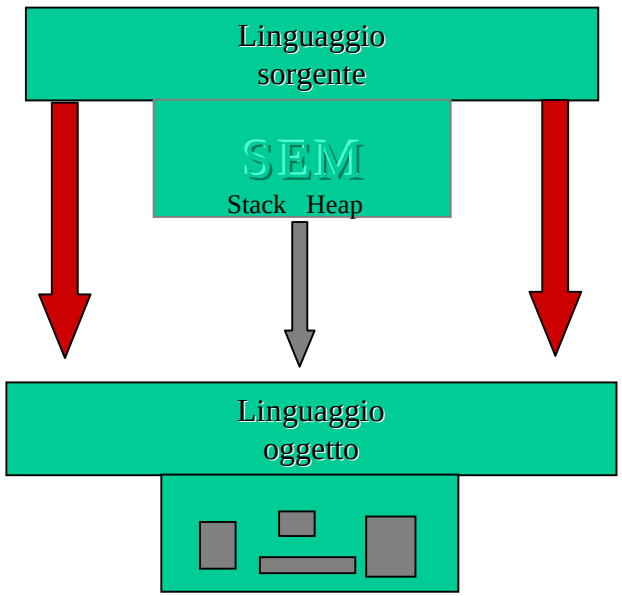
Generazione di codice

- Gerachia di macchine: sorgente, target
- Traduzioni guidate dalla sintassi: l’attributo codice
- Generiamo codice per le espressioni su macchine a stack: codice postfisso
- Un linguaggio intermedio: codice a tre indirizzi
- Generazione di codice su L.I. - *parte 1*

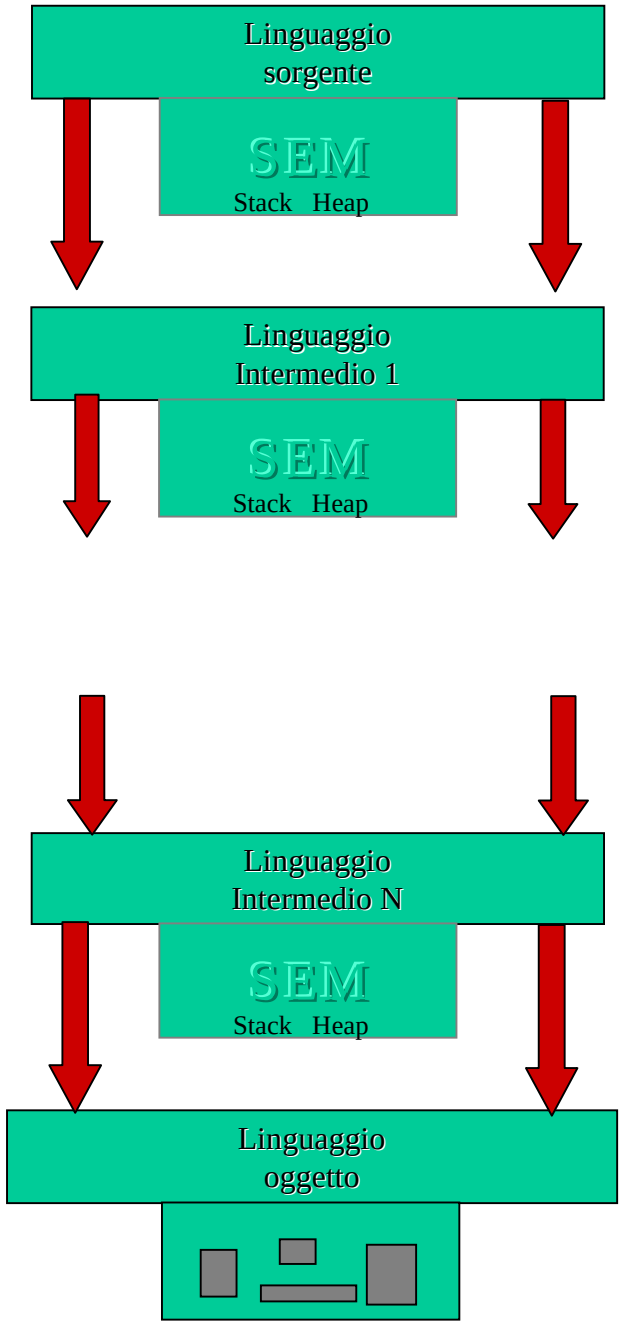
Da strutture di una **M**acchina **A**stratta a
strutture equivalenti (semantica)
di una macchina (concreta)

generazione di codice corrispondente

mediante una traduzione di strutture



Metodologia:
gerarchia di macchine alla cui base e' una macchina concreta dotata di un esecutore



TECNICA

ancora grammatiche ad attributi o SDD

codice generato = attributo

codice = valore di un’opportuna algebra

i.e.

opportune operazioni per:

manipolare e

comporre codice

un semplice e vantaggioso *codice* per le espressioni.

