Valutazione parziale, specializzazione di interpreti, compilazione

Contenuti

- la valutazione parziale
- la specializzazione di interpreti
  - prima proiezione di Futamura
  - seconda proiezione di Futamura
- caratteristiche dell'interprete da specializzare
- cosa fa lo specializzatore di interpreti
- un esempio sul frammento funzionale
  - simulazione "manuale" di specializzazione con generazione del codice

Trasformazione sistematica di un interprete in un compilatore

- la valutazione parziale è una tecnica che permette di generare automaticamente un compilatore a partire da un interprete
  - corrispondente "semantico" dei generatori automatici di analizzatori sintattici a partire dalla "specifica" della grammatica del linguaggio
    - semantica denotazionale (o operazionale) come vera specifica
    - i raffinamenti dei vari interpreti sono passi della generazione
  - al di là della sua utilità pratica, permette di capire meglio l'essenza della differenza fra interpretazione e compilazione

La valutazione parziale: il problema

- fissati
  - un programma P
  - una n-upla di valori $v_1,\ldots,v_n$ per i suoi "primi" n dati d'ingresso

  vogliamo determinare un programma P'
  - specializzazione di P per $v_1,\ldots,v_n$

  che si comporta esattamente come P
  - per ogni valore degli altri dati
  - quando i suoi "primi" n dati d'ingresso sono $v_1,\ldots,v_n$

  ed è più efficiente di P

La valutazione parziale: un esempio

```
f(x, y) = if x=0 then g(x) else h(y)
```

- vogliamo specializzare `f` per il valore `x = 2`

```
f2(y) = f(2, y) =  h(y)
```

- `f2` si ottiene "valutando" il corpo di `f` quanto possibile a tempo di specializzazione
  - la guardia del condizionale è sempre falsa
  - si può rimpiazzare il condizionale con il suo ramo `else`
- `f2` è più efficiente di `f` per qualunque valore di `y`

Il problema ha una soluzione

- il teorema *s-m-n* di Kleene
- dati
  - una funzione $f = \lambda x_1,..,x_n.e$
  - una *k*-upla di valori $a_1,...,a_k$

  è possibile calcolare la funzione

  $$f' = \lambda x_{k+1},..,x_n.e'$$

  tale che

  $$\forall x_{k+1},..,x_n.f(a_1,...,a_k,x_{k+1},..,x_n) = f'(x_{k+1},..,x_n)$$

Come si calcola effettivamente la soluzione

- il valutatore parziale esegue simbolicamente il programma
  - valutando una volta per tutte ed eliminando dal codice le istruzioni che possono essere eseguite
    - quando c'è abbastanza informazione per farlo
  - lasciando le altre nel codice "residuo"
    - eventualmente semplificandolo utilizzando le regole di una semantica algebrica
- valutare una volta per tutte, eliminare dal codice o lasciare nel codice si applica in particolare anche alle *strutture dati*
- miglioramenti notevoli si ottengono
  - semplificando i condizionali (e sfogliando i cicli) quando siamo in grado di valutare una guardia
  - rimpiazzando le procedure (non ricorsive) con la loro definizione (ma attenzione alla dimensione del codice residuo!)

Il valutatore parziale del linguaggio `M`

- `peval` è un valutatore parziale capace di specializzare programmi scritti nel linguaggio `M`
  - è molto simile ad un interprete di `M`
- un generico programma `P` di `M` ha i propri dati raggruppati in due tuple
  - la prima `D` è quella dei dati forniti nella specializzazione
  - la seconda `X` è la tupla di dati non conosciuti
- `peval: Prog`$_M$ `* dati -> Prog`$_M$

  `peval(P, D)= P'` tale che

  $\forall X.$ `P'(X) = P(D, X)`
- queste sono le equazioni che danno le proprietà del valutatore parziale

Specializziamo un interprete

- `peval: Prog`$_M$ `* dati -> Prog`$_M$

  `peval(P, D)= P'` tale che

  `∀X. P'(X) = P(D, X)`

- specializziamo con `peval` il programma `int`
  - un interprete del linguaggio `L` scritto nel linguaggio `M`, che ha come dati
    - un programma `prog` in `L` (che supponiamo noto nella specializzazione)
    - uno stato iniziale `s`
- applichiamo le equazioni della valutazione parziale

  `peval(int, prog)= int'` tale che

  `∀s. int'(s) = int(prog, s)`

- cos'è `int'`?
  - dato che `int` è un interprete, `int(prog, s)` fornisce lo stato finale per ogni stato iniziale `s` ottenuto dalla semantica di `prog`
  - `int'` fa la stessa cosa ed è un programma di `M` (come `int`)
  - `int'` è la traduzione di `prog` da `L` a `M`

Prima proiezione di Futamura: il codice compilato

- `peval` valutatore parziale di `M`
- `int` interprete del linguaggio `L` scritto nel linguaggio `M`
- `prog` programma di `L`

```
peval(int, prog)= int'
∀s. int'(s) = int(prog, s)
```

- `int'` risultato della compilazione di `prog` sulla macchina astratta di `M`
  - può essere eseguito direttamente su `M` senza aver bisogno dell'interprete
  - è funzionalmente equivalente a `prog`
  - dato che non c'è più di mezzo un interprete, l'esecuzione diretta di `int'` dovrebbe essere più efficiente dell'esecuzione interpretativa di `prog`

Seconda proiezione di Futamura: il compilatore

- `peval` valutatore parziale di `M`, scritto in `M` (autoapplicabile!)
- `int` interprete del linguaggio `L` scritto nel linguaggio `M`
- `prog` programma di `L`

```
peval(int, prog)= int'
∀s. int'(s) = int(prog, s)
```

- `int'` risultato della compilazione di `prog` sulla macchina astratta di `M`

```
peval(peval, int)= peval'
∀prog. peval'(prog) = peval(int, prog) = int'
```

- `peval'` compilatore da `L` a `M`

Cosa succede nella generazione del codice compilato

- `peval` valutatore parziale di `M`
- `int` interprete del linguaggio `L` scritto nel linguaggio `M`
- `prog` programma di `L`

```
peval(int, prog)= int'
```

- `int'` risultato della compilazione di `prog` sulla macchina astratta di `M`
- per capire come `int` si semplifica nella specializzazione, bisogna guardare la sua definizione
  - può essere costruito, valutato ed eliminato tutto ciò che dipende solo da `prog` (che è statico!)
    - le strutture dati destinate a contenere i costrutti sintattici (la pila di pile di costrutti etichettati)
    - il ciclo di decodifica determinato dal contenuto di tali strutture dati
    - gli eventuali analizzatori statici (per esempio, l'inferenza dei tipi)
    - se il linguaggio ha lo scoping statico, le pile di array di nomi e di link statici
      - con conseguente ottimizzazione delle `applyenv`
  - le strutture dati che restano finiscono nel supporto a tempo di esecuzione

