Mao (Modello Astratto Operazionale)

Funzioni: dichiarazione, chiamata, passaggio dei parametri

Funzioni

Le funzioni ci servono per dare un nome ad un frammento di codice che calcola un valore in modo da poter riutilizzare il codice tutte le volte che vogliamo utilizzando parametri (input) diversi

Distinguiamo due momenti diversi:

- Definizione di funzione: il momento in cui scriviamo il codice dell'espressione che vogliamo calcolare, cioè il codice che vogliamo riutilizzare specificando l'input parametrico che dovrà utilizzare (parametri formali)

- Invocazione di funzione: il momento nel programma in cui chiamo la funzione per eseguire il calcolo dell'espressione specificata nella sostituzione di funzione, tale calcolo avviene su valori effettivi (parametri attuali)

Dichiarazione di funzione

int è il tipo restituito dalla funzione: tipo del valore di ritorno

max è l'identificatore che associamo al codice della funzione: nome di funzione

a e b sono i placeholder per i valori su cui la funzione deve essere eseguita: parametri formali
Rendono la funzione parametrica: questa funzione si aspetta due interi

```
int max(int a, int b) {
    int m = a;
    if (b>m) { m := b; }
    return m;
}
```

m è il valore calcolato dalla funzione: valore di ritorno
verrà restituito al codice che invoca la funzione

Questa funzione si aspetta due interi e restituisce un intero che è il massimo tra i due valori in modo da poter riutilizzare il codice tutte le volte che vogliamo, scegliendo parametri (input) diversi

Invocazione di funzione

Una volta definita una funzione, la possiamo chiamare usando il suo nome e passandogli gli input sui quali vogliamo calcolare il risultato

Inoltre, la possiamo invocare quante volte vogliamo, passandogli di volta in volta parametri diversi

```
if (max(x,y)<10) {
      z := max(x+2,y*3);
} else {
      z := max(x/10,y-10);
}
```

Invocazione di funzione

Proviamo a scrivere un programma per calcolare il massimo tra tre valori x,y,z usando la nuova funzione che abbiamo appena definito

Invocazione di funzione

Proviamo a scrivere un programma per calcolare il massimo tra tre valori x,y,z usando la nuova funzione che abbiamo appena definito

```
int mx;
mx := max(x,y);
mx := max(mx,z);
```

Control Flow nell'invocazione di funzione

Cosa succede quando nel corpo del programma invochiamo una funzione?

quando si incontra la chiamata di funzione il controllo passa
ad eseguire il codice della definizione
passando opportunamente i parametri (prossima slide)

```
int mx;
mx:= max(x,y);
```

```
int max(int a,int b)
{    int m = a;
     if (b>m) { m := b; }
     return m;
}
```

```
mx:= max(mx,z);
```

```
int max(int a,int b)
{    int m = a;
     if (b>m) { m := b; }
     return m;
}
```

questo succede ad ogni invocazione di funzione

Funzioni

passaggio dei parametri

Corrispondenza tra parametri formali e attuali

Come fa la chiamata di funzione a comunicare gli argomenti alla funzione?

La chiamata di funzione deve contenere espressioni corrispondenti in numero e in tipo a quelle dichiarate nell'intestazione della funzione

La comunicazione avviene tramite il passaggio di parametri

Ad ogni parametro formale della funzione deve corrispondere un'espressione nella stessa posizione tra i parametri attuali

Queste invocazioni della funzione max sono corrette?

```
mx:=max(12+3,2==3);
```

```
mx:=max(12,x,y);
```

Passaggio dei parametri per valore

Come vengono passati gli argomenti su cui applicare la funzione?

Il passaggio per valore è la modalità più comune: ogni parametro formale viene inizializzato con una copia del valore del corrispondente parametro attuale

Se i parametri formali vengono modificati all'interno della funzione, tali modifiche non influenzano i parametri attuali originali

Definizione di funzione in Pascal

```
// passaggio per valore
function inc(a: integer):integer
begin
      a := a+1;
      inc := a;
end
```

Chiamata di funzione in Pascal

```
b := inc(c);
```

Se c valeva 2 qual è il valore di b dopo la chiamata?
E quello di c?

Passaggio dei parametri per riferimento

Un'altra modalità di passaggio di parametri è quella per riferimento:
si inizializzano i parametri formali con il riferimento ai corrispondenti argomenti attuali

Se i parametri formali vengono modificati all'interno della funzione, tale modifica influenza i parametri attuali originali

Definizione di funzione in Pascal

```
// passaggio per riferimento
function inc(var a: integer):integer
begin
      a := a+1;
      inc := a;
end
```

Chiamata di funzione in Pascal

```
b := inc(c);
```

Se c valeva 2 qual è il valore di b dopo la chiamata?
E quello di c?

Passaggio dei parametri in Mao

In Mao (come in JavaScript):
i tipi base (interi, booleani e caratteri) vengono passati per valore,
mentre per il tipo array per riferimento: si copia l'indirizzo base l_b dell'array,
creando un alias per l'intero array

Attenzione!! Quando passiamo un array come argomento attuale ad una funzione stiamo implicitamente passando il riferimento all'intero array stesso, non una copia dei suoi elementi

Le modifiche apportate all'array all'interno della funzione influenzano l'array originale passato come argomento attuale

Passaggio dei parametri

Rivediamo l'esempio precedente (con i parametri passati per valore)

i parametri formali sono visibili solo all'interno della funzione!

```
int mx;
mx:= max(x,y);
```

```
int max(int a,int b)
{    int m = a;
     if (b>m) { m := b; }
     return m;
}
```

```
mx:= max(mx,z);
```

```
int max(int a,int b)
{    int m = a;
     if (b>m) { m := b; }
     return m;
}
```

Passaggio dei parametri

Rivediamo l'esempio precedente (con i parametri passati per valore)

i parametri formali sono visibili solo all'interno della funzione!

```
int mx;
mx:= max(x,y);
```

```
{   int a,int b = x,y;
    int m = a;
    if (b>m) { m := b; }
    return m;
}
```

```
mx:= max(mx,z);
```

```
{   int a,int b = mx,z;
    int m = a;
    if (b>m) { m := b; }
    return m;
}
```

Passaggio dei parametri

Scriviamo una funzione che preso un array e un intero i restituisca il valore attualmente presente in posizione i azzerandolo

Passaggio dei parametri

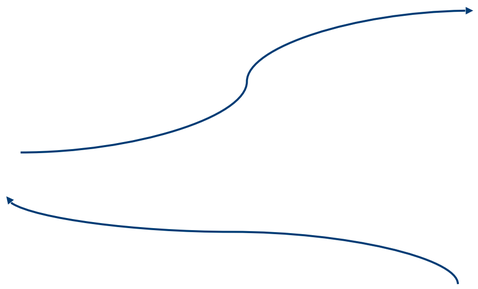
Scriviamo una funzione che preso un array e un intero i restituisca il valore attualmente presente in posizione i azzerandolo

```
// a viene passato per riferimento (all'array)
// i viene passato per valore (tipo base)
int estrai(int[] a, int i) {
     int v := a[i];
     a[i] := 0;
     return v;
}
```

