

Semântica de CALCF (2)

A função semântica **I** de CALCF:

$$I : \text{CALCF} \times \text{ENV} \rightarrow \text{RESULT}$$

CALCF = conjunto dos programas abertos

ENV = conjunto dos ambientes válidos

RESULT = conjunto dos significados (denotações)

- Um significado pode ser um valor inteiro ou uma função(representada por uma abstracção):

$$\text{RESULT} = \text{Integer} \cup \text{Abstraction} \cup \{ \text{error} \}$$

Interpretador de CALCF (2)

- Algoritmo $\text{eval}(E, \text{env})$ para calcular a denotação de uma expressão E de CALCF:

$\text{eval} : \text{CALCF} \times \text{ENV} \rightarrow \text{RESULT}$

```
eval( num(n) , env )       $\triangleq$  n
eval( id(s) , env )        $\triangleq$  env.Find(s)
...
eval( fun(s, B), env )     $\triangleq$  abstraction(s, B)
eval( call(E1, E2) , env )  $\triangleq$  fun = eval(E1, env )
    if fun is abstraction(s, B) then
    [ env'=env.BeginScope();
      env'.Assoc(s, eval(E2, env )) ;
      val = eval(B, env' );
      env'.EndScope();
      val ]
    else error
```

Interpretador de CALCF (2)

- Avaliação do programa:

```
decl f=(fun x -> x+1) in
decl g = (fun y -> f(y)+2) in
      decl x = g(2)
      in x+x
```

Interpretador de CALCF (2)

- Avaliação do programa:

```
decl f=(fun x -> x+1) in
decl g = (fun y -> f(y)+2) in
      decl x = g(2)
      in x+x
```

eval(P1,0) =

Interpretador de CALCF (2)

- Avaliação do programa:

```
decl f=(fun x -> x+1) in
decl g = (fun y -> f(y)+2) in
decl x = g(2)
in x+x
```

eval(P1,0) =

eval(P2, [f=abs(x,x+1)]) =

Interpretador de CALCF (2)

- Avaliação do programa:

```
decl f=(fun x -> x+1) in
decl g = (fun y -> f(y)+2) in
    decl x = g(2)
    in x+x
```

eval(P1,0) =

eval(P2, [f=abs(x,x+1)]) =

eval(P3, [g=abs(y,f(y)+2), f=abs(x,x+1)]) =

Interpretador de CALCF (2)

- Avaliação do programa:

```
decl f=(fun x -> x+1) in
decl g = (fun y -> f(y)+2) in
      decl x = g(2)
      in x+x
```

$\text{eval}(P1, 0) =$

$\text{eval}(P2, [f=\text{abs}(x, x+1)]) =$

$\text{eval}(P3, [g=\text{abs}(y, f(y)+2), f=\text{abs}(x, x+1)]) =$

$\text{eval}(g(2), [g=\text{abs}(y, f(y)+2), f=\text{abs}(x, x+1)]) =$

Interpretador de CALCF (2)

- Avaliação do programa:

```
decl f=(fun x -> x+1) in
decl g = (fun y -> f(y)+2) in
      decl x = g(2)
      in x+x
```

$\text{eval}(P1, 0) =$

$\text{eval}(P2, [f=\text{abs}(x, x+1)]) =$

$\text{eval}(P3, [g=\text{abs}(y, f(y)+2), f=\text{abs}(x, x+1)]) =$

$\text{eval}(g(2), [g=\text{abs}(y, f(y)+2), f=\text{abs}(x, x+1)]) =$

$\text{eval}(g, [g=\text{abs}(y, f(y)+2), f=\text{abs}(x, x+1)]) = \text{abs}(y, f(y)+2)$

Interpretador de CALCF (2)

- Avaliação do programa:

```
decl f=(fun x -> x+1) in
decl g = (fun y -> f(y)+2) in
      decl x = g(2)
      in x+x
```

$\text{eval}(P1, 0) =$

$\text{eval}(P2, [f=\text{abs}(x, x+1)]) =$

$\text{eval}(P3, [g=\text{abs}(y, f(y)+2), f=\text{abs}(x, x+1)]) =$

$\text{eval}(g(2), [g=\text{abs}(y, f(y)+2), f=\text{abs}(x, x+1)]) =$

$\text{eval}(f(y)+2, [y=2, g=\text{abs}(y, f(y)+2), f=\text{abs}(x, x+1)]) =$

Interpretador de CALCF (2)

- Avaliação do programa:

```
decl f=(fun x -> x+1) in
decl g = (fun y -> f(y)+2) in
      decl x = g(2)
      in x+x
```

$\text{eval}(P1, 0) =$

$\text{eval}(P2, [f=\text{abs}(x, x+1)]) =$

$\text{eval}(P3, [g=\text{abs}(y, f(y)+2), f=\text{abs}(x, x+1)]) =$

$\text{eval}(g(2), [g=\text{abs}(y, f(y)+2), f=\text{abs}(x, x+1)]) =$

$\text{eval}(f(y)+2, [y=2, g=\text{abs}(y, f(y)+2), f=\text{abs}(x, x+1)]) =$

$\text{eval}(f, [y=2, g=\text{abs}(y, f(y)+2), f=\text{abs}(x, x+1)]) = \text{abs}(x, x+1)$

Interpretador de CALCF (2)

- Avaliação do programa:

```
decl f=(fun x -> x+1) in
decl g = (fun y -> f(y)+2) in
      decl x = g(2)
      in x+x
```