Aggiustamenti sull'interprete

- vanno inseriti tutti gli eventuali analizzatori (nomi, tipi, dimensionamenti)
  - gli errori vengono rilevati a tempo di specializzazione (compilazione)
  - le verifiche (per esempio, il type checking) sono fatte una volta per tutte e scompaiono dal codice residuo
  - le informazioni calcolate (per esempio, il tipo dell'espressione) sono pre-calcolate
- un discorso particolare sulle funzioni (e costrutti simili come procedure e classi)
  - vogliamo specializzare il loro codice
    - una volta sola
    - al momento della definizione
  - conviene reintrodurre nell'interprete un trattamento in stile "denotazionale"

```
type efun = eval list -> unit
let makefun (Fun(ii,aa),r) = function d ->
  newframes(aa,bindlist(r, ii, d))
let applyfun (ev1, ev2) = match ev1 with Funval(f) -> f ev2
```

  - volendo una "compilazione separata" può addirittura convenire reintrodurre la ricorsione (al posto delle chiamate di `newframes`)

Le scelte sulla specializzazione: strutture dati

- vediamole nel caso del frammento funzionale
- la scelta principale riguarda le strutture che compongono la pila dei records di attivazione
  - strutture che possono essere costruite ed eliminate

  `cstack`, pila di pile di espressioni etichettate

  `namestack`, pila di array di identificatori

  - strutture che possono essere costruite (in versione statica) ma devono essere lasciate nel supporto a tempo di esecuzione

  `slinkstack`, pila di (puntatori ad) ambienti

  - strutture che devono essere lasciate nel supporto a tempo di esecuzione

  `tempvalstack`, pila di pile di valori esprimibili

  `evalstack`, pila di array di valori denotati

  `tagstack`, pila di etichette per la retention

- se lo scoping fosse dinamico, non potrei eliminare `namestack`
- nel linguaggio imperativo, tutto uguale e non posso eliminare `storestack`
- nel linguaggio orientato ad oggetti, posso costruire ed eliminare anche la parte nomi degli ambienti permanenti degli oggetti

Le scelte sulla specializzazione: funzioni

- per quali funzioni usiamo l'unfolding?
  - rimpiazzamento della chiamata con il corpo e valutazione parziale del corpo
- la `newframes`
- le operazioni relative all'ambiente
  - dato che sono noti `namestack` e `slinkstack`
  - l'unfolding e valutazione di `applyenv` produce un effetto analogo a quello della traduzione dei nomi in coppie di interi
- il ciclo di interpretazione (i due `while` annidati) è controllato da informazione sintattica (il contenuto di `cstack`)
  - può essere sfogliato completamente ed eliminato

Un esempio di simulazione manuale 0

```
let sem ((e:exp), (r:eval env)) =
    push(emptystack(1,Unbound),tempvalstack); newframes(e,r);
    while not(empty(cstack)) do
     while not(empty(top(cstack))) do
      (match top(top(cstack)) with
       |Expr1(x) -> (pop(top(cstack)); push(Expr2(x), top(cstack));
         (match x with
            | Sum(a,b) -> push(Expr1(a), top(cstack) ); push(Expr1(b), top(cstack) )
            | Let(i,e1,e2) -> push(Expr1(e1), top(cstack) )
            | _ -> ()))
      |Expr2(x) -> (pop(top(cstack) ); (match x with
          | Eint(n) ->  push(Int(n),top(tempvalstack) )
            | Den(i) ->   push(applyenv(topenv() ,i), top(tempvalstack) )
            | Let(i,e1,e2) -> let arg=top(top(tempvalstack)) in
               pop(top(tempvalstack) );  newframes(e2, bind(topenv() , i, arg))
      done;
      let valore= top(top(tempvalstack)) in
      pop(top(tempvalstack)); pop(cstack); popenv(); pop(tempvalstack);
      push(valore,top(tempvalstack))
    done;
    let valore = top(top(tempvalstack)) in pop(tempvalstack); valore
```

```
e = Let("x", Eint 3, Let("y", Eint 5, Sum(Den "x",Den "y")))
```

Un esempio di simulazione manuale 1

```
let sem ((e:exp), (r:eval env)) =
    push(emptystack(1,Unbound),tempvalstack); newframes(e,r);
    while not(empty(cstack)) do
     while not(empty(top(cstack))) do  (match top(top(cstack)) with
       |Expr1(x) -> (pop(top(cstack)); push(Expr2(x), top(cstack));
        (match x with
           | Sum(a,b) -> push(Expr1(a), top(cstack) ); push(Expr1(b), top(cstack) )
           | Let(i,e1,e2) -> push(Expr1(e1), top(cstack) )
           | _ -> ()))
      |Expr2(x) -> (pop(top(cstack) ); (match x with
           | Eint(n) ->  push(Int(n),top(tempvalstack) )
           | Den(i) ->   push(applyenv(topenv() ,i), top(tempvalstack) )
           | Let(i,e1,e2) -> let arg=top(top(tempvalstack)) in pop(top(tempvalstack) );
              newframes(e2, bind(topenv() , i, arg)) done;
     let valore= top(top(tempvalstack)) in  pop(top(tempvalstack)); pop(cstack); popenv(); pop(tempvalstack);
      push(valore,top(tempvalstack))
    done; let valore = top(top(tempvalstack)) in pop(tempvalstack); valore
```

```
e = Let("x", Eint 3, Let("y", Eint 5, Sum(Den "x",Den "y")))
```

```
  let newframes(e,rho) =
     let cframe = emptystack(cframesize(e),Expr1(e)) in
     let tframe = emptystack(tframesize(e),Unbound) in
     push(Expr1(e),cframe); push(cframe,cstack); push(tframe,tempvalstack);
     if rho = !currentenv then (push([||],namestack); push([||],evalstack); push(rho,slinkstack);
     currentenv := lungh(namestack) ) else currentenv := rho
```

```
cstack
[|[|Expr1 (Let ("x", Eint 3, Let ("y", Eint 5, Sum (Den "x", Den "y"))));....|],{contents=0};
  [|Expr1 (Eint 0)|], {contents=-1}; ...; ...|],...
```

```
let semres r = push(emptystack(1,Unbound),tempvalstack); let tframe = emptystack(tframesize(e),Unbound) in
push(tframe,tempvalstack); currentenv := r;
```

