**Table of Contents**

1. **Introduction ............ ................................................... 2  
     1.1 Why Verification Matters ..... ........................................ 3  
     1.2 Problem ......................... ............................................ 3**
2. **Dafny ....................................... 4  
     2.1 Purpose of This Paper ............ ..................................... 4  
     2.2 Dafny Features ..................... ........................................ 4  
       2.2.1 Examples .................. ......................................... 5  
     2.3 Main Idea & Details ............. ........................................ 6  
     2.4 Syntax and Language Constructs ... 6  
       2.4.1 requires and ensures .... ....................................... 6  
       2.4.2 assert and invariant ... ........................................ 6  
       2.4.3 Z3-Integration ............ ....................................... 6  
     2.5 Own Experience ................... ...................................... 7  
     2.6 Literature Critique ................. ..................................... 7  
       2.6.1 Especially straightforward  
       to understand ..................... ..................................... 7  
       2.6.2 Difficult to understand ..... ..................................... 8  
     2.7 Future Work & Conclusion ...... ..................................... 8  
     2.8 Relevance of Dafny in Modern  
       Development .............. ............................................ 9**
3. **Sources .............................. ....................................... 10**
4. **Introduction**

Software is everywhere today. Whether in smartphones, cars or ATMs, it ensures that everything runs smoothly. This makes reliable software even more important. Not only can errors be annoying, but they can also be dangerous in safety-critical areas. Usually, errors are identified through testing, whereby the programme is executed with selected inputs and checked to ensure that everything works as expected. However, tests have their limitations: they only check certain scenarios and never cover all possible inputs. Therefore, a programme can pass all tests and still malfunction in other cases. This is where formal verification come into play. They enable us to prove programmes mathematically, in the same way that you prove a theorem in mathematics. Rather than testing a few examples, they ensure that the programme always works correctly. No matter what inputs are chosen.

Although this may sound complicated at first, modern tools such as Dafny make the whole process much more accessible. Dafny is a programming language that enables you to add conditions such as pre- and post-conditions, loop invariants, and termination checks directly to programmes. This enables you to verify the correctness of a program while writing it, without executing it first. This enables many errors to be identified and resolved at an early stage, before they can cause significant issues further down the line. This can make a big difference, especially in safety-critical systems. Unfortunately, real-life examples repeatedly demonstrate that software errors are not just theoretical. For example, Tesla recently had to recall around two million vehicles due to software errors [1] in the Autopilot system. The system failed to recognise certain obstacles correctly, which could have led to accidents in the worst case. Another example is NASA. A memory leak caused the Mars Rover Spirit's system to crash repeatedly [2]. This problem was difficult to identify and nearly resulted in the mission failing.

* 1. **Why Verification Matters**

Such cases demonstrate the importance of not only testing programmes, but also mathematically verifying their correctness from the outset. A test may not have identified the Tesla errors because it only covers a small number of possible scenarios. Formal Verification, on the other hand, can ensure that the programme works correctly in all conceivable cases. This gives developers and, above all, the people who must rely on these systems more security.

