Interpretação e Compilação de Linguagens de Programação

Mestrado Integrado em Engenharia Informática
Departamento de Informática
Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa

2016-2017

João Costa Seco (joao.seco@di.fct.unl.pt)

Lecture 07
Abstração funcional

Interpretação e Compilação de Linguagens de Programação

Luís Caires, João Costa Seco

Edição 2010-2011

Regência: João Costa Seco (joao.seco@di.fct.unl.pt)

Licenciatura em Engenharia Informática

Departamento de Informática

Faculdade de Ciências e Tecnologia

Universidade Nova de Lisboa

Unidade 7: Abstracção funcional

Nas linguagens de programação existem várias formas de tornar código mais abstrato e reutilizável em diferentes contextos. A abstração tem por objetivo aumentar o nível de detalhe com que se olha para um problema e pode ser conseguida ao nível da funcionalidade, dos dados, da iteração de coleções, através da hierarquização da informação, etc.
Todas as linguagens de programação modernas possuem uma ou mais destas formas de abstração, e normalmente todas incluem suporte primitivo para a abstração funcional por parametrização.

- Abstração por parametrização
- Equivalência alfa
- A linguagem CALCF
- Semântica com substituição da linguagem CALCF
- Algoritmo interpretador da linguagem CALCF com ambiente
- Estratégias de resolução de nomes
- Algoritmo compilador da linguagem CALCF
- Declarações recursivas
- Passagem de parâmetros (por valor, por nome)

Abstracção por Parametrização

- As linguagens de programação têm um número finito de construções que podem ser usadas para representar um número infinito de computações.
- Podemos capturar grupos de computações semelhantes através da abstração de alguns das suas componentes (as que variam).
- A abstração é uma forma de enriquecer as linguagens de programação com novas operações, com um nível mais alto (manipulam entidades mais complexas como um todo) definidas a partir de outras construções da linguagem.

```
1*1 + 2*2

1*1 + 1*2

2*2 + 1*1

1*2 + 2*1

x*x + y*y

2*2 + 3*2

5*5 + 7*7

2*2 + 9*9

1*1 + 2*2

3*1 + 1*2
```

Equivalência-alfa ($=_\alpha$)

- Duas abstrações dizem-se alfa-equivalentes se uma pode ser obtida a partir da outra através da renomeação dos seus parâmetros, evitando conflitos com os seus nomes livres:

  (x → x+y) $=_\alpha$ (z → z+y)

  (x → x+y) $\neq_\alpha$ (y → y+y)

- Abstrações alfa-equivalentes são semanticamente equivalentes (são interpretadas como denotando a mesma função):

  (x → x+1) $=_\alpha$ (z → z+1)

- Por isso, um interpretador pode traduzir os nomes dos parâmetros em algo mais conveniente e conhecido apenas localmente...

O que denota uma abstracção?

- Uma abstração é uma expressão sintática que denota uma certa função.
- Uma função é uma entidade semântica primitiva que suporta uma operação de aplicação, tal como um valor inteiro é uma entidade semântica primitiva que suporta a operação de adição, etc.
- Uma linguagem pode suportar funções mas não abstrações (exemplo: C, C++, Pascal)
- Várias linguagens suportam abstrações (ML, Smalltalk, Python, Javascript, Java (usando objetos anónimos))

Exemplo: A Linguagem CALCF

- A linguagem CALCF estende a linguagem CALCI com funções, representadas por abstrações:

```
fun Id -> Expression end
```

- As funções podem ser aplicadas ao seu argumento, usando a expressão de aplicação:

```
Expression1 ( Expression2 )
```

- Semântica pretendida:
  Se ***Expression1*** denota uma função f, e ***Expression2*** denota um valor qualquer v, então ***Expression1*(Expression2)*** denota o valor que resulta de aplicar f a v.

