CSCI 334: Principles of Programming Languages

Lecture 7: Evaluation by Rewriting

Instructor: Dan Barowy

Williams

Topics

Lambda calculus—how to evaluate it

Your to-dos

1. Lab 4, **due Sunday 9/9** (partner lab)
2. Reading quiz, **due Wednesday 9/5**.

Lambda calculus: relevance

Fundamental technique for building programming languages that work **correctly** (and **intuitively**!).

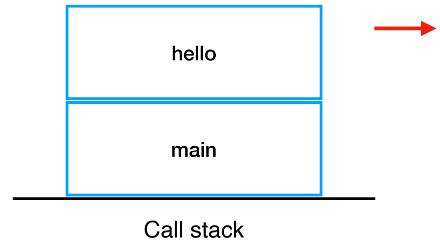
But it can also be leveraged to do some **seemingly magical** things, like **type inference**:

```
Vector<Association<String,FrequencyList>> table =
   new Vector<Association<String,FrequencyList>>();

Vector<Association<String,FrequencyList>> table = new Vector<>();

let table = new Vector<>()

…
```

Class Lambda Grammar

```
<expr>    ::= <value>
           |  <abs>
           |  <app>
           |  <parens>
<var>     ::= α ∈ { a ... z }
<abs>     ::= λ<var>.<expr>
<app>     ::= <expr><expr>
<parens>  ::= (<expr>)
<value>   ::= v ∈ ℕ
           |  <var>
```

Evaluation: You know how Java does it




```
class Program {
  public static void hello() {
    println("Hello world!");
  }

  public static void main(…) {
    hello();
  }
}
```

Evaluation: Lambda calculus is like algebra

(λx.x)x

Evaluation consists of simplifying an expression using text substitution.

Only two simplification rules:

α-reduction

β-reduction

α-Reduction

(λx.x)x

This expression has two **different** x variables

Which should we rename?

Rule:

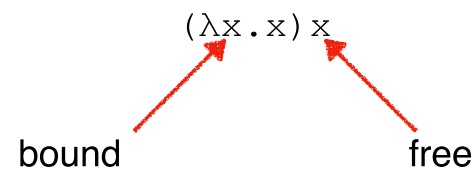
⟦λx.<expr>⟧ $=_{\alpha}$ ⟦λy.[y/x]<expr>⟧

[y/x]<expr> means “substitute y for x in <expr>”

α-Reduction

(λx.x) x	given
(λy.[y/x]x) x	α-reduce y for x (binding)
(λy.y) x	α-reduce y with x (expr)

Free vs bound variables



Watch out!

λx.xy	given
λy.[y/x]xy	α-reduce y for x
λy.yy	inner α-reduction
	this is incorrect!

The lambda has "captured" the free y.
Substitution must be **capture-avoiding**.

β-Reduction

(λx.x) y

How we "call" or **apply** a function to an argument

Rule:

⟦(λx.<expr>) y⟧ $=_\beta$ ⟦[y/x]<expr>⟧

Let's reduce this

(λx.x)x

Watch out!

(λx.λx.x)x	given
([x/x]λx.x)	β-reduce x for x
(λx.x)	β-reduce inner expr
	done

The inner lambda term **redefines** x and therefore "blocks" substitution of x.

How far do we go?

We keep going until there is **nothing left to simplify**.

x	← done
xx	← done
λx.y	← done
(λx.xy)z	← not done

That "most simplified" expression is called a **normal form**.

An expression that can be simplified is a called a **redex**.

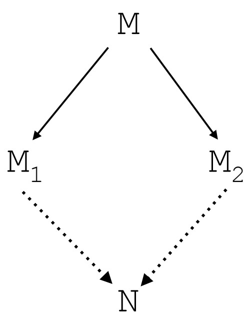
Try this one with a partner

(λx.λy.yx)xy

(don't forget precedence/associativity rules)

Sometimes multiple simplifications

Order (mostly) does not matter



If $M \to M_1$ and $M \to M_2$
then $M_1 \to^* N$ and $M_2 \to^* N$
for some N

"confluence"

Activity

Normal order reduction:

```
(λf.λx.f(f x))(λz.(+ x z))2
```

Recap & Next Class

Today:

Lambda calculus: how to evaluate

Next class:

Computability