# IMPLEMENTAZIONE CAML DELLO STATO

$\rho : Ide \longrightarrow Loc_{\perp}$ $\quad$ $\mu : Loc \longrightarrow Val_{\perp}$

$Val = \underline{IN \cup B}$

```
type ide = string ;;
type val = ValN of int |
           ValB of bool |
           unknown
type mloc = int ;;
type amb = ide -> mloc Bottom ;;
type mem = mloc -> val bottom ;;
```

# ESPRESSIONI

Exp ::= Num | Ide | Exp Aop Exp | Exp Bop Exp |
( Exp ) | Uop Exp

Aop ::= + | - | * | / | %

/ → quoziente divisione intera

% → resto divisione intera

Bop ::= < | > | <= | >= | != | && | || | ==

Uop ::= !

! → not logico

Usiamo Exp per rappresentare il linguaggio generato da Exp

Per dire che "e è una stringa generabile da Exp"

scriviamo

$$e \in Exp$$

o ——— o

# SEMANTICA DELLE ESPRESSIONI

$$Sem_{exp} : Exp \to P \to M \to Val$$

↑ oggetto sintattico

definita da EXP PER CASI sulle alternative sintattiche

EXP ::= Num

$n \in Num$ → stringa generabile da Num

EXP
|
Num
△
n

$$Sem_{exp}\ n\ \rho\ \mu = val\ n$$

$val : \underline{Num} \longrightarrow \underline{\mathbb{N}}$

→ val 14 = 14

↑ stringa di simboli

↑ $\in \mathbb{N}$

Num ::= numeri romani

↖

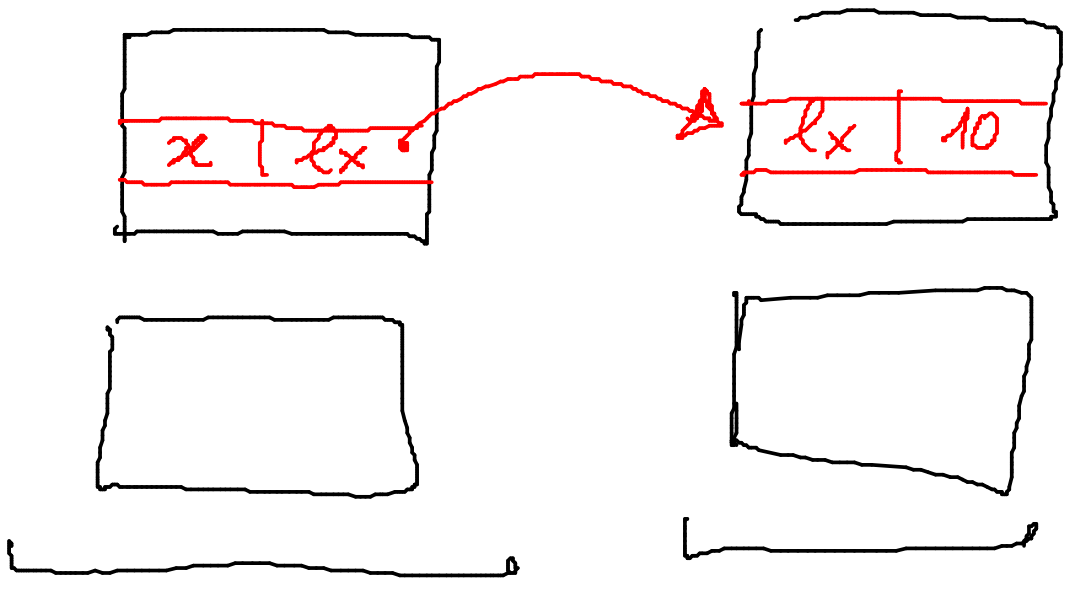
val XIV = 14

$Exp ::= Ide \qquad x \in Ide$

$$Sem_{exp}\ x\ \rho\ \mu = \mu(\rho\ x)$$

$\rho : Ide \to Loc_{\perp}$

$\mu : Loc \to Val_{\perp}$



$Exp ::= Exp\ Aop\ Exp$

$\ell_1$ op $\ell_2$

$Sem_{aop} : Aop \rightarrow Funzione$

$Sem_{aop}$ + = "somma tra numeri"