Passaggio dei parametri

Rivediamo l'esempio precedente con il parametro array passato per riferimento

```
int[] b = [10,2];
int val := estrai(b,0);
```



```
{   int[] a,int i = b,0;
    int v := a[i];
    a[i] := 0;
    return v;
}
```

```
// a viene passato per riferimento (all'array)
// i viene passato per valore (tipo base)
int estrai(int[] a, int i) {
     int v := a[i];
     a[i] := 0;
     return v;
}
```

Funzioni

sintassi

Sintassi della definizione di funzione

Sintassi della dichiarazione di una funzione, dove solitamente l'espressione E = {C return E';}

$$C ::= \ldots \mid T\ Id(Ds)\ E$$

T è il tipo di ritorno, Id è il nome della funzione, $Ds = T_1\ Id_1, \ldots, T_n\ Id_n$ è la lista dei parametri formali e {C return E';} è l'espressione che determina il valore restituito dalla funzione, detta corpo della funzione

La dichiarazione di funzione associa l'identificatore Id al corpo della funzione

La lista di parametri formali introduce l variabili locali che sono visibili solo all'interno della funzione e non possono essere utilizzate al di fuori di essa.

Esempio: clonazione di array

Scrivere una funzione che clona un array di interi

```
int[] clone(int[] v) {
      int[] a = new int[v.length];
      int i = 0;
      while (i<v.length) {
            a[i] := v[i];
            i:=i+1;
      }
      return a;
}
```

Esempio: test ordinato

Scrivere una funzione che controlla se gli elementi di un array sono ordinati in maniera strettamente crescente

```
bool ordinato(int[] v) {
     bool ord = true;
     int i = 0;
     while ((i < a.length-1) && ord)  {
          ord := (a[i] < a[i+1]);
          i := i + 1;
     }
     return ord;
}
```

Esempio: calcolo del massimo

Scrivere una funzione che restituisce il massimo in un array

```
int max(int[] v) {
      int m = v[0];
      int i = 1;
      while (i < v.length)  {
            if (m < v[i]) { m := v[i]; }
            i := i + 1;
      }
      return m;
}
```

Sintassi della chiamata di funzione

E ::= ... | Id(Es)

L'invocazione di una funzione è un'espressione che, dopo avere instanziato i parametri formali della funzione Id con i valori degli argomenti attuali Es = E_1, …, E_n, restituisce il valore ottenuto valutando il corpo della funzione

Esercizio

Scrivere una funzione che dati n, m (con $n \leq m$) calcoli la somma degli interi nell'intervallo [n,m]

Esercizio

Scrivere una funzione che dati n, m (con $n \leq m$) calcoli la somma degli interi nell'intervallo [n,m]

```
int somma_int(int n, int m) {
    int somma=0;
    while(n<=m){
        somma := somma+n;
        n:=n+1;
    }
    return somma;
}
```

Esercizio

Usare la precedente funzione per calcolare l'ennesimo numero triangolare

Esercizio

Usare la precedente funzione per calcolare l'ennesimo numero triangolare

```
int tri := somma_int(0,n);
```

Esercizio

Scrivere una funzione che dato un array e un intero m restituisca il numero di occorrenze di m nell'array

Esercizio

Scrivere una funzione che dato un array e un intero m restituisca il numero di occorrenze di m nell'array

```
int num_occ(int[] a, int m) {
      int i = 0;
      int occ = 0;
      while (i<a.length) {
            if (a[i]==m) { occ:=occ+1; }
            i:=i+1;
      }
      return occ;
}
```

Esercizio

Utilizzando la funzione precedente scrivere un programma che, dato un array a e un intero m, controlli che a contenga almeno due occorrenze del valore m

```
bool controllo=(num_occ(a,m)>1);
```

Funzioni

sistema di tipi

Controllo dei tipi per le funzioni

Dobbiamo estendere i tipi per poterli associare alle funzioni.

Definiamo un tipo di funzione con la sintassi

$$(T_1 \times \cdots \times T_n) \rightarrow T$$

Diamo anche la definizione di variabili libere di una funzione

$$fv(\ T\ Id(T_1\ Id_1,\ \ldots,\ T_n\ Id_n)\ E\ ) = fv(E) \setminus \{\ Id_1,\ \ldots,\ Id_n\ \}$$

Controllo dei tipi per le funzioni

$C ::= \ldots \mid T\ Id(T_1\ Id_1, \ldots, T_n\ Id_n)\ E$

Assumeremo sempre che i parametri formali $Id_1, \ldots, Id_n$ siano tutti distinti

Per controllare il tipo di questa comando ci serve di assumere nell'ambiente di tipo le associazioni tra parametri formali e loro tipi e con tali assunzioni riuscire a tipare il corpo E con il tipo T

qui stiamo richiedendo che il corpo non abbia variabili libere tranne i parametri formali.
Con la ricorsione rilasseremo questa ipotesi

$$\frac{\{\ T_1\ Id_1, \ldots, T_n\ Id_n\ \} \vdash E : T}{\Gamma \vdash T\ Id\ (T_1\ Id_1, \ldots, T_n\ Id_n)\ E : \{\ Id : (T_1 \times \cdots \times T_n) \rightarrow T\ \}}$$

Controllo dei tipi per l'invocazione

E ::= ... | Id(E_1, ..., E_n)

Per quanto riguarda l'invocazione di una funzione, dobbiamo controllare che il numero e i tipi degli argomenti forniti corrispondano a quelli definiti nella funzione

$$\frac{\Gamma(\mathrm{Id}) = (T_1 \times \cdots \times T_n) \rightarrow T \qquad \forall i \in [1,n]\ \Gamma \vdash E_i : T_i}{\Gamma \vdash \mathrm{Id}(E_1, \ldots, E_n) : T}$$

Esercizio

Scrivere una funzione che dato un intero restituisca il suo valore assoluto

```
int abs(int m) {
      int n = m;
      if (n<0) { n := -n; }
      return n;
}
```

Esercizio

Controllare che il comando

```
C = int abs(int m) { int n=m; if (n<0) {n:=-n;} return n; }
```

sia ben tipato nell'ambiente di tipo $\Gamma = \varnothing$

Esercizio

Controllare che il comando

```
C = int abs(int m) { int n=m; if (n<0) {n:=-n;} return n; }
```

sia ben tipato nell'ambiente di tipo $\Gamma = \emptyset$

$$\frac{\{ T_1\ Id_1, \ldots, T_n\ Id_n \} \vdash E : T}{\Gamma \vdash T\ Id\ (T_1\ Id_1, \ldots, T_n\ Id_n)\ E : \{ Id : (T_1 \times \cdots \times T_n) \to T \}}$$

$\emptyset \vdash$ C : { abs: $int \to int$ }

{ m : int } ⊢ { int n=m; if (n<0) {n:= -n;} return n; } : int

Esercizio

Controllare che il comando

```
C = int abs(int m) { int n=m; if (n<0) {n:=-n;} return n; }
```

sia ben tipato nell'ambiente di tipo $\Gamma = \emptyset$

$\emptyset \vdash$ C : { abs: $int \rightarrow int$ }

{ m : int } ⊢ { int n=m; if (n<0) {n:=-n;} return n; } : int

$$\frac{\Gamma \vdash C : \Gamma_1 \quad \Gamma[\Gamma_1] \vdash E : T}{\Gamma \vdash \{\, C \text{ return } E; \,\} : T}$$