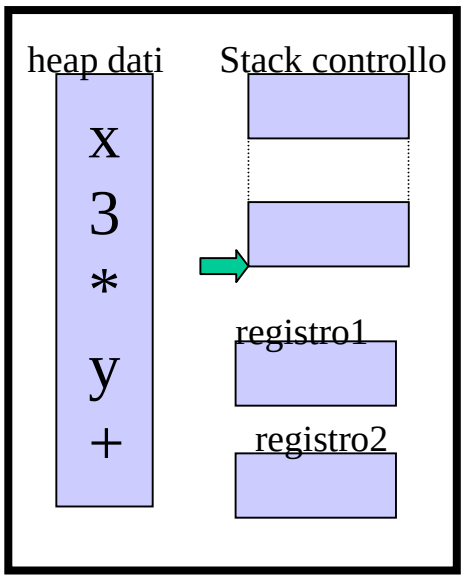
postfissa: opd1 opd2 opt

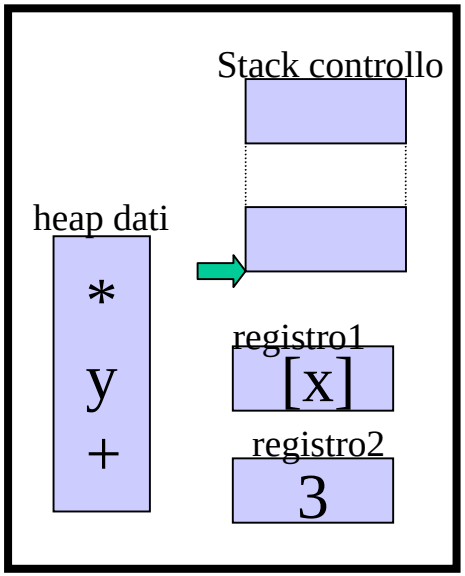
regole di inferenza:

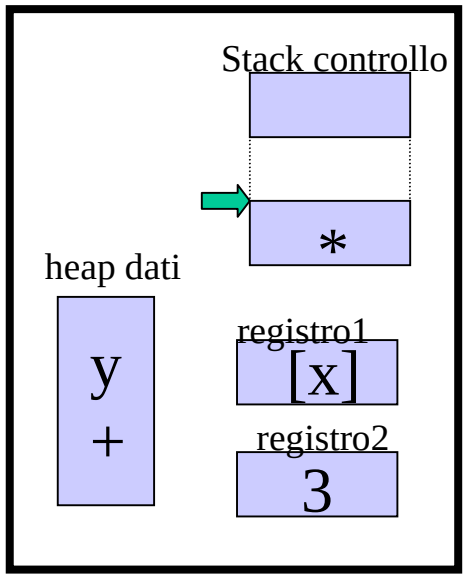
operando:
$$\frac{I=opd\ seq \qquad S=T}{I:=seq \quad S:=push(opd,T)}$$

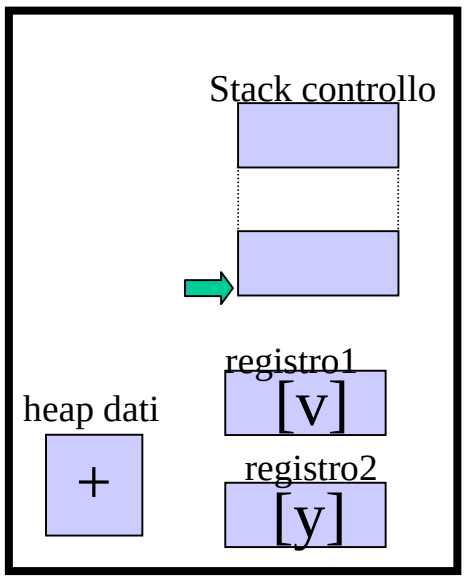
operatore: binario
$$\frac{I=opt\ seq \qquad S=T \qquad [opt](val(top(S)),val(top(S)-1))=v}{val(top(S)-1):=v \quad S:=pop(T,1)}$$

