

# CSCM985, Lab Dafny-2

To be ticked off:

- on **13.11.2025** or
- a week later, on **20.11.2025** at the beginning of the lab.

Please work in teams of two. There are Computer Instructions at the begin of this Lab Sheet.

- There are is on task with three sub-items.
- There is one challange task.
- All group participants need to be present to be ticked off.
- Check your marks on Lab-Tracker after having been ticked off, marks cannot be changed after.

## Learning Outcomes

- ◎ Understanding of Program Verificaion
- ◎ Coming out with Loop Invariants

## Required Resources

- ☒ Files: Task2.dfy, TimeMeasurement.java

## Where to Look Up Things

- 👁 Lecture Slides on Program Verification 7.11.
- 👁 Dafny Cheat Sheet: <https://dafny.org/latest/DafnyCheatsheet.pdf>

## Computer Instructions

- sign in to OneDrive in order to have your files synched to the cloud
- Under Teaching Software  
Faculty  
choose VScode [Do not choose Visual Studio!!!]  
start Visual Studio Code (it might be the case that it still needs to be installed)  
note: the window is often hidden
- In VScode choose extensions (left sidebar)  
enter Dafny  
select Dafny  
click install  
create a new file with extension .dfy, say hugo.dfy  
click into this file  
if there are migration instruction [likely], accept the update to version 4.6.0.0, and you are ready to go
- In case there is no migration pop-up  
click on Dafny in the extensions field  
click uninstall  
click install  
(possibly repeat this cycle)  
until you are offered restart extension  
click restart extension  
a pop-up upgrade version appears accept the upgrade to 4.6.0
- click on hugo.dfy  
check for Dafny 4.6.0.0 in the bottom right corner - then things are correctly installed
- Seeing Dafny in Action:
  - Download the file hello.dfy
  - Open it in VScode
  - Press F5: this executes hello.dfy
    - you should see “hello” and the number “34” on the screen
  - Change the assertion “assert 42 > 0;” to “assert 42 < 0;”
    - you see a right marker on the left of this line

## Task 1 - Proving Correctness for Loops with the Help of Invariants

### Introduction to the Task:

Program verification becomes undecidable the moment loops are involved. Only with some additional ‘help’ in form of so-called (loop) invariants we can expect Dafny to tell us if a contract is correct. We will explore this on the example of verifying different methods for computing the integer square root.

### Explanation - What Invariants are About

The word *invariant* means “not changing”. In the context of program verification, an invariant is a predicate concerning a loop. The invariant holds both, *before* the body of the loop is executed and *after* the body of the loop is executed: the validity of the invariant does not change.

In Dafny, we write invariants as follows:

```
while B
    invariant J
{
    Body
}
```

For such an annotation, the Dafny verifier tries to prove:

$$(B \&& J) ==> WP [[ Body, J ]]$$

When entering the loop, we know that  $B$  holds. Suppose further that also  $J$  holds. In order for  $J$  to be an invariant,  $(B \&& J)$  must imply the weakest precondition of  $Body$  with respect to  $J$ .

The *integer square root of a non-negative integer  $y$*  is defined to be the natural number  $x$  with  $x * x \leq y < (x + 1) * (x + 1)$ . We write

$$x = isqrt(y).$$

For example,  $isqrt(27) = 5$ , as  $5 * 5 \leq 27 < 6 * 6$ .

Another way to think about this is that we are first looking for the square root of a number  $y$ . This root is usually a real number. The number  $x$  is then  $y$  rounded down to the next natural number. In the example  $\sqrt{27} = 5.196\dots$ , we then round down to get  $x = 5$ . If we consider  $\sqrt{35} = 5.916\dots$ , we would again round down to get  $x = 5$ . This means, 27 and 35 both have 5 the integer square root 5.

## Explanation - How to Come Up with Invariants

Invariants need to be invented or found. They represent insight into the algorithm at hand. In the lectures we discussed two design principles for invariants.

Invariant design principle 1: Express relations (which value is larger than another one) as an invariant. The below invariant  $i - 1 < n$  is of this type, it relates the values of the variables  $i$  and  $n$  by saying which value is smaller than the other.

```
method count0_3 (n:int) returns (r: int)
    requires n >= 0;
{
    var i:int := 0;
    while (i < n)
        invariant i - 1 < n;
    {
        i:=i+1;
    }
    r := 0;
}
```

Invariant design principle 2: State the intermediate result computed in the  $i$ -th run of the loop. The below invariant  $\text{sum} == i * (i+1)/2$  is of this type, it uses the formula  $r == n*(n+1)/2$  stated after the keyword `ensures`. However, it is

- replacing variables  $r$  and  $n$  used to state the end result
- with variables  $\text{sum}$  and  $i$  used in the loop to compute an intermediate result.

```
method sum(n:int) returns(r:int)
    requires n >= 0
    ensures r == n*(n+1)/2
{
    var sum : int := 0;
    var i: int := 0;
    while (i < n)
        invariant sum == i * (i+1)/2;
    {
        i := i+1;
        sum := sum + i;
    }
    r := sum;
}
```

➊ Consider the method `SquareRootAscendingSearch` from the file `Task2.dfy`: Without an invariant, Dafny can't establish the contract. Follow the hints in the file and prove that the method is correct.

➋ Consider the method `SquareRootDescendingSearch` in the file `Task2.dfy`. Without an invariant, Dafny can't establish the contract. Follow the hints in the file and prove that the method is correct.

➤ Consider the method `SquareRootAscendingUsingAddition` in the file `Task2.dfy`. Astonishingly, Dafny can prove its contract without requiring an invariant. Thus, there is nothing for you to do :-)

➤ Consider the method `SquareRootBinarySearch` in the file `Task2.dfy`. Without an invariant, Dafny can't establish the contract. Follow the hints in the file and prove that the method is correct.

**Assessment:** The file `Task2.dfy` in Visual Studio with a green bar from top to bottom.

### Assessment Check List

- ✓ Three items to be ticked off.

## Challeng Task – Using Verified Dafny Methods in Java

**Introduction to the Task:** Now, that we have verified methods, let's try to call them from Java. Any code that we call is supposed to be correct w.r.t. its contract! [up to translation errors].

To do this in a nice way is possibly, but also a bit of work. Thus, we will use a shortcut here.

➊ After verifying everything in Task 2, we can compile things to Java. To this end:

➋ Choose under View the Command Pallet

➌ search dafny

select Dafny: Build with Custom Arguments

enter build –target:java –spill-translation

➍ Effect: this produces a folder and a .jar file

➎ ignore the .jar file

➏ in the folder you find produced code including the dafny runtime

➐ we are interested in `_System/_default.java`

➑ copy from `_default.java` the code for the first three methods into IntelliJ.

➒ Call call the first 3 methods from within the main method of the Main class file `TimeMeasurement.java`.

➓ Find a number for which there is a runtime difference for the 3 methods – there ought to be one, as the integer square root is closer to 0 than to  $y$ ; also, the algorithm with addition is supposed to be faster than the other two.