$\text{eval}(P1, 0) =$

$\text{eval}(P2, [f=\text{abs}(x, x+1)]) =$

$\text{eval}(P3, [g=\text{abs}(y, f(y)+2), f=\text{abs}(x, x+1)]) =$

$\text{eval}(g(2), [g=\text{abs}(y, f(y)+2), f=\text{abs}(x, x+1)]) =$

$\text{eval}(f(y)+2, [y=2, g=\text{abs}(y, f(y)+2), f=\text{abs}(x, x+1)]) =$

$\text{eval}(x+1, [x=2, g=\text{abs}(y, f(y)+2), f=\text{abs}(x, x+1)]) = 3$

Interpretador de CALCF (2)

- Avaliação do programa:

```
decl f=(fun x -> x+1) in
decl g = (fun y -> f(y)+2) in
      decl x = g(2)
      in x+x
```

$\text{eval}(P1, 0) =$

$\text{eval}(P2, [f=\text{abs}(x, x+1)]) =$

$\text{eval}(P3, [g=\text{abs}(y, f(y)+2), f=\text{abs}(x, x+1)]) =$

$\text{eval}(g(2), [g=\text{abs}(y, f(y)+2), f=\text{abs}(x, x+1)]) =$

$\text{eval}(f(y)+2, [y=2, g=\text{abs}(y, f(y)+2), f=\text{abs}(x, x+1)]) = 5$

Interpretador de CALCF (2)

- Avaliação do programa:

```
decl f=(fun x -> x+1) in
decl g = (fun y -> f(y)+2) in
decl x = g(2)
in x+x
```

$\text{eval}(P1, 0) =$

$\text{eval}(P2, [f=\text{abs}(x, x+1)]) =$

$\text{eval}(P3, [g=\text{abs}(y, f(y)+2), f=\text{abs}(x, x+1)]) =$

$\text{eval}(x+x, [x=5, g=\text{abs}(y, f(y)+2), f=\text{abs}(x, x+1)]) = 10$

Interpretador de CALCF (2)

- Outro exemplo:

```
decl x=1 in
  decl f = (fun y -> y+x) in
    decl g = (fun x -> x+f(x))
              in g(2)
```

$E = [g = \text{abs}(x, x + f(x)); f = \text{abs}(y, y + x); x = 1]$

$\text{eval}(g(2), E) =$

$\text{eval}(x + f(x), [x = 2; g = \text{abs}(x, x + f(x)), f = \text{abs}(y, y + x)]) =$

$2 + \text{eval}(f(x), [x = 2; g = \text{abs}(x, x + f(x)), f = \text{abs}(y, y + x)]) =$

$2 + \text{eval}(y + x, [y = 2; x = 2; g = \text{abs}(x, x + f(x)), f = \text{abs}(y, y + x)]) =$

$2 + 2 + 2 = 6$

- Seguindo a sua intuição, qual é o valor deste programa?

Interpretador de CALCF (2)

- Outro exemplo:

```
decl x=1 in
  decl f = (fun y -> y+x) in
    decl g = (fun x -> x+f(x))
              in g(2)
```

$E = [g = \text{abs}(x, x+f(x)); f = \text{abs}(y, y+x); x=1]$

$\text{eval}(g(2), E) =$

$\text{eval}(x+f(x), [x=2; g = \text{abs}(x, x+f(x)), f = \text{abs}(y, y+x)]) =$

$2 + \text{eval}(f(x), [x=2; g = \text{abs}(x, x+f(x)), f = \text{abs}(y, y+x)]) =$

$2 + \text{eval}(y+x, [y=2; x=2; g = \text{abs}(x, x+f(x)), f = \text{abs}(y, y+x)]) =$

$2 + 2 + 2 = 6$

- O valor do programa deveria ser 5 ! O que falhou???

Resolução dinâmica de nomes

- A semântica simplificada (2) que definimos para a linguagem CALCF adopta a "regra dinâmica" de resolução de nomes (**dynamic scoping**).
- Segundo esta regra, os valores dos nomes **não locais** são interpretados no contexto da chamada da função, em vez do contexto da definição.
- Historicamente, algumas linguagens de programação (ex: Lisp, JavaScript 1.0) adoptaram a resolução dinâmica de nomes, por ser de implementação mais simples.
- Actualmente, é considerado um mecanismo indesejável e até **incorrecto** (por não respeitar o princípio da substituição), apesar de às vezes "dar jeito" interpretar nomes não locais no contexto da chamada (por exemplo: "hostname").

Princípio da substitutividade

- O valor de qualquer expressão permanece inalterado sempre que nela se substitui uma subexpressão por outra expressão com o mesmo significado / valor.
- A semântica da linguagem CALCF viola este princípio, pois os dois programas seguintes têm valores diferentes:

```
decl x=1 in
  decl f = (fun y→y+x) in
    decl g = (fun x→x+f(x))
      in g(2)
```

```
decl x=1 in
  decl f = (fun y→y+1) in
    decl g = (fun x→x+f(x))
      in g(2)
```

Resolução estática de nomes

- Todas as linguagens de programação modernas adoptam a regra da **resolução estática** de identificadores (**static scoping**).
- Segundo esta regra, os valores dos identificadores livres que ocorram no corpo de abstrações são interpretados no contexto em que as abstrações ocorrem (na definição das funções), e não no contexto da chamada.
- Para implementar esta semântica de forma eficiente é necessário usar um domínio de resultados mais rico, em que as funções são representadas por entidades chamadas "**fechos**".