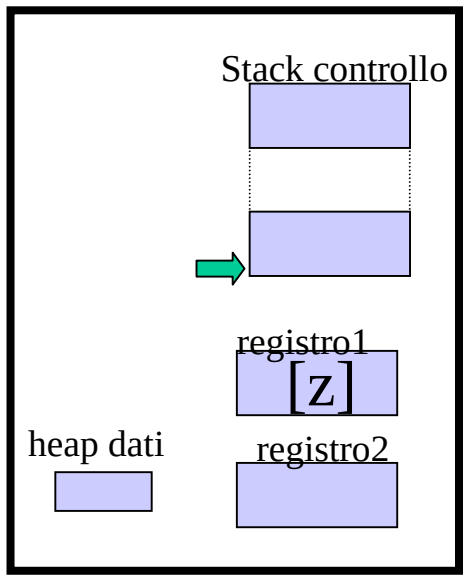
(x*3)+y diventa x 3 * y +











osserva:

[x]=valore di x

[x]*[3]=v

[v]+[y]=z

Piano di traduzione

attributi: in= codice sottoespressione sin. (ereditato di E', F')
code= codice espressione(sintetizzato di E,E',F,F',T)
loc= locazione di memoria per variabili (sintetiz)
operazioni: *= concatena (giustappone) codice postfisso

[0]**P**::= **Ds Cs**	
[1]**P**::= **Cs**	
[2]**D**::= **var ide O** [3]$\mathbf{O_1}$::= **, ide** $\mathbf{O_2}$ [4]**O**::=ε	?
[5]$\mathbf{Cs_1}$::= **; C** $\mathbf{Cs_2}$	
[6]**Cs**::= ε	
[7]**C**::= **A**	
[8]**C**::=**W**	
[9]**A**::= **ide := E**	
[10]**W**::= **while E do C endw**	

Questa parte della grammatica non e' coinvolta nella traduzione

[11]**E**::= **F E'**	**E'**.in:=**F**.code, **E**.code:=**E'**.code
[12]**E'**₁::= **op-l F E'**₂	**E'**₂.in:=**E'**₁.in*__F__.code*__op-l__.value, **E'**₁.code:=**E'**₂.code
[13]**E'**::= ε	**E'**.code:=**E'**.in
[14]**F**::= **T F'**	**F'**.in:=**T**.code, **F**.code:=**F'**.code
[15]**F'**₁::= **op-h T F'**₂	?
[16]**F'**::= ε	**F'**.code:=**F'**.in
[17]**T**::= **num**	**T**.code:=**num**.value
[18]**T**::= **ide**	**T**.code:=**ide**.loc
[19]**T**::= **(E)**	**T**.code:=**E**.code

UN LINGUAGGIO INTERMEDIO

1. (assegnamento)

x:= y op z
x:= op y

$$\frac{S \equiv \langle S\rho, S_M \rangle \vdash loc(x) \to lx = S\rho(x) \quad S \vdash r(y) \to ry = S_M(S\rho(y)) \quad S \vdash r(z) \to rz \quad \vdash [op](ry,rz) = v}{S \vdash x := y\ op\ z \to S[lx/v] = S_M(lx) \leftarrow v}$$

2. (copy)

x:= y

$$\frac{S \vdash loc(x) \to lx \quad S \vdash r(y) \to ry}{S \vdash x := y \to S[lx/ry]}$$

3. (u-jump)

GoTo l

$$\frac{S \vdash code(l) \to P = S\rho(l) \quad S \vdash P \to S'}{S \vdash goto\ l \| Ps \to S'}$$

4. (c-jump)

if x rel y GoTo l

$$\frac{S \vdash code(l) \to P \quad S \vdash r(x) \to rx \quad S \vdash r(y) \to ry \quad \vdash [rel](rx,ry) = false \quad S \vdash Ps \to S'}{S \vdash \text{if } x \text{ rel } y \text{ goto } l \| Ps \to S'}$$

?

5. (P-call)

```
param x1
param x2
...
param xn
call p,n
```

$$\frac{S \vdash r(x) \rightarrow rx}{S \vdash \text{param } x \rightarrow S[./rx]}$$

$$\frac{S \vdash \text{code}(p) \rightarrow P \qquad S \vdash P \rightarrow S'}{S \vdash \text{call } p,n \rightarrow S|_M S' = \langle S\rho, S'_M(S\rho)\rangle}$$

7. (i-structure)

```
x:= y[i]
x[i]:=y
```

$$\frac{\begin{array}{l} S \vdash \text{loc}(x) \rightarrow lx \\ S \vdash \text{loc}(y) \rightarrow ly \\ S \vdash r(i) \rightarrow ri \\ \vdash ly + ri = add \\ S \vdash S_M(add) = v \end{array}}{S \vdash x := y[i] \rightarrow S[lx/v]}$$

?

