Lab 2: Type Signatures and More Higher-order Functions

Wednesday September 13, 2023

Weekly Lab Agenda

- Go over reminders/goals
- Review past material
- Work in groups of 2-3 to solve a few exercises
    - Please sit with your group from last week.
- Discussion leaders will walk around and answer questions
- Solutions to exercises will be reviewed as a class
- Attendance taken at the end

Reminders

- Homework 1 is due tonight at 11:59pm
    - Come to office hours for help!
- Homework 2 will be released soon
- If you need to miss lab and have a valid reason according to the syllabus (medical, other personal) please fill out the questionnaire on moodle before the start time of your lab.
    - Waking up late, bus was late are NOT valid reasons to miss lab.

Today's Goals

- Practice writing correct type signatures
- Practice with more higher-order functions

Review of Type Signatures

We can infer types based on the operations done on values

```
// f(x: number, y: number): number     or  f: (number, number) => number
function f(x, y) {
  return x + (2*y);
  // product with y: y is number, x and result is number
}
```

Sometimes, we have several possibilities

```
// g: (number, number) => number    or    g: (string, string) => string
function g(x, y) { return x + y; }  // + can be string concatenation

// h(a: string): boolean   or   h<T>(a: T[]): boolean
function h(a) { return a.length > 5; } // both strings & arrays have length
```

The array could have any element type. We call T a **type variable.**
This lets us write **generic functions.**

Exercise 1: Type Signatures

What is the type signature for the following code?

```
const h = (a, g, f) => f(a.map(g)).filter(g);
```

Start by considering an arbitrary element type for the array, and continue to derive types for the map and filter callback functions.

Remember:

```
map<A,B>(arr: A[], f: (x: A) => B): B[]
```

```
filter<T>(arr: T[], f: (x: T) => boolean): T[]
```

Review of Reduce

```
function reduce<T, U>(
     a: T[],
     f: (acc: U, e: T) => U,
     init: U
): U {
     let result = init;
     for (let i = 0; i < a.length; ++i) {
          result = f(result, a[i]);
     }
     return result;
}
```

Reduce is used to combine array elements with the same function.

Example: Find the product of all elements of an array a = [3, 2, 6, 2, 2, 0]

```
a.reduce((prod, e) => prod * e, 1);
```

Exercise 2: Reduce

Write a function that counts how many pixels in an array are “mainly blue”.

- Input: An array of type Color[] (where Color is a triple [R, G, B])
- Output: The number of pixels satisfying the following condition: the blue value is at least twice the red value and at least twice the green value
- Note: Define “`type Color = [number, number, number];`”

Can you solve the same problem for a 2D array of type Color[][] ?

Exercise 3: Map, Filter & Reduce

Write a function without using loops or recursion to calculate the sum of the natural log of all positive numbers in an array.

Note: Use Math.log() to calculate the logarithm.

Example: Input: [-1, 0, 2, 10, -5] => Output: log(2) + log(10) = 2.9957…