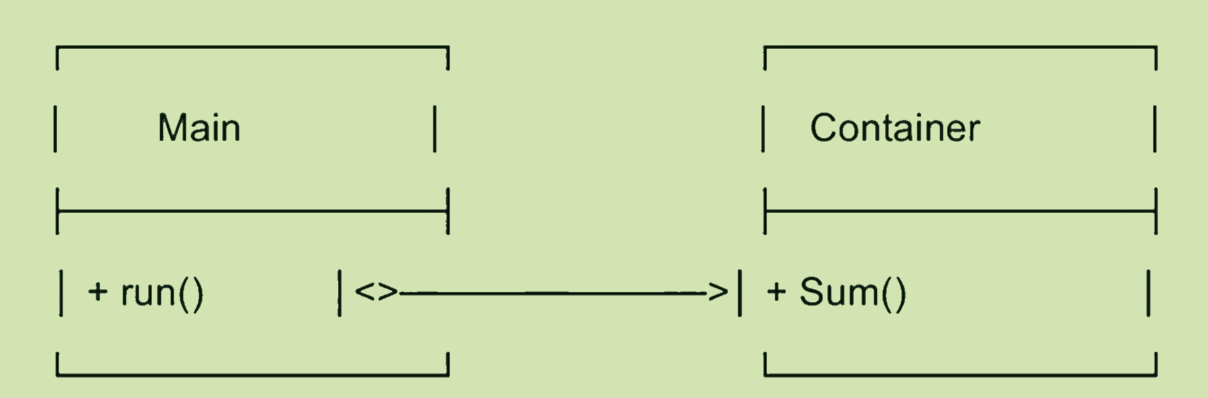
* 1. **Problem**

In practice, creating software that is free from errors is one of the biggest challenges. Even when writing code, developers must ensure that no errors occur that will be difficult to identify later on. This is particularly important in safety-critical systems, where software can mean the difference between life and death — errors here are not just annoying, they can have serious consequences. Although tests and code reviews can help detect errors, they are often insufficient to guarantee a programme's correctness in all situations.

This is precisely where formal verification comes in. The goal is to develop programmes in such a way at the writing stage that their correctness can be mathematically proven. This ensures that programmes function correctly for all possible inputs. This is where Dafny comes into play. Dafny is a modern tool specially developed for programme verification. It enables developers to write and prove the correctness of their programmes simultaneously. This means that software is checked for errors before execution. This enormously facilitates the development of reliable software and ensures that you won't have to worry about difficult-to-trace bugs later on.

1. **Dafny**
   1. Purpose of This Paper

This paper examines how Dafny helps guarantee the correctness of programmes during the writing stage. The following figure shows a class diagram to illustrate the functionality of the example programme (under DAFNY/code):



**Description:**

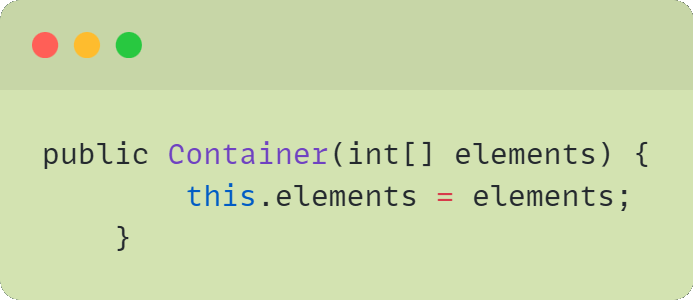
* Main calls Container.Sum().
* Container is a class that has a method.
* Sum() is a method that adds a sequence.

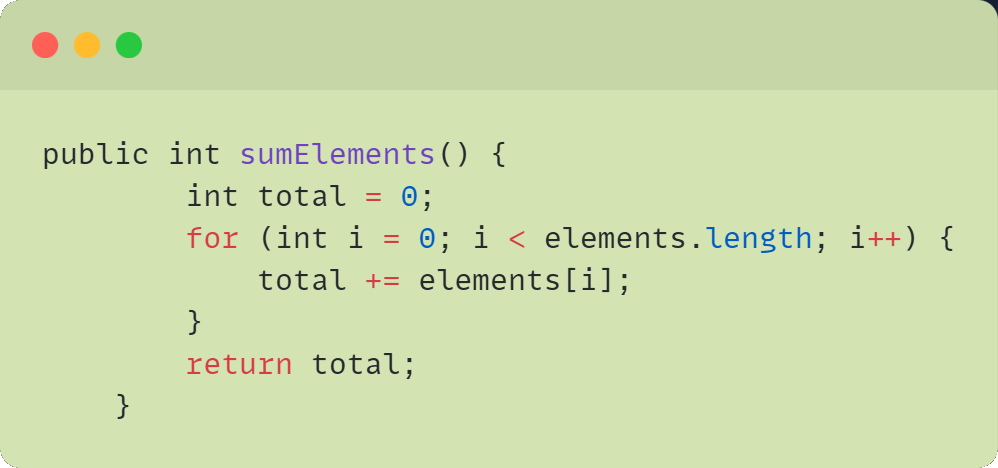
To tackle these issues, the following section introduces Dafny’s features.