Un esempio di simulazione manuale 2

```
let sem ((e:exp), (r:eval env)) =
     push(emptystack(1,Unbound),tempvalstack); newframes(e,r);
     while not(empty(top(cstack))) do  (match top(top(cstack)) with
        |Expr1(x) -> (pop(top(cstack)); push(Expr2(x), top(cstack));
         (match x with
              | Sum(a,b) -> push(Expr1(a), top(cstack) ); push(Expr1(b), top(cstack) )
              | Let(i,e1,e2) -> push(Expr1(e1), top(cstack) )
              | _ -> ()))
       |Expr2(x) -> (pop(top(cstack) ); (match x with
              | Eint(n) ->  push(Int(n),top(tempvalstack) )
              | Den(i) ->   push(applyenv(topenv() ,i), top(tempvalstack) )
              | Let(i,e1,e2) -> let arg=top(top(tempvalstack)) in pop(top(tempvalstack) );
               newframes(e2, bind(topenv() , i, arg)) done;
      let valore= top(top(tempvalstack)) in  pop(top(tempvalstack)); pop(cstack); popenv(); pop(tempvalstack);
       push(valore,top(tempvalstack));
      while not(empty(cstack)) do
      while not(empty(top(cstack))) do  (match top(top(cstack)) with
        |Expr1(x) -> (pop(top(cstack)); push(Expr2(x), top(cstack));
         (match x with
              | Sum(a,b) -> push(Expr1(a), top(cstack) ); push(Expr1(b), top(cstack) )
              | Let(i,e1,e2) -> push(Expr1(e1), top(cstack) )
              | _ -> ()))
       |Expr2(x) -> (pop(top(cstack) ); (match x with
              | Eint(n) ->  push(Int(n),top(tempvalstack) )
              | Den(i) ->   push(applyenv(topenv() ,i), top(tempvalstack) )
              | Let(i,e1,e2) -> let arg=top(top(tempvalstack)) in pop(top(tempvalstack) );
               newframes(e2, bind(topenv() , i, arg)) done;
      let valore= top(top(tempvalstack)) in  pop(top(tempvalstack)); pop(cstack); popenv(); pop(tempvalstack);
       push(valore,top(tempvalstack))
     done; let valore = top(top(tempvalstack)) in pop(tempvalstack); valore


cstack
[|[|Expr1 (Let ("x", Eint 3, Let ("y", Eint 5, Sum (Den "x", Den "y"))));....|],{contents=0};
  [|Expr1 (Eint 0)|], {contents=-1}; ...; ...|],...


let semres r = push(emptystack(1,Unbound),tempvalstack); let tframe = emptystack(tframesize(e),Unbound) in
push(tframe,tempvalstack); currentenv := r;
```

Un esempio di simulazione manuale 3

```
let sem ((e:exp), (r:eval env)) = ...
     (match top(top(cstack)) with
        |Expr1(x) -> (pop(top(cstack)); push(Expr2(x), top(cstack));
         (match x with
              | Sum(a,b) -> push(Expr1(a), top(cstack) ); push(Expr1(b), top(cstack) )
              | Let(i,e1,e2) -> push(Expr1(e1), top(cstack) )
              | _ -> ()))
       |Expr2(x) -> (pop(top(cstack) ); (match x with
              | Eint(n) ->  push(Int(n),top(tempvalstack) )
              | Den(i) ->   push(applyenv(topenv() ,i), top(tempvalstack) )
              | Let(i,e1,e2) -> let arg=top(top(tempvalstack)) in pop(top(tempvalstack) );
               newframes(e2, bind(topenv() , i, arg))
    while not(empty(top(cstack))) do  (match top(top(cstack)) with
       |Expr1(x) -> (pop(top(cstack)); push(Expr2(x), top(cstack));
        (match x with
              | Sum(a,b) -> push(Expr1(a), top(cstack) ); push(Expr1(b), top(cstack) )
              | Let(i,e1,e2) -> push(Expr1(e1), top(cstack) )
              | _ -> ()))
       |Expr2(x) -> (pop(top(cstack) ); (match x with
              | Eint(n) ->  push(Int(n),top(tempvalstack) )
              | Den(i) ->   push(applyenv(topenv() ,i), top(tempvalstack) )
              | Let(i,e1,e2) -> let arg=top(top(tempvalstack)) in pop(top(tempvalstack) );
               newframes(e2, bind(topenv() , i, arg)) done;
     let valore= top(top(tempvalstack)) in  pop(top(tempvalstack)); pop(cstack); popenv(); pop(tempvalstack);
      push(valore,top(tempvalstack));
     while not(empty(cstack)) do
     while not(empty(top(cstack))) do  (match top(top(cstack)) with
       |Expr1(x) -> (pop(top(cstack)); push(Expr2(x), top(cstack));
        (match x with
              | Sum(a,b) -> push(Expr1(a), top(cstack) ); push(Expr1(b), top(cstack) )
              | Let(i,e1,e2) -> push(Expr1(e1), top(cstack) )
              | _ -> ()))
       |Expr2(x) -> (pop(top(cstack) ); (match x with
              | Eint(n) ->  push(Int(n),top(tempvalstack) )
              | Den(i) ->   push(applyenv(topenv() ,i), top(tempvalstack) )
              | Let(i,e1,e2) -> let arg=top(top(tempvalstack)) in pop(top(tempvalstack) );
               newframes(e2, bind(topenv() , i, arg)) done;
     let valore= top(top(tempvalstack)) in  pop(top(tempvalstack)); pop(cstack); popenv(); pop(tempvalstack);
      push(valore,top(tempvalstack))
    done; let valore = top(top(tempvalstack)) in pop(tempvalstack); valore
```

Un esempio di simulazione manuale 4

```
let sem ((e:exp), (r:eval env)) = ...
     while not(empty(top(cstack))) do  (match top(top(cstack)) with
        |Expr1(x) -> (pop(top(cstack)); push(Expr2(x), top(cstack));
         (match x with
              | Sum(a,b) -> push(Expr1(a), top(cstack) ); push(Expr1(b), top(cstack) )
              | Let(i,e1,e2) -> push(Expr1(e1), top(cstack) )
              | _ -> ()))
       |Expr2(x) -> (pop(top(cstack) ); (match x with
              | Eint(n) ->  push(Int(n),top(tempvalstack) )
              | Den(i) ->   push(applyenv(topenv() ,i), top(tempvalstack) )
              | Let(i,e1,e2) -> let arg=top(top(tempvalstack)) in pop(top(tempvalstack) );
               newframes(e2, bind(topenv() , i, arg)) done;
      let valore= top(top(tempvalstack)) in  pop(top(tempvalstack)); pop(cstack); popenv(); pop(tempvalstack);
       push(valore,top(tempvalstack));
      while not(empty(cstack)) do
      while not(empty(top(cstack))) do  (match top(top(cstack)) with
        |Expr1(x) -> (pop(top(cstack)); push(Expr2(x), top(cstack));
         (match x with
              | Sum(a,b) -> push(Expr1(a), top(cstack) ); push(Expr1(b), top(cstack) )
              | Let(i,e1,e2) -> push(Expr1(e1), top(cstack) )
              | _ -> ()))
       |Expr2(x) -> (pop(top(cstack) ); (match x with
              | Eint(n) ->  push(Int(n),top(tempvalstack) )
              | Den(i) ->   push(applyenv(topenv() ,i), top(tempvalstack) )
              | Let(i,e1,e2) -> let arg=top(top(tempvalstack)) in pop(top(tempvalstack) );
               newframes(e2, bind(topenv() , i, arg)) done;
      let valore= top(top(tempvalstack)) in  pop(top(tempvalstack)); pop(cstack); popenv(); pop(tempvalstack);
       push(valore,top(tempvalstack))
     done; let valore = top(top(tempvalstack)) in pop(tempvalstack); valore
```

```
cstack
[|[|Expr2 (Let ("x", Eint 3, Let ("y", Eint 5, Sum (Den "x", Den "y")))); Expr1 (Eint 3);...|],{contents=1};
[|Expr1 (Eint 0)|], {contents=-1};...|],...
```

```
let semres r = push(emptystack(1,Unbound),tempvalstack); let tframe = emptystack(tframesize(e),Unbound) in
push(tframe,tempvalstack); currentenv := r;
```

