5.- Assume, Lemmas, Calculations and Sequences

Using **assert** and **assume** in software development

- **assert** φ
  - Dafny first tries to prove φ, and if successful, then φ can be used in the rest of proof.
  - Provides a non-instantiable lemma φ:
    A property that is previously and separately proved and helps to prove other properties.
- **assume** φ
  - Dafny assumes that φ is true: it is enabled to use φ in the current proof (without proving it).
  - Dafny does not consider verified any file with one **assume**.
- In verified software development:
  - **assume** φ for checking if φ is the required property;
  - If OK then change from **assume** to **assert**;
  - If "assertion violation" then φ must be proved in a **lemma** or some previous **assert**(s) must be inserted (as lemmas).

Lemmas in Dafny

- **lemma** N(x1: T1,...,xn: Tn)
  requires α;
  ensures β;
- For a concrete call N(a1,...,an):
  wp(N(a1,...,an),ψ) = β[a1,...,an/x1,...,xn]$\rightarrow \psi$ provided that α[a1,...,an/x1,...,xn] is satisfied.
- In other words:

  N(a1,...,an);
  assert ψ;

  is equivalent to

  assert α[a1,...,an/x1,...,xn];
  assume β[a1,...,an/x1,...,xn];
  assert ψ;

Verified Calculations

- A calculation in Dafny is an statement that proves a property by a chain of expressions, each transformed into the next.
- The grammar for calculations is:

```
CalcStatement ::= calc {
                  CalcBody
                  }
CalcBody ::= Line
             (Op Hint
              Line)*
Line ::= Expression ;
Op ::= ≤ | < | ≥ | > | ⟹ | ⟸ | ⟺ | ≠ | =

Hint ::= { (BlockStatement | CalcStatement)* }
```

where a BlockStatement is one or more **assert**/**assume** clauses and lemma calls.

Proof by contradiction

A proof by contradiction *of* φ *can be made supposing that* $\neg\varphi$ *holds and getting* **false** .

```
lemma X()
   requires ...
   ensures Q(x)
{
if ¬Q(x) { calc {
                  // derive contradiction
                  ⟹ false;
                  }
           }
}
```

Value types

- Value types are types whose members represent some information that does not depend on the state of the heap (inmutable), whereas objects have a state (represented in the heap) that is mutable.
- Values have a mathematical flair: they cannot be modified once they are created.
- Value types can be stored in fields (i.p. var) on the heap, and used in real code in addition to specifications. Variables that contain a value type can be updated to have a new value of that type.
- Dafny's built in *value types are sets, sequences, multisets, and maps.*

Sequences

- Sequences are an immutable value type.
- A sequence can be formed using an ordered list of expressions enclosed in square brackets.

```
[3, 1, 4, 1, 5, 9, 3] [4+2, 1+5, a*b]
```

- For any type T, a value of type **seq**⟨T⟩ denotes a sequence of elements of type T, that is, a mapping from a finite set of consecutive natural numbers (called indicies) to T values.

```
[]          // empty sequence
s + r       // concatenation
≤           // prefix
<           // proper prefix
```

```
predicate sorted(s: seq⟨int⟩)
{
    forall i,j • 0 ≤ i < j < |s| ⟹ s[i] ≤ s[j]
}
```

A generic predicate (requiring equality)

```
predicate palindrome⟨T(=)⟩ (s:seq⟨T⟩)
{
  // forall i • 0 ≤ i ≤(|s|-1)/2 ⟹ s[i]=s[|s|-i-1]
   forall i • 0 ≤ i < |s| ⟹ s[i] = s[|s|-1-i]
                          //easier to understand and to validate
}
```