Semântica de CALCF (1)

A função semântica I de CALCF:

$$I : CALCF \rightarrow RESULT$$

CALCF = conjunto dos programas fechados

RESULT = conjunto dos significados (denotações)

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

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

Semântica de CALCF (2)

A função semântica I de CALCF:

$$I : CALCF \times ENV \rightarrow RESULT$$

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):

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

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) adotaram a resolução dinâmica de nomes, por ser de implementação mais simples.
- Atualmente, é considerado um mecanismo indesejável e até incorreto (por não respeitar o princípio da substituição), apesar de às vezes "dar jeito" interpretar nomes não locais no contexto da chamada (por exemplo: "hostname").

Semântica de CALCF (3)

- A (nova) função semântica I de CALCF:

$$I : CALCF \times ENV \rightarrow RESULT$$

CALCF = conjunto dos programas abertos

ENV = conjunto dos ambientes

RESULT = conjunto dos significados (denotações)

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

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

Avaliação de Funções

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

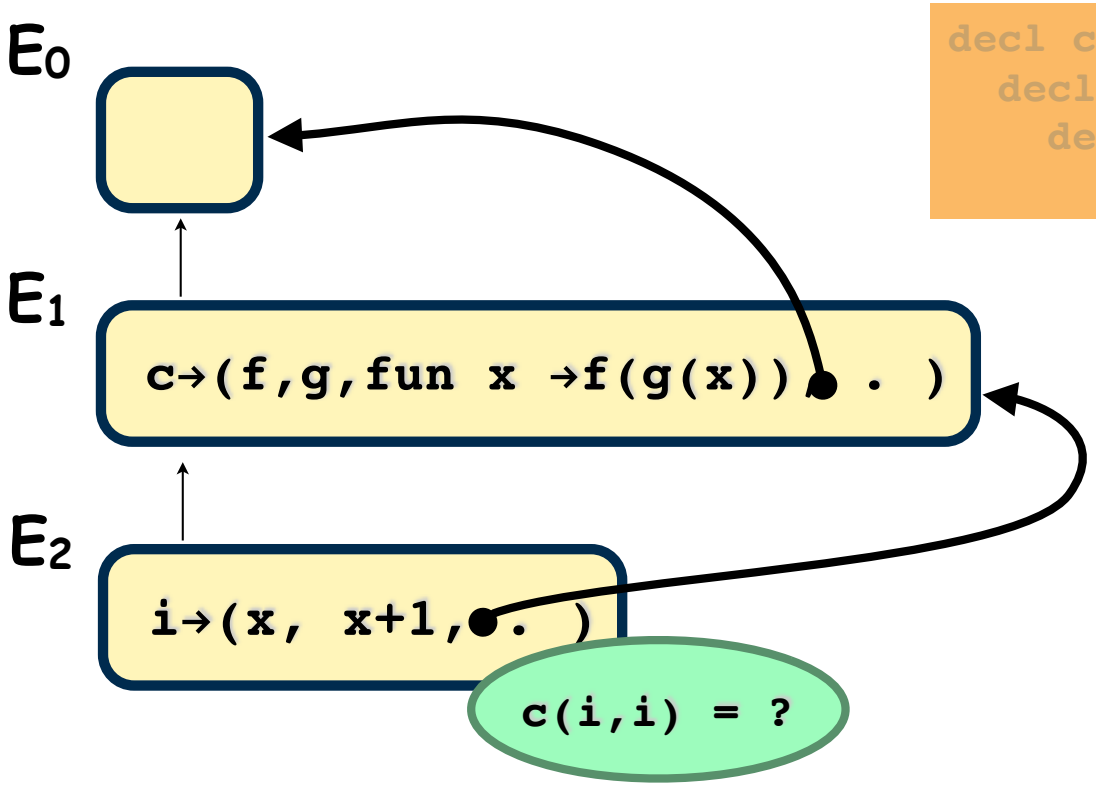
```
decl x=1 in
  decl f = (fun y -> y+x) in
    decl g = (fun x -> x+f(x))
      in g(2)
```