Esercizio

Controllare che il comando

```
C = int abs(int m) { int n=m; if (n<0) {n:=-n;} return n; }
```

sia ben tipato nell'ambiente di tipo $\Gamma = \emptyset$

$\emptyset \vdash$ C : { abs: $int \rightarrow int$ }

{ m : int } ⊢ { int n=m; if (n<0) {n:=-n;} return n; } : int

{ m : int } ⊢ int n=m; if (n<0) {n:= -n;} : $\Gamma_{1?}$

$$\frac{\Gamma \vdash C : \Gamma_1 \quad \Gamma[\Gamma_1] \vdash E : T}{\Gamma \vdash \{ C \text{ return } E; \} : T}$$

{ m : int }[$\Gamma_{1?}$] ⊢ n : int

Esercizio

Controllare che il comando

```
C = int abs(int m) { int n=m; if (n<0) {n:=-n;} return n; }
```

sia ben tipato nell'ambiente di tipo $\Gamma = \emptyset$

$\emptyset \vdash$ C : { abs: $int \rightarrow int$ }

{ m : int } ⊢ { int n=m; if (n<0) {n:=-n;} return n; } : int

{ m : int } ⊢ int n=m; if (n<0) {n:= -n;} : $\Gamma_{1?}$

$$\frac{\Gamma \vdash C_1 : \Gamma_1 \quad \Gamma[\Gamma_1] \vdash C_2 : \Gamma_2}{\Gamma \vdash C_1\ C_2 : \Gamma_1[\Gamma_2]}$$

{ m : int }[$\Gamma_{1?}$] ⊢ n : int

Esercizio

Controllare che il comando

C = `int abs(int m) { int n=m; if (n<0) {n:=-n;} return n; }`

sia ben tipato nell'ambiente di tipo $\Gamma = \emptyset$

$\emptyset \vdash$ C : { abs: $int \to int$ }

{ m : int } ⊢ { int n=m; if (n<0) {n:= -n;} return n; } : int

{ m : int } ⊢ int n=m; if (n<0) {n:= -n;} : $\Gamma_{1?}$ con $\Gamma_{1?} = \Gamma_{2?}[\Gamma_{3?}]$

{ m : int } ⊢ int n=m; : $\Gamma_{2?}$

{ m : int }[$\Gamma_{2?}$] ⊢ if (n<0) {n:= -n;} : $\Gamma_{3?}$

$$\frac{\Gamma \vdash C_1 : \Gamma_1 \quad \Gamma[\Gamma_1] \vdash C_2 : \Gamma_2}{\Gamma \vdash C_1\ C_2 : \Gamma_1[\Gamma_2]}$$

{ m : int }[$\Gamma_{1?}$] ⊢ n : int

Esercizio

Controllare che il comando

```
C = int abs(int m) { int n=m; if (n<0) {n:=-n;} return n; }
```

sia ben tipato nell'ambiente di tipo $\Gamma = \emptyset$

$\emptyset \vdash$ C : { abs: $int \rightarrow int$ }

{ m : int } ⊢ { int n=m; if (n<0) {n:= -n;} return n; } : int

{ m : int } ⊢ int n=m; if (n<0) {n:= -n;} : $\Gamma_{1?}$ con $\Gamma_{1?} = \Gamma_{2?}[\Gamma_{3?}]$

{ m : int } ⊢ int n=m; : $\Gamma_{2?}$ con $\Gamma_{2?}$ = { n : int } ✓

{ m : int }[$\Gamma_{2?}$] ⊢ if (n<0) {n:= -n;} : $\Gamma_{3?}$

{ m : int }[$\Gamma_{1?}$] ⊢ n : int

Esercizio

Controllare che il comando

```
C = int abs(int m) { int n=m; if (n<0) {n:=-n;} return n; }
```

sia ben tipato nell'ambiente di tipo $\Gamma = \emptyset$

$\emptyset \vdash$ C : { abs: $int \rightarrow int$ }

{ m : int } ⊢ { int n=m; if (n<0) {n:= -n;} return n; } : int

{ m : int } ⊢ int n=m; if (n<0) {n:= -n;} : $\Gamma_{1?}$ con $\Gamma_{1?}$ = { n : int }[$\Gamma_{3?}$]

{ m : int } ⊢ int n=m; : { n : int } ✓

{ m : int }[n : int] ⊢ if (n<0) {n:= -n;} : $\Gamma_{3?}$

{ m : int }[$\Gamma_{1?}$] ⊢ n : int

Esercizio

Controllare che il comando

```
C = int abs(int m) { int n=m; if (n<0) {n:=-n;} return n; }
```

sia ben tipato nell'ambiente di tipo $\Gamma = \emptyset$

$\emptyset \vdash$ C : { abs: $int \rightarrow int$ }

{ m : int } ⊢ { int n=m; if (n<0) {n:= -n;} return n; } : int

{ m : int } ⊢ int n=m; if (n<0) {n:= -n;} : $\Gamma_{1?}$ con $\Gamma_{1?}$ = { n : int }[$\Gamma_{3?}$]

{ m : int } ⊢ int n=m; : { n : int } ✓

{ m : int }[n : int] ⊢ if (n<0) {n:= -n;} : $\Gamma_{3?}$

$$\frac{\Gamma \vdash E : bool \quad \Gamma \vdash C_1 : \Gamma_1 \quad \Gamma \vdash C_2 : \Gamma_2}{\Gamma \vdash \text{if (E) } \{C_1\} \text{ else } \{C_2\} : \emptyset}$$

{ m : int }[$\Gamma_{1?}$] ⊢ n : int

Esercizio

Controllare che il comando

C = `int abs(int m) { int n=m; if (n<0) {n:=-n;} return n; }`

sia ben tipato nell'ambiente di tipo $\Gamma = \emptyset$

$\emptyset \vdash$ C : { abs: $int \rightarrow int$ }

{ m : int } ⊢ { int n=m; if (n<0) {n:= -n;} return n; } : int

{ m : int } ⊢ int n=m; if (n<0) {n:= -n;} : $\Gamma_{1?}$ con $\Gamma_{1?}$ = { n : int }[$\Gamma_{3?}$]

{ m : int } ⊢ int n=m; : { n : int } ✓

{ m : int }[n : int] ⊢ if (n<0) {n:= -n;} : $\Gamma_{3?}$ con $\Gamma_{3?} = \emptyset$

{ m : int }[n : int] ⊢ n<0 : bool

{ m : int }[n : int] ⊢ n:= -n; : $\Gamma_{4?}$

{ m : int }[$\Gamma_{1?}$] ⊢ n : int

$$\frac{\Gamma \vdash E : bool \quad \Gamma \vdash C_1 : \Gamma_1 \quad \Gamma \vdash C_2 : \Gamma_2}{\Gamma \vdash \text{if (E) } \{C_1\} \text{ else } \{C_2\} : \emptyset}$$

