# Translating Pedagogical Verification Exercises to Viper

Benjamin Frei
Supervised by Prof. Dr. Peter Müller, Nicolas Klose and João Pereira
ETH Zürich
Zürich, Switzerland
March 2023

## 1 Introduction

Today's world wouldn't be imaginable without software. There is an enormous amount of software being designed, written, and implemented for very different and very specific systems. In many applications, it would be valuable to be able to guarantee that such programs will behave as they are supposed to.

Rather than relying solely on the testing of a program, with the help of verification languages such as Viper and Dafny [1] [2], we are able to prove the correctness and termination of such a program with respect to its formal specification. In order to ensure the correctness of such software, Viper uses multiple concepts to allow a user to write verified code. In general, Viper builds on separation logic, which is an extension of Hoare logic, designed to reason about heap manipulation and aliasing. This can give us a guarantee that such a program behaves according to its specification, ruling out any unexpected program behaviours.

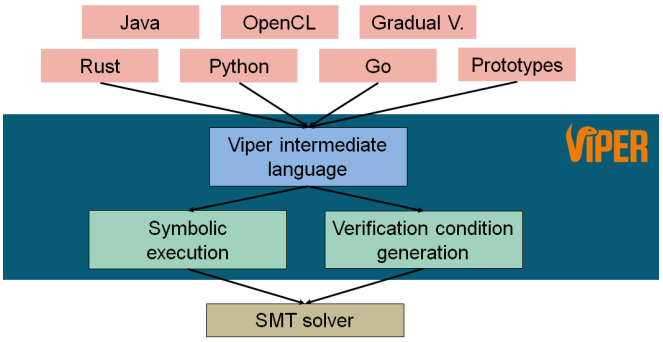


Figure 1: This figure shows how Viper is used to verify code written in other languages, by translating programs in these languages to a Viper program. [3]

As we see in Figure 1, Viper consists of the Viper intermediate language, the symbolic execution backend, as well as of the Verification Condition generation

backend. There are frontends that transform code written in other languages into Viper (more on this in section 3.3). After this transformation, they can use the Viper infrastructure to prove the correctness of the translated program, giving them the certainty that their code is correct.

Program verification is hard, and it is very helpful to be able to look at and analyse code. The motivation for this project is to create such pedagogical material that introduces Viper to programmers without previous experience in program verification. The book Program Proofs will be used as a source of examples and exercises for pedagogical material in Viper. This book is aimed at developers without previous experience in program verification. To that end, the user is gradually introduced to the Dafny programming language, a verification-oriented language that includes features such as functional contracts and ghost code. To obtain new pedagogical material for Viper, we will translate the examples and exercises from this book and we will document the most important differences between the Dafny and Viper versions in a way that is easy to understand to beginners of both languages. This gives them an intuition for most concepts provided by a verification language, helping them in verifying their programs.

# 2 Background

## 2.1 The Book: Program Proofs

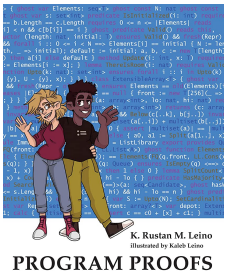


Figure 2: This figure shows the cover of the Book Program Proofs [4]

An important resource for this thesis will be the book Program Proofs. It will provide us with the code examples in Dafny that will be translated to Viper. The book is aimed at university students and can also be used as an introduction for industrial software engineers, with at least a bit of prior knowledge in programming and program verification. It uses the verification language Dafny and is designed to give readers a way to get familiar with how to write proofs that connect specifications and programs. The book was written by K. Rustan M. Leino and released on March 7th, 2023. The book is not meant for complete beginners in programming, but rather for people who would like to get started with automated program verification concepts.

## 2.2 Dafny

The book, from which the code examples and code exercises will be translated, is based on the verification-aware programming language named Dafny developed and maintained by Microsoft research.

As you write your code with Dafny, it constantly verifies whether the specification you set are met by the code you are writing. Once the code you wanted to write has been written in Dafny, the language lets you compile the code into other languages like C#, Go, Python, Java, or JavaScript. Dafny itself supports multiple concepts seen in imperative languages like loops and also supports concepts seen more often in functional languages such as inductive data types, as well as recursive functions. Dafny also offers tools used to create mathematical proofs, such as unbounded and bounded quantifiers, calculation proofs, pre-condition and post-conditions, termination conditions, loop invariants, and read/write specifications.