$Sem_{aop}$ * = "prodotto " "

IN CAML

$Sem_{aop}$ "+" = prefx + ;;

$Sem_{aop}$ "+" = prefx * ;;

$Sem_{exp}\ (\ell_1\ op\ \ell_2)\ \rho\ \mu =$

$(Sem_{aop}\ op)\ (Sem_{exp}\ \ell_1\ \rho\ \mu)$

$(Sem_{bop}\ op)\ (Sem_{exp}\ \ell_2\ \rho\ \mu)$

$Exp ::= Exp\ \ Bop\ Exp$

$Sem_{bop} : Bop \rightarrow$ Funzione

&& $\rightsquigarrow$ $\wedge$ logico

> $\rightsquigarrow$ maggiore tra numeri

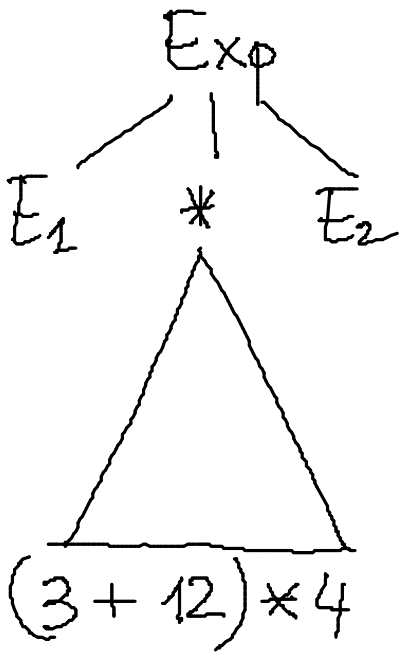
$$Exp ::= (Exp) \qquad (e) \in Exp$$

$$Sem_{exp}\ (e)\ \rho\ \mu = Sem_{exp}\ e\ \rho\ \mu$$

$$Exp ::= Uop\ Exp \qquad !\, e \in Exp$$

$$Sem_{exp}\ (!\, e)\ \rho\ \mu = not\ (Sem_{exp}\ e\ \rho\ \mu)$$

# VALUTAZIONE DI UNA EXP usando $Sem_{exp}$




$\langle \rho, \mu \rangle = \sigma$

$Sem_{exp}\ (3+12)*4\ \rho\ \mu$

=

$(Sem_{oop}\ *)\ (Sem_{exp}\ (3+12)\ \rho\ \mu)\ (Sem_{exp}\ 4\ \rho\ \mu)$

=

$(Sem_{oop}\ *)\ (Sem_{exp}\ 3+12\ \rho\ \mu)\ (val\ 4)$

=

$(Sem_{oop}\ *)\ ((Sem_{oop}\ +)\ (Sem_{exp}\ 3\ \rho\ \mu)\ (Sem_{exp}\ 12\ \rho\ \mu))\ (val\ 4)$