Esercizio

Controllare che il comando

```
C = int abs(int m) { int n=m; if (n<0) {n:=-n;} return n; }
```

sia ben tipato nell'ambiente di tipo $\Gamma = \emptyset$

$\emptyset \vdash$ C : { abs: $int \rightarrow int$ }

{ m : int } ⊢ { int n=m; if (n<0) {n:= -n;} return n; } : int

{ m : int } ⊢ int n=m; if (n<0) {n:= -n;} : $\Gamma_{1?}$ con $\Gamma_{1?}$ = { n : int }[$\emptyset$]

{ m : int } ⊢ int n=m; : { n : int } ✓

{ m : int }[n : int] ⊢ if (n<0) {n:= -n;} : $\emptyset$

{ m : int }[n : int] ⊢ n<0 : bool

{ m : int }[n : int] ⊢ n:= -n; : $\Gamma_{4?}$

{ m : int }[$\Gamma_{1?}$] ⊢ n : int

Esercizio

Controllare che il comando

```
C = int abs(int m) { int n=m; if (n<0) {n:=-n;} return n; }
```

sia ben tipato nell'ambiente di tipo $\Gamma = \emptyset$

$\emptyset \vdash$ C : { abs: $int \rightarrow int$ }

{ m : int } ⊢ { int n=m; if (n<0) {n:= -n;} return n; } : int

{ m : int } ⊢ int n=m; if (n<0) {n:= -n;} : $\Gamma_{1?}$ con $\Gamma_{1?}$ = { n : int }[$\emptyset$]

{ m : int } ⊢ int n=m; : { n : int } ✓

{ m : int }[n : int] ⊢ if (n<0) {n:= -n;} : $\emptyset$

{ m : int }[n : int] ⊢ n<0 : bool ✓

{ m : int }[n : int] ⊢ n:= -n; : $\Gamma_{4?}$

{ m : int }[$\Gamma_{1?}$] ⊢ n : int

Esercizio

Controllare che il comando

```
C = int abs(int m) { int n=m; if (n<0) {n:=-n;} return n; }
```

sia ben tipato nell'ambiente di tipo $\Gamma = \emptyset$

$\emptyset \vdash$ C : { abs: $int \rightarrow int$ }

{ m : int } ⊢ { int n=m; if (n<0) {n:= -n;} return n; } : int

{ m : int } ⊢ int n=m; if (n<0) {n:= -n;} : $\Gamma_{1?}$ con $\Gamma_{1?}$ = { n : int }[$\emptyset$]

{ m : int } ⊢ int n=m; : { n : int } ✓

{ m : int }[n : int] ⊢ if (n<0) {n:= -n;} : $\emptyset$

{ m : int }[n : int] ⊢ n<0 : bool ✓

{ m : int }[n : int] ⊢ n:= -n; : $\Gamma_{4?}$ con $\Gamma_{4?}$ = $\emptyset$ ✓

{ m : int }[$\Gamma_{1?}$] ⊢ n : int

Esercizio

Controllare che il comando

C = `int abs(int m) { int n=m; if (n<0) {n:=-n;} return n; }`

sia ben tipato nell'ambiente di tipo $\Gamma = \emptyset$

$\emptyset \vdash$ C : { abs: $int \rightarrow int$ }

{ m : int } ⊢ { int n=m; if (n<0) {n:= -n;} return n; } : int

{ m : int } ⊢ int n=m; if (n<0) {n:= -n;} : $\Gamma_{1?}$ con $\Gamma_{1?}$ = { n : int }[$\emptyset$]

{ m : int } ⊢ int n=m; : { n : int } ✓

{ m : int }[n : int] ⊢ if (n<0) {n:= -n;} : $\emptyset$ ✓

{ m : int }[n : int] ⊢ n<0 : bool ✓

{ m : int }[n : int] ⊢ n:= -n; : $\emptyset$ ✓

{ m : int }[$\Gamma_{1?}$] ⊢ n : int

Esercizio

Controllare che il comando

```
C = int abs(int m) { int n=m; if (n<0) {n:=-n;} return n; }
```

sia ben tipato nell'ambiente di tipo $\Gamma = \emptyset$

$\emptyset \vdash$ C : { abs: $int \rightarrow int$ }

{ m : int } ⊢ { int n=m; if (n<0) {n:= -n;} return n; } : int

{ m : int } ⊢ int n=m; if (n<0) {n:= -n;} : { n : int } ✓

{ m : int } ⊢ int n=m; : { n : int } ✓

{ m : int }[n : int] ⊢ if (n<0) {n:= -n;} : ∅ ✓

{ m : int }[n : int] ⊢ n<0 : bool ✓

{ m : int }[n : int] ⊢ n:= -n; : ∅ ✓

{ m : int }[n : int] ⊢ n : int

Esercizio

Controllare che il comando

```
C = int abs(int m) { int n=m; if (n<0) {n:=-n;} return n; }
```

sia ben tipato nell'ambiente di tipo $\Gamma = \emptyset$

$\emptyset \vdash$ C : { abs: $int \rightarrow int$ }

{ m : int } ⊢ { int n=m; if (n<0) {n:= -n;} return n; } : int

{ m : int } ⊢ int n=m; if (n<0) {n:= -n;} : { n : int } ✓

{ m : int } ⊢ int n=m; : { n : int } ✓

{ m : int }[n : int] ⊢ if (n<0) {n:= -n;} : ∅ ✓

{ m : int }[n : int] ⊢ n<0 : bool ✓

{ m : int }[n : int] ⊢ n:= -n; : ∅ ✓

{ m : int }[n : int] ⊢ n : int ✓

Esercizio

Controllare che il comando

```
C = int abs(int m) { int n=m; if (n<0) {n:=-n;} return n; }
```

sia ben tipato nell'ambiente di tipo $\Gamma = \emptyset$

$\emptyset \vdash$ C : { abs: $int \rightarrow int$ }

{ m : int } ⊢ { int n=m; if (n<0) {n:= -n;} return n; } : int ✓

{ m : int } ⊢ int n=m; if (n<0) {n:= -n;} : { n : int } ✓

{ m : int } ⊢ int n=m; : { n : int } ✓

{ m : int }[n : int] ⊢ if (n<0) {n:= -n;} : ∅ ✓

{ m : int }[n : int] ⊢ n<0 : bool ✓

{ m : int }[n : int] ⊢ n:= -n; : ∅ ✓

{ m : int }[n : int] ⊢ n : int ✓

Esercizio

Controllare che il comando

```
C = int abs(int m) { int n=m; if (n<0) {n:=-n;} return n; }
```

sia ben tipato nell'ambiente di tipo $\Gamma = \emptyset$

$\emptyset \vdash$ C : { abs: $int \to int$ } ✓

{ m : int } ⊢ { int n=m; if (n<0) {n:= -n;} return n; } : int ✓

{ m : int } ⊢ int n=m; if (n<0) {n:= -n;} : { n : int } ✓

{ m : int } ⊢ int n=m; : { n : int } ✓

{ m : int }[n : int] ⊢ if (n<0) {n:= -n;} : $\emptyset$ ✓

{ m : int }[n : int] ⊢ n<0 : bool ✓

{ m : int }[n : int] ⊢ n:= -n; : $\emptyset$ ✓

{ m : int }[n : int] ⊢ n : int ✓

Esercizio

Dato l'ambiente di tipo restituito dalla definizione

C = `int abs(int m) {int n=m; if (n<0) {n:= -n;} return n;}`

controllare il tipo dell'espressione

E = `abs(-3)`

Per la definizione C abbiamo calcolato l'ambiente Γ={ abs: $int \rightarrow int$ } quindi