Dafny proves by default that all programs terminate. Meaning, it will always check for the termination of loops and recursive functions.

```
1 method Triple(x: int) returns (r: int)
2   requires x % 2 == 0
3   ensures r == 3 * x
4 {
5   var y := x / 2;
6   r := 6 * y;
7 }
```

Figure 3: This figure shows an example code written in Dafny. It is taken from the book Program Proofs. [4]

In Figure 3, we see a method written in Dafny. It takes an integer as input and returns an integer. The output is three times the input. This is ensured by Dafny with the pre-condition x%2 == 0 and post-condition r == 3 * x. With those pre- and post-conditions, Dafny guarantees the correctness of the code regarding the specifications given in those conditions. There is a lot of material available on Dafny [1] [4], which makes it a good language for people who are just beginning with program verification.

## 2.3 Viper

There are many verifiers that are built on top of Viper, such as Gobra for Go, Nagini for Python, or Prusti for Rust. Those frontends are tools that allow developers to automatically create Viper encodings of their code. Those descriptions will then be analysed by Viper and its formal verification tools. This helps to ensure that the code behaves correctly according to its specifications. The intermediate verification language Viper was developed by ETH Zürich. It is currently used for educational purposes by a number of different institutions, such as Brown University, Columbia University, etc.

One of Viper's key purposes is its focus on correctness and security.

```
1 method Triple(x : Int) returns (r : Int)
2     requires x % 2 == 0
3     ensures r == 3 * x
4 {
5     var y : Int := x / 2
6     r := 6 * y
7 }
```

Figure 4: This figure shows an example code written in Viper.

The code shown in Figure 4 is an example code written in Viper with the intention of showing how Viper code looks. Just like in Dafny, we can also use pre-conditions and post-conditions to prove correctness. Unlike Dafny however, Viper does not automatically check for termination. To do this, the programmer must annotate a decreases condition to the method.

# 3 Core Goals

The thesis will be mainly based on the book Program Proofs. The book teaches program verification concepts in the setting of an actual programming language that is designed for verification.

## 3.1 Translating Pedagogical Verification code examples to Viper

The main core goal of this thesis is to analyse and translate the code snippets, examples and exercises from the book Program Proofs. The examples are written in the language Dafny and the goal is to translate them into Viper. Since the examples are meant to be used for pedagogical reasons, the implementation in Viper will be carefully documented. The main focus will be put onto part 1 of the book, including chapters 6 to 12, excluding chapter 9 Abstraction. Chapter 9 is excluded since Viper does not have any kind of support for modules.

## 3.2 Identifying examples that are difficult to implement in Viper

Another core goal is to identify code examples that are not directly implementable in Viper due to a lack of features Viper might provide. We then reason about those examples, why they might not be implementable, and what features Viper could additionally provide to make those examples easier.

### 3.3 Comparing the annotation overhead and verification time between Viper and Dafny

The third core goal for this thesis is to analyse the annotation overhead of the two languages, Viper and Dafny compared to each other. The verification time will also be compared. We also analyse what causes the differences in annotation overhead and verification time.

## 4 Extension goals

After the core goals have been met, one or more of the extended goals below could be targeted in the thesis.

### 4.1 Presenting the code examples to the reader

An extension goal is to work on a way to present the code examples to the reader in a comfortable way. One possible way of making such a presentation would be, to create a web page similar to the one created for the Viper Tutorial. [5] This would give someone an easy way to interact with the code examples, without having to install Viper itself.

### 4.2 Translating more chapters

A possible extension goal is, in addition to the already translated chapters, to translate more of the chapters from part 2. This would include chapters 13 to chapter 17 of the book.

### 4.3 Identify relevant features missing in Viper and implement them

Once we started translating code examples, we will see which features are missing in Viper and make it difficult to implement certain code constructs in Viper. Another possible extension goal is to implement some of those missing features and concepts in Viper.

## References


[1] The Dafny Programming and Verification Language, Available: link.

[2] Programming Methodology Group ETH Zürich, Viper Programming Language, Available: link.

[3] Programming Methodology Group ETH Zürich, ETH Viper Webpage Image, Available: link.

[4] K Rustan M Leino. *Program Proofs*. MIT Press, 2023.

[5] Programming Methodology Group ETH Zürich, Viper Tutorial, Available: link.