8. (pointer)

x:= &y
x:=*y
*x:=y

$$\frac{S \vdash loc(x) \to lx \quad S \vdash loc(y) \to ly}{S \vdash x := \&y \to S[lx/ly]}$$

$$\frac{S \vdash loc(x) \to lx \quad S \vdash r(y) \to ry \quad S \vdash S_M(ry) \to v}{S \vdash x := *y \to S[lx/v]}$$

$$\frac{S \vdash r(x) \to rx \quad S \vdash r(y) \to ry}{S \vdash *x := \to S[rx/ry]}$$

significato dei simboli:

l/r = updade di contenuto di l con r

. = locazione corrente stm nel prog.

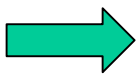
|| = concatenazione di codice

Traduzione delle espressioni di Semplice nel linguaggio intermedio

x+y*3 sara' tradotto in:

```
t1:= y.loc * 3
t2:= x+t1
```

Dove:
t1 e t2 = locazioni del linguaggio intermedio
.loc = attributo il cui valore e' una locazione di L.I.
:= e' un comando di L.I.



Operazioni nel meta:
newtemp: -> **loc**
gen: expr -> **cmd**
||: **cmd cmd** -> **cmd**

Piano di traduzione

code=sintetizzato per il codice prodotto

loc=sintetizzato per la locazione
[ogni sottoespressione calcola la locazione ove e’ posto il risultato]

in=ereditato per la locazione della sottoespressione sinistra

Aggiungiamo la traduzione del comando di assegnamento

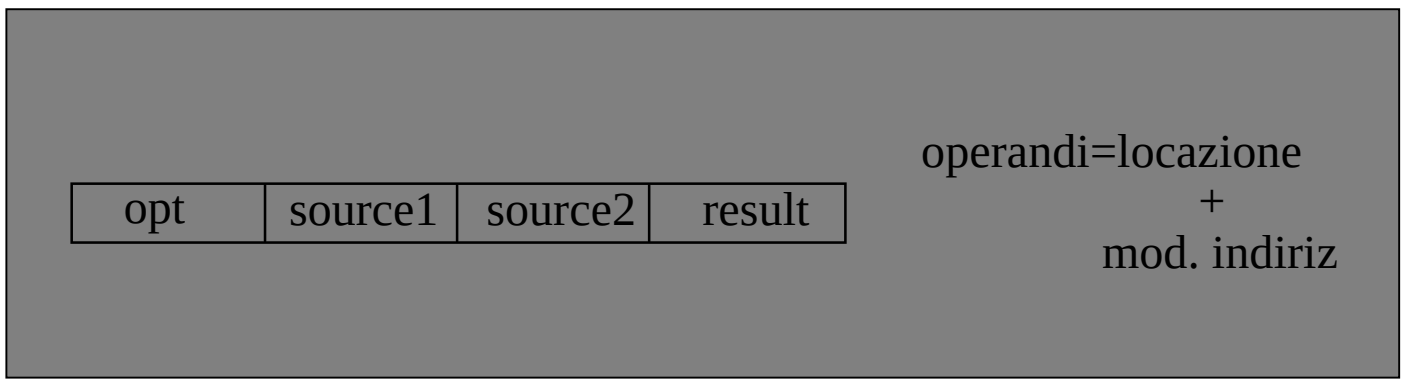
[0]**P**::= **Ds Cs**	
[1]**P**::= **Cs**	
[2]**D**::= **var ide O** [3]**O**$_1$::= **, ide O**$_2$ [4]**O**::=ε	?
[5]**Cs**$_1$::= **; C Cs**$_2$	
[6]**Cs**::= ε	
[7]**C**::= **A**	
[8]**C**::=**W**	
[9]**A**::= **ide := E**	**A**.code:=**E**.code||gen(**ide**.loc ':='**E**.loc)
[10]**W**::= **while E do C endw**	

Questa parte della grammatica non e' coinvolta nella traduzione

[11] **E** ::= **F E'**	**E**.loc:= **E'**.loc; **E'**.in:=**F**.loc, **E**.code:=(**F**.code || **E'**.code)
[12] **E'**$_1$::= **op-l F E'**$_2$	t:=newtemp,**E'**$_2$.in:=t, **E'**$_1$.loc:=**E'**$_2$.loc **E'**$_1$.code:=**F**.code||gen(t:=**E'**$_1$.in **op-l**.value **F**.loc)||**E'**$_2$.code
[13] **E'** ::= ε	**E**.code:=nop,**E'**.loc:=**E'**.in
[14] **F** ::= **T F'**	**F**.loc:= **F'**.loc; **F'**.in:=**T**.loc, **F**.code:=(**T**.code || **F'**.code)
[15] **F'**$_1$::= **op-h T F'**$_2$	t:=newtemp,**F'**$_2$.in:=t, **F'**$_1$.loc:=**F'**$_2$.loc **F'**$_1$.code:=**T**.code||gen(t:=**F'**$_1$.in **op-l**.value **T**.loc)||**F'**$_2$.code
[16] **F'** ::= ε	**F'**.code:=nop, **F'**.loc:=**F'**.in
[17] **T** ::= **num**	**T**.code:=nop,**T**.loc:=**num**.value
[18] **T** ::= **ide**	**T**.code:=nop,**T**.loc:=**ide**.loc
[19] **T** ::= **(E)**	**T**.code:=**E**.code, **T**.loc:=**E**.loc