$\Gamma \vdash$ abs(-3) : $T_?$

$$\frac{\Gamma(Id) = (T_1 \times \cdots \times T_n) \rightarrow T \qquad \forall i \in [1,n]\ \Gamma \vdash E_i : T_i}{\Gamma \vdash Id(E_1, \ldots, E_n) : T}$$

Esercizio

Dato l'ambiente di tipo restituito dalla definizione

C = `int abs(int m) {int n=m; if (n<0) {n:= -n;} return n;}`

controllare il tipo dell'espressione

E = `abs(-3)`

Per la definizione C abbiamo calcolato l'ambiente Γ={ abs: $int \rightarrow int$ } quindi

$\Gamma \vdash$ abs(-3) : $T_? = int$

Γ (abs) = $int \rightarrow int$

$\Gamma \vdash$ -3 : int

$$\frac{\Gamma(Id) = (T_1 \times \cdots \times T_n) \rightarrow T \qquad \forall i \in [1,n]\ \Gamma \vdash E_i : T_i}{\Gamma \vdash Id(E_1, \ldots, E_n) : T}$$

Esercizio

Dato l'ambiente di tipo restituito dalla definizione

C = `int abs(int m) {int n=m; if (n<0) {n:= -n;} return n;}`

controllare il tipo dell'espressione

E = `abs(-3)`

Per la definizione C abbiamo calcolato l'ambiente Γ={ abs: $int \rightarrow int$ } quindi

$\Gamma \vdash$ abs(-3) : $T_? = int$

Γ (abs) = $int \rightarrow int$

$\Gamma \vdash$ -3 : int ✓

$$\frac{\Gamma(Id) = (T_1 \times \cdots \times T_n) \rightarrow T \qquad \forall i \in [1,n]\ \Gamma \vdash E_i : T_i}{\Gamma \vdash Id(E_1, \ldots, E_n) : T}$$

Esercizio

Dato l'ambiente di tipo restituito dalla definizione

C = `int abs(int m) {int n=m; if (n<0) {n:= -n;} return n;}`

controllare il tipo dell'espressione

E = `abs(-3)`

Per la definizione C abbiamo calcolato l'ambiente Γ={ abs: *int* → *int* } quindi

$\Gamma \vdash$ abs(-3) : *int* ✓

Γ (abs) = *int* → *int*

$\Gamma \vdash$ -3 : *int* ✓

$$\frac{\Gamma(Id) = (T_1 \times \cdots \times T_n) \to T \qquad \forall i \in [1,n]\ \Gamma \vdash E_i : T_i}{\Gamma \vdash Id(E_1, \ldots, E_n) : T}$$

Funzioni

semantica operazionale

Semantica operazionale delle funzioni

Prima delle regole dobbiamo definire la rappresentazione delle funzioni nell'ambiente:
ad ogni identificatore di funzione associamo la rappresentazione della funzione

Usiamo la seguente rappresentazione

$$\lambda\ (Ds).\ E$$

che codifica, in qualche modo opportuno, la lista dei parametri formali e il corpo della funzione

Semantica operazionale della definizione

C ::= ... | T Id(T$_1$ Id$_1$, ..., T$_n$ Id$_n$) E

La regola per la dichiarazione di funzione aggiunge l'associazione
tra l'identificatore della funzione e la sua rappresentazione nell'ambiente ρ,
senza modificare la memoria σ

$$\frac{}{\langle \text{T Id (Ds) E} , \rho , \sigma \rangle \rightarrow (\rho[\text{Id} \mapsto \lambda(\text{Ds}).\text{E}], \sigma)}$$

Semantica operazionale della chiamata

E ::= ... | Id(E_1, ..., E_n)

Per valutare la chiamata dobbiamo

- trovare in ρ l'associazione tra l'identificatore della funzione e la sua rappresentazione,
- legare la lista dei parametri formali con i parametri attuali e valutare il corpo della funzione

$$\frac{\rho(\mathrm{Id}) = \lambda(\mathrm{Ds}).\{\mathrm{C\ return\ E;}\} \qquad \langle \{\ \mathrm{Ds=Es;\ C\ return\ E;}\ \} , \rho , \sigma \rangle \Downarrow (v, \sigma')}{\langle \mathrm{Id(Es)}, \rho , \sigma \rangle \Downarrow (v, \sigma')}$$

Esercizio

Valutare il comando

```
C1 = int abs(int m) {C return n;}
```

```
C = int n=m; if (n<0) {n:= -n;}
```

nell'ambiente e memoria vuoti

$$\frac{}{\langle \text{T Id (Ds) E} , \rho , \sigma \rangle \rightarrow (\rho[\text{Id} \mapsto \lambda(\text{Ds}).\text{E}], \sigma)}$$

$$\langle \text{C1}, \varnothing, \varnothing \rangle \rightarrow \langle \{ \text{abs} \mapsto \lambda(\text{int m}).\{\text{C return n;}\} \}, \varnothing \rangle$$

Esercizio

Valutare l'espressione

`abs(-3)`

nell'ambiente e memoria dopo la definizione della funzione abs:

$(\rho = \{\texttt{abs} \mapsto \lambda(\texttt{int m}).\{\texttt{C return n;}\}\}, \sigma = \emptyset)$ dove `C = int n=m; if (n<0) {n:= -n;}`

$\langle\ \text{abs(-3)}\ , \rho\ , \sigma\ \rangle \Downarrow \langle\ v_?\ , \sigma_?\ \rangle$

$$\frac{\rho(\text{Id}) = \lambda(\text{Ds}).\{\text{C return E;}\} \qquad \langle \{\ \text{Ds=Es; C return E;}\ \}\ , \rho\ , \sigma\ \rangle \Downarrow (\text{v}, \sigma')}{\langle \text{Id(Es)}, \rho\ , \sigma\ \rangle \Downarrow (\text{v}, \sigma')}$$

Esercizio

Valutare l'espressione

abs(−3)

nell'ambiente e memoria dopo la definizione della funzione abs:

$(\rho = \{\texttt{abs} \mapsto \lambda(\texttt{int m}).\{\texttt{C return n;}\}\}, \sigma = \emptyset)$ dove C = int n=m; if (n<0) {n:= −n;}

$\langle$ abs(-3) , ρ , σ $\rangle \Downarrow \langle$ $v_?$, $\sigma_?$ $\rangle$