Un esempio di simulazione manuale 5

```
let sem ((e:exp), (r:eval env)) = ...
     (match top(top(cstack)) with
        |Expr1(x) -> (pop(top(cstack)); push(Expr2(x), top(cstack));
        (match x with
              | Sum(a,b) -> push(Expr1(a), top(cstack) ); push(Expr1(b), top(cstack) )
              | Let(i,e1,e2) -> push(Expr1(e1), top(cstack) )
              | _ -> ()))
       |Expr2(x) -> (pop(top(cstack) ); (match x with
              | Eint(n) ->  push(Int(n),top(tempvalstack) )
              | Den(i) ->   push(applyenv(topenv() ,i), top(tempvalstack) )
              | Let(i,e1,e2) -> let arg=top(top(tempvalstack)) in pop(top(tempvalstack) );
             newframes(e2, bind(topenv() , i, arg))
     while not(empty(top(cstack))) do  (match top(top(cstack)) with
        |Expr1(x) -> (pop(top(cstack)); push(Expr2(x), top(cstack));
        (match x with
              | Sum(a,b) -> push(Expr1(a), top(cstack) ); push(Expr1(b), top(cstack) )
              | Let(i,e1,e2) -> push(Expr1(e1), top(cstack) )
              | _ -> ()))
       |Expr2(x) -> (pop(top(cstack) ); (match x with
              | Eint(n) ->  push(Int(n),top(tempvalstack) )
              | Den(i) ->   push(applyenv(topenv() ,i), top(tempvalstack) )
              | Let(i,e1,e2) -> let arg=top(top(tempvalstack)) in pop(top(tempvalstack) );
             newframes(e2, bind(topenv() , i, arg)) done;
      let valore= top(top(tempvalstack)) in  pop(top(tempvalstack)); pop(cstack); popenv(); pop(tempvalstack);
        push(valore,top(tempvalstack));
       while not(empty(cstack)) do
      while not(empty(top(cstack))) do  (match top(top(cstack)) with
        |Expr1(x) -> (pop(top(cstack)); push(Expr2(x), top(cstack));
        (match x with
              | Sum(a,b) -> push(Expr1(a), top(cstack) ); push(Expr1(b), top(cstack) )
              | Let(i,e1,e2) -> push(Expr1(e1), top(cstack) )
              | _ -> ()))
       |Expr2(x) -> (pop(top(cstack) ); (match x with
              | Eint(n) ->  push(Int(n),top(tempvalstack) )
              | Den(i) ->   push(applyenv(topenv() ,i), top(tempvalstack) )
              | Let(i,e1,e2) -> let arg=top(top(tempvalstack)) in pop(top(tempvalstack) );
             newframes(e2, bind(topenv() , i, arg)) done;
      let valore= top(top(tempvalstack)) in  pop(top(tempvalstack)); pop(cstack); popenv(); pop(tempvalstack);
        push(valore,top(tempvalstack))
      done; let valore = top(top(tempvalstack)) in pop(tempvalstack); valore
```

Un esempio di simulazione manuale 6

```
let sem ((e:exp), (r:eval env)) = ...
      while not(empty(top(cstack))) do  (match top(top(cstack)) with
         |Expr1(x) -> (pop(top(cstack)); push(Expr2(x), top(cstack));
         (match x with
               | Sum(a,b) -> push(Expr1(a), top(cstack) ); push(Expr1(b), top(cstack) )
               | Let(i,e1,e2) -> push(Expr1(e1), top(cstack) )
               | _ -> ()))
        |Expr2(x) -> (pop(top(cstack) ); (match x with
               | Eint(n) ->  push(Int(n),top(tempvalstack) )
               | Den(i) ->   push(applyenv(topenv() ,i), top(tempvalstack) )
               | Let(i,e1,e2) -> let arg=top(top(tempvalstack)) in pop(top(tempvalstack) );
              newframes(e2, bind(topenv() , i, arg)) done;
      let valore= top(top(tempvalstack)) in  pop(top(tempvalstack)); pop(cstack); popenv(); pop(tempvalstack);
       push(valore,top(tempvalstack));
       while not(empty(cstack)) do
      while not(empty(top(cstack))) do  (match top(top(cstack)) with
         |Expr1(x) -> (pop(top(cstack)); push(Expr2(x), top(cstack));
         (match x with
               | Sum(a,b) -> push(Expr1(a), top(cstack) ); push(Expr1(b), top(cstack) )
               | Let(i,e1,e2) -> push(Expr1(e1), top(cstack) )
               | _ -> ()))
        |Expr2(x) -> (pop(top(cstack) ); (match x with
               | Eint(n) ->  push(Int(n),top(tempvalstack) )
               | Den(i) ->   push(applyenv(topenv() ,i), top(tempvalstack) )
               | Let(i,e1,e2) -> let arg=top(top(tempvalstack)) in pop(top(tempvalstack) );
              newframes(e2, bind(topenv() , i, arg)) done;
      let valore= top(top(tempvalstack)) in  pop(top(tempvalstack)); pop(cstack); popenv(); pop(tempvalstack);
        push(valore,top(tempvalstack))
      done; let valore = top(top(tempvalstack)) in pop(tempvalstack); valore
```

```
cstack
[|[|Expr2 (Let ("x", Eint 3, Let ("y", Eint 5, Sum (Den "x", Den "y")))); Expr2 (Eint 3);...|],{contents=1};
[|Expr1 (Eint 0)|], {contents=-1};...|],...
```

```
let semres r = push(emptystack(1,Unbound),tempvalstack); let tframe = emptystack(tframesize(e),Unbound) in
push(tframe,tempvalstack); currentenv := r;
```