come risulta il codice generato per
E.code allorche'
E derivi la frase (x+2)*(y-x)

il linguaggio target e' a 3 indirizzi
perche' ogni stm
e' rappresentabile con una word=32
bit secondo
il formato:



:=/+	x.loc	2	t1
:=/-	y.loc	x.loc	t2
:=/*	t1	t2	t3

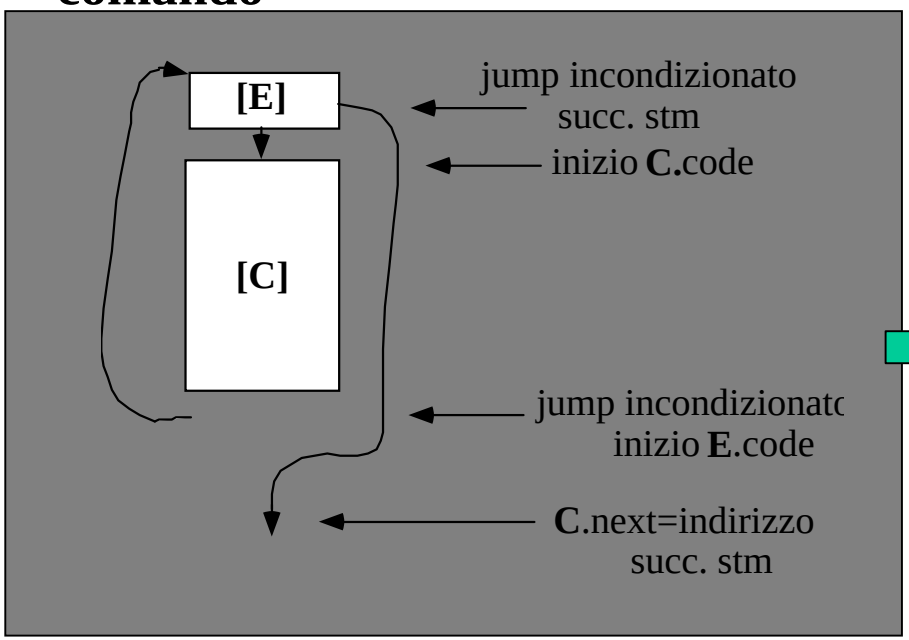
Osserva:

il linguaggio a tre indirizzi prevede assegnamenti differenti per differenti operatori di espressioni

Traduciamo l'iterazione di Semplice

[10]W::= while E do C endw

Schema del Controllo del comando



Operazioni nel meta

indirizzo del codice cui trasferire controllo

Piano di traduzione

code=sintetizzato per il codice prodotto

loc=sintetizzato per la locazione
[ogni sottoespressione calcola la locazione ove e' posto il risultato]

begin,next=sintetizzati per locazione
inizio codice
trasferimento controllo finale

[10]**W**::= **while E do C endw**

```
W.begin:=newtemp,W.next:=newtemp,
W.code:= gen(W.begin ':')||
          E.code||
          gen('if' E.loc '=' 'false' 'goto' W.next)||
          C.code||gen('goto'W.begin)
          gen(W.next ':')
```

Traduciamo la sequenzializzazione di Semplice

[5] **C_{S1} ::= ; C C_{S2}**

C_{S1}.code := **C**.code || **C_{S2}**.code

Come risulta il codice generato per C.code allorche' C derivi la frase:

```
while x=0
    do  y:=(x+2)*(y-x);
        x:=x-y endw
```

L1:	:=/=	x.loc	0	t1
	if/=	t1	false	L2
	:=/+	x.loc	2	t2
	:=/-	y.loc	x.loc	t3
	:=/*	t2	t3	y.loc
	:=/-	x.loc	y.loc	x.loc
	goto	L1		
L2:	?	?	?	?

Questo codice e' codice del nostro linguaggio intermedio?

sono previsti stm con label?