$\langle$int m=-3; C return n; , ρ , $\sigma\rangle \Downarrow \langle$ $v_?$, $\sigma_?$ $\rangle$

$$\frac{\rho(\text{Id}) = \lambda(\text{Ds}).\{\text{C return E;}\} \qquad \langle \{\ \text{Ds=Es; C return E;}\ \}, \rho, \sigma \rangle \Downarrow (v, \sigma')}{\langle \text{Id(Es)}, \rho, \sigma \rangle \Downarrow (v, \sigma')}$$

Esercizio

Valutare l'espressione

`abs(-3)`

nell'ambiente e memoria dopo la definizione della funzione abs:

$(\rho = \{\texttt{abs} \mapsto \lambda(\texttt{int m}).\{\texttt{C return n;}\}\}, \sigma = \emptyset)$ dove `C = int n=m; if (n<0) {n:= -n;}`

$\langle\ \text{abs(-3)}\ , \rho\ , \sigma\ \rangle \Downarrow \langle\ v_?\ , \sigma_?\ \rangle$

$\langle \text{int m=-3; C return n;}\ , \rho\ , \sigma \rangle \Downarrow \langle\ v_?\ , \sigma_?\ \rangle$

$$\frac{\langle\ C, \rho\ , \sigma\ \rangle \rightarrow (\rho_1\ , \sigma_1\ ) \qquad \langle\ E\ , \rho_1\ , \sigma_1\ \rangle \Downarrow (\ v, \sigma_2\ )}{\langle\ \{\ C \text{ return } E;\ \}\ , \rho\ , \sigma\ \rangle \Downarrow (\ v, \sigma_2\ )}$$

Esercizio

Valutare l'espressione

abs(−3)

nell'ambiente e memoria dopo la definizione della funzione abs:

$(\rho = \{\texttt{abs} \mapsto \lambda(\texttt{int m}).\{\texttt{C return n;}\}\}, \sigma = \emptyset)$ dove `C = int n=m; if (n<0) {n:= -n;}`

$\langle\ \text{abs(-3)}\ ,\ \rho\ ,\ \sigma\ \rangle \Downarrow \langle\ v_?\ ,\ \sigma_?\ \rangle$

$\langle \text{int m=-3; C return n;}\ ,\ \rho\ ,\ \sigma \rangle \Downarrow \langle\ v_?\ ,\ \sigma_?\ \rangle$

$\langle\ \text{int m=-3; C}\ ,\ \rho\ ,\ \sigma \rangle \rightarrow \langle\ \rho_1\ ,\ \sigma_1\ \rangle$

$$\frac{\langle\ \text{C}, \rho\ ,\ \sigma\ \rangle \rightarrow (\ \rho_1\ ,\ \sigma_1\ ) \qquad \langle\ \text{E}\ ,\ \rho_1\ ,\ \sigma_1\ \rangle \Downarrow (\ v, \sigma_2\ )}{\langle\ \{\ \text{C return E;}\ \}\ ,\ \rho\ ,\ \sigma\ \rangle \Downarrow (\ v, \sigma_2\ )}$$

$\langle\ \text{n}\ ,\ \rho_1\ ,\ \sigma_1\ \rangle \Downarrow \langle\ v_?\ ,\ \sigma_?\ \rangle$

Esercizio

Valutare l'espressione

`abs(-3)`

nell'ambiente e memoria dopo la definizione della funzione abs:

$(\rho = \{\texttt{abs} \mapsto \lambda(\texttt{int m}).\{\texttt{C return n;}\}\}, \sigma = \emptyset)$ dove `C = int n=m; if (n<0) {n:= -n;}`

$\langle$ abs(-3) , ρ , σ $\rangle \Downarrow \langle$ $v_?$, $\sigma_?$ $\rangle$

$\langle$int m=-3; C return n; , ρ , $\sigma\rangle \Downarrow \langle$ $v_?$, $\sigma_?$ $\rangle$

$\langle$ int m=-3; C , ρ , $\sigma\rangle \rightarrow \langle$ ρ_1 , σ_1 $\rangle$

$\langle$ int m=-3; , ρ , $\sigma\rangle \rightarrow \langle$ ρ_2 , σ_2 $\rangle$

$\langle$ int n=m; , ρ_2 , $\sigma_2\rangle \rightarrow \langle$ ρ_3 , σ_3 $\rangle$

$\langle$ if (n< 0) {n := -n;} , ρ_3 , $\sigma_3\rangle \rightarrow \langle$ ρ_1 , σ_1 $\rangle$

$\langle$ n , ρ_1 , σ_1 $\rangle \Downarrow \langle$ $v_?$, $\sigma_?$ $\rangle$

Esercizio

Valutare l'espressione

abs(−3)

nell'ambiente e memoria dopo la definizione della funzione abs:

$(\rho = \{\texttt{abs} \mapsto \lambda(\texttt{int m}).\{\texttt{C return n;}\}\}, \sigma = \emptyset)$ dove `C = int n=m; if (n<0) {n:= −n;}`

$\langle$ abs(-3) , ρ , σ $\rangle \Downarrow \langle$ $v_?$, $\sigma_?$ $\rangle$

$\langle$int m=-3; C return n; , ρ , $\sigma\rangle \Downarrow \langle$ $v_?$, $\sigma_?$ $\rangle$

$\langle$ int m=-3; C , ρ , $\sigma\rangle \rightarrow \langle$ ρ_1 , σ_1 $\rangle$

$\langle$ int m=-3; , ρ , $\sigma\rangle \rightarrow \langle$ ρ_2 , σ_2 $\rangle$ con ρ_2=ρ[m $\mapsto$ l_m] e σ_2=σ[$l_m \mapsto -3$]

$\langle$ int n=m; , ρ_2 , $\sigma_2\rangle \rightarrow \langle$ ρ_3 , σ_3 $\rangle$

$\langle$ if (n< 0) {n := -n;} , ρ_3 , $\sigma_3\rangle \rightarrow \langle$ ρ_1 , σ_1 $\rangle$

$\langle$ n , ρ_1 , σ_1 $\rangle \Downarrow \langle$ $v_?$, $\sigma_?$ $\rangle$

Esercizio

Valutare l'espressione

abs(−3)

nell'ambiente e memoria dopo la definizione della funzione abs:

$(\rho = \{\texttt{abs} \mapsto \lambda(\texttt{int m}).\{\texttt{C return n;}\}\}, \sigma = \emptyset)$ dove C = int n=m; if (n<0) {n:= −n;}

$\langle$ abs(-3) , ρ , σ $\rangle \Downarrow \langle$ $v_?$, $\sigma_?$ $\rangle$

$\langle$int m=-3; C return n; , ρ , $\sigma\rangle \Downarrow \langle$ $v_?$, $\sigma_?$ $\rangle$

$\langle$ int m=-3; C , ρ , $\sigma\rangle \rightarrow \langle$ ρ_1 , σ_1 $\rangle$

$\langle$ int m=-3; , ρ , $\sigma\rangle \rightarrow \langle$ ρ[m $\mapsto$ l_m] , [$l_m \mapsto -3$] $\rangle$ ✓

$\langle$ int n=m; , ρ[m $\mapsto$ l_m] , [$l_m \mapsto -3$]$\rangle \rightarrow \langle$ ρ_3 , σ_3 $\rangle$