Semântica de CALCF (3)

- A (nova) função semântica **I** de CALCF:

$$\mathbf{I : CALCF \times ENV \rightarrow RESULT}$$

CALCF = conjunto dos programas **abertos**

ENV = conjunto dos ambientes

RESULT = conjunto dos significados (denotações)

Os resultados podem ser valores inteiros, fechos (uma abstracção + um ambiente), ou um erro.

$$\mathbf{RESULT = Integer \cup Closure \cup \{ error \}}$$

Resultados de CALCF

- Os significados de programas da linguagem CALCF podem ser apresentados como um tipo indutivo

Tipo de dados **RESULT** com os constructores num e closure

```
num:      Integer → RESULT
closure:  String × CALCF × ENV → RESULT
error:    void → RESULT
```

Um fecho representa uma função através de um triplo contendo o **parâmetro**, o **corpo**, e o **ambiente** que regista os valores dos nomes livres no corpo

Assim, ao contrário de uma abstracção, um fecho é efectivamente um valor “fechado” (não depende de nenhum nome externo).

Ambiente "mutável"

- Na prática, é conveniente implementar ambientes usando uma estrutura de dados mutável:

Environ BeginScope()

- Cria um novo nível **vazio**, onde serão colocadas as ligações de um novo âmbito local.
- Não pode existir mais que uma ligação para um mesmo identificador no mesmo nível.

Environ EndScope()

- Devolve o ambiente no estado anterior à última operação BeginScope().
- Mas **não destrói** o último nível pois podem existir no contexto de execução fechos que o referem!

Interpretador de CALCF (3)

- Algoritmo $\text{eval}(E, \text{env})$ para calcular o valor de uma expressão E de CALCF:

$\text{eval} : \text{CALCF} \times \text{ENV} \rightarrow \text{RESULT}$

```
eval( fun( $s$ ,  $B$ ),  $\text{env}$  )       $\triangleq$  closure( $s$ ,  $\text{env}$ ,  $B$ )  
eval( call( $E_1$ ,  $E_2$ ) ,  $\text{env}$  )  $\triangleq$   $\text{fun} = \text{eval}(E_1, \text{env})$   
    if  $\text{fun}$  is closure( $s$ ,  $\text{envc}$ ,  $B$ ) then  
    [  $\text{envloc} = \text{envc}.\text{BeginScope}()$ ;  
       $\text{envloc}.\text{Assoc}(s, \text{eval}(E_2, \text{env}))$  ;  
       $\text{val} = \text{eval}(B, \text{envloc})$  ;  
       $\text{envc} = \text{envloc}.\text{EndScope}()$  ;  
       $\text{val}$  ]  
    else error
```

Avaliação de Funções

- Exemplo: avaliar o seguinte programa, na semântica usando ambientes e fechos.

```
decl x=1 in
  decl f = (fun y -> y+x) in
    decl g = (fun x -> x+f(x))
      in g(2)
```

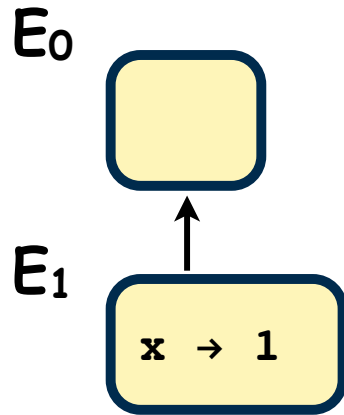
Exemplo

E_0



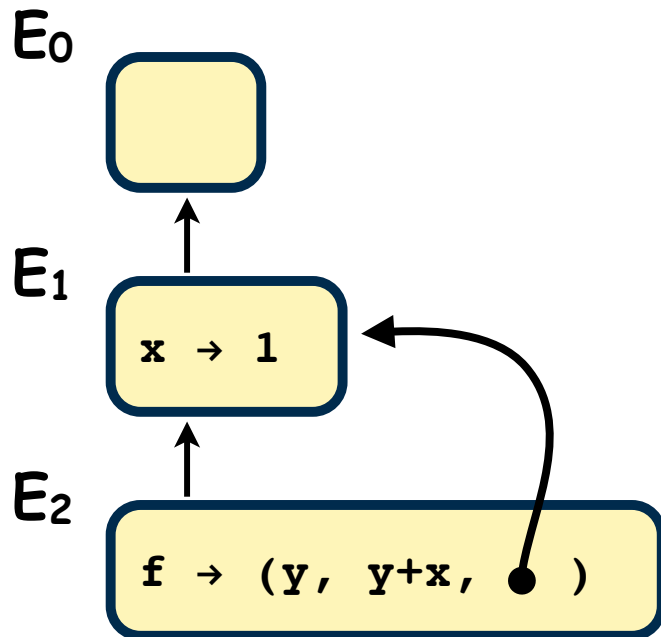
```
decl x=1 in
  decl f = (fun y -> y+x) in
    decl g = (fun x -> x+f(x))
      in g(2)
```


