"")) then {t=newtemp;
                                                emit(t":="S2.loc"[+] T.loc);
                                                S1.exp = S2.exp;}
                else if ((S2.exp=T.exp)or(T.exp="") then {S1.exp = S2.exp;
                                                          S1.loc = S2.loc;}
                        else {S1.exp=T.exp; S1.loc=T.loc;}
                S1.val = S2.val + T.val;}
S::= T {S.exp = T.exp; S.val=T.val; S.loc=T.loc}
T::= ide  {T.exp=ide; T.val=int(0); T.loc=ide.loc}
     | num {T.exp=""; T,val=int(num.lexeme; T.loc=null)
```