

Tutorial

Making a start in Dafny

Albert Nymeyer

11 July, 2022

Dafny version 3.7.1 runs on the network in the Department of Computer Science. Dafny is a combined verifier and compiler: there is code that looks much like the C-language, but amongst the code, there are also instructions that are based on formal logic. These instructions play a crucial role in software development.

You can check that the version of Dafny by typing in:

```
dafny \version
```

Another command you might like to use is:

```
dafny \help
```

but beware, there are lots of options. On the *Dafny* link on the course website there is a *Dafny Options* document that provides some useful command-line options in this course.

Exercise I: Say hello

1. To write your first Dafny program on the CSE network, use an editor to input the following program. Call it `hello.dfy`.

```
method Main()
{
    print "hello, world\n";
    assert 42 > 0;
}
```

Convention in Dafny is to use 3-space indentation by the way.

2. You can verify the program is correct by using the command:

```
dafny hello.dfy
```

This command is actually short for the command:

```
dafny /compile:1 hello.dfy
```

If you type in either command you should get the messages:

```
Dafny program verifier finished with 1 verified, 0 errors
Compiled assembly into hello.dll
```

The generated file *hello.dll* is a dynamic-link-library file. Also generated is another file, *hello.runtimeconfig.json*. These files can be combined with other *.dll* files to build an application in some target language. We do not build applications in this course so these files can be thrown away.

You can avoid generating the *.dll* and *.json* files altogether by using the command:

```
dafny /compile:0 hello.dfy
```

3. Alternatively, you can verify AND compile and execute the program. Observing that *hello.dfy* contains a *print* statement this would seem desirable.

Use the command:

```
dafny /compile:3 hello.dfy
```

which results in the messages:

```
Dafny program verifier finished with 1 verified, 0 errors
Running ...
hello, world
```

Notice that the program has verified successfully and has executed, resulting in the *hello, world* message. Dafny has not report that the program compiled successfully, but it clearly must have. Note that this option still generates *.dll* and *.json* files.

Exercise II: What's in a name?

4. Use the editor to change the name of the method in the file *hello.dfy* from *Main* to *Spiderman* (or any other name that takes your fancy). By the way, it is convention in Dafny that the first letter in a method name is upper-case.

5. Verify the program using the option */compile:0* say. This should result in:

```
Dafny program verifier finished with 1 verified, 0 errors
```

6. Now try to verify, compile and execute the program using the */compile:3* option. It should result in the messages:

```
Dafny program verifier finished with 1 verified, 0 errors
Compiled assembly into hello.dll
Program compiled successfully
```

Notice Dafny has told you that the program has compiled, but the program has NOT executed. (The message *Running...* is missing and there is no message *hello, world*.)

7. The reason it does not execute is the runtime system looks for a method called *Main* to start execution. There is no method of this name, so there is no execution.

The lesson here is that you must have a method *Main* for execution to occur.

Exercise III: Make a bad assertion

The integer 42 is greater than 0, so the assertion in *hello.dfy* is obviously true. What happens if the assertion is false?

8. Edit the file *hello.dfy* and change the predicate in the *assert* statement to something false, say $42 < 0$.

9. We could verify the program by doing:

```
dafny /compile:3 hello.dfy
```

but using this option is pointless because there is no method *Main* so whether it verifies or not, it cannot execute. All we need to do is:

```
dafny hello.dfy
```

It should generate the messages:

```
hello.dfy(4,13): Error: assertion might not hold
Dafny program verifier finished with 0 verified, 1 error
```

10. Dafny has failed to verify the program. It has generated an error at position line 4, character 13, and then halted. This position is the space before the less-than sign, and is the position at which Dafny (actually the theorem prover in Dafny) realises it cannot prove the assertion.

You can override this (force it to continue with compilation) using command-line arguments but we NEVER override the verifier in this course.

The lesson here is that verification must succeed before compilation can occur.

Exercise IV: Can an assertion be *almost always* true?

The predicate $42 < 0$ is obviously always false, so clearly this as an error, and Dafny should report this error.

Consider the following assertion. If n is an integer, then $n^2 > 0$ is true. Actually, if $n = 0$, this assertion is false, but it is true for every other value of n . So it is true for an infinite number of values of n from $-\infty$ to $+\infty$. How does Dafny handle this kind of assertion?

11. Create a new file `square.dfy` consisting of the following code.

```
method SquareMe(n:int)
{
    assert n*n > 0;
}
```

The method *SquareMe* has an integer parameter n , and asserts that $n^2 > 0$.

12. Verify the program using:

```
dafny square.dfy
```

It should generate the messages:

```
square.dfy(3,13): Error: assertion might not hold
Dafny program verifier finished with 0 verified, 1 error
```

13. The program has again failed to verify. There's just one value that makes the assertion false, but that is enough for verification to fail.

The lesson is the verifier must prove an assertion is true for ALL possible input.

Exercise V: Bring it home

We can fix `square.dfy` by excluding the $n = 0$ case from the assertion. The correct assertion is *for any integer n , if $n \neq 0$ then n^2 is positive*. In Dafny, logical implication is represented rather inelegantly by `==>`, so we express the assertion as `n!=0 ==> n*n>0`.

14. Edit the program `square.dfy` to make this assertion, and verify. It should work.

Epilogue: A perspective

Some general comments:

Ordinary Programming We started in Exercise I with a Dafny program `hello.dfy` that prints *hello, world*. Printing is used a lot in conventional languages like Python, Java and C. In normal, ordinary software development, printing is often used as debug, and to build confidence.

Dafny programming In Dafny we are mainly (only) interested in verification. A Dafny program usually doesn't contain output statements, and it is not necessary to execute programs. As we saw in the exercises above, Dafny finds errors during verification, which happens before the program is even compiled, let alone executed. A *print* statement is often pointless (Dafny won't get that far if there is an error). Also, a *print* statement generally will not reveal where an error in code and/or logic is.

Specifications Dafny is all about assertions, where an *assert* statement is just one kind of assertion. The more common form of assertion is a *specification*. A specification consists of pre- and post-conditions, which define what the program should do. These specifications allow a user to guarantee correctness.

Visual Studio Dafny

The exercises above used the CSE network Dafny verifier/compiler. For those of you that have a strong urge to work on your own machine, and want the support of an IDE, it is also possible to download a plug-in for Visual Studio. I stress that you do not need to do this in this course: *CSE dafny* is adequate (and all I use).

In the Visual Studio version, the IDE runs continuously in the background. The error messages it provides are more informative than *CSE dafny*. See <https://marketplace.visualstudio.com/items?itemName=correctnessLab.dafny-vscode> for instructions on the download process. The release notes can be found at <https://github.com/dafny-lang/dafny/releases>. If you wish to see some nifty animations of *Visual Studio dafny*, check out the website <https://correctness-lab.ch/projects/dafny-vscode>. Remember though:

- there are differences between *Visual Studio dafny* and *CSE dafny*, so before you submit code in the course, you MUST check that your code runs warning-free and error-free with *CSE dafny*
- all assessment is carried out using *CSE Dafny*
- in the final exam, you may have to use *CSE dafny*