Exemplo



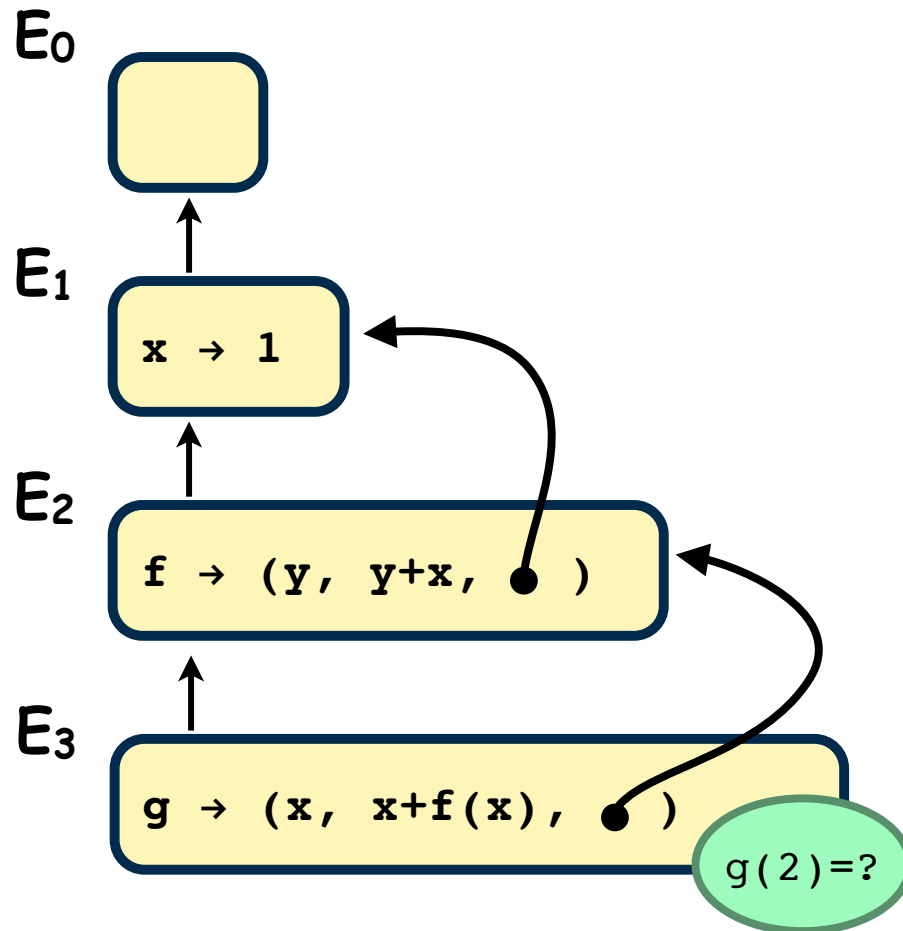
```
decl x=1 in  
  decl f = (fun y -> y+x) in  
    decl g = (fun x -> x+f(x))  
      in g(2)
```

Exemplo



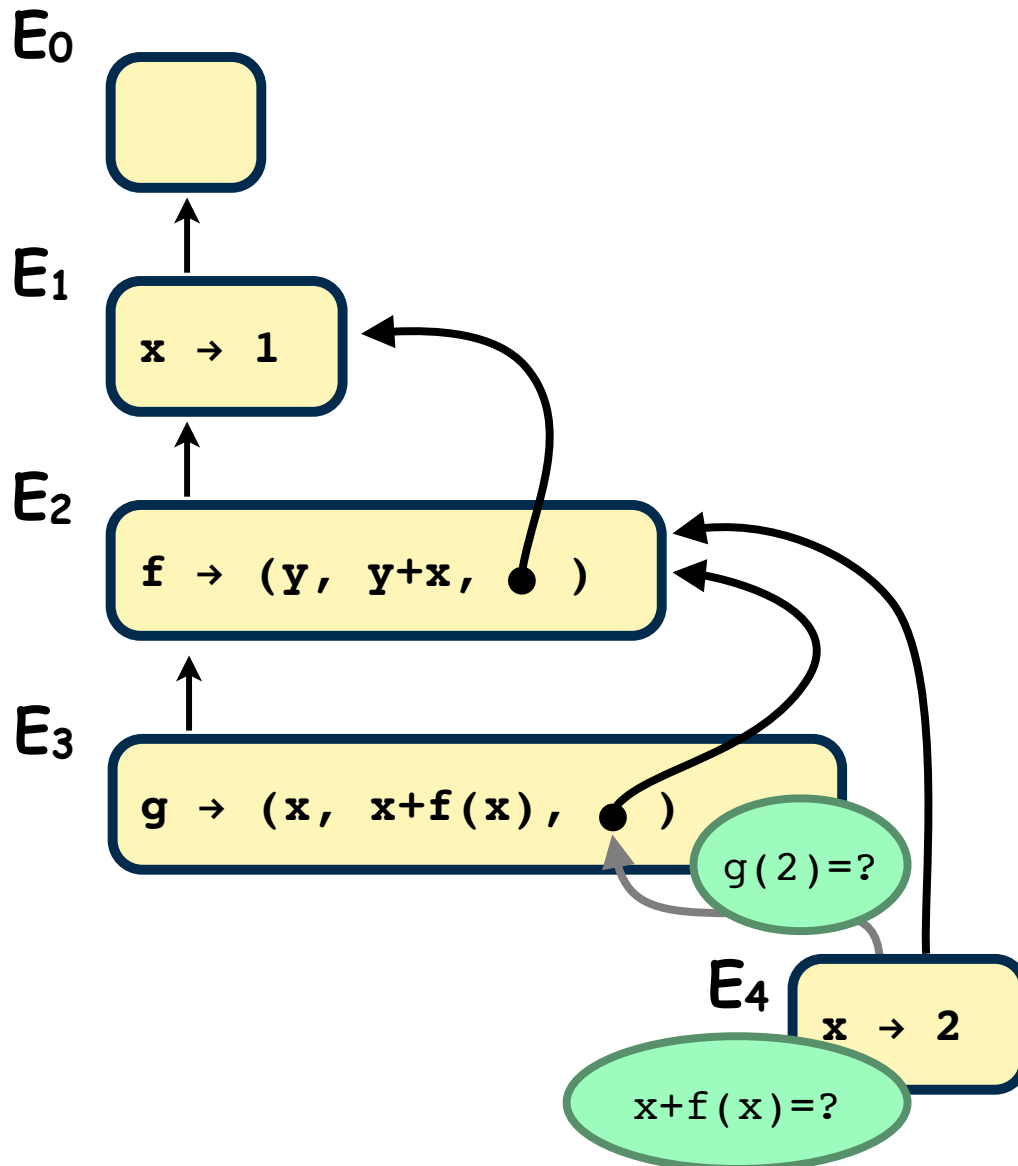
```
decl x=1 in
  decl f = (fun y -> y+x) in
    decl g = (fun x -> x+f(x))
      in g(2)
```

