CS450

Structure of Higher Level Languages

Lecture 19: Language λ_E: fast function calls

Tiago Cogumbreiro

Today we will...

1. Motivate the need for environments
2. Introduce the λ_E language formally
3. Discuss the implementation details of the λ_E-Racket
4. Discuss test-cases

In this unit we learn about...

- Implementing a formal specification
- Growing a programming language interpreter

The λ-calculus is slow

Recall the λ-calculus

Syntax

$$e ::= v \mid x \mid (e_1\ e_2) \qquad v ::= n \mid \lambda x.e$$

Semantics

$$v \Downarrow v\ \texttt{(E-val)}$$

$$\frac{e_f \Downarrow \lambda x.e_b \qquad e_a \Downarrow v_a \qquad e_b \overbrace{[x \mapsto v_a]}^{\text{Complexity?}} \Downarrow v_b}{(e_f\ e_a) \Downarrow v_b}\ \texttt{(E-app)}$$

A complexity analysis on function-call

Let us focus consider our implementation of Micro-Racket, and draw our attention to function substitution.

Given a function call $(e_f\ e_a)$

1. We evaluate e_f down to a function $(\lambda(x)\ e_b)$
2. We evaluate e_a down to a value v_a
3. We evaluate $e_b[x \mapsto v_a]$ down to a value v_b

> What is the complexity of the substitution operation $[x \mapsto v_a]$?

A complexity analysis on function-call

Let us focus consider our implementation of Micro-Racket, and draw our attention to function substitution.

Given a function call $(e_f\ e_a)$

1. We evaluate e_f down to a function $(\lambda(x)\ e_b)$
2. We evaluate e_a down to a value v_a
3. We evaluate $e_b[x \mapsto v_a]$ down to a value v_b

> What is the complexity of the substitution operation $[x \mapsto v_a]$?

The run-time grows **linearly** on the size of the expression, as we must replace x by v_a in every sub-expression of e_b.

Can we do better?

Can we do better?

Yes, we can sacrifice some **space**

to improve the run-time **speed**.

Decreasing the run time of substitution

Idea 1: Use a lookup-table to bookkeep the variable bindings

Idea 2: Introduce closures/environments

λ_E-calculus: λ-calculus with environments

We introduce the evaluation of expressions down to values, parameterized by environments:

$$e \Downarrow_E v$$

The evaluation takes two arguments: an expression e, and an environment E. The evaluation returns a value v.

Attention!

Homework Assignment 4:

- Evaluation $e \Downarrow_E v$ is implemented as function `(e:eval env exp)` that returns a value `e:value`, an environment `env` is a `hash`, and expression `exp` is an `e:expression`.
- functions and structs prefixed with `s:` correspond to the λ_S language (Section 1).
- functions and structs prefixed with `e:` correspond to the λ_E language (Section 2)

λ_E-calculus: λ-calculus with environments

Syntax

$$e ::= v \mid x \mid (e_1\ e_2) \mid \lambda x.e \qquad v ::= n \mid (E, \lambda x.e)$$

Semantics

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

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

$$\lambda x.e \Downarrow_E (E, \lambda x.e) \qquad \texttt{(E-clos)}$$

$$\frac{e_f \Downarrow_E (E_b, \lambda x.e_b) \qquad e_a \Downarrow_E v_a \qquad e_b \Downarrow_{\mathbf{E_b[x \mapsto v_a]}} v_b}{(e_f\ e_a) \Downarrow_E v_b} \qquad \texttt{(E-app)}$$

Overview of λ_E-calculus

Notable differences

1. Declaring a function is an ***expression*** that yields a function value (a closure), which packs the environment at creation-time with the original function declaration.
2. Calling a function unpacks the environment E_b from the closure and extends environment E_b with a binding of parameter x and the value v_a being passed

Environments

> An environment E maps variable bindings to values.

Constructors

- Notation $\emptyset$ represents the empty environment (with zero variable bindings)
- Notation $E[x \mapsto v]$ extends an environment with an new binding (overwriting any previous binding of variable x).

Accessors

- Notation $E(x) = v$ looks up value v of variable x in environment E

Church's encoding

Chuch's encoding

- Alonzo Church created the λ-calculus
- Church's Encoding is a treasure trove of λ-calculus expressions: it shows how natural numbers can be encoded
- Let us go through Church's encoding of booleans
- Examples taken from Colin Kemp's PhD thesis (page 17)

Encoding Booleans with λ-terms

- Why? Because you will be needing test-cases.

```
(require rackunit)
(define ns (make-base-namespace))
(define (run-bool b) (((eval b ns) #t) #f))


(define TRUE '(lambda (a) (lambda (b) a)))
(define FALSE '(lambda (a) (lambda (b) b)))
(define (OR a b) (list (list a TRUE) b))
(define (AND a b) (list (list a b) FALSE))
(define (NOT a) (list (list a FALSE) TRUE))
(define (EQ a b) (list (list a b) (NOT b)))

(check-equal?
    (run-bool (EQ TRUE (OR (AND FALSE TRUE) TRUE)))
    (equal? #t (or (and #f #t) #t)))
```