Un esempio di simulazione manuale 7

```
let sem ((e:exp), (r:eval env)) = ...
     (match top(top(cstack)) with
        |Expr1(x) -> (pop(top(cstack)); push(Expr2(x), top(cstack));
        (match x with
              | Sum(a,b) -> push(Expr1(a), top(cstack) ); push(Expr1(b), top(cstack) )
              | Let(i,e1,e2) -> push(Expr1(e1), top(cstack) )
              | _ -> ()))
       |Expr2(x) -> (pop(top(cstack) ); (match x with
              | Eint(n) ->  push(Int(n),top(tempvalstack) )
              | Den(i) ->   push(applyenv(topenv() ,i), top(tempvalstack) )
              | Let(i,e1,e2) -> let arg=top(top(tempvalstack)) in pop(top(tempvalstack) );
             newframes(e2, bind(topenv() , i, arg))
     while not(empty(top(cstack))) do  (match top(top(cstack)) with
        |Expr1(x) -> (pop(top(cstack)); push(Expr2(x), top(cstack));
        (match x with
              | Sum(a,b) -> push(Expr1(a), top(cstack) ); push(Expr1(b), top(cstack) )
              | Let(i,e1,e2) -> push(Expr1(e1), top(cstack) )
              | _ -> ()))
       |Expr2(x) -> (pop(top(cstack) ); (match x with
              | Eint(n) ->  push(Int(n),top(tempvalstack) )
              | Den(i) ->   push(applyenv(topenv() ,i), top(tempvalstack) )
              | Let(i,e1,e2) -> let arg=top(top(tempvalstack)) in pop(top(tempvalstack) );
             newframes(e2, bind(topenv() , i, arg)) done;
      let valore= top(top(tempvalstack)) in  pop(top(tempvalstack)); pop(cstack); popenv(); pop(tempvalstack);
        push(valore,top(tempvalstack));
       while not(empty(cstack)) do
      while not(empty(top(cstack))) do  (match top(top(cstack)) with
        |Expr1(x) -> (pop(top(cstack)); push(Expr2(x), top(cstack));
        (match x with
              | Sum(a,b) -> push(Expr1(a), top(cstack) ); push(Expr1(b), top(cstack) )
              | Let(i,e1,e2) -> push(Expr1(e1), top(cstack) )
              | _ -> ()))
       |Expr2(x) -> (pop(top(cstack) ); (match x with
              | Eint(n) ->  push(Int(n),top(tempvalstack) )
              | Den(i) ->   push(applyenv(topenv() ,i), top(tempvalstack) )
              | Let(i,e1,e2) -> let arg=top(top(tempvalstack)) in pop(top(tempvalstack) );
             newframes(e2, bind(topenv() , i, arg)) done;
      let valore= top(top(tempvalstack)) in  pop(top(tempvalstack)); pop(cstack); popenv(); pop(tempvalstack);
        push(valore,top(tempvalstack))
     done; let valore = top(top(tempvalstack)) in pop(tempvalstack); valore
```

Un esempio di simulazione manuale 8

```
let sem ((e:exp), (r:eval env)) = ...
      while not(empty(top(cstack))) do  (match top(top(cstack)) with
         |Expr1(x) -> (pop(top(cstack)); push(Expr2(x), top(cstack));
         (match x with
               | Sum(a,b) -> push(Expr1(a), top(cstack) ); push(Expr1(b), top(cstack) )
               | Let(i,e1,e2) -> push(Expr1(e1), top(cstack) )
               | _ -> ()))
        |Expr2(x) -> (pop(top(cstack) ); (match x with
               | Eint(n) ->  push(Int(n),top(tempvalstack) )
               | Den(i) ->   push(applyenv(topenv() ,i), top(tempvalstack) )
               | Let(i,e1,e2) -> let arg=top(top(tempvalstack)) in pop(top(tempvalstack) );
              newframes(e2, bind(topenv() , i, arg)) done;
      let valore= top(top(tempvalstack)) in  pop(top(tempvalstack)); pop(cstack); popenv(); pop(tempvalstack);
       push(valore,top(tempvalstack));
       while not(empty(cstack)) do
      while not(empty(top(cstack))) do  (match top(top(cstack)) with
         |Expr1(x) -> (pop(top(cstack)); push(Expr2(x), top(cstack));
         (match x with
               | Sum(a,b) -> push(Expr1(a), top(cstack) ); push(Expr1(b), top(cstack) )
               | Let(i,e1,e2) -> push(Expr1(e1), top(cstack) )
               | _ -> ()))
        |Expr2(x) -> (pop(top(cstack) ); (match x with
               | Eint(n) ->  push(Int(n),top(tempvalstack) )
               | Den(i) ->   push(applyenv(topenv() ,i), top(tempvalstack) )
               | Let(i,e1,e2) -> let arg=top(top(tempvalstack)) in pop(top(tempvalstack) );
              newframes(e2, bind(topenv() , i, arg)) done;
      let valore= top(top(tempvalstack)) in  pop(top(tempvalstack)); pop(cstack); popenv(); pop(tempvalstack);
       push(valore,top(tempvalstack))
      done; let valore = top(top(tempvalstack)) in pop(tempvalstack); valore
```

```
cstack
[|[|Expr2 (Let ("x", Eint 3, Let ("y", Eint 5, Sum (Den "x", Den "y")))); Expr2 (Eint 3);...|],{contents=0};
[|Expr1 (Eint 0)|], {contents=-1};...|],...
```

```
let semres r = push(emptystack(1,Unbound),tempvalstack); let tframe = emptystack(tframesize(e),Unbound) in
push(tframe,tempvalstack); currentenv := r; push(Int(3),top(tempvalstack));
```

Un esempio di simulazione manuale 9

```
let sem ((e:exp), (r:eval env)) = ...
     (match top(top(cstack)) with
         |Expr1(x) -> (pop(top(cstack)); push(Expr2(x), top(cstack));
         (match x with
                | Sum(a,b) -> push(Expr1(a), top(cstack) ); push(Expr1(b), top(cstack) )
                | Let(i,e1,e2) -> push(Expr1(e1), top(cstack) )
                | _ -> ()))
       |Expr2(x) -> (pop(top(cstack) ); (match x with
                | Eint(n) ->  push(Int(n),top(tempvalstack) )
                | Den(i) ->   push(applyenv(topenv() ,i), top(tempvalstack) )
                | Let(i,e1,e2) -> let arg=top(top(tempvalstack)) in pop(top(tempvalstack) );
              newframes(e2, bind(topenv() , i, arg))
     while not(empty(top(cstack))) do  (match top(top(cstack)) with
         |Expr1(x) -> (pop(top(cstack)); push(Expr2(x), top(cstack));
         (match x with
                | Sum(a,b) -> push(Expr1(a), top(cstack) ); push(Expr1(b), top(cstack) )
                | Let(i,e1,e2) -> push(Expr1(e1), top(cstack) )
                | _ -> ()))
       |Expr2(x) -> (pop(top(cstack) ); (match x with
                | Eint(n) ->  push(Int(n),top(tempvalstack) )
                | Den(i) ->   push(applyenv(topenv() ,i), top(tempvalstack) )
                | Let(i,e1,e2) -> let arg=top(top(tempvalstack)) in pop(top(tempvalstack) );
              newframes(e2, bind(topenv() , i, arg)) done;
      let valore= top(top(tempvalstack)) in  pop(top(tempvalstack)); pop(cstack); popenv(); pop(tempvalstack);
        push(valore,top(tempvalstack));
       while not(empty(cstack)) do
      .....
      let valore= top(top(tempvalstack)) in  pop(top(tempvalstack)); pop(cstack); popenv(); pop(tempvalstack);
        push(valore,top(tempvalstack))
      done; let valore = top(top(tempvalstack)) in pop(tempvalstack); valore
```

```
let newframes(e,rho) =
   let cframe = emptystack(cframesize(e),Expr1(e)) in
   let tframe = emptystack(tframesize(e),Unbound) in
   push(Expr1(e),cframe); push(cframe,cstack); push(tframe,tempvalstack);
   if rho = !currentenv then (push([||],namestack); push([||],evalstack); push(rho,slinkstack);
   currentenv := lungh(namestack) ) else currentenv := rho

let bind ((r:eval env),i,d) = push(Array.create 1 i,namestack); push(Array.create 1 d,evalstack);
 push(r,slinkstack);(lungh(namestack): eval env)
```

