Formal Methods in Software Engineering
Laboratory 2

1. Read from Dafny Reference Manual how to define algebraic data types in Dafny.
2. Define the natural numbers as an algebraic data type and use it for:
   (a) proving that the successor constructor is injective and that zero is different from successor(x), for any x;
   (b) inductively defining the addition off natural numbers;
   (c) proving that the addition is associative and commutative;
   (d) defining a predicate $lt(m, n)$ that holds when m is less than n;
   (e) proving that lt is transitive.
3. Define the parametric lists as an algebraic data type and use it for:
   (a) defining the size of a list (using natural numbers defined above);
   (b) defining the concatenation of two lists;
   (c) proving that the size of the concatenation of two lists is the sum of the lists size;
   (d) defining a function reversing a list;
   (e) proving that reversing a list twice we obtain the initial list.

A Induction in Dafny

A.1 Inductive Sets

Natural numbers by rules:

$$\frac{}{\mathsf{0}} \qquad \frac{n}{\mathsf{s}(n)}$$

The same definition in Dafny:

```
datatype Nat = Zero | Succ(Pred: Nat)
```

where `0` is renamed by `Zero` and `s` by `Succ`.

A.2 Discriminator Predicates

```
lemma Discr(n: Nat)
ensures n.Zero? || n.Succ?
{
    //
}
```

A.3 Recursive Definition

Example:

$$
\begin{aligned}
&\mathit{double} : \mathit{Nat} \to \mathit{Nat}\\
&\mathit{double}(\mathsf{0}) = \mathsf{0}\\
&\mathit{double}(\mathsf{s}(n)) = \mathsf{s}(\mathsf{s}(\mathit{double}(n)))
\end{aligned}
$$

```
function double(n: Nat) : Nat
{
    match n
    case Zero => Zero
    case Succ(n’) => Succ(Succ(double(n’)))
}
```

which is equivalent to

```
function double(n: Nat) : Nat
{
    if n.Zero? then Zero else  Succ(Succ(double(n.Pred)))
}
```

A.4 Recursive Predicates

Example:

```
predicate evenNat(n: Nat)
{
    match n
    case Zero => true
    case Succ(n’) => ! evenNat(n’)
}
```

A.5 Proofs by Induction

Example:

```
lemma doubleIsEven(n: Nat)
ensures evenNat(double(n))
{
    match n
    case Zero => assert evenNat(Zero);
    case Succ(n’) => doubleIsEven(n’);
}
```