* 1. **Dafny Features**

Dafny enhances programme development by providing practical functions that ensure programmes are correct as they are written. These features facilitate mathematical proof of programmes and the early detection of potential errors:

* Pre- und Postconditions:  
  They define the conditions that must be met before and after a function is executed. This precisely defines what a function can do and what it is guaranteed to return (1).
* Loop Invariants:  
  They describe a condition that must be fulfilled at the beginning and after each loop pass. This can be used to mathematically prove that the loop always works correctly (2).
* Termination Checking:  
  This feature checks that the programme or loop will end at some point and will not get stuck in an infinite loop (3).
* SMT-Solver (Z3):  
  A tool that is both powerful and automatic, which checks compliance with all specified conditions and thereby supports the verification of the code (4).
  + 1. **Examples**

Z1:

Z2:

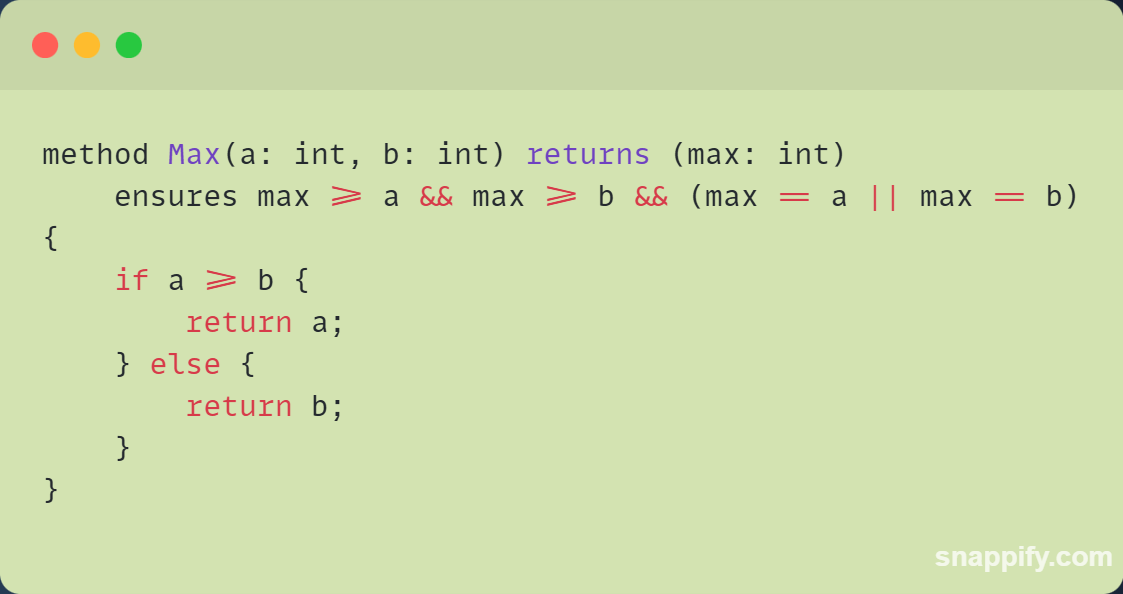
Z3:

* 1. **Main Idea & Details**

To better understand the practical application of Dafny, the central language constructs are presented below through a simple example. The ability to formulate mathematical conditions directly in the code and have their correctness checked automatically is particularly important here.

* 1. **Syntax and Language Constructs (Code as an Example)**

Dafny is based on the familiar syntax patterns of modern programming languages, supplementing them with special keywords for specifications. For instance, it is possible to write a function that calculates the maximum of two numbers:



This example shows how a postcondition (ensures) can be defined directly in the code so that verification takes place when the code is written.

* + 1. **requires and ensures**

'Requires' describes the preconditions that must be met before a function can be called, while 'ensures' defines the conditions that should apply after the function has been executed. Together, these two keywords can be used to precisely define and verify the desired behaviour of a function.

* + 1. **assert und invariant:**

The keyword 'assert' is used to formulate conditions that must always apply at any point in the code. If a condition is not met, the verifier reports an error. The invariant keyword can be used to define loop conditions that must be met before, during and after each iteration. This ensures that loops do not reach any undesired states.