$\langle$ if (n< 0) {n := -n;} , ρ_3 , $\sigma_3\rangle \rightarrow \langle$ ρ_1 , σ_1 $\rangle$

$\langle$ n , ρ_1 , σ_1 $\rangle \Downarrow \langle$ $v_?$, $\sigma_?$ $\rangle$

Esercizio

Valutare l'espressione

abs(−3)

nell'ambiente e memoria dopo la definizione della funzione abs:

$(\rho = \{\texttt{abs} \mapsto \lambda(\texttt{int m}).\{\texttt{C return n;}\}\}, \sigma = \emptyset)$ dove `C = int n=m; if (n<0) {n:= -n;}`

$\langle$ abs(-3) , ρ , σ $\rangle \Downarrow \langle$ $v_?$, $\sigma_?$ $\rangle$

$\langle$int m=-3; C return n; , ρ , $\sigma\rangle \Downarrow \langle$ $v_?$, $\sigma_?$ $\rangle$

$\langle$ int m=-3; C , ρ , $\sigma\rangle \rightarrow \langle$ ρ_1 , σ_1 $\rangle$

$\langle$ int m=-3; , ρ , $\sigma\rangle \rightarrow \langle$ ρ[m $\mapsto$ l_m] , [$l_m \mapsto -3$] $\rangle$ ✓

$\langle$ int n=m; , ρ[m $\mapsto$ l_m] , [$l_m \mapsto -3$]$\rangle \rightarrow \langle$ ρ_3 , σ_3 $\rangle$ con $\rho_3 = \rho$[m $\mapsto$ l_m,n $\mapsto$ l_n]

$\langle$ if (n< 0) {n := -n;} , ρ_3 , $\sigma_3\rangle \rightarrow \langle$ ρ_1 , σ_1 $\rangle$ e $\sigma_3 = [l_m \mapsto -3, l_n \mapsto -3]$

$\langle$ n , ρ_1 , σ_1 $\rangle \Downarrow \langle$ $v_?$, $\sigma_?$ $\rangle$

Esercizio

Valutare l'espressione

`abs(-3)`

nell'ambiente e memoria dopo la definizione della funzione abs:

$(\rho = \{\texttt{abs} \mapsto \lambda(\texttt{int m}).\{\texttt{C return n;}\}\}, \sigma = \emptyset)$ dove `C = int n=m; if (n<0) {n:= -n;}`

$\langle$ abs(-3) , ρ , σ $\rangle \Downarrow \langle$ $v_?$, $\sigma_?$ $\rangle$

$\langle$int m=-3; C return n; , ρ , $\sigma\rangle \Downarrow \langle$ $v_?$, $\sigma_?$ $\rangle$

con $\rho_3 = \rho[\text{m} \mapsto l_m, \text{n} \mapsto l_n]$

e $\sigma_3 = [l_m \mapsto -3, l_n \mapsto -3]$

$\langle$ int m=-3; C , ρ , $\sigma\rangle \rightarrow \langle$ ρ_1 , σ_1 $\rangle$

$\langle$ int m=-3; , ρ , $\sigma\rangle \rightarrow \langle$ $\rho[\text{m} \mapsto l_m]$, $[l_m \mapsto -3]$ $\rangle$ ✓

$\langle$ int n=m; , $\rho[\text{m} \mapsto l_m]$, $[l_m \mapsto -3]\rangle \rightarrow \langle \rho_3$, σ_3 $\rangle$ ✓

$\langle$ if (n< 0) {n := -n;} , ρ_3 , $\sigma_3\rangle \rightarrow \langle$ ρ_1 , σ_1 $\rangle$

$\langle$ n , ρ_1 , σ_1 $\rangle \Downarrow \langle$ $v_?$, $\sigma_?$ $\rangle$

Esercizio

Valutare l'espressione

abs(−3)

nell'ambiente e memoria dopo la definizione della funzione abs:

$(\rho = \{\texttt{abs} \mapsto \lambda(\texttt{int m}).\{\texttt{C return n;}\}\}, \sigma = \emptyset)$ dove `C = int n=m; if (n<0) {n:= -n;}`

$\langle$ abs(-3) , ρ , σ $\rangle \Downarrow \langle$ $v_?$, $\sigma_?$ $\rangle$

$\langle$int m=-3; C return n; , ρ , $\sigma\rangle \Downarrow \langle$ $v_?$, $\sigma_?$ $\rangle$

con $\rho_3 = \rho[\text{m} \mapsto l_m, \text{n} \mapsto l_n]$

e $\sigma_3 = [l_m \mapsto -3, l_n \mapsto -3]$

$\langle$ int m=-3; C , ρ , $\sigma\rangle \rightarrow \langle$ ρ_1 , σ_1 $\rangle$

$\langle$ int m=-3; , ρ , $\sigma\rangle \rightarrow \langle$ $\rho[\text{m} \mapsto l_m]$, $[l_m \mapsto -3]$ $\rangle$ ✓

$\langle$ int n=m; , $\rho[\text{m} \mapsto l_m]$, $[l_m \mapsto -3]\rangle \rightarrow \langle \rho_3$, σ_3 $\rangle$ ✓

$\langle$ if (n< 0) {n := -n;} , ρ_3 , $\sigma_3\rangle \rightarrow \langle$ ρ_1 , σ_1 $\rangle$

$\langle$ n<0 , ρ_3 , $\sigma_3\rangle \Downarrow \langle$true,$\sigma_3\rangle$ $\langle$ n := -n; , ρ_3 , $\sigma_3\rangle \rightarrow \langle$ ρ_1 , σ_1 $\rangle$

$\langle$ n , ρ_1 , σ_1 $\rangle \Downarrow \langle$ $v_?$, $\sigma_?$ $\rangle$

Esercizio

Valutare l'espressione

abs(−3)

nell'ambiente e memoria dopo la definizione della funzione abs:

$(\rho = \{\text{abs} \mapsto \lambda(\text{int m}).\{\text{C return n;}\}\}, \sigma = \emptyset)$ dove C = int n=m; if (n<0) {n:= −n;}

$\langle$ abs(-3) , ρ , σ $\rangle \Downarrow \langle$ $v_?$, $\sigma_?$ $\rangle$

$\langle$int m=-3; C return n; , ρ , $\sigma\rangle \Downarrow \langle$ $v_?$, $\sigma_?$ $\rangle$

con $\rho_3 = \rho[\text{m} \mapsto l_m, \text{n} \mapsto l_n]$

e $\sigma_3 = [l_m \mapsto -3, l_n \mapsto -3]$

$\langle$ int m=-3; C , ρ , $\sigma\rangle \rightarrow \langle$ ρ_1 , σ_1 $\rangle$

$\langle$ int m=-3; , ρ , $\sigma\rangle \rightarrow \langle$ $\rho[\text{m} \mapsto l_m]$, $[l_m \mapsto -3]$ $\rangle$ ✓