Un esempio di simulazione manuale 10

```
let sem ((e:exp), (r:eval env)) = ...
      while not(empty(top(cstack))) do  (match top(top(cstack)) with
         |Expr1(x) -> (pop(top(cstack)); push(Expr2(x), top(cstack));
         (match x with
               | Sum(a,b) -> push(Expr1(a), top(cstack) ); push(Expr1(b), top(cstack) )
               | Let(i,e1,e2) -> push(Expr1(e1), top(cstack) )
               | _ -> ()))
        |Expr2(x) -> (pop(top(cstack) ); (match x with
               | Eint(n) ->  push(Int(n),top(tempvalstack) )
               | Den(i) ->   push(applyenv(topenv() ,i), top(tempvalstack) )
               | Let(i,e1,e2) -> let arg=top(top(tempvalstack)) in pop(top(tempvalstack) );
              newframes(e2, bind(topenv() , i, arg)) done;
      let valore= top(top(tempvalstack)) in  pop(top(tempvalstack)); pop(cstack); popenv(); pop(tempvalstack);
       push(valore,top(tempvalstack));
       while not(empty(cstack)) do
      while not(empty(top(cstack))) do  (match top(top(cstack)) with
         |Expr1(x) -> (pop(top(cstack)); push(Expr2(x), top(cstack));
         (match x with
               | Sum(a,b) -> push(Expr1(a), top(cstack) ); push(Expr1(b), top(cstack) )
               | Let(i,e1,e2) -> push(Expr1(e1), top(cstack) )
               | _ -> ()))
        |Expr2(x) -> (pop(top(cstack) ); (match x with
               | Eint(n) ->  push(Int(n),top(tempvalstack) )
               | Den(i) ->   push(applyenv(topenv() ,i), top(tempvalstack) )
               | Let(i,e1,e2) -> let arg=top(top(tempvalstack)) in pop(top(tempvalstack) );
              newframes(e2, bind(topenv() , i, arg)) done;
      let valore= top(top(tempvalstack)) in  pop(top(tempvalstack)); pop(cstack); popenv(); pop(tempvalstack);
        push(valore,top(tempvalstack))
      done; let valore = top(top(tempvalstack)) in pop(tempvalstack); valore
```

```
cstack
[|[|Expr2 (Let ("x", Eint 3, Let ("y", Eint 5, Sum (Den "x", Den "y")))); Expr2 (Eint 3);...|],{contents=-1};
[|Expr1 (Let ("y", Eint 5, Sum (Den "x", Den "y")); ...|], {contents=0};...|],...

namestack
[|[| "x" ; "dummy" ;...|],{contents=0};...
slinkstack
[|[| -1 ; 0 ;...|],{contents=0};...


let semres r = push(emptystack(1,Unbound),tempvalstack); let tframe = emptystack(2,Unbound) in
push(tframe,tempvalstack); currentenv := r; push(Int(3),top(tempvalstack)); let arg=top(top(tempvalstack)) in
pop(top(tempvalstack) ); push(!currentenv, slinkstack); push(Array.create 1 arg,evalstack);
let tframe = emptystack(2,Unbound) in push(tframe,tempvalstack); currentenv := !currentenv + 1;
```

Un esempio di simulazione manuale 11

```
let sem ((e:exp), (r:eval env)) = ...
     (match top(top(cstack)) with
        |Expr1(x) -> (pop(top(cstack)); push(Expr2(x), top(cstack));
        (match x with
              | Sum(a,b) -> push(Expr1(a), top(cstack) ); push(Expr1(b), top(cstack) )
              | Let(i,e1,e2) -> push(Expr1(e1), top(cstack) )
              | _ -> ()))
       |Expr2(x) -> (pop(top(cstack) ); (match x with
              | Eint(n) ->  push(Int(n),top(tempvalstack) )
              | Den(i) ->   push(applyenv(topenv() ,i), top(tempvalstack) )
              | Let(i,e1,e2) -> let arg=top(top(tempvalstack)) in pop(top(tempvalstack) );
             newframes(e2, bind(topenv() , i, arg))
     while not(empty(top(cstack))) do  (match top(top(cstack)) with
        |Expr1(x) -> (pop(top(cstack)); push(Expr2(x), top(cstack));
        (match x with
              | Sum(a,b) -> push(Expr1(a), top(cstack) ); push(Expr1(b), top(cstack) )
              | Let(i,e1,e2) -> push(Expr1(e1), top(cstack) )
              | _ -> ()))
       |Expr2(x) -> (pop(top(cstack) ); (match x with
              | Eint(n) ->  push(Int(n),top(tempvalstack) )
              | Den(i) ->   push(applyenv(topenv() ,i), top(tempvalstack) )
              | Let(i,e1,e2) -> let arg=top(top(tempvalstack)) in pop(top(tempvalstack) );
             newframes(e2, bind(topenv() , i, arg)) done;
      let valore= top(top(tempvalstack)) in  pop(top(tempvalstack)); pop(cstack); popenv(); pop(tempvalstack);
        push(valore,top(tempvalstack));
       while not(empty(cstack)) do
      .....
      let valore= top(top(tempvalstack)) in  pop(top(tempvalstack)); pop(cstack); popenv(); pop(tempvalstack);
        push(valore,top(tempvalstack))
     done; let valore = top(top(tempvalstack)) in pop(tempvalstack); valore
```

