Handout 2
Verified Dynamic Programming

Software Verification

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

Version of October 26, 2025

The goal of this handout is to verify a Why3 program that implements various forms of computing Delannoy numbers, using dynamic programming. A good description of what are Delannoy numbers is given in the Wikipedia page: `https://en.wikipedia.org/wiki/Delannoy_number`.

During this handout you are supposed to complete the specification, implementation, and proof given in the companion `.mlw` file. This includes:

- pre-conditions (*i.e.*, `requires` clauses)
- post-conditions (*i.e.*, `ensures` clauses)
- termination measures (*i.e.*, `variant` clauses)
- loop invariants
- filling in pieces of code

The functions computing Delannoy numbers are already implemented. **Do not change such implementations. Only fill in the missing pieces of code, where asked**.

Your submission must contain the following items:

- The proposed skeleton file `dynamic.mlw` completed with your solution
- The content of the sub-directory `dynamic` generated by Why3. In particular, this directory **must contain** session files `why3session.xml` and `why3shapes.gz`.
- A PDF document named `report.pdf` in which you report on your work.

The report can be written either in Portuguese or English. The structure should follow the sections and the questions of this document. For each question, detail your approach, focusing in particular on the design choices that you made regarding the implementations and specifications. In particular, loop invariants that you added should be explained in your report: what they mean and how they help to complete the proof.

You can use the following structure to answer a question: *"For this function, I propose the following implementation: [give pseudo-code]. The contract of this function is [give a copy-paste of the contract]. It captures the fact that [rephrase the contract in natural language]. To prove this code correct, I need to add extra annotations [give the loop invariants, etc.] capturing that [rephrase the annotations in english]. This invariant is initially true because [explain]. It is preserved at each iteration because [explain]. The post-condition then follows because [explain]."*

The reader of your report should be convinced at each step that the contracts are the right ones, and should be able to understand why your program is correct, *e.g.*, why a loop invariant is initially true, why it is preserved, and why it suffices to prove the post-condition. It is legitimate to copy-paste parts of your Why3 code in the report, yet you should only copy the most relevant parts, not all of your code.

1 Delannoy Numbers

The Delannoy number $\mathcal{D}(n, m)$ is the number of possible paths in the integer grid $\mathbb{N}^2$ that go from the origin $(0, 0)$ to the point (n, m) by using any combination of elementary steps that are vertical (adding $(0, 1)$), diagonal (adding $(1, 1)$) or horizontal (adding $(1, 0)$).

By a simple case analysis on the last step, we see that a path to (n, m) is either a path to $(n − 1, m)$ followed by an horizontal step, a path to $(n−1, m−1)$ followed by a diagonal step, or a path to $(n, m−1)$ followed by a vertical step. This leads to the following recursive definition:

$$\mathcal{D}(n,m) \triangleq \begin{cases} 1 & \text{if} \quad n = 0 \quad \wedge \quad m \geq 0 \\ 1 & \text{if} \quad m = 0 \quad \wedge \quad n \geq 0 \\ \mathcal{D}(n-1,m) + \mathcal{D}(n-1,m-1) + \mathcal{D}(n,m-1) & \text{if} \quad n \geq 1 \quad \wedge \quad m \geq 1 \end{cases}$$

2 Recursive Specification

The given skeleton file `dynamic.mlw` provides a module `Delannoy_spec` for the mathematical definition of Delannoy numbers.

Exercise 1. Fill in the definition of the *let-function* d of module `Delannoy_spec`, that defines Delannoy numbers. Explain what you need to specify to make your definition accepted by Why3. □

Exercise 2. The *lemma-function* `d_is_positive` states that all Delannoy numbers are non-negative. Complete its proof. □

3 Computing Delannoy Numbers in a Matrix

The module `Delannoy_matrix` of the skeleton file proposes two versions of a program to compute Delannoy numbers in a matrix, using two nested loops. The code relies on the `Matrix` module from the Why3 standard library.

Exercise 3. Fill the holes in the logic annotations of the function `delannoy_matrix` so has to to prove the total correctness of this program. □

Exercise 4. Do the same with the alternative version `delannoy_matrix_alt`. Comment the differences with the first version, regarding the loop invariants in particular. □

4 Computing Delannoy Numbers in an Array

The module `Delannoy_array` proposes an implementation that further explores the power of dynamic programming. It avoids the use a matrix to compute $\mathcal{D}(n, m)$ by using only an array of size $m + 1$ as temporary storage.

Exercise 5. Fill in the code of the program `delannoy_dynamic`, by providing an appropriate definition for the local variable `y`. Explain how you came up with this expression. □

Exercise 6. Complete the missing logic annotations so as to prove total correctness of this program. □

5 Experimental Evaluation of Complexities

Using the proposed `Makefile`, you can run some tests using the command `‘make test_delannoy‘`. The initial recursive definition of `d` is tested, together with the second matrix implementation and the array-based implementation. For each test, its execution time is given inside square brackets. See the file `test_delannoy.ml` for the sequence of tests performed. Some tests may take too long time or exhaust your computer's memory, so you can comment them out.

Exercise 7. Explain, informally, the results of the tests, regarding the time and space complexities of the three implementations: the recursive one, the matrix one and the array one. □

Important note: in order to be able to run the OCaml extracted code from your Why3 proof, you must have the `zarith` library installed in your system. If that is not the case, please run the command ‘`opam install zarith`‘.

6 Conclusions

You should end your report with a (small) conclusion that summarizes your achievements and explain the issues you couldn’t solve if any.