Exemplo



```
decl x=1 in
  decl f = (fun y -> y+x) in
    decl g = (fun x -> x+f(x))
      in g(2)
```

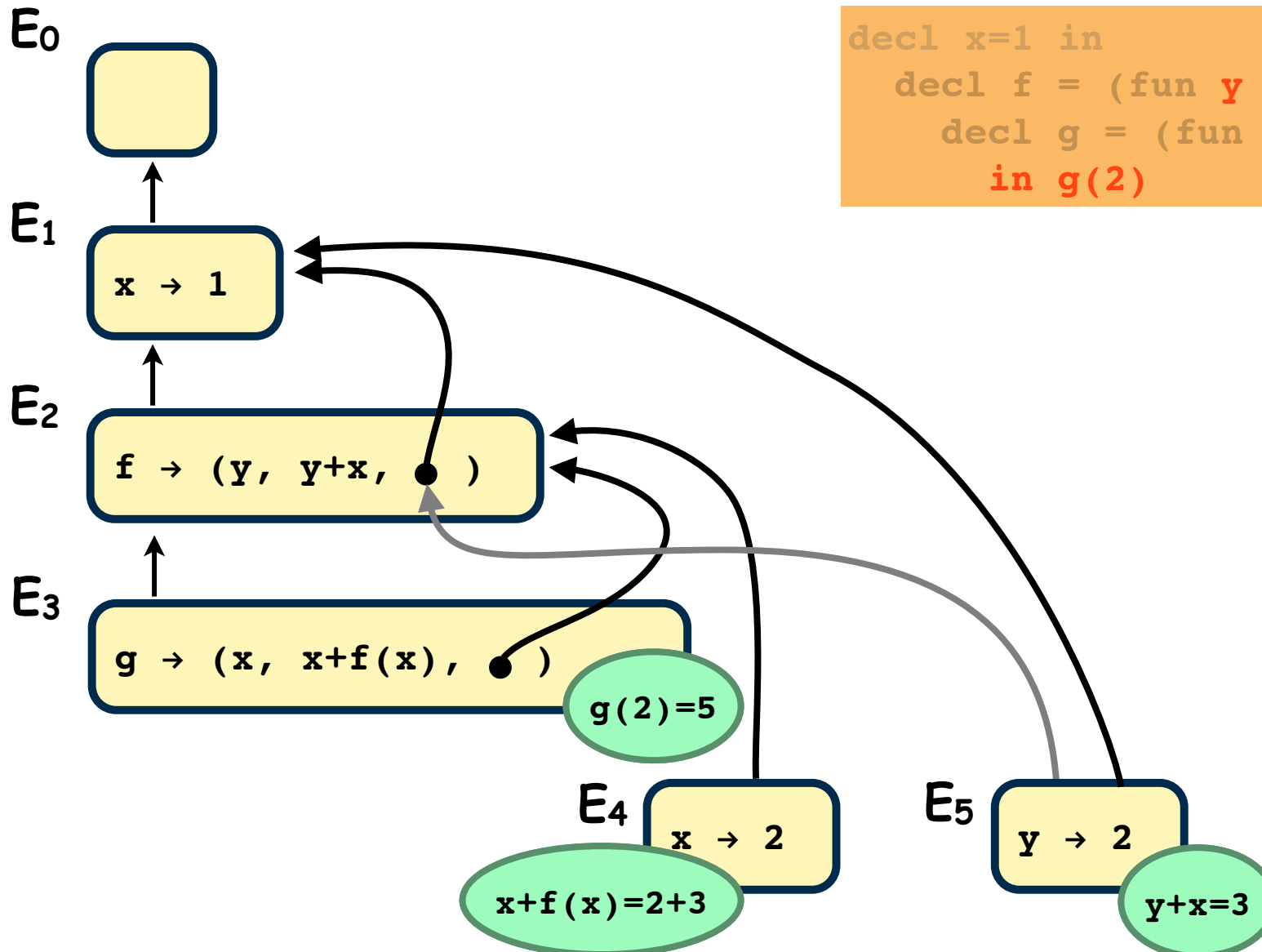
Exemplo



```
decl x=1 in
  decl f = (fun y -> y+x) in
    decl g = (fun x -> x+f(x))
      in g(2)
```

Exemplo

```
decl x=1 in
  decl f = (fun y -> y+x) in
    decl g = (fun x -> x+f(x))
      in g(2)
```