Un esempio di simulazione manuale 12

```
let sem ((e:exp), (r:eval env)) = ...
      while not(empty(top(cstack))) do  (match top(top(cstack)) with
         |Expr1(x) -> (pop(top(cstack)); push(Expr2(x), top(cstack));
         (match x with
               | Sum(a,b) -> push(Expr1(a), top(cstack) ); push(Expr1(b), top(cstack) )
               | Let(i,e1,e2) -> push(Expr1(e1), top(cstack) )
               | _ -> ()))
        |Expr2(x) -> (pop(top(cstack) ); (match x with
               | Eint(n) ->  push(Int(n),top(tempvalstack) )
               | Den(i) ->   push(applyenv(topenv() ,i), top(tempvalstack) )
               | Let(i,e1,e2) -> let arg=top(top(tempvalstack)) in pop(top(tempvalstack) );
              newframes(e2, bind(topenv() , i, arg)) done;
      let valore= top(top(tempvalstack)) in  pop(top(tempvalstack)); pop(cstack); popenv(); pop(tempvalstack);
       push(valore,top(tempvalstack));
       while not(empty(cstack)) do
      while not(empty(top(cstack))) do  (match top(top(cstack)) with
         |Expr1(x) -> (pop(top(cstack)); push(Expr2(x), top(cstack));
         (match x with
               | Sum(a,b) -> push(Expr1(a), top(cstack) ); push(Expr1(b), top(cstack) )
               | Let(i,e1,e2) -> push(Expr1(e1), top(cstack) )
               | _ -> ()))
        |Expr2(x) -> (pop(top(cstack) ); (match x with
               | Eint(n) ->  push(Int(n),top(tempvalstack) )
               | Den(i) ->   push(applyenv(topenv() ,i), top(tempvalstack) )
               | Let(i,e1,e2) -> let arg=top(top(tempvalstack)) in pop(top(tempvalstack) );
              newframes(e2, bind(topenv() , i, arg)) done;
      let valore= top(top(tempvalstack)) in  pop(top(tempvalstack)); pop(cstack); popenv(); pop(tempvalstack);
        push(valore,top(tempvalstack))
      done; let valore = top(top(tempvalstack)) in pop(tempvalstack); valore
```

```
cstack
[|[|Expr2 (Let ("x", Eint 3, Let ("y", Eint 5, Sum (Den "x", Den "y")))); Expr2 (Eint 3);...|],{contents=-1};
[|Expr2 (Let ("y", Eint 5, Sum (Den "x", Den "y")); Expr2(Eint 5); ..|], {contents=1};...|],...
namestack
[|[| "x" ; "dummy" ;...|],{contents=0};...
slinkstack
[|[| -1 ; 0 ;...|],{contents=0};...

let semres r = push(emptystack(1,Unbound),tempvalstack); let tframe = emptystack(2,Unbound) in
push(tframe,tempvalstack); currentenv := r; push(Int(3),top(tempvalstack)); let arg=top(top(tempvalstack)) in
pop(top(tempvalstack) ); push(!currentenv, slinkstack); push(Array.create 1 arg,evalstack);
let tframe = emptystack(2,Unbound) in push(tframe,tempvalstack); currentenv := !currentenv + 1;
```

Un esempio di simulazione manuale 13

```
let sem ((e:exp), (r:eval env)) = ...
     (match top(top(cstack)) with
        |Expr1(x) -> (pop(top(cstack)); push(Expr2(x), top(cstack));
        (match x with
              | Sum(a,b) -> push(Expr1(a), top(cstack) ); push(Expr1(b), top(cstack) )
              | Let(i,e1,e2) -> push(Expr1(e1), top(cstack) )
              | _ -> ()))
       |Expr2(x) -> (pop(top(cstack) ); (match x with
              | Eint(n) ->  push(Int(n),top(tempvalstack) )
              | Den(i) ->   push(applyenv(topenv() ,i), top(tempvalstack) )
              | Let(i,e1,e2) -> let arg=top(top(tempvalstack)) in pop(top(tempvalstack) );
             newframes(e2, bind(topenv() , i, arg))
     while not(empty(top(cstack))) do  (match top(top(cstack)) with
        |Expr1(x) -> (pop(top(cstack)); push(Expr2(x), top(cstack));
        (match x with
              | Sum(a,b) -> push(Expr1(a), top(cstack) ); push(Expr1(b), top(cstack) )
              | Let(i,e1,e2) -> push(Expr1(e1), top(cstack) )
              | _ -> ()))
       |Expr2(x) -> (pop(top(cstack) ); (match x with
              | Eint(n) ->  push(Int(n),top(tempvalstack) )
              | Den(i) ->   push(applyenv(topenv() ,i), top(tempvalstack) )
              | Let(i,e1,e2) -> let arg=top(top(tempvalstack)) in pop(top(tempvalstack) );
             newframes(e2, bind(topenv() , i, arg)) done;
      let valore= top(top(tempvalstack)) in  pop(top(tempvalstack)); pop(cstack); popenv(); pop(tempvalstack);
        push(valore,top(tempvalstack));
       while not(empty(cstack)) do
      .....
      let valore= top(top(tempvalstack)) in  pop(top(tempvalstack)); pop(cstack); popenv(); pop(tempvalstack);
        push(valore,top(tempvalstack))
     done; let valore = top(top(tempvalstack)) in pop(tempvalstack); valore
```

Un esempio di simulazione manuale 14

```
let sem ((e:exp), (r:eval env)) = ...
      while not(empty(top(cstack))) do  (match top(top(cstack)) with
         |Expr1(x) -> (pop(top(cstack)); push(Expr2(x), top(cstack));
         (match x with
               | Sum(a,b) -> push(Expr1(a), top(cstack) ); push(Expr1(b), top(cstack) )
               | Let(i,e1,e2) -> push(Expr1(e1), top(cstack) )
               | _ -> ()))
        |Expr2(x) -> (pop(top(cstack) ); (match x with
               | Eint(n) ->  push(Int(n),top(tempvalstack) )
               | Den(i) ->   push(applyenv(topenv() ,i), top(tempvalstack) )
               | Let(i,e1,e2) -> let arg=top(top(tempvalstack)) in pop(top(tempvalstack) );
              newframes(e2, bind(topenv() , i, arg)) done;
      let valore= top(top(tempvalstack)) in  pop(top(tempvalstack)); pop(cstack); popenv(); pop(tempvalstack);
        push(valore,top(tempvalstack));
       while not(empty(cstack)) do
      while not(empty(top(cstack))) do  (match top(top(cstack)) with
         |Expr1(x) -> (pop(top(cstack)); push(Expr2(x), top(cstack));
         (match x with
               | Sum(a,b) -> push(Expr1(a), top(cstack) ); push(Expr1(b), top(cstack) )
               | Let(i,e1,e2) -> push(Expr1(e1), top(cstack) )
               | _ -> ()))
        |Expr2(x) -> (pop(top(cstack) ); (match x with
               | Eint(n) ->  push(Int(n),top(tempvalstack) )
               | Den(i) ->   push(applyenv(topenv() ,i), top(tempvalstack) )
               | Let(i,e1,e2) -> let arg=top(top(tempvalstack)) in pop(top(tempvalstack) );
              newframes(e2, bind(topenv() , i, arg)) done;
      let valore= top(top(tempvalstack)) in  pop(top(tempvalstack)); pop(cstack); popenv(); pop(tempvalstack);
        push(valore,top(tempvalstack))
      done; let valore = top(top(tempvalstack)) in pop(tempvalstack); valore
```

```
cstack
[|[|Expr2 (Let ("x", Eint 3, Let ("y", Eint 5, Sum (Den "x", Den "y")))); Expr2 (Eint 3);...|],{contents=-1};
[|Expr2 (Let ("y", Eint 5, Sum (Den "x", Den "y")); Expr2(Eint 5); ..|], {contents=0};...|],...

namestack
[|[| "x" ; "dummy" ;...|],{contents=0};...
slinkstack
[|[| -1 ; 0 ;...|],{contents=0};...

let semres r = push(emptystack(1,Unbound),tempvalstack); let tframe = emptystack(2,Unbound) in
push(tframe,tempvalstack); currentenv := r; push(Int(3),top(tempvalstack)); let arg=top(top(tempvalstack)) in
pop(top(tempvalstack) ); push(!currentenv, slinkstack); push(Array.create 1 arg,evalstack);
let tframe = emptystack(2,Unbound) in push(tframe,tempvalstack); currentenv := !currentenv + 1;
push(Int(5),top(tempvalstack));
```