=

$= (\text{Sem}_{\text{oop}}\ *)\ ((\text{Sem}_{\text{oop}}\ +)\ 3\ 12)\ 4$

$=$ times $(\text{plus}\ 3\ 12)\ 4$

$=$ times $15\ 4$

$=$ 60

$Sem_{exp}$ $x + 10$ $\rho$ $\mu$

=

plus $(Sem_{exp}\ x\ \rho\ \mu)$ $(Sem_{exp}\ 10\ \rho\ \mu)$

=

plus $\underline{\mu(\rho\ x)}$ 10

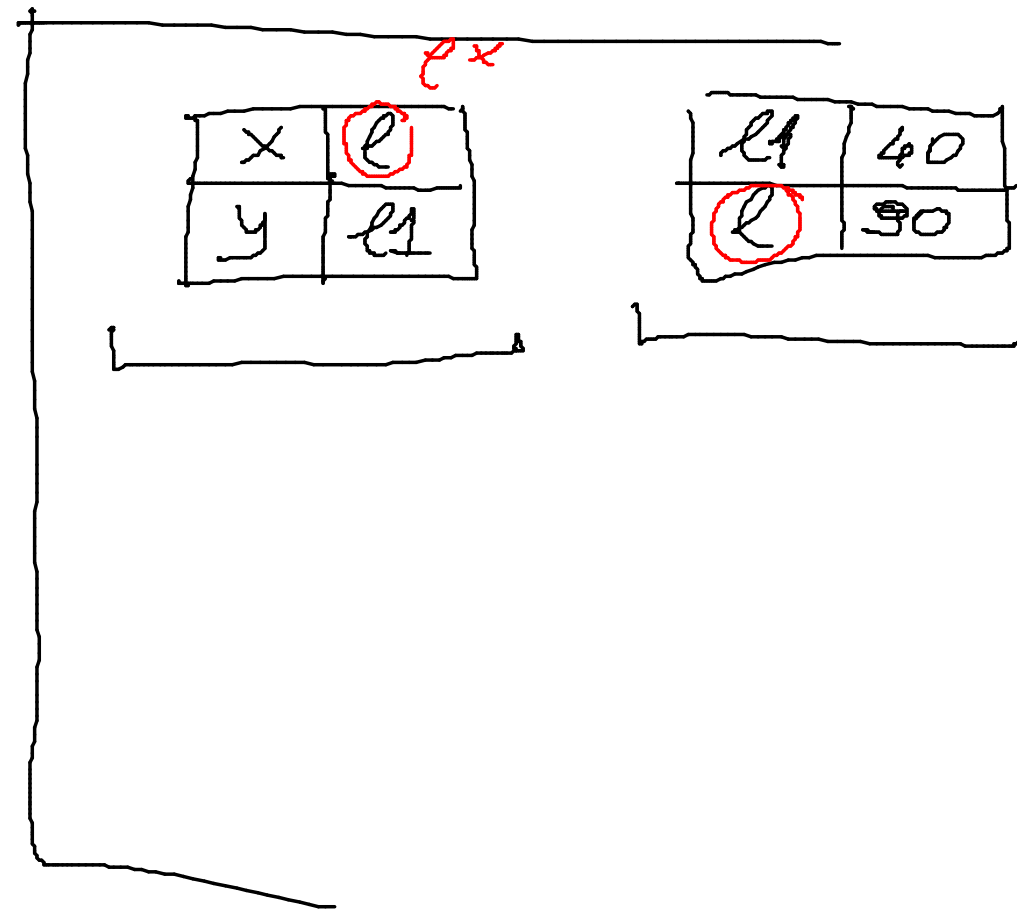
dipende dallo STATO

=

plus 90 10

=

100



# STRUTTURA DI UN PROGRAMMA

```
{ int x = 10;
  int y = 20; int z;

  x = x + 2;

  y = x + y;

  z = x * y;

  .
  .
  .
}
```

PARTE DICHIARATIVA
dichiarano i nomi che usiamo nel programma

PARTE ESECUTIVA

Sequenza di COMANDI
(assegnamenti, condizionali, iterazioni, blocchi ...)

Le dichiarazioni "costruiscono" lo stato INIZIALE

```
int x = 10;
int y = 20;
int z;
```

| x | l1 |
|---|---|
| z | l2 |
| y | l3 |

| l2 | ? |
|---|---|
| l3 | 20 |
| l1 | 10 |

? è un valore, ma non sappiamo quale.

