Homework 3

Due date: Monday, October 5, 2015

The purpose of this assignment is to get practice with structural recursion and induction. Moreover, you will do some experiments with Node.js to understand JavaScript's type conversion mechanism.

Like last time, you will work on this assignment in pairs. However, note that each student needs to submit a write-up and are individually responsible for completing the assignment. You are welcome to talk about these exercises in larger groups. However, we ask that you write up your answers in pairs. Also, be sure to acknowledge those with which you discussed, including your partner and those outside of your pair.

Try to make your code as concise and clear as possible. Challenge yourself to find the most crisp, concise way of expressing the intended computation. This may mean using ways of expressing computation currently unfamilar to you.

Finally, make sure that your file compiles and runs (using Scala 2.11.7). A program that does not compile will *not* be graded.

Submission instructions. Upload to NYU classes exactly three files named as follows:

- `hw03-YourNYULogin.pdf` with your answers to the written questions (scanned, clearly legible handwritten write-ups are acceptable).
- `hw03-YourNYULogin.scala` with your answers to the coding exercises.

Replace `YourNYULogin` with your NYU login ID (e.g., for me, I would submit files named `hw03-tw47.pdf` and `hw03-tw47.scala`). Don't use your student identification number. To help with managing the submissions, we ask that you rename your uploaded files in this manner.

Getting started. Download the code pack `hw03.zip` from the assignment section on the NYU classes page.

Problem 1 Structural Recursion and Induction (22 Points)

Recall the definition of the set N, whose elements represent the natural numbers:

$$\frac{}{\emptyset \in N} \qquad \frac{x \in N}{\{x\} \in N}$$

On N we defined an addition function as follows

$$\begin{aligned} + &: N \times N \to N \\ \emptyset + y &= y \\ \{x\} + y &= \{x + y\} \end{aligned}$$

(a) Define a multiplication function $\cdot : N \times N \to N$ **(4 Points)**.

(b) Proof by structural induction $\forall x \in N : \{x\} = x + \{\emptyset\}$ **(4 Points)**.

Recall the definition of the set *List* of all lists of integer numbers:

$$\frac{}{\langle\rangle \in \mathit{List}} \qquad \frac{\mathit{hd} \in \mathbb{Z} \quad \mathit{tl} \in \mathit{List}}{\langle \mathit{hd}, \mathit{tl}\rangle \in \mathit{List}}$$

(c) Define a function $\mathit{filter} : \mathbb{Z} \times \mathit{List} \to \mathit{List}$ that takes an integer number $n \in \mathbb{Z}$ and a list $\ell \in \mathit{List}$ and computes the list that is obtained by removing all numbers from ℓ that are smaller than n. For example

$$\mathit{filter}(3, \langle 5, \langle 2, \langle -1, \langle 3, \mathit{Nil}\rangle\rangle\rangle\rangle) = \langle 5, \langle 3, \mathit{Nil}\rangle\rangle$$

(6 Points)

(d) Write a tail-recursive Scala function

```
def filter(n: Int, l: List): List
```

that implements the function *filter*. **(4 Points)**

(e) Prove by structural induction that for all $\ell \in \mathit{List}$ the following property holds:

$$\mathit{reverse}(\mathit{reverse}(\ell)) = \ell$$

In your proof you may use the following property without proving it: for all $\ell_1, \ell_2 \in \mathit{List}$

$$\mathit{reverse}(\mathit{append}(\ell_1, \ell_2)) = \mathit{append}(\mathit{reverse}(\ell_2), \mathit{reverse}(\ell_1))$$

Use the definitions of *reverse* and *append* provided in the course notes. **(4 Points)**

Problem 2 JakartaScript: Type Conversions (18 Points)

One aspect that makes the JavaScript specification complex is the presence of implicit conversions (e.g., string values may be implicitly converted to numeric values depending on the context in which values are used). For example, consider the following Javascript expression:

```
3 * "4"
```

which multiplies the numeric value `3` with the string value `"4"`. Evaluating this expression using a JavaScript interpreter such as Node.js yields the numeric value 12. Thus, the string value `"4"` is first converted to the numeric value `4` before the actual multiplication operation is performed.

In this exercise, we will explore some of the complexity of JavaScript's type conversion mechanism by writing a few small JavaScript programs that illustrate how implicit conversion works.

(a) The goal of this first exercise is to understand how the implicit conversions between the different primitive types of JavaScript work. To do so, we can write simple JavaScript expressions using operators that expect values of specific types and then use these operators with values of other types. For example, the multiplication operator `*` expects numeric values as arguments. So we can this operator on string and Boolean values to experiment with the implicit conversion of these values to numeric values. Similarly, we can use the Boolean negation operator `!` to experiment with the implicit conversion of string and numeric values to Boolean values. Write at least 10 simple JavaScript expressions that illustrate the following implicit conversions:

- string values to numeric values
- Boolean values to numeric values
- string values to Boolean values
- numeric values to Boolean values

Evaluate your expressions using Node.js and report both the expressions and the result of the evaluation. Do not forget to consider corner cases where the conversion is not obvious. Infer general rules that describe how the considered implicit type conversions work.

Hint: avoid the operators +, `&&`, `||`, and == in your experiments. We will explore some of these operators in more detail below. **(6 Points)**

(b) One of the confusing aspects of implicit type conversion in JavaScript is how it interacts with operator overloading. For example, JavaScript's + operator can mean addition of numeric values but also concatenation of strings. Which of the two operations is performed for a specific occurrence of + in a JavaScript program depends on the types of the arguments of + that are observed at run-time. Write at least six JavaScript expressions that involve the + operator and evaluate them using Node.js. Use string, numeric, and Boolean values as arguments and also mix the different argument types. Report the expressions and the results of their evaluation. Infer a general rule that describes how + is interpreted in JavaScript. **(6 Points)**

(c) The semantics of the Boolean operators `&&` and `||` in JavaScript is complicated by so-called *short-circuit evaluation*. In particular, the type of the result value of these two operators depends on the type of their second argument. Write at least six simple JavaScript expressions where the operators `&&` and `||` are applied to a mix of string, numerical, and Boolean values. Evaluate your expressions using Node.js and report your results. Infer general rules that describe how the two operators work. **Hint:** For `&&` and `||` type conversion is only performed on the first argument, but not on the second argument.