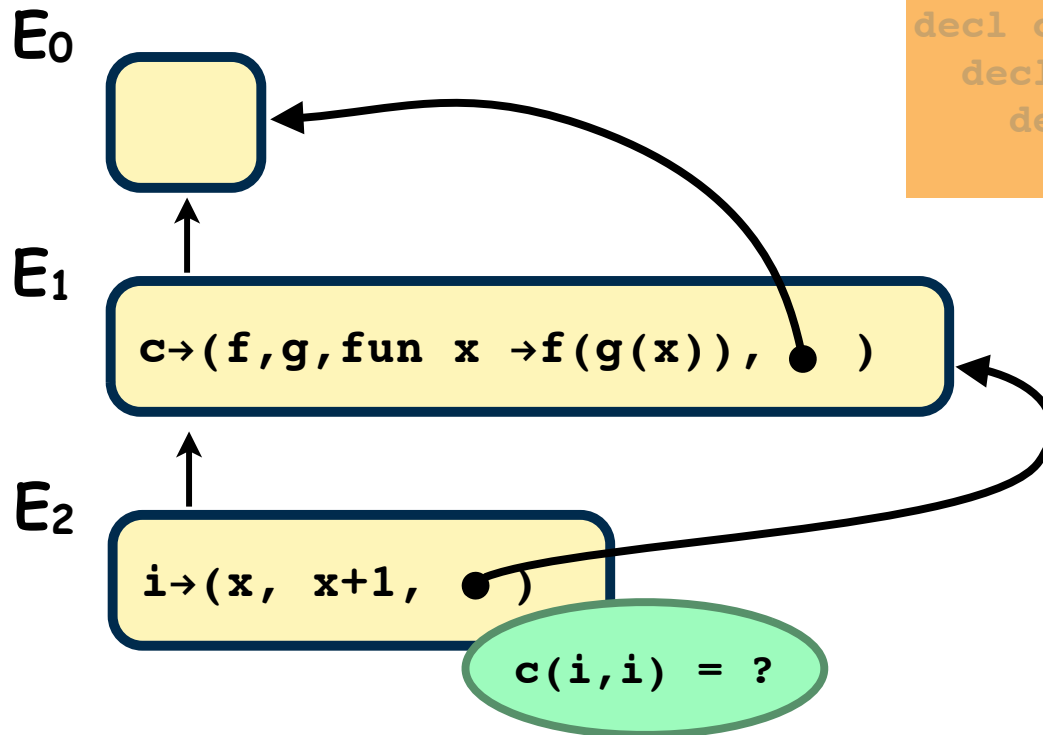


Avaliação de Funções

- Exemplo: avaliar o seguinte programa, na semântica usando ambientes e fechos.

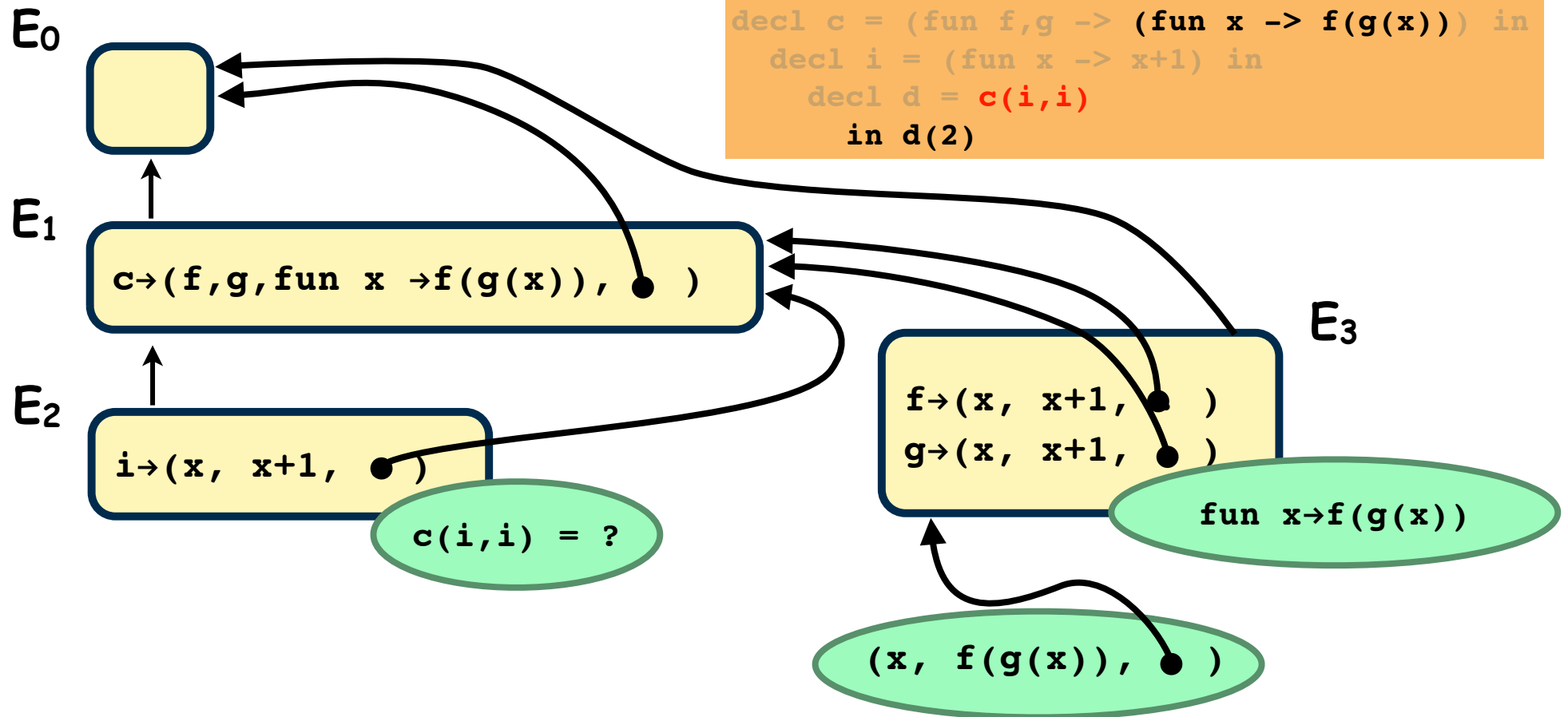
```
decl comp = (fun f,g -> (fun x -> f(g(x)))) in
  decl inc = (fun x -> x+1) in
    decl dup = comp(inc,inc)
    in dup(2)
```

