Lernziele

- Werte und Ausdrücke
- Algebraische Typen und Pattern-Matching
- Listen

Aufgabe 1

Diskutieren Sie Unterschiede der folgenden Funktionen:

```
let f x =
    let mutable y = 0
    for i in 1..10 do
        printfn "%i" y
        y <- y + 1
    x
```

```
let f x =
    let y = 0
    for i in 1..10 do
        printfn "%i" y
        let y = y + 1
    x
```

Aufgabe 2

(a) Implementieren Sie einen Typ `NatNumber` entsprechend der Definition:

> *Eine natürliche Zahl ist entweder Null oder Nachfolger einer natürlichen Zahl.*

(b) Implementieren Sie eine Funktion `eval: NatNumber -> uint32`, die natürliche Zahlen "auswertet" und eine Funktion `interpret`, die sich zu `eval` "dual" verhält.

(c) Definieren Sie mit einem Pattern-Match eine Funktion `fact: NatNumber -> NatNumber`. Halten Sie sich dabei möglichst exakt[1] an die mathematische Definition:

$$fakt(0) = 1$$
$$fakt(n+1) = (n+1) \cdot fakt(n)$$

[1] Achten Sie insbesondere darauf, dass links "$n+1$" steht und nicht rechts "$n-1$".

(d) Definieren Sie einen Datentyp `IntNumber`, als Record, entsprechend der **hier** gegebenen Konstruktion. Implementieren Sie folgende Funktionen:

- `embed: NatNumber -> IntNumber` entsprechend der Zuordnung $x \mapsto x$.
- `eval: IntNumber -> int` und
  `interpret: int -> IntNumber`,
  um `IntNumber` als `int` zu evaluieren und umgekehrt.
- `add, sub, neg` um die gewöhnliche Addition, Subtraktion und Negation auf `IntNumber` zu ermöglichen.

Aufgabe 3

Lesen Sie diesen Eintrag über die "Programmiersprache" Fractran. Implementieren Sie einen Fractran interpreter (beachten Sie dazu die Anmerkungen in der Vorlesung)

```
run: (I * I) list -> I -> I
```

wobei `I` für `System.Numerics.BigInteger` stehe.