Midterm 2 Study Guide

Handout 25
CSCI 334: Spring 2025

The second midterm is cumulative, drawing from all topics covered throughout the semester. More emphasis will be placed on topics from the second half of the semester.

First half	Second half
What is a Programming Language?	*Gumptionology 101*
An Introduction to F#	*Beating the Averages*
More F#	*Parsing*
F#: The Cool Stuff	*Evaluation*
Higher Order Functions	*Implementing Variables*
Syntax	*Implementing Scope*
Lambda Calculus, Part 1 and *Part 2*	*Type Inference* (handout)
Function Graphs (handout)	

Problems

Q1. Terminology

Review the semester's required readings, specifically looking for definitions of new terms. 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 "More F#", one of the first italicized phrases is *source code*. So write something like:

Source code: a textual form of code written by a programmer which is not directly interpretable by a machine.

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

Q2. F# Coding

Write a function:

```
isPalindrome: string -> bool
```

that returns `true` if the given string is a palindrome, otherwise `false`. Recall that a palindrome is a `string` that reads the same forwards and backwards.

```
isPalindrome "mom" = true
isPalindrome "dad" = true
isPalindrome "amanaplanacanalpanama" = true
```

Remember that you can use array indexing to access a single `char` inside a string.

```
> "mom"[1];;
val it: char = 'o'
```

Solve this problem using recursion. Your solution may not use a built-in function like `List.rev` or `Array.rev`.

Q3. ..... Lambda Calculus Reductions

Reduce the following expressions. If the expression does not have a normal form, explain why.

- `(λf.fz)(λx.x)`
- `(λx.λy.x(yy))λz.z`
- `(λx.λy.xxy)(λx.λy.xxy)`
- `(λf.λx.fx)(λy.z)((λx.xx)(λx.xx))`

Q4. (10 points) ..... Partial and Total Functions

Find the function graphs for the following functions and state whether the function is total or partial. Use the domain $\mathbb{Z}$ as input. Assume that an `int` can represent any $z \in \mathbb{Z}$.

(a)
```
let rec fibonacci n =
    match n with
    | 0 -> 0
    | 1 -> 1
    | _ -> fibonacci (n - 1) + fibonacci (n - 2)
```

(b)
```
let rec gcd a b =
    if b = 0 then
      a
    else
      gcd b (a % b)
```

(c)
```
let abs x =
    if x < 0 then
      -x
    else
      x
```

Q5. (10 points) ..... Parsing and Evaluation

The EPHSPRESSO language is a sequence of operations on a tape having the following grammar.

```
<expr> ::= <op>*
  <op> ::= + | - | < | > | p
```

A *tape* is an array of length 10, with all elements initially set to zero. A *tape head* is the current index of the tape, also initially zero. The `+` and `-` operations increment and decrement the number stored under the tape head, respectively. The `<` and `>` operations move the tape head left (decrement the index) or right (increment the index), respectively. The `p` operation prints a character representation of the number written on the tape under the tape head according to the following scheme:

$$0 \to \texttt{a}$$
$$1 \to \texttt{b}$$
$$2 \to \texttt{c}$$
$$\ldots$$
$$25 \to \texttt{z}$$

Moving the tape head off the tape causes the machine to terminate abnormally (i.e., with an error). Otherwise, the machine terminates normally when the program finishes executing.

Implement EPHSPRESSO in F#. You should be able to write a program that prints `ephs` to the console.

Q6. ..... Type Inference

Infer the type of the following function.

```
let f (x: int) y = string (x + 1) + y
```

Assume the functions shown below have the following types.

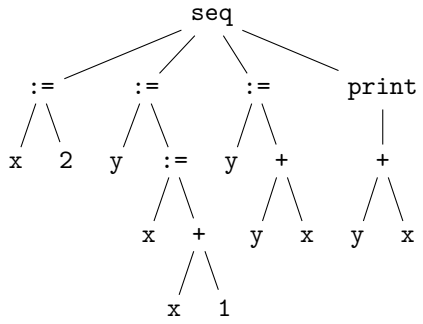
function	type
+	`int -> int -> int`
+	`string -> string -> string`
string	`int -> string`

Q7. ..... Global Variables

Suppose you have the following program, in a language with globally-scoped variables, where `:=` stands for assignment.

```
x := 2
y := x := x + 1
y := y + x
print y + x
```

This program parses into the following AST.



Determine what the final `print` statement prints by simulating evaluation with a global variable environment. The initial environment is empty, written `{ }`. Assume that environments are always passed from the left hand side of a `seq` to the right hand side and that assignment returns a value. To get you started, here is the same tree with just the leftmost assignment evaluated. The environment is shown in red and return values are shown in blue.