Avaliação de Funções

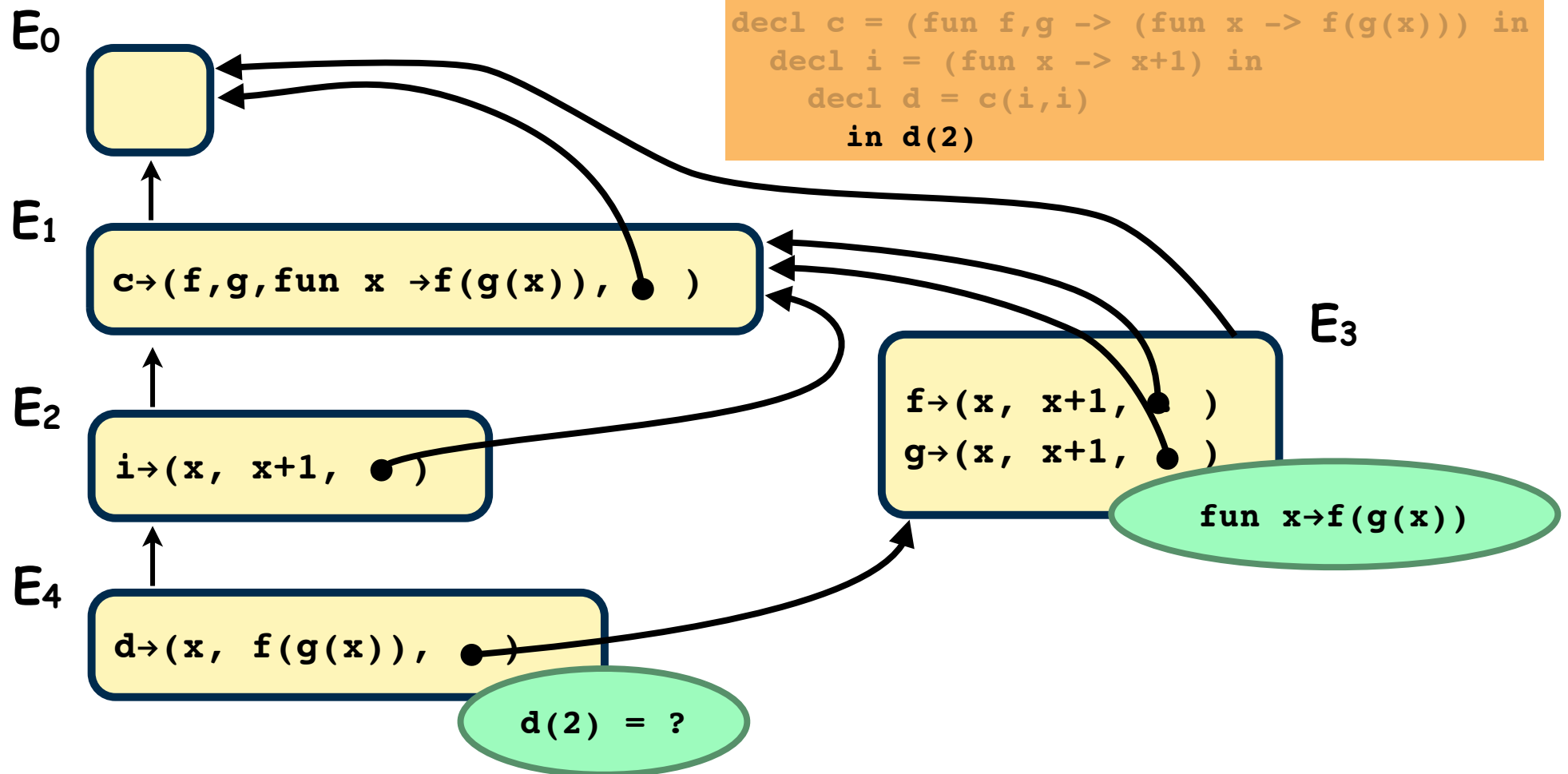


```
decl c = (fun f,g -> (fun x -> f(g(x)))) in
  decl i = (fun x -> x+1) in
    decl d = c(i,i)
      in d(2)
```

