Midterm Study Guide

Handout 16
CSCI 334: Spring 2025

Reading

1. "A Brief Introduction to F#"

2. "More F#'

3. "F#: The Cool Stuff"

4. "Higher-Order Functions"

5. "Syntax"

6. "Introduction to the Lambda Calculus, Part 1"

7. "Introduction to the Lambda Calculus, Part 2"

Problems

Q1. (0 points) ..... Terminology

Review the required readings for this assignment, specifically looking for definitions for new terminology. Write down the term and a brief definition.

Hint: Look for italicized words. Italics are sometimes used for emphasis, but it is also often used to draw your attention to new technical terms. For example, in "Higher-Order Functions", one of the first italicized phrases is "first class." So write something like:

First-class: any value that can be assigned to a varible, passed to a function, or returned from a function.

Find as many technical terms as you can and define them.

Q2. (0 points) ..... Practice Reductions

Reduce each of the following expressions to their normal forms.

(a) $(\lambda x.x)(\lambda x.xx)(\lambda x.xa)$

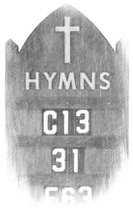
(b) $(\lambda x.x)(\lambda y.yy)(\lambda z.za)$

(c) $(\lambda x.\lambda y.xyy)(\lambda a.a)b$

(d) $(\lambda x.xx)(\lambda y.yx)z$

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

Q3. (0 points) ..... Church Numerals



Church encoding is a means of representing data and operators purely in the lambda calculus. The data and operators form a mathematical structure which is embedded in the lambda calculus. The Church numerals are a representation of the natural numbers using 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	$\lambda f.\lambda x.f^n x$

Subtraction by one can be achieved using the `pred` function.

$$\texttt{pred} \equiv \lambda n.\lambda f.\lambda x.n(\lambda g.\lambda h.h(gf))(\lambda u.x)(\lambda u.u)$$

Prove that 1 - 1 = 0 by performing a reduction using Church numerals and the above `pred` definition.

Q4. (0 points) ..... Backus-Naur Form

Suppose you start with the following (slightly augmented) grammar for the lambda calculus.

```
<expression>   ::= <value>
                |  <abstraction>
                |  <application>
                |  <parens>
                |  <arithmetic>

<value>        ::= v ∈ ℕ
                | <variable>
<variable>     ::=  α ∈ { a ... z }
<abstraction>  ::=  λ <variable>.<expression>
<application>  ::= <expression><expression>
<parens>       ::= (<expression>)
<arithmetic>   ::= (<op> <expression> ... <expression>)
<op>           ::= o  ∈ { +, - }
```

Add grammar support for exponentiation. For example, one should be able to parse the following expression:

$$((\lambda x.(+\ x\ 2))1)^2$$

In particular, the resulting parse tree should ensure that it correctly evaluates to `9`.

Q5. (0 points) ..... Currying and Partial Application

In this problem, we will use currying and partial application to produce convenience functions.

(a) Begin by writing a logging function. This function is intended to be called by another program and should write a message to the given file. Here is a sample output.

```
[2023-10-23 08:20][mysqld] FAIL: Tried (and failed) to read database too many times. Quitting.
```

Here is another sample output.

```
[2022-12-30 21:53][logind] WARN: User 'dbarowy' attempted to login without password.
```

Observe that the above string has four fields:

- The date, in `yyyy-MM-dd HH:mm` format.
- The name of the program (the "daemon") calling the log function.
- The severity of the message (which is either `INFO`, `WARN`, or `FAIL`).
- A message.

Your `log` function should have the following declaration.

```
let log (date: System.DateTime)(severity: string)(daemon: string)(message: string)(file: string) : unit = ...
```

To write to a file, `log` should use the `System.IO.File.AppendAllText` function (see `File.AppendAllText` documentation). You will also need to convert the `System.DateTime` object into a `string` having the format shown above (see `System.DateTime` documentation). Hint: if you find this to be difficult, be sure to look at `DateTime`'s `ToString` method.

(b) Next, write the following convenience function.

```
let lognow = ...
```

Importantly, `lognow` should be defined only by partial application of `log`. Specifically, `lognow` should be called `log` with a single argument corresponding to the current `DateTime` (see the documentation for `Now`). If you have done this correctly, then the return type for `lognow` will be `string -> string -> string -> string -> unit`.

(c) Next, write the following three convenience functions.

```
let info = ...

let warn = ...

let fail = ...
```

Each definition should be defined by calling `lognow` with a single argument corresponding to the `severity` parameter. If you have done this correctly, then the return type for each of the above functions will be `string -> string -> string -> unit`. You should be able to call the above functions like so,

```
info "daemon" "message" "foobar.txt"
```

and the effect will be that text resembling the following will be written to the file `foobar.txt`:

```
[2023-10-23 08:20][daemon] INFO: message
```

(d) Finally, make it so that the user can call your program from the command line with an expression of the form:

```
$ dotnet run <severity> <daemon> <message> <file>
```