Avaliação de Funções

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

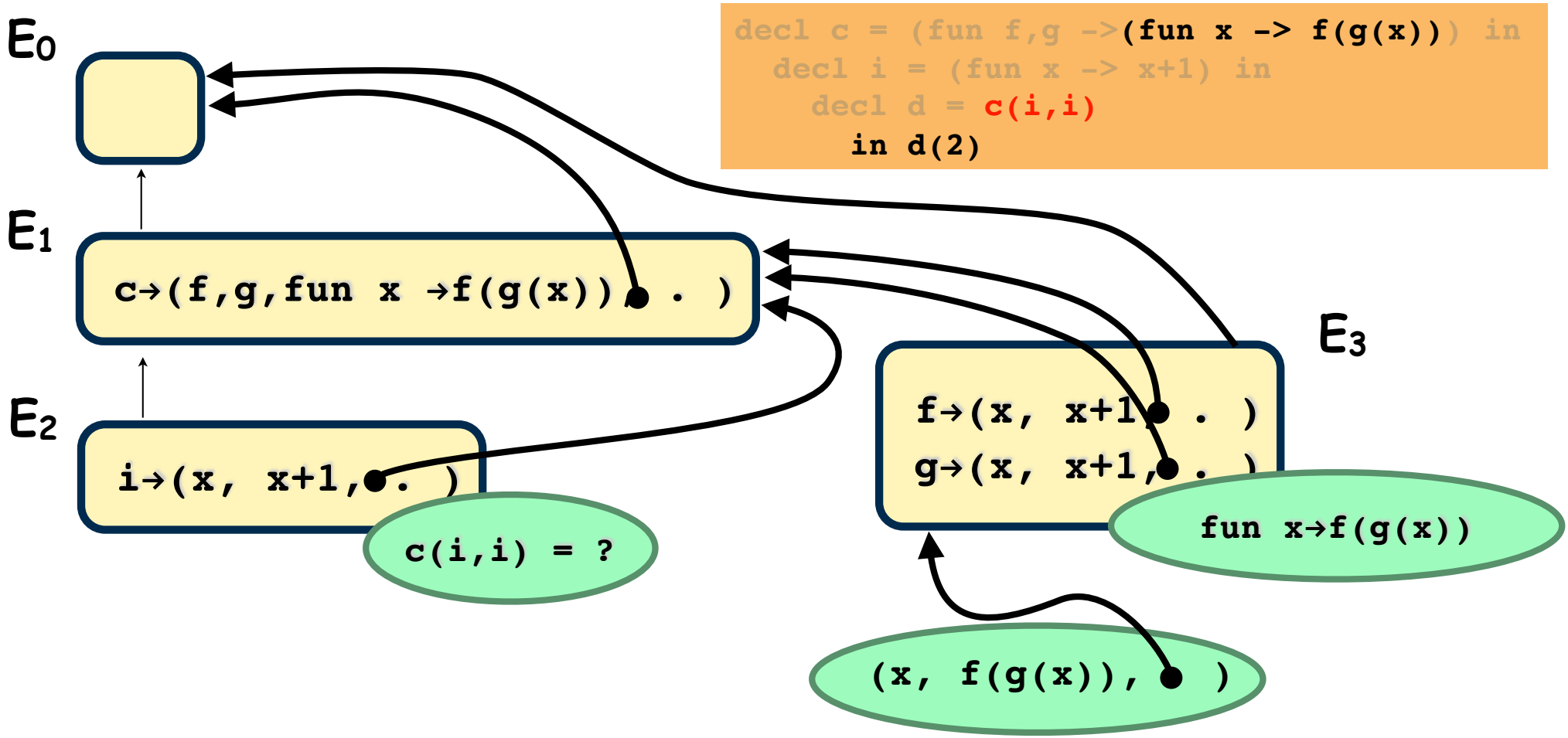
```
decl comp = (fun f,g -> (fun x -> f(g(x)))) in
  decl inc = (fun x -> x+1) in
    decl dup = comp(inc,inc)
      in dup(2)
```