Avaliação de Funções

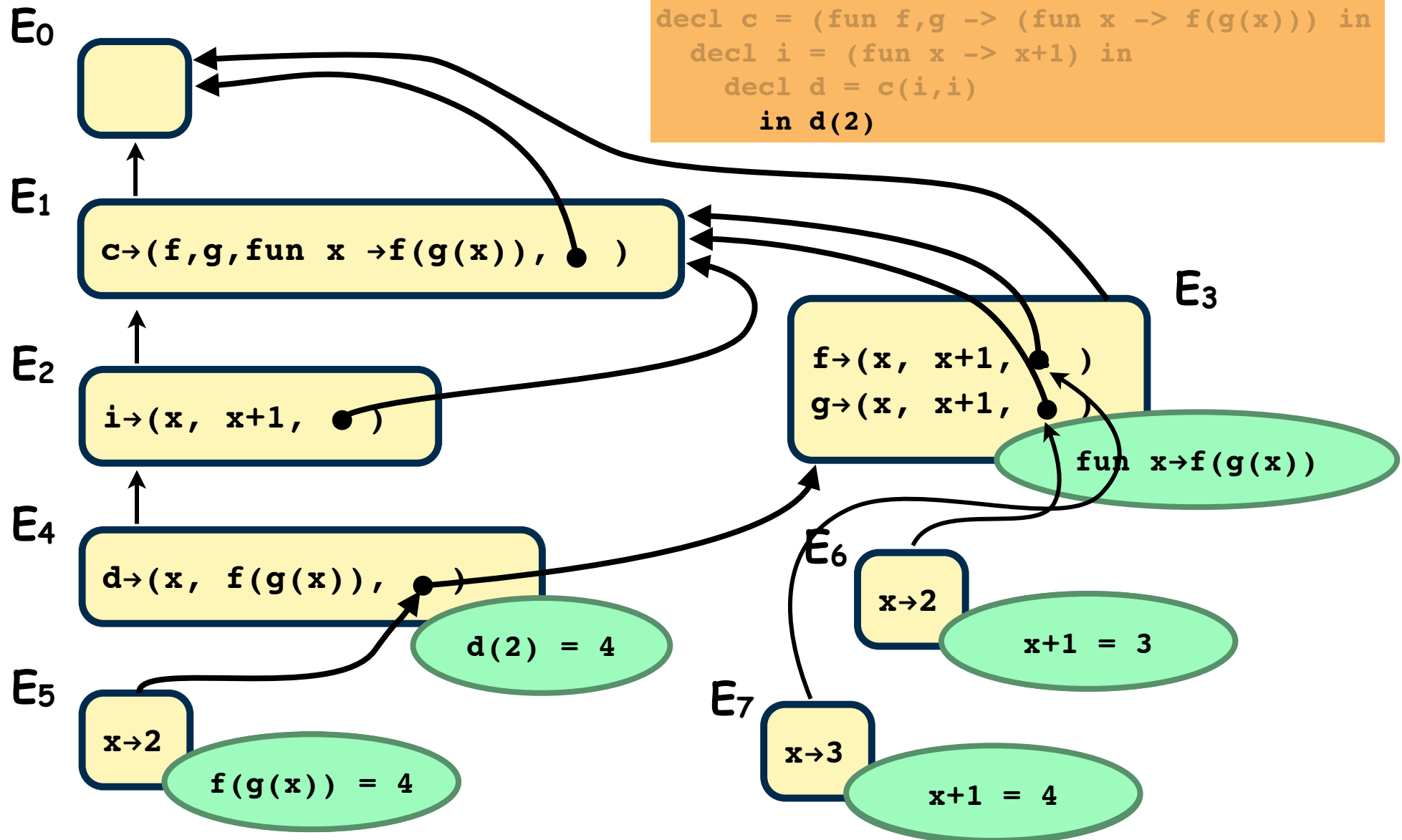


Avaliação de Funções



Avaliação de Funções

```
decl c = (fun f,g -> (fun x -> f(g(x)))) in
  decl i = (fun x -> x+1) in
    decl d = c(i,i)
    in d(2)
```



Quiz

- Qual o valor (se existir) das seguintes expressões:

```
decl f = (fun x -> x+1) in  
  decl g = (fun y -> y(2)) in g(f) end end  
  
decl f = (fun x -> x(x)) in f(f) end
```

- Qual o valor da expressão seguinte quando avaliada pela regra dinâmica e pela regra estática de resolução de nomes:

```
decl x=2 in  
  decl g = (fun y -> y-x) in  
    decl x = 4 in g(x) end end end
```

- Considera que as duas expressões seguintes têm sempre o mesmo valor? Porquê?

```
decl id = E1 in E2 end  
(fun id -> E2)( E1)
```

Quiz (solução)

- Considera que as duas expressões seguintes têm sempre o mesmo valor? Porquê?

```
decl id = E1 in E2 end  
  
(fun id -> E2)( E1 )
```

- Temos (aplicando as regras de avaliação):

```
eval(decl id = E1 in E2 end, env) =  
  eval(E2, env.Assoc(id, eval(E1, env)))
```

- Por outro lado:

```
eval((fun id -> E2), env) = closure(id, E2, env)  
  
eval((fun id -> E2) (E1), env) =  
  eval(E2, env.Assoc(id, eval(E1, env)))
```

Resumo e Leituras

- Abstração por parameterização é um mecanismo aplicado a um subprograma que o generaliza, abstraindo o valor de certas subexpressões em parâmetros bem identificados) e permite a sua instanciação e reutilização aplicada a diferentes contextos.
- Uma abstração é uma construção sintática composta por um conjunto de parâmetros e um subprograma.
- Cada instanciação de uma abstracção é activada em associação com uma interpretação dos seus parâmetros e nomes livres.
- Leituras:
 - Liskov, Guttag "Program Development in Java"
 - Friedmand "Essentials of programming languages" 3rd edition, Cap 3.
 - Mitchell, "Concepts in programming languages", Cap 7
 - Appel, Cap. 15, "Modern Compiler Implementation"