Lab 5

Due Wednesday, March 12 by midnight

Handout 10
CSCI 334: Spring 2025

Encoding Proofs

Each question in this assignment must be answered by writing a function that returns a proof. To facilitate working with proofs, I provide some starter code, `LambdaViz.fs`, in your repository. You will need to create a new F# console program that opens this library's module, `LambdaViz`. Add the following to your `.fsproj`:

```
<Compile Include="Combinator.fs" />
<Compile Include="LambdaParser.fs" />
<Compile Include="LambdaViz.fs" />
```

`LambdaViz` comes with the following record datatype for encoding trees.

```
type Proof = { expr: string; justification: Justification }
```

where `Justification` is defined as

```
type Justification =
| Given
| EliminateParens
| AlphaReduce of replacement: char * original: char
| BetaReduce of replacement: string * variable: char
```

Suppose you have the following reduction proof:

$(\lambda x.x)(\lambda x.x)$	given
$(\lambda a.a)(\lambda x.x)$	α-reduce a for x
$((\lambda x.x))$	β-reduce $(\lambda x.x)$ for a
$(\lambda x.x)$	eliminate parentheses
$\lambda x.x$	eliminate parentheses

Then we encode the proof above as a `Proof list` like so.

```
[
    {
        expr = "(Lx.x)(Lx.x)";
        justification = Given
    };
    {
        expr = "(La.a)(Lx.x)";
        justification = AlphaReduce('a','x')
    };
    {
        expr = "((Lx.x))";
        justification = BetaReduce("(Lx.x)",'a')
    };
    {
        expr = "(Lx.x)";
        justification = EliminateParens
    };
    {
        expr = "Lx.x";
        justification = EliminateParens
    };
]
```

The only important difference between the grammar used by `LambdaViz` and the one we use in class is that you may type `L` instead of λ.

The `prettyproof` function in `LambdaViz` visualizes the proof, which is helpful to check your work. Use the following in your `main` function to print the proof to the console.

```
let proof1_str = prettyproof proof1
printfn "%s" proof1_str
```

The output will be a proof in two-column form. If any lambda expression in your proof contains a syntax error, `prettyproof` will indicate the line and string position of the error.

Pay attention to the types of the arguments for the `AlphaReduce` and `BetaReduce` justifications. `AlphaReduce` takes two `char`s. The first `char` is the `replacement` variable and the second `char` is the `original` variable. In other words, `AlphaReduce` represents the α reduction `[replacement/original]`. `BetaReduce` takes a `string` and a `char`. The first `string` is the `replacement` expression and the second `char` is the `original` variable. In other words, `BetaReduce` represents the β reduction `[replacement/original]`.

Coding Guidelines

Answers to all questions in this assignment should go into the root directory of your repository.

Your single program should be split into two pieces: a "`Program.fs`" file that contains the `main` method and associated program-startup helpers (if needed), and another "`Library.fs`" file that contains the function(s) of interest in the question. Library code should be contained within a module named "`CS334`". For full credit, your program should both build and run correctly.

Turn-In Instructions

Turn in your work using your assigned `git` repository. The name of your repository will have the form `https://aslan.barowy.net/cs334-s25/cs334-lab05-<USERNAME>.git`. For example, if your CS username is `abc1`, the repository would be `https://aslan.barowy.net/cs334-s25/cs334-lab05-abc1.git`.

You should have received an invite to commit to the repository via email. If you did not receive an email, please contact me right away!

Group Programming Assignment

This is a partner lab. You may work with another classmate if you wish, and you may co-develop solutions. Remember: although you can work on code together, you must each independently write up and submit your solution. No code copying is allowed. Tell me who your partner is by committing a `collaborators.txt` file to your repository. **Be sure to commit this file whether you worked with a partner or not.** If you worked by yourself, `collaborators.txt` should contain something like "I worked by myself." (5 points)

This assignment is due on Wednesday, March 12 by midnight.

Reading

1. **(Required)** "Introduction to the Lambda Calculus, Part 2"

Problems

Q1. (35 points) ..... Reduction #1

Reduce the following lambda expression.

$$(\lambda x.\lambda y.xy)(\lambda x.xy)$$

Do all possible reductions to find the normal form. Be sure to remove any unnecessary parens from your final answer. The following `Library.fs` function should return your proof.

```
let q1() : Proof list = ...
```

Q2. (30 points) ..... Reduction #2

Reduce the following lambda expression to a normal form.

$$(\lambda x.mx)((\lambda x.\lambda y.\lambda z.x)ohn)o$$

The following `Library.fs` function should return your proof.

```
let q2() : Proof list = ...
```

Q3. (30 points) ..... Reduction #3: Church Numerals

Church encoding is a scheme for representing data and operators in the lambda calculus. The Church numerals embed the set of natural numbers in lambda notation. The method is named for Alonzo Church, who first encoded data in the lambda calculus this way.

The natural numbers are written using Church numerals as follows.

Number	Lambda Expression
0	$\lambda f.\lambda x.x$
1	$\lambda f.\lambda x.fx$
2	$\lambda f.\lambda x.f(fx)$
3	$\lambda f.\lambda x.f(f(fx))$
...	...
n	$\lambda f.\lambda x.f^n x$

Addition by one can be achieved using the successor function, defined as

$$\text{succ} \equiv \lambda n.\lambda f.\lambda x.f(nfx)$$

Prove that $0 + 1 = 1$. The following `Library.fs` function should return your proof.

```
let q3() : Proof list = ...
```

Q4. ($\frac{1}{10}^{\text{th}}$ bonus point) ..... Optional: Feedback

I always appreciate hearing back about how easy or difficult an assignment is.

For $\frac{1}{10}^{\text{th}}$ of a bonus to your final grade, please fill out the following Google Form.