In corrispondenza di ogni DICHIARAZIONE viene OCCUPATA una nuova locazione nel frame memoria, che viene associata (nel frame ambiente) al nome che si sta dichiarando.
Si esegue la parte esecutiva a partire dallo stato predisposto dalle dichiarazioni

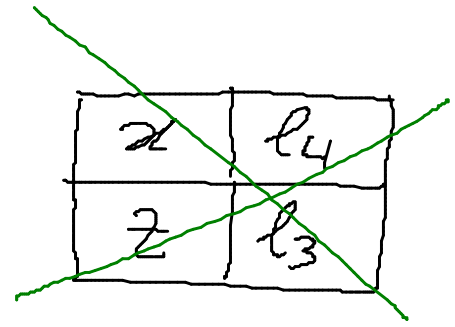
x = x+2;
y = x+y;
z = x*y;

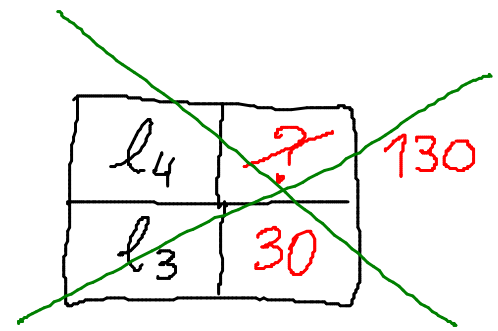
| x | $l_1$ |
|---|---|
| z | $l_2$ |
| y | $l_3$ |

| $l_1$ | ~~10~~ 12 |
|---|---|
| $l_3$ | ~~20~~ 32 |
| $l_2$ | ? 384 |

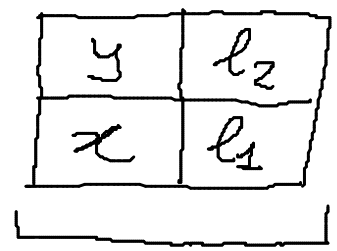
Blocco Principale

```
{ int x = 10; int y = 20;        <- DICH.

  x = x + 2; y = x + y;          <- PARTE ESEC.

  { int z = 30;                  <- BLOCCO ANNIDATO
    int x;

    x = z + 100;
    y = x + 200;
  }                              <- (*)

  y = x + 300;
  :
  :
}
```




312





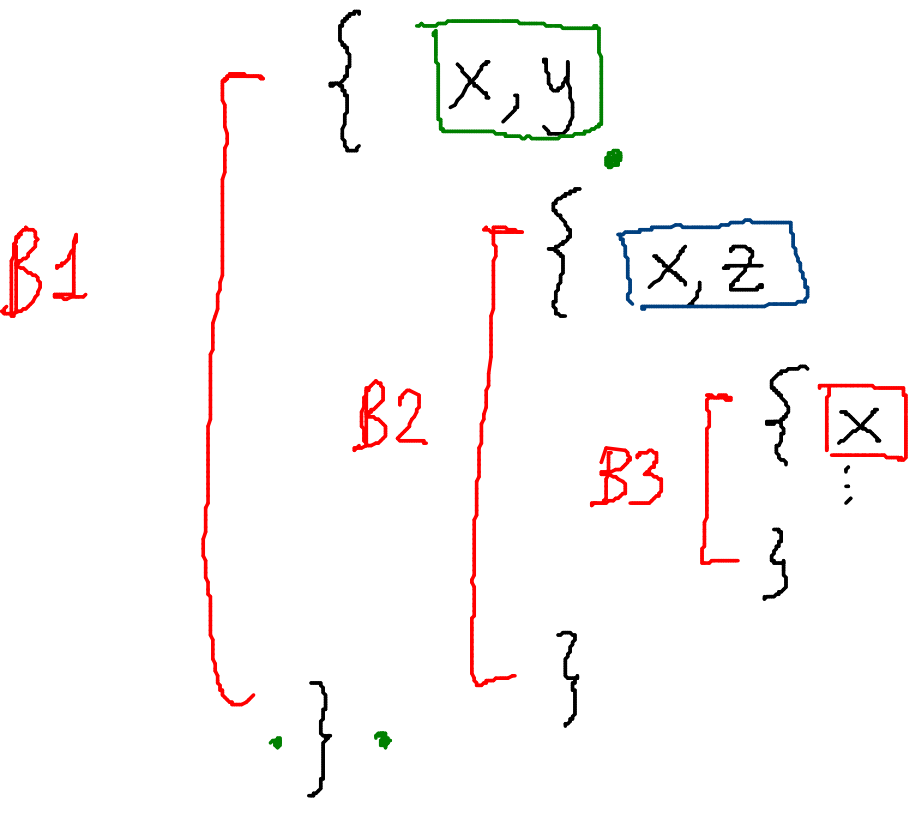
32 330

12

Dentro la parte esecutiva di un blocco
si possono usare variabili (nomi) dichiarate

