Lab Session 5
Introduction to the Why3 Tool

Software Verification

Nova School of Science and Technology
Mário Pereira   mjp.pereira@fct.unl.pt

Version of October 6, 2025

1 Verification of Programs Without Loops

Exercise 1. Implement, specify and verify the following methods. Try to define the strongest post-condition and the weakest pre-condition possible.

1. Absolute value – `let abs (x: int) : int`
2. Maximum of two integers – `let max2 (x y: int) : int`.
   (a) Use a function to specify the post-condition.
   (b) Don't use a function to specify the post-condition.
3. Maximum of three integers:
   `let max3 (x y w: int) : int`.
   Use the specification and functions above to specify and implement function `max3`. Do a version with and without the use of a function in the specification.
4. Comparison of two integers:
   `let compare_to (x y: int) : int`

Write as many intermediate assertions in the code as you can, to illustrate the proof behind a verified method in Why3. □

Exercise 2. Implement and verify (i.e. define suitable pre- and post- conditions) a function:

```
let interval_contains (low high low_b high_b: int
```
WhyML

That takes four natural numbers, specifying two time intervals A and B. The interval A is defined as [`low_a`, `high_a`], while B is defined as [`low_b`, `high_b`]. The method tests whether (*i.e.*, returns `true` iff) interval A fully contains interval B.

Important note: integer values in Why3 stand for the mathematical $\mathcal{Z}$ set. So, it is your responsibility to provide enough pre-conditions to guarantee correct usage of the function arguments.

□

Exercise 3. Consider the following Why3 functions:

WhyML
```
method mystery1(n:nat,m:nat) returns (res:nat)
   ensures true
{
   if (n==0) {
         return m;
   } else {
      var aux := mystery1 (n-1,m);
      return 1+aux;
   }
 }
method mystery2(n:nat,m:nat) returns (res:nat)
   ensures true
{
   if (n==0) {
      return 0;
   }
   else {
      var aux := mystery2(n-1,m);
      var aux2 := mystery1(m,aux);
      return aux2;
   }
}
```

Change the post-condition of both functions so that it is the *strongest* possible. Also, provide a *termination measure* to prove the termination of both functions. □

Exercise 4. Consider the following specification of a `min` function:

WhyML
```
let min2 (a b: int) : int
  ensures { result = a <-> a <= b }
= if a>=b then b
  else a
```

Produce a code snippet that uses `min2` according to its specification but contains an assertion on the result of the function that fails to verify, despite being *a true statement.* Fix the specification of `min2` to make the assertion go through. □

Exercise 5. Implement the simplest function possible that satisfies the given specification:

1. WhyML
```
let m1 (x y: int) : int
  requires { 0 < x < y }
  ensures  { result >= 0 && result < y && result <> x }
```

2. WhyML
```
let m2 (x: int) : int
  requires { x >= 0 }
  requires { x <= -1 }
  ensures  { result > x && result < x }
```

3. *WhyML*

```
let m3 (x y: int) : bool
  ensures { result -> x = y }
```

4. *WhyML*

```
let m4 (x y:int) : bool
  ensures { result <-> x = y }
```

□

Exercise 6. [⋆⋆⋆] Consider the following Why3 method:

WhyML

```
function square (n: int) : int = n * n

let q (n: int) : int
  requires { n >= 0 }
  ensures  { true }
= let ref count = 0 in
  let ref sum = 1 in
  while sum <= n do
    invariant { 0 <= count }
    invariant { square count <= n }
    invariant { sum = square (count + 1) }
    count <- count + 1;
    sum <- sum + 2 * count + 1;
  done;
  count
```

What is the strongest post condition for function `q`? □

2 Loop Invariants

Exercise 7. Consider the following Why3 implementation of Robert Floyd's example from the *Assigning Meaning to Programs* paper:

WhyML

```
let division (x y: int) : (q: int, r: int)
= let ref q = 0 in
  let ref r = x in
  while r >= y do
    r <- r - y;
    q <- q + 1
  done;
  (q, r)
```

a) Derive proper loop invariants.

b) Provide proper loop termination measures.

c) Specify the weakest pre-condition and strongest post-condition for this function.

□

Exercise 8. Consider the following Why3 implementation of Turing's famous *Checking a Large Routine* paper:

WhyML
```
let rec function fact (n: int) : int
  requires { n >= 0 }
  variant  { n }
= if n = 0 then 1
  else n * fact(n - 1)

let routine (n: int) : int
=
  let ref r = 0 in
  let ref u = 1 in

  while r < n do
    let ref s = 1 in
    let ref v = u in
    while s <= r do
      u <- u + v;
      s <- s + 1
    done;
    r <- r + 1
  done;
  u
```

a) Derive proper loop invariants.

b) Provide proper loop termination measures.

c) Specify the weakest pre-condition and strongest post-condition for this method.

d) Change the above implementation to an equivalent one, but using a `for`-loop.

□

Exercise 9. Specify and write an iterative function that computes the n-th Fibonacci number. Use the following recursive definition as a specification:

WhyML
```
let rec function fib (n: int) : int
  requires { n >= 0 }
  variant  { n }
= if n = 0 then 1
  else if n = 1 then 1
  else fib (n - 1) + fib (n - 2)
```

You will need to write a while loop with an appropriate loop invariant that establishes the post-condition. Also, prove the termination of your implementation. □

Exercise 10. Specify and write an iterative function that adds all the elements of a list of integers, according to the following:

WhyML
```
type list 'a = Nil | Cons 'a (list 'a)

let rec function add (l : list int) : int
```

```
= match l with
  | Nil -> 0
  | Cons x xs -> x + add xs
  end

let predicate is_nil (l:list 'a)
  ensures { result <-> l = Nil }
= match l with Nil -> true | Cons _ _ -> false end

let add_imp (l : list int) : int
  ensures { result = add l }
```

□