CS450

Structure of Higher Level Languages

Lecture 23: Language λ_D: adding definitions correctly

Tiago Cogumbreiro

Today we will...

- Introduce the **mutable** semantics of λ-calculus with environments
- Study mutation as a side-effect

Introducing the λ_D

Language λ_D: Terms

We highlight in red an operation that produces a side effect: ***mutating an environment***.

$$\frac{e \Downarrow_E v \qquad \color{red}{E \leftarrow [x := v]}}{(\texttt{define}\ x\ e) \Downarrow_E \texttt{void}}\ (\texttt{E-def})$$

$$\frac{t_1 \Downarrow_E v_1 \qquad t_2 \Downarrow_E v_2}{t_1 ; t_2 \Downarrow_E v_2}\ (\texttt{E-seq})$$

Language λ_D: Expressions

> Because we have side-effects, the order in which we evaluate each sub-expression is important.

$$v \Downarrow_E v \qquad (\texttt{E-val})$$

$$x \Downarrow_E E(x) \qquad (\texttt{E-var})$$

$$\lambda x.t \Downarrow_E (E, \lambda x.t) \qquad (\texttt{E-lam})$$

$$\frac{e_f \Downarrow_E (E_f, \lambda x.t_b) \qquad e_a \Downarrow_E v_a \qquad E_b \leftarrow E_f + [x := v_a] \qquad t_b \Downarrow_{E_b} v_b}{(e_f\ e_a) \Downarrow_E v_b}\ (\texttt{E-app})$$

> Can you explain why the order is important?

Language λ_D: Expressions

> Because we have side-effects, the order in which we evaluate each sub-expression is important.

$$v \Downarrow_E v \qquad (\texttt{E-val})$$

$$x \Downarrow_E E(x) \qquad (\texttt{E-var})$$

$$\lambda x.t \Downarrow_E (E, \lambda x.t) \qquad (\texttt{E-lam})$$

$$\frac{e_f \Downarrow_E (E_f, \lambda x.t_b) \qquad e_a \Downarrow_E v_a \qquad E_b \leftarrow E_f + [x := v_a] \qquad t_b \Downarrow_{E_b} v_b}{(e_f\ e_a) \Downarrow_E v_b} \; (\texttt{E-app})$$

> Can you explain why the order is important? Otherwise, we might evaluate the body of the function e_b without observing the assignment $x := v_a$ in E_b.

Mutable operations on environments

Mutable operations on environments

Put

$$E \leftarrow [x := v]$$

Take a reference to an environment E and mutate its contents, by adding a new binding.

Push

$$E \leftarrow E' + [x := v]$$

Create a new environment referenced by E which copies the elements of E' and also adds a new binding.

Making side-effects explicit

Mutation as a side-effect

- Let us use a triangle ▶ to represent the order of side-effects.

$$\frac{e \Downarrow_E v \quad \blacktriangleright \quad E \leftarrow [x := v]}{(\texttt{define}\ x\ e) \Downarrow_E \texttt{void}}\ (\texttt{E-def})$$

$$\frac{t_1 \Downarrow_E v_1 \quad \blacktriangleright \quad t_2 \Downarrow_E v_2}{t_1 ; t_2 \Downarrow_E v_2}\ (\texttt{E-seq})$$

$$\frac{e_f \Downarrow_E (E_f, \lambda x.t_b) \ \blacktriangleright\ e_a \Downarrow_E v_a \ \blacktriangleright\ E_b \leftarrow E_f + [x := v_a] \ \blacktriangleright\ t_b \Downarrow_{E_b} v_b}{(e_f\ e_a) \Downarrow_E v_b}\ (\texttt{E-app})$$

Implementing side-effect mutation

Making the heap explicit

> We can annotate each triangle with a heap, to make explicit which how the global heap should be passed from one operation to the next. In this example, defining a variable takes an input global heap H and produces an output global heap H_2.

$$\frac{\blacktriangleright_H \quad e \Downarrow_E v \quad \blacktriangleright_{H_1} \quad E \leftarrow [x := v] \quad \blacktriangleright_{H_2}}{\blacktriangleright_H \ (\texttt{define}\ x\ e) \Downarrow_E \texttt{void} \blacktriangleright_{H_2}} \ \texttt{(E-def)}$$

Let us use our rule sheet!

$$\frac{e \Downarrow_E v \quad \blacktriangleright \quad E \leftarrow [x := v]}{(\texttt{define}\ x\ e) \Downarrow_E \texttt{void}}\ (\texttt{E-def})$$

$$\frac{t_1 \Downarrow_E v_1 \quad \blacktriangleright \quad t_2 \Downarrow_E v_2}{t_1 ; t_2 \Downarrow_E v_2}\ (\texttt{E-seq})$$

$$\frac{e_f \Downarrow_E (E_f, \lambda x.t_b) \quad \blacktriangleright \quad e_a \Downarrow_E v_a \quad \blacktriangleright \quad E_b \leftarrow E_f + [x := v_a] \quad \blacktriangleright \quad t_b \Downarrow_{E_b} v_b}{(e_f\ e_a) \Downarrow_E v_b}\ (\texttt{E-app})$$

$$v \Downarrow_E v \qquad (\texttt{E-val})$$

$$x \Downarrow_E E(x) \qquad (\texttt{E-var})$$

$$\lambda x.t \Downarrow_E (E, \lambda x.t) \qquad (\texttt{E-lam})$$

Examples

Evaluating Example 2

```
(define b (lambda (x) a))
(define a 20)
(b 1)
```

Input

```
E0: []
---
Env: E0
Term: (define b (lambda (y) a))
```

Evaluating Example 2

```
(define b (lambda (x) a))
(define a 20)
(b 1)
```

Input

```
E0: []
---
Env: E0
Term: (define b (lambda (y) a))
```

Output

```
E0: [
  (b . (closure E0 (lambda (y) a)))
]
Value: #<void>
```

$$\frac{\dfrac{}{\lambda y.a \Downarrow_{E_0} (E_0, \lambda y.a)} \qquad \blacktriangleright \qquad \dfrac{}{E_0 \leftarrow [b := (E_0, \lambda y.a)]}}{(\texttt{define}\ b\ \lambda y.a) \Downarrow_{E_0} \texttt{void}}$$