1) nella parte dichiarativa del blocco

2) nella parte dichiarativa di blocchi ESTERNI
(blocchi in cui il blocco è annidato)




B2 è annidato in B1

B3 è " " in B2

in B1 posso usare x, y

in B2 " " x, z, y

in B3 x, z, y

{ x, y

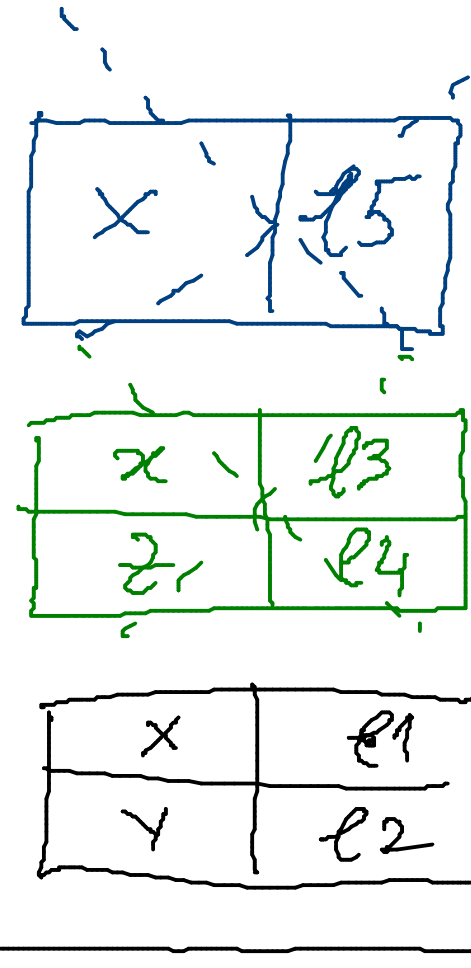
{ x, z

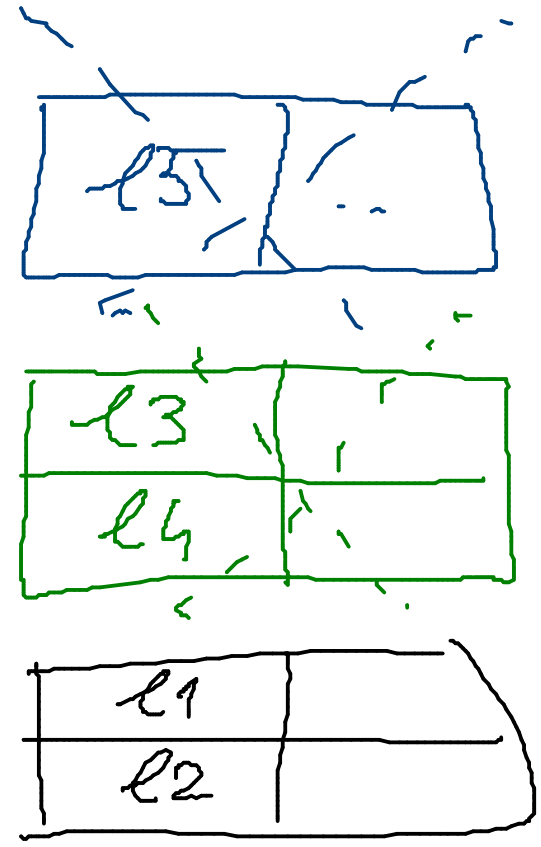
{ x

}

}

}



```
{ int x = 10;
  x = x + 2;
    { int z = 10;
      z = x + z;
    }
  z = z + x;
}
```

NO!

↑ ERRORE STATICO

```
let y = x + 1;;
```

↑ unbound

si può rilevare qualizzando la struttura del programma