HW 1 — Lambda Calculus

CS 421
Revision 1.0

Assigned February 3, 2016
Due February 10, 2016

1 Objectives and Background

The objective for this homework is to give you practice performing lambda calculus reductions.

1.1 An Overview of Lambda Calculus

The lambda calculus is a method by which one can capture the core essence of functions, function applications, and evaluation. There are two types of lambda calculus: typed and untyped. We will be working with the untyped lambda calculus in this class.

There are three types of expressions in the untyped lambda calculus:

- Variables (like x, y, a, etc.),
- Abstractions (like $(\lambda x.\ e)$), and
- Application (like $(e_1\ e_2)$, $(f\ x)$, etc.)

1.2 Lambda Calculus Terminology

There are also a few bits of terminology that you should know when working with the lambda calculus:

- **Lambda Abstraction**: Refers to the entire anonymous function (e.g., $(\lambda x.\ e)$). Note that lambda abstractions extend as far right as possible, the end of which is defined by either its closing parenthesis or the end of the expression. For example, in $(\lambda x.xy(yz))$, the body of the lambda abstraction is $xy(yz)$.
- **Variable Binding**: In the expression $(\lambda x.\ e)$, x is a variable binding within e. That is, all occurrences of x in the expression e are bound by the x following the λ.
- **Bound Occurrence**: Refers to the occurrence of x in expression e within $(\lambda x.\ e)$.
- **Unbound Occurrence**: Refers to the occurrence of a variable that is not bound.
- **Scope of Binding**: In the expression $(\lambda x.\ e)$, occurrences of the variable x in e are said to be bound by the expression $(\lambda x.\ e)$, assuming those occurrences are not in a subterm of e in the form of $(\lambda x.\ e')$.
- **Free Variables**: All variables in an expression with unbound occurrences.

1.3 Lambda Calculus Terminology: An Example

For example, in the expression $(\lambda x.xy)\ x$, which we'll annotate as $(\lambda x_0.\ x_1\ y_0)\ x_2$:

- Lambda Abstraction: $(\lambda x.xy)$
- Variable Bindings:
  - x_0 (which binds all occurrences of x in the expression xy, namely x_1)
- Bound Occurrences:
  - The occurrence of x (namely, x_1) in the lambda abstraction, which is bound by x_0.
- Unbound Occurrences:
  - x_2 is unbound, because it is bound to no instances of x.
  - y_0 is unbound, because it is bound to no instances of y.

- Scope of Bindings:
  - The occurrence of x, that we call x_1, is bound within the expression $(\lambda x.xy)$ by x_0.
  - Note that the occurrence of x that we call x_2 is not bound by the expression $(\lambda x.xy)$.
- Free Variables:
  - y_0, because it is unbound within its scope.
  - x_2, because it is unbound within its scope.

1.4 Computations in Lambda Calculus

There are two types of computations that you can do with the untyped lambda calculus that we'll worry about: alpha (α)-conversion, and beta (β)-reduction.

α**-conversion** is a way to avoid variable capture by substituting the associated variable (and its bound occurrences) with a new name that is not used in the related expression. More concretely, given a lambda calculus expression $\lambda x.e$, we are allowed to select a new name for x (say, y) given that:

- y is not free in e, and
- there are no free occurrences of x in e that become bound within e when replaced with y.

That is, we only change variable names for the binding variable and its binding occurrences.

For example:

$$\lambda x.xy \xrightarrow{\alpha} \lambda z.zy$$

but:

$$\lambda x.xy \xrightarrow{\alpha} \lambda y.yy$$

is invalid because y is free in the expression (namely, xy).

β**-reduction** is, trivially put, function application. More precisely, given $(\lambda x.e_1)\ e_2$, we replace every occurrence of x in e_1 with e_2, taking into account any potential variable capture (and handling it via α-conversion). We evaluate lambda expressions left to right, meaning that:

$$(\lambda x.\lambda y.xy)\ (\lambda y.y)\ (\lambda z.z) \xrightarrow{\beta} (\lambda y.(\lambda y.y)y)\ (\lambda z.z) \xrightarrow{\beta} (\lambda y.y)\ (\lambda z.z) \xrightarrow{\beta} (\lambda z.z)$$

is valid, whereas:

$$(\lambda x.\lambda y.xy)\ (\lambda y.y)\ (\lambda z.z) \xrightarrow{\beta} (\lambda y.xy)\ (\lambda z.z) \xrightarrow{\beta} ...$$

and:

$$(\lambda x.\lambda y.xy)\ (\lambda y.y)\ (\lambda z.z) \xrightarrow{\beta} (\lambda y.(\lambda z.z)y)\ (\lambda y.y) \xrightarrow{\beta} ...$$

are invalid because the original expression was not evaluated left to right.

2 Handing In

You should commit and push a PDF copy of your solution to your student repository in the **hw1-lambdacalc** folder and name the file **hw1-submission.pdf**.

To commit and push your work:

```
git add "hw1-submission.pdf"; git commit -m "handing in hw1"; git push
```

3 Your Task

Evaluate each of the following lambda expressions, showing each step of the computation along the way. Note that some (but not all) of the following expressions may require α-conversion. You should not perform α-conversion unless you believe there is variable capture that warrants it. As an example computation:

$$(\lambda x.(\lambda y.xy)z)\ (\lambda x.xy) \xrightarrow{\alpha} (\lambda x.(\lambda a.xa)z)\ (\lambda x.xy) \xrightarrow{\beta} (\lambda a.(\lambda x.xy)a)z \xrightarrow{\beta} (\lambda x.xy)z \xrightarrow{\beta} zy$$

Problem 1. $(\lambda x.x)\ y$

Problem 2. $(\lambda x.y)\ x$

Problem 3. $(\lambda x.x\ y)\ (\lambda y.y\ z)$

Problem 4. $(\lambda x.x\ y)\ (\lambda a.a\ b)\ p$

Problem 5. $(\lambda x.x\ y)\ (\lambda a.b\ a)\ p$

Problem 6. $(\lambda x.(\lambda y.x\ y))\ y$

Problem 7. $(\lambda x.y\ x)\ y$

Problem 8. $(\lambda x.\lambda y.x\ y\ z)\ (\lambda x.x\ y)\ z$

Problem 9. $(\lambda x.y\ x)\ x$

Problem 10. $(\lambda y.y\ x)\ (\lambda z.z\ y)$