$\langle$ int n=m; , $\rho[\text{m} \mapsto l_m]$, $[l_m \mapsto -3]\rangle \rightarrow \langle \rho_3$, σ_3 $\rangle$ ✓

con $\rho_1 = \rho_3$

e $\sigma_1 = [l_m \mapsto -3, l_n \mapsto 3]$

$\langle$ if (n< 0) {n := -n;} , ρ_3 , $\sigma_3\rangle \rightarrow \langle$ ρ_1 , σ_1 $\rangle$

$\langle$ n<0 , ρ_3 , $\sigma_3\rangle \Downarrow \langle$true,$\sigma_3\rangle$ $\langle$ n := -n; , ρ_3 , $\sigma_3\rangle \rightarrow \langle$ ρ_1 , σ_1 $\rangle$

$\langle$ n , ρ_1 , σ_1 $\rangle \Downarrow \langle$ $v_?$, $\sigma_?$ $\rangle$

Esercizio

Valutare l'espressione

abs(−3)

nell'ambiente e memoria dopo la definizione della funzione abs:

$(\rho = \{\texttt{abs} \mapsto \lambda(\texttt{int m}).\{\texttt{C return n;}\}\}, \sigma = \emptyset)$ dove `C = int n=m; if (n<0) {n:= -n;}`

$\langle$ abs(-3) , ρ , σ $\rangle \Downarrow \langle$ $v_?$, $\sigma_?$ $\rangle$

$\langle$int m=-3; C return n; , ρ , $\sigma\rangle \Downarrow \langle$ $v_?$, $\sigma_?$ $\rangle$

con $\rho_3 = \rho[\text{m} \mapsto l_m, \text{n} \mapsto l_n]$

e $\sigma_3 = [l_m \mapsto -3, l_n \mapsto -3]$

$\langle$ int m=-3; C , ρ , $\sigma\rangle \rightarrow \langle$ ρ_3 , σ_1 $\rangle$ ✓

$\langle$ int m=-3; , ρ , $\sigma\rangle \rightarrow \langle$ ρ[m $\mapsto$ l_m] , [$l_m \mapsto -3$] $\rangle$ ✓

$\langle$ int n=m; , ρ[m $\mapsto$ l_m] , [$l_m \mapsto -3$]$\rangle \rightarrow \langle \rho_3 , \sigma_3 \rangle$ ✓ con $\rho_1 = \rho_3$

$\langle$ if (n< 0) {n := -n;} , ρ_3 , $\sigma_3\rangle \rightarrow \langle$ ρ_3 , σ_1 $\rangle$ ✓ e $\sigma_1 = [l_m \mapsto -3, l_n \mapsto 3]$

$\langle$ n<0 , ρ_3 , $\sigma_3\rangle \Downarrow \langle$true,$\sigma_3\rangle$ $\langle$ n := -n; , ρ_3 , $\sigma_3\rangle \rightarrow \langle$ ρ_3 , σ_1 $\rangle$ ✓

$\langle$ n , ρ_3 , σ_1 $\rangle \Downarrow \langle$ $v_?$, $\sigma_?$ $\rangle$

Esercizio

Valutare l'espressione

abs(−3)

nell'ambiente e memoria dopo la definizione della funzione abs:

$(\rho = \{\text{abs} \mapsto \lambda(\text{int m}).\{\text{C return n;}\}\}, \sigma = \emptyset)$ dove C = int n=m; if (n<0) {n:= −n;}

$\langle$ abs(-3) , ρ , σ $\rangle \Downarrow \langle$ $v_?$, $\sigma_?$ $\rangle$

$\langle$int m=-3; C return n; , ρ , $\sigma\rangle \Downarrow \langle$ $v_?$, $\sigma_?$ $\rangle$

con $\rho_3 = \rho[\text{m} \mapsto l_m, \text{n} \mapsto l_n]$
e $\sigma_3 = [l_m \mapsto -3, l_n \mapsto -3]$

$\langle$ int m=-3; C , ρ , $\sigma\rangle \rightarrow \langle$ ρ_3 , σ_1 $\rangle$ ✓

$\langle$ int m=-3; , ρ , $\sigma\rangle \rightarrow \langle$ ρ[m $\mapsto$ l_m] , [$l_m \mapsto -3$] $\rangle$ ✓

$\langle$ int n=m; , ρ[m $\mapsto$ l_m] , [$l_m \mapsto -3$]$\rangle \rightarrow \langle \rho_3$, σ_3 $\rangle$ ✓ con $\rho_1 = \rho_3$

$\langle$ if (n< 0) {n := -n;} , ρ_3 , $\sigma_3\rangle \rightarrow \langle$ ρ_3 , σ_1 $\rangle$ ✓ e $\sigma_1 = [l_m \mapsto -3, l_n \mapsto 3]$

$\langle$ n<0 , ρ_3 , $\sigma_3\rangle \Downarrow \langle$true,$\sigma_3\rangle$ $\langle$ n := -n; , ρ_3 , $\sigma_3\rangle \rightarrow \langle$ ρ_3 , σ_1 $\rangle$ ✓

$\langle$ n , ρ_3 , σ_1 $\rangle \Downarrow \langle$ $v_?$, $\sigma_?$ $\rangle$ con $v_?$ = 3 e $\sigma_?$ = σ_1

Esercizio

Valutare l'espressione

abs(−3)

nell'ambiente e memoria dopo la definizione della funzione abs:

$(\rho = \{\text{abs} \mapsto \lambda(\text{int m}).\{\text{C return n;}\}\}, \sigma = \emptyset)$ dove C = int n=m; if (n<0) {n:= −n;}

$\langle$ abs(-3) , ρ , σ $\rangle \Downarrow \langle$ 3 , σ_1 $\rangle$ ✓

$\langle$int m=-3; C return n; , ρ , $\sigma\rangle \Downarrow \langle$ 3 , σ_1 $\rangle$ ✓

con $\rho_3 = \rho$[m $\mapsto l_m$,n $\mapsto l_n$]
e σ_3=[$l_m \mapsto -3, l_n \mapsto -3$]

$\langle$ int m=-3; C , ρ , $\sigma\rangle \rightarrow \langle \rho_3 , \sigma_1 \rangle$ ✓

$\langle$ int m=-3; , ρ , $\sigma\rangle \rightarrow \langle \rho$[m $\mapsto l_m$] , [$l_m \mapsto -3$] $\rangle$ ✓

$\langle$ int n=m; , ρ[m $\mapsto l_m$] , [$l_m \mapsto -3$]$\rangle \rightarrow \langle \rho_3 , \sigma_3 \rangle$ ✓

con $\rho_1 = \rho_3$
e σ_1=[$l_m \mapsto -3, l_n \mapsto 3$]

$\langle$ if (n< 0) {n := -n;} , ρ_3 , $\sigma_3\rangle \rightarrow \langle \rho_3 , \sigma_1 \rangle$ ✓

$\langle$ n<0 , ρ_3 , $\sigma_3\rangle \Downarrow \langle$true,$\sigma_3\rangle$ $\langle$ n := -n; , ρ_3 , $\sigma_3\rangle \rightarrow \langle \rho_3 , \sigma_1 \rangle$ ✓

$\langle$ n , ρ_3 , σ_1 $\rangle \Downarrow \langle$ 3 , σ_1 $\rangle$ ✓