Example 2: step 2

Input

```
E0: [
  (b . (closure E0 (lambda (y) a)))
]
---
Env: E0
Term: (define a 20)
```

Example 2: step 2

Input

```
E0: [
  (b . (closure E0 (lambda (y) a)))
]
---
Env: E0
Term: (define a 20)
```

Output

```
E0: [
  (a . 20)
  (b . (closure E0 (lambda (y) a)))
]
Value: #<void>
```

$$\dfrac{\dfrac{}{20 \Downarrow_{E_0} 20} \quad \blacktriangleright \quad \dfrac{}{E_0 \leftarrow [a := 20]}}{(\mathbf{define}\ a\ 20) \Downarrow_{E_0} \mathtt{void}}$$

Example 2: step 3

Input

```
E0: [
  (a . 20)
  (b . (closure E0 (lambda (y) a)))
]
---
Env: E0
Term: (b 1)
```

Example 2: step 3

Input

```
E0: [
  (a . 20)
  (b . (closure E0 (lambda (y) a)))
]
---
Env: E0
Term: (b 1)
```

Output

```
E0: [
  (a . 20)
  (b . (closure E0 (lambda (y) a)))
]
E1: [ E0
  (y . 1)
]
Value: 20
```

$$\frac{b \Downarrow_{E_0} (E_0, \lambda y.a) \;\blacktriangleright\; 1 \Downarrow_{E_0} 1 \;\blacktriangleright\; E_1 \leftarrow E_0 + [y := 1] \;\blacktriangleright\; a \Downarrow_{E_1} 20}{(b\;1) \Downarrow_{E_0} 20}$$

Example 3

```
(define (f x) (lambda (y) x))
(f 10)
```

Input

```
E0: []
---
Env: E0
Term: (define (f x) (lambda (y) x))
```

Example 3

```
(define (f x) (lambda (y) x))
(f 10)
```

Input

```
E0: []
---
Env: E0
Term: (define (f x) (lambda (y) x))
```

Output

```
E0: [
  (f . (closure E0
          (lambda (x) (lambda (y) x))))
]
Value: void
```

Example 3

```
(define (f x) (lambda (y) x))
(f 10)
```

Input

```
E0: []
---
Env: E0
Term: (define (f x) (lambda (y) x))
```

Output

```
E0: [
  (f . (closure E0
          (lambda (x) (lambda (y) x))))
]
Value: void
```

$$\frac{\lambda x.\lambda y.x \Downarrow_{E_0} (E_0, \lambda x.\lambda y.x)}{(\mathtt{define}\ f\ \lambda x.\lambda y.x) \Downarrow_{E_0} \mathtt{void}}$$

Example 3

```
(define (f x) (lambda (y) x))
(f 10)
```

Input

```
E0: []
---
Env: E0
Term: (define (f x) (lambda (y) x))
```

Output

```
E0: [
  (f . (closure E0
          (lambda (x) (lambda (y) x))))
]
Value: void
```

$$\frac{\lambda x.\lambda y.x \Downarrow_{E_0} (E_0, \lambda x.\lambda y.x) \qquad \blacktriangleright \qquad E_0 \leftarrow [f := (E_0, \lambda x.\lambda y.x)]}{(\texttt{define}\ f\ \lambda x.\lambda y.x) \Downarrow_{E_0} \texttt{void}}$$

Example 3

Input

```
E0: [
  (f . (closure E0
          (lambda (x) (lambda (y) x))))
]
---
Env: E0
Term: (f 10)
```

Example 3

Input

```
E0: [
  (f . (closure E0
          (lambda (x) (lambda (y) x))))
]
---
Env: E0
Term: (f 10)
```

Output

```
E0: [
  (f . (closure E0
          (lambda (x) (lambda (y) x))))
]
E1: [ E0 (x . 10) ]
Value: (closure E1 (lambda (y) x))
```

Example 3

Input

```
E0: [
  (f . (closure E0
          (lambda (x) (lambda (y) x))))
]
---
Env: E0
Term: (f 10)
```

Output

```
E0: [
  (f . (closure E0
          (lambda (x) (lambda (y) x))))
]
E1: [ E0 (x . 10) ]
Value: (closure E1 (lambda (y) x))
```

$$\dfrac{\dfrac{E_0(f) = (E_0, \lambda x.\lambda y.x)}{f \Downarrow_{E_0} (E_0, \lambda x.\lambda y.x)} \quad \dfrac{}{10 \Downarrow_{E_0} 10} \quad \dfrac{}{E_1 \leftarrow E_0 + [x := 10]} \quad \dfrac{}{\lambda y.x \Downarrow_{E_1} (E_1, \lambda y.x)}}{(f\ 10) \Downarrow_{E_0} (E_1, \lambda y.x)}$$