Un esempio di simulazione manuale 15

```
let sem ((e:exp), (r:eval env)) = ...
     (match top(top(cstack)) with
        |Expr1(x) -> (pop(top(cstack)); push(Expr2(x), top(cstack));
        (match x with
              | Sum(a,b) -> push(Expr1(a), top(cstack) ); push(Expr1(b), top(cstack) )
              | Let(i,e1,e2) -> push(Expr1(e1), top(cstack) )
              | _ -> ()))
       |Expr2(x) -> (pop(top(cstack) ); (match x with
              | Eint(n) ->  push(Int(n),top(tempvalstack) )
              | Den(i) ->   push(applyenv(topenv() ,i), top(tempvalstack) )
              | Let(i,e1,e2) -> let arg=top(top(tempvalstack)) in pop(top(tempvalstack) );
             newframes(e2, bind(topenv() , i, arg))
     while not(empty(top(cstack))) do  (match top(top(cstack)) with
        |Expr1(x) -> (pop(top(cstack)); push(Expr2(x), top(cstack));
        (match x with
              | Sum(a,b) -> push(Expr1(a), top(cstack) ); push(Expr1(b), top(cstack) )
              | Let(i,e1,e2) -> push(Expr1(e1), top(cstack) )
              | _ -> ()))
       |Expr2(x) -> (pop(top(cstack) ); (match x with
              | Eint(n) ->  push(Int(n),top(tempvalstack) )
              | Den(i) ->   push(applyenv(topenv() ,i), top(tempvalstack) )
              | Let(i,e1,e2) -> let arg=top(top(tempvalstack)) in pop(top(tempvalstack) );
             newframes(e2, bind(topenv() , i, arg)) done;
      let valore= top(top(tempvalstack)) in  pop(top(tempvalstack)); pop(cstack); popenv(); pop(tempvalstack);
       push(valore,top(tempvalstack));
      while not(empty(cstack)) do
      .....
      let valore= top(top(tempvalstack)) in  pop(top(tempvalstack)); pop(cstack); popenv(); pop(tempvalstack);
       push(valore,top(tempvalstack))
     done; let valore = top(top(tempvalstack)) in pop(tempvalstack); valore

let newframes(e,rho) =
   let cframe = emptystack(cframesize(e),Expr1(e)) in
   let tframe = emptystack(tframesize(e),Unbound) in
   push(Expr1(e),cframe); push(cframe,cstack); push(tframe,tempvalstack);
   if rho = !currentenv then (push([||],namestack); push([||],evalstack); push(rho,slinkstack);
   currentenv := lungh(namestack) ) else currentenv := rho

let bind ((r:eval env),i,d) = push(Array.create 1 i,namestack); push(Array.create 1 d,evalstack);
 push(r,slinkstack);(lungh(namestack): eval env)
```

eccetera!

Quando si trova un `Den x`

```
let applyenv ((x: eval env), (y: ide)) =
       let n = ref(x) in
       let den = ref(Unbound) in
       while !n > -1 do
          let lenv = access(namestack,!n) in
          let nl = Array.length lenv in
          let index = ref(0) in
          while !index < nl do
                    if Array.get lenv !index = y then
                    (den := Array.get (access(evalstack,!n)) !index;
                    index := nl)
                    else index := !index + 1
          done;
          if not(!den = Unbound) then  n := -1 else n := access(slinkstack,!n)
       done;
       !den
```

- `namestack` è completamente noto
- `slinkstack` è noto nella "versione statica"
- un riferimento che si risolve in due passi lungo la catena statica in seconda posizione

```
Array.get (access(evalstack,
   access(slinkstack,access(slinkstack,  !currentenv)))) 1
```

Il codice residuo per il piccolo esempio

```
let semres r = push(emptystack(1,Unbound),tempvalstack);
  let tframe = emptystack(1,Unbound) in push(tframe,tempvalstack); currentenv := r;
  push(Int(3),top(tempvalstack));
  let arg=top(top(tempvalstack)) in pop(top(tempvalstack) );
  push(!currentenv, slinkstack); push(Array.create 1 arg,evalstack);
  let tframe = emptystack(1,Unbound) in push(tframe,tempvalstack);
  currentenv := !currentenv + 1;
  push(Int(5),top(tempvalstack));
  let arg=top(top(tempvalstack)) in pop(top(tempvalstack) );
  push(!currentenv, slinkstack); push(Array.create 1 arg,evalstack);
  let tframe = emptystack(2,Unbound) in
  push(tframe,tempvalstack); currentenv := !currentenv + 1;
  push(Array.get(access(evalstack, !currentenv)) 0, top(tempvalstack));
  push(Array.get(access(evalstack, access(slinkstack,!currentenv))) 0, top(tempvalstack));
  let arg1 = top(top(tempvalstack)) in pop(top(tempvalstack));
  let arg2 = top(top(tempvalstack)) in pop(top(tempvalstack));
  push(plus(arg1,arg2),top(tempvalstack));
  let valore= top(top(tempvalstack)) in  pop(top(tempvalstack));
  pop(evalstack); pop(slinkstack); pop(tempvalstack);
  push(valore,top(tempvalstack));
  let valore= top(top(tempvalstack)) in  pop(top(tempvalstack));
  pop(evalstack); pop(slinkstack); pop(tempvalstack);
  push(valore,top(tempvalstack));
  let valore = top(top(tempvalstack)) in pop(tempvalstack); valore
```

Il supporto a tempo di esecuzione

- restano nel supporto solo le strutture dati e le funzioni utilizzate nel codice residuo
  - `tempvalstack, slinkstack, evalstack, tagstack`
  - l'interprete, `cstack`, `namestack` e la maggior parte delle funzioni ausiliarie per gestire i frames e l'ambiente vengono eliminate
- la situazione è simile per i linguaggi imperativi e a oggetti
  - memoria a pila e heap ovviamente restano nel supporto