Homework 5

Due Wednesday, 14 March

Handout 19
CSCI 334: Spring 2018

Notes

This homework has two types of problems:

Problems: You will turn in answers to these questions.

Pair Programming: This part involves writing ML code. You are required to work with a partner for this section. I will assume that you plan to work with the same partner you worked with on the last assignment unless you email me (dbarowy@cs.williams.edu) with your partner's name **by the evening of Wednesday, March 7**.

Turn-In Instructions

You must turn in your work for this assignment in two parts, one for the Problems part and one for the Pair Programming part. You will be assigned two GitHub repositories.

Note that this assignment should be anonymized: neither your name nor your partner's name should appear in any of the files submitted with this assignment except the file "`collaborators.txt`" (see below).

Problems: Problem sets should be typed up using LaTeX and submitted using your `cs334_hw5_<username>` repository. For example, if your GitHub username is `dbarowy`, then your repository will be called `cs334_hw5_dbarowy`. Be sure that your work is committed and pushed to your repository by Wednesday, March 14 at 11:59pm.

If you discuss the problem set with your partner or with a study group, please be sure to include their names in a `collaborators.txt` file in your repository.

Pair Programming: Programming solutions should be typed up and submitted using your partner repository. For example, if your GitHub username is `dbarowy` and you are working with a partner whose username is `wjannen`, look for a repository called `cs334_hw5_dbarowy-wjannen` (usernames will be in alphabetical order). Be sure that your work is committed and pushed to your repository by Wednesday, March 14 at 11:59pm.

Standard ML files should have a "`.sml`" suffix. For example, the pair programming problem **P1a** should appear as the file `p1a.sml`. Your code should be documented using comments. Comment lines appear inside `(*` and `*)` braces.

Reading

1. **(Required)** Read Mitchell, Chapters 6 and 7.

Problems

Q1. (25 points) ..... Infer the Types

Infer the ML types for the following function using the Hindley-Milner algorithm.

```
fun f (x::xs) = (x,x)::(f xs)
 |  f nil     = nil
```

Specifically,

(a) convert the function to a λ expression;

(b) draw the corresponding parse tree for the λ expression;

(c) using a table, label each subexpression with a type variable;

(d) generate constraints using the λ-calculus constraint rules;

(e) and solve the constraints, yielding the type of `f`.

Note that this problem includes a case not discussed in class, but which you can find in Mitchell pp. 135–145.

Assume that the cons operator `::` is a curried function of type `'a → 'a list → 'a list`. You may also assume that the tuple constructor (a, b) is a curried function of type `'a → 'b → 'a*'b`.

Q2. (10 points) ..... Static and Dynamic Scope

Consider the following program fragment, written in ML:

```
1  let val x = 2 in
2    let val f = fn y => x + y in
3      let val x = 7 in
4        x +
5            f x
6      end
7    end
8  end;
```

(a) Under static scoping, what is the value of `x + f x` in this code? During the execution of this code, the value of `x` is needed three different times (on lines 2, 4, and 5). For each line where `x` is used, state what numeric value is used when the value of `x` is requested and explain why these are the appropriate values under static scoping.

(b) Under dynamic scoping, what is the value of `x + f x` in this code? For each line where `x` is used, state which value is used for `x` and explain why these are the appropriate values under dynamic scoping.

Q3. (15 points) ..... Function Calls and Memory Management

This question asks about memory management in the evaluation of the following statically-scoped ML expression.

```
val x = 5;
fun f(y) = (x+y)-2;
fun g(h) = let val x = 7 in h(x) end;
let val x = 10 in  g(f) end;
```

(a) (12 points) Fill in the missing information in the following depiction of the run-time stack after the call to `h` inside the body of `g`. Remember that function values are represented by closures, and that a closure is a pair consisting of an environment (pointer to an activation record) and compiled code.

In this drawing, a bullet (•) indicates that a pointer should be drawn from this slot to the appropriate closure or compiled code. Since the pointers to activation records cross and could become difficult to read, each activation record is numbered at the far left. In each activation record, place the number of the activation record of the statically enclosing scope in the slot labeled "access link." The first two are done for you. Also use activation record numbers for the environment pointer part of each closure pair. Write the values of local variables and function parameters directly in the activation records.

Activation Records *Closures* *Compiled Code*

(1)		access link	(0)
		x	
(2)		access link	(1)
		f	•
(3)		access link	( )
		g	•
(4)		access link	( )
		x	
(5)	g(f)	access link	( )
		h	•
		x	
(6)	h(x)	access link	( )
		y	

⟨( ), • ⟩

⟨( ), • ⟩

code for f

code for g
... ...

