P. Thiemann, G. Radanne Wintersemester 2017/18

Functional Programming

http://proglang.informatik.uni-freiburg.de/teaching/functional-programming/2017/

Exercise Sheet 8 – GADTs

24/01/2017

Note: For the following exercises, you must turn on the GHC extension `GADTs` . To do this, add the following special comment (pragma) at the beginning of your Haskell source file:

```
{- # LANGUAGE GADTs # -}
```

In addition, it also recommended to use the pragma

```
{- # OPTIONS_GHC -fwarn-incomplete-patterns # -}
```

at the beginning of the file. This will make GHC warn you about pattern matchings that do not cover all the cases.

Exercise 1 (Safe List)
Define a list type `SafeList` that supports a “safe” head `safeHead` operation. That is, the type checker should allow the use of `safeHead` only if the argument is a non-empty `SafeList`:

```
safeHead (Cons 4 Nil) - ok
- safeHead Nil - Type error
```

This operation is called “safe” because it does not cause a runtime error on incorrect inputs, unlike `head`.
In addition, implement `safeDrop` and `safeAppend` in a meaningful way.

Exercise 2 (Stack Calculator)
We previously implemented a stack calculator. This one was quite simple:

- only arithmetic operations
- always returns 0 on underflow

Now, we can do it better! The stack should now have a finite size and contain both `Int` and `Bool` values. The stack programs should consist of the following commands:

```
sprog ::=
  | Noop -- Doesn't do anything
  | Pop -- Removes the top element from the stack
  | Push v -- Puts the value v on the stack
  | Dup -- Places a duplicate of the top element on the stack
  | Dup2 -- Places duplicates of the top two elements on the stack
  | Flip -- Swaps the two top elements
  | Add | Substract | Multiply
    -- Perform an arithmetic operation on the top elements and puts
    -- the result on the stack
  | Le | Ge -- Compares the two top elements and puts the result on the stack
  | Not | And | Or -- Performs a logical operation
  | sprog1; sprog2 -- Executes sprog1 first, then sprog2
  | if sprog1 sprog2 -- Executes sprog1 if True is at the top and sprog2 otherwise
```

1. Define the data type `SProg` so that it only accepts programs that can be executed without errors.

2. Implement a tag-free interpreter for `SProg`.
3. Write a `SProg` program that calculates the max of the two top elements. Write two more `SProg` toy programs and test them.
4. Now add the loop construct `while` to `SProg`.

   ```
   sprog ::= ... | while sprog
   -- Perform sprog as long as the top element of the stack is not True
   ```

5. Write a `SProg` program which, given two integers x, y at the top of the stack, places $x \bmod y$ at the top of the stack. Use only the operations from the above syntax (and `while`, of course)
6. Give an example of a simple error-free stack program that you can *not* express with your `SProg` data type.