# Refinement types in Haskell: Exercise Sheet 2

April 10, 2025

**Exercise 1.** Give a well-typed definition of the Ackermann function in Liquid Haskell. Hint: you'll need to use termination metrics to prove that it terminates.

**Exercise 2.** Recall the following definition of de Bruijin lambda terms in Liquid Haskell:

```
{-@ type Nat = { n:Int | n >= 0 } @-}

data Expr = Var Int | Lam Expr | App Expr Expr
{-@ data Expr = Var Nat | Lam Expr | App Expr Expr @-}
```

**Part 1**. Using measures (and reflection if you wish) define the type of *closed* lambda terms.

**Part 2.** Define a function that $\beta$-normalises a closed lambda term.

*Bonus exercise:* define the type of lambda expressions indexed over the number of free variables i.e. de Bruijin indexed lambda terms. This can be done either with a record that contains an explicit representative of the number of free variables or with type level indexing.

**Exercise 3.** Recall that Hutton's razor is the simple expression language defined as follows:

```
data Expr = Val Int | Add Expr Expr

eval :: Expr -> Int
eval (Val n)   = n
eval (Add x y) = eval x + eval y
```

**Part 1.** Define your own list type `List` with a custom length measure and a concatenation function.

**Part 2.** Consider the following types for a stack, instruction and program:

```
type Stack = List Int

data Instr = PUSH Int | ADD

type Code = List Instr
```

**Part 3.** Define a stack-based execution function for Hutton's razor of the form:

```
exec :: Code -> Stack -> Stack
```

**Part 4.** Define a compiler for Hutton's razor of the form:

```
comp :: Expr -> Code
```

**Part 5.** Construct a proof of the following correctness theorem for your compiler:

```
{-@ correctness :: e:Expr -> c:Code -> s:Stack ->
                   { exec (comp e ++ c) s = exec c (Cons (eval e) s) }
@-}
```

*Hint:* Ensure that you make liberal use of reflection and proof by logical evaluation!