(b) (3 points) What is the value of this expression? Why?

Pair Programming

P1. (5 points) ..... Lisp Conditional

A Lisp programmer learning ML doesn't like the syntax of the ML conditional expression, `if ...then ...else`. Instead, the programmer decides to define a conditional function:

```
fun Cond(test, trueCase, falseCase) =
          if test then trueCase else falseCase;
```

You foresee problems if this programmer uses this `Cond` function exclusively in place of `if ...then ...else`. Write two short programs that demonstrate the problem.

(a) Write a short program using `Cond`.

(b) Write the supposedly equivalent program using `if ...then ...else` that produces a different result.

P2. (10 points) ..... Exceptions in ML

The function `stringToNum` defined below uses two auxiliary functions to convert a string of digits into a non-negative integer.

```
(* Convert one character to a numeric digit. *)
fun charToNum c = ord c - ord #"0";

fun calcList (nil,n) = n
  | calcList (fst::rest,n) =
                   calcList(rest,10 * n + charToNum fst);

(* Convert a string of digits to a number.  The explode function
   converts a string to a list of characters. *)
fun stringToNum s = calcList(explode s, 0);
```

For instance, `stringToNum "3405"` returns the integer 3405. (The function `explode` converts a string into a list of characters, and `ord` returns the ASCII integer value for a character.)

Unfortunately, `calcList` returns a spurious result if the string contains any non-digits. For instance, `stringToNum "3a05"` returns 7905, while `stringToNum " 405"` returns ~15595. This occurs because `charToNum` will convert any character, not just digits. We can attempt to fix this by having `charToNum` raise an exception if it is applied to a non-digit.

(a) Revise the definition of `charToNum` to raise an exception, and then modify the function `stringToNum` so that it handles the exception, returning ~1 if there is a non-digit in the string. You should make no changes to `calcList`.

(b) Implement ML functions to provide the same behavior (including returning ~1 if the string includes a non-digit) as in the first part, but without using exceptions. While you may change any function, try to preserve as much of the structure of the original program as possible.

(c) Which implementation do you prefer? Why? Answer this as a comment in the code.

P3. (20 points) ..... Folding Fun

Use `foldl` or `foldr` to solve the following problems.

(a) Write a function `concatWords: string list -> string`. This function should return return a string with all strings in the list concatenated:

```
- concatWords nil;
val it = "" : string
- concatWords ["Three", "Short", "Words"];
val it = "ThreeShortWords" : string
```

(b) Write a function `words_length: string list -> int`. This function should return the total length of all words appearing in a list of strings. For example:

```
- words_length nil;
val it = 0 : int
- words_length ["Three", "Short", "Words"];
val it = 15 : int
```

(c) Can we always use `foldl` in place of `foldr`? If yes, explain. If no, give an example function `f`, list `l`, and initial value `v` such that `foldr f v l` and `foldl f v l` behave differently.

(d) Write a function `count: ''a -> ''a list -> int`. It computes the number of times a value appears in a list. For example:

```
- count "sheep" ["cow", "sheep", "sheep", "goat"];
val it = 2 : int
- count 4 [1,2,3,4,1,2,3,4,1,2,3,4];
val it = 3 : int
```

(e) Write a function `partition: int -> int list -> int list * int list` that takes an integer p and a list of integers l, and that returns a pair of lists containing the elements of l smaller than p and those greater than or equal to p. The ordering of the original list should be preserved in the returned lists. (We wrote a recursive form during lecture as part of quicksort.)

```
- partition 10 [1,4,55,2,44,55,22,1,3,3];
val it = ([1,4,2,1,3,3],[55,44,55,22]) : int list * int list
```

(f) Write a function `poly: real list -> (real -> real)` that takes a list of reals `[`a_0`,` a_1`, ...,` a_{n-1}`]` and returns a function that takes an argument `b` and evaluates the polynomial

$$a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$$

at $x = b$; that is, it computes $\sum_{i=0}^{n-1} a_i b_i$. For example,

```
- val g = poly [1.0, 2.0];
val it = fn: real -> real
- g(3.0);
val it = 7.0: real
- val g = poly [1.0, 2.0, 3.0];
val it = fn: real -> real
- g(2.0);
val it = 17.0: real
```

(Hint: $a_0 + a_1x + a_2x^2 + a_3x^3 = a_0 + x(a_1 + x(a_2 + xa_3))$. This is an example of Horner's Rule. Horner's Rule demonstrates that we can evaluate a degree n polynomial with only $O(n)$ multiplies.)