CSCI 334:
Principles of Programming Languages

Lecture 7: Evaluation by Rewriting

Instructor: Dan Barowy
Williams

Topics

Lambda calculus—how to parse it

Lambda calculus—how to evaluate it

Your to-dos

1. Read *Introduction to the Lambda Calculus, Part 1*, **for Lab 3**.
2. Lab 3, **due Monday 2/26** (solo lab)
   Give yourself enough time to learn a small amount of LaTeX

Announcements

- CS Colloquium this **Friday, Feb 23 @ 2:35pm in Wege Auditorium (TCL 123)**

How Big is YouTube?
Prof. Ethan Zuckerman ('93, UMass Amherst)

Social media and user-generated content have thoroughly transformed the media landscape, giving birth to powerful companies, transforming news and political participation. Despite the influence of platforms run by Google and Meta, it is difficult to answer the simplest questions about these technologies, like "How many videos are hosted on YouTube?" Our lab has published a novel method for estimating the size of YouTube and learning other essential facts about the platform. In the process, we have uncovered a set of legal and ethical questions that will be essential for other "unpermissioned research" about social media platforms.

4

No Quiz

So this is just an ungraded activity

5

See the solution posted online.

λ Syntax Refresher

6

Memorize This†

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

†I rarely ask students to memorize things.

7

You should memorize this.

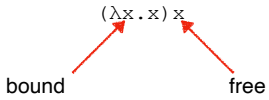
Let's parse this together

```
xλy.a(xx)
```

8

See your classmates for the derivation tree.

Free vs bound variables



9

One very important aspect of the lambda calculus is whether a variable is "free" or "bound." This expression has two different x variables in it. Be on the lookout for this distinction. The bound variable is the x that appears within the lambda. The free x is the one outside the lambda. They just happen to have the same name.

Evaluating λ expressions

10

Lambda calculus: relevance

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

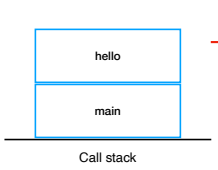
But it can also be leveraged to do some **seemingly magical** things, like **type inference**:

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

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

let table = new Vector<>()

…
```

11

Why are we learning this? At its heart, the study of programming languages is about how a language "desugars" into a core mathematical idea. You do not *need* the lambda calculus to build a programming language. However, unless you understand the relationship between your language and the lambda calculus, certain kinds of insights about programs will be difficult or impossible to obtain.

Evaluation: You know how Java does it




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

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

12

Now, let's talk about how a program is evaluated. You might have some sense of how some languages are evaluated, like Java. C works essentially the same way as Java in this regard.

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

13

However, the lambda calculus is different. It is more like algebra. You evaluate by rewriting an expression with some kind of text substitution.

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>`"

14

There are only two "evaluation rules" in the lambda calculus. We call these rules "reductions." The first is alpha reduction, which is used to rename a variable in an expression.

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)

15

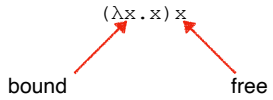
For example, we can alpha reduce the expression (λx.x)x to (λy.y)x. This is OK because we're just renaming a bound variable. Your intuition may already tell you that this is OK! For example, you probably already know that the following two Java programs are the same.

```
public static int id(int x) {
  return x;
}
```

```
public static int id(int y) {
  return y;
}
```

Note that in this class, you must write your reductions in two-column format, just like you did in your high school geometry class.

Free vs bound variables



16

Note that there is a very important distinction between free and bound variables. The inner (leftmost) x is defined by the abstraction. The outer (rightmost) x is a TOTALLY DIFFERENT VARIABLE that happens to have the same name. We do not know how it is defined in this expression, so we must treat it with caution. We cannot rename a free variable, but we can rename a bound one.

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**.

17

Be careful not to “capture” a variable when performing an alpha reduction.

β-Reduction

`(λx.x)y`

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

Rule:

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

18

The second reduction rule is beta reduction, which has essentially the same meaning as a "function call." It passes an argument into a function definition, discards the lambda, and then rewrites the body of the function definition.

Let's reduce this

`(λx.x)x`

19

For example, let's reduce this expression. See your classmates for the step-by-step reduction. The result is ultimately x.

Recap & Next Class

Today:

- Lambda calculus: how to parse
- Lambda calculus: how to evaluate

Next class:

- Lambda calculus: how to survive

20