Avaliação de Funções

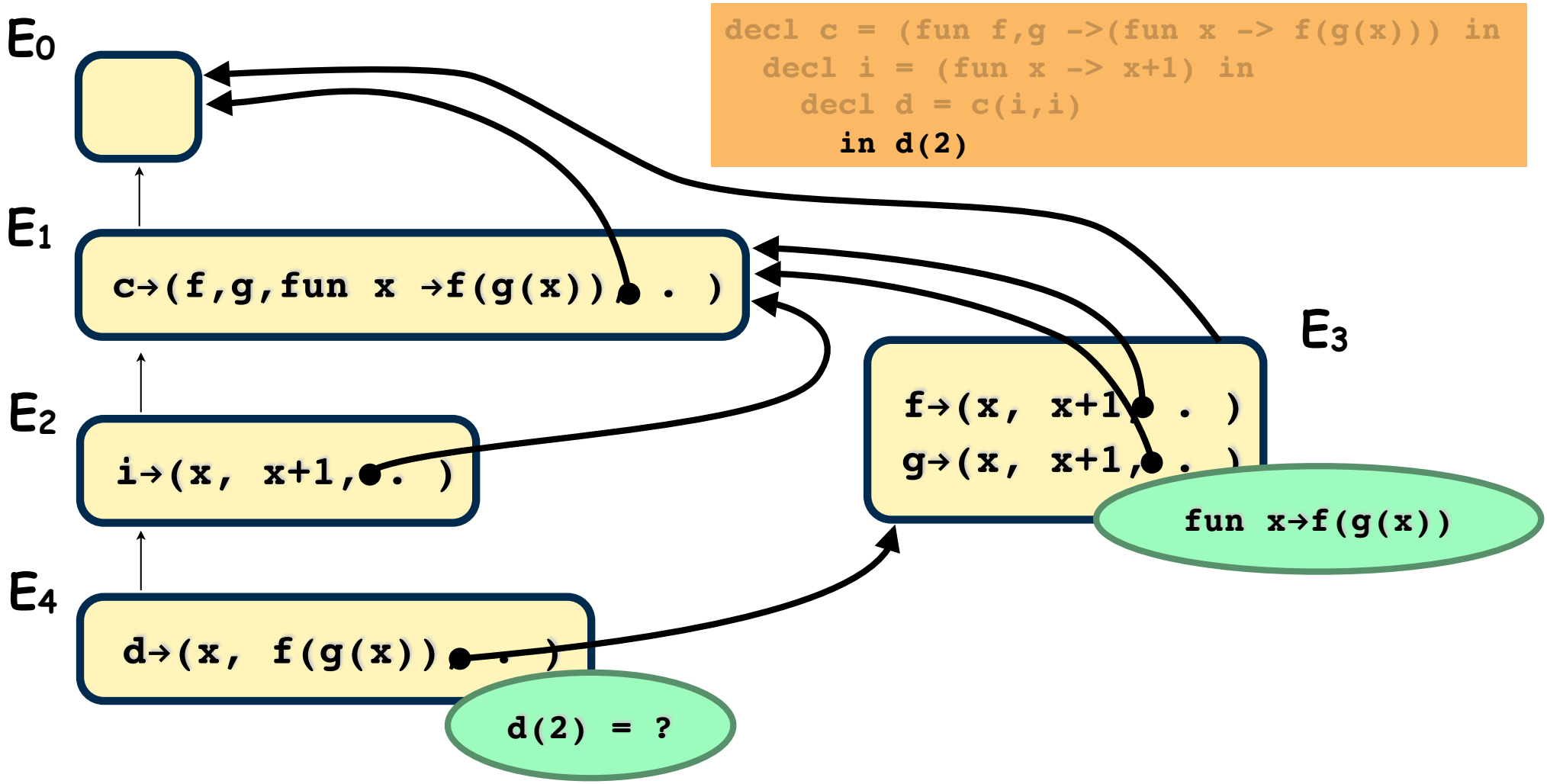



```
decl c = (fun f,g ->(fun x -> f(g(x))) in
  decl i = (fun x -> x+1) in
    decl d = c(i,i)
      in d(2)
```

Avaliação de Funções



Avaliação de Funções



Avaliação de Funções