* + 1. **Z3-Integration**

Dafny uses the Z3 SMT solver in the background to automatically verify defined conditions. This enables complex conditions to be checked quickly and reliably, eliminating the need for developers to perform each proof manually.

* 1. **Own Experience**

We have not yet gained any experience with Dafny, so we cannot report any hurdles or challenges at this time. This section is a placeholder and will be updated in my next submission, once we have worked with Dafny in practice.

* 1. **Literature Critique**

This literature review examines the paper 'An Automatic Program Verifier for Functional Correctness' by K. Rustan M. Leino (Microsoft Research, 2010). Many of the statements in this paper are based on the results and findings of the paper, which provides a comprehensive description of Dafny and explains it using case studies. Although other papers are available, we found this one to be the most appropriate, as it provides a good balance of theory and practice and covers many basic Dafny features in detail.

Literature criticism is important for understanding the strengths and weaknesses of a paper, and for assessing its relevance to your own work and applications. Structured assessments help you recognise gaps in your own research and provide an objective overview of the literature used.

* + 1. **Especially straightforward to understand**

Overview of Dafny features: This paper provides an accessible explanation of key concepts such as pre- and postconditions, loop invariants, and termination checking, offering beginners a comprehensive overview.

* Integration of the Z3 SMT solver: This demonstrates the functionality of the SMT solver in a practical way, making it easier to understand how Dafny automatically verifies mathematical conditions.
* Case studies with code and metrics: A detailed analysis of case studies, including lines of code and proof obligations, illustrates the practical use of Dafny.

**2.6.2 Difficult to understand:**

* Dynamic Frames: This concept was only briefly mentioned and was not explained in a way that beginners could easily understand.
* Proof strategies: The presentation of the lemmas and invariants sometimes required a higher level of formal logic, which may pose a challenge for less experienced readers.

So far, we have not found a better online source that caters for both beginners and advanced users so well. One probable reason for this is that other papers are often either too theoretical or too specialised; this paper, however, offers a good overview and provides practical examples. Additionally, Dafny has only been used more extensively worldwide since around 2010, which explains why there are not yet many accessible papers on it.

* 1. **Future Work & Conclusion**

This paper has demonstrated the value of using Dafny as a modern verification tool alongside classic test. This is particularly important in safety-critical systems, where errors can have serious consequences. Dafny makes it possible to detect errors at an early stage during the writing process. It is not only useful for small projects but also offers students a practical introduction to formal verification.

Looking to the future, the focus should be on integrating Dafny more closely into larger teams and agile development processes. Coordinating code reviews and tests is becoming increasingly complex, particularly in projects involving many developers working in parallel. With its formal specifications, Dafny can provide a reliable foundation that has often been lacking until now. Currently, using Dafny in large teams is still challenging, as many companies prioritise quick results and are reluctant to allocate additional resources to formal verification. The fact that the costs of errors in operation or safety-critical applications are significantly higher than the investment in thorough verification is often overlooked.

* 1. **Relevance of Dafny in Modern Development**

Dafny is particularly relevant in an age where artificial intelligence is an integral part of our everyday lives. Nowadays, AI-powered systems make decisions that can have a significant impact. This is precisely why it is important that the code is error-free, and the results are reliable. Dafny can help to ensure the correctness of such systems, whether they are used for industrial projects, critical systems or even software for space missions.

At the same time, bear in mind that Dafny consumes resources and is not necessary for every project. Not all software is safety critical. Therefore, it is up to each team to decide whether a project is critical enough to warrant the use of Dafny. This is precisely when Dafny should be used.

Another exciting prospect would be to integrate Dafny more closely into modern DevOps environments, such as Docker, to help avoid errors in databases or complex systems. The challenge remains in convincing companies, large teams, and the community that the initial extra effort is an investment in quality and security.

In today's climate, where many companies are focusing on quick profits and cost reduction, the fact that a later error can be more expensive than solid verification is often overlooked.

This is an area in which Dafny could bridge the gap between modern development processes and secure software a goal that the community should work towards together.

1. **Sources:**
2. See *Die Presse*, “Problems with Autopilot: Tesla recalls two million cars,” 2025.
3. See NASA Mars Exploration Rover Spirit – Technical Report (NASA JPL), 2004.