Deductive Verification of Imperative Graph Algorithms

Project Description

ETH Zürich

Gishor Sivanrupan
Supervisor: Arshavir Ter-Gabrielyan

1 Introduction

This project focuses on verifying graph algorithms using the Viper infrastructure. Viper is a set of tools for verifying properties of programs written in an intermediate verification language called the Viper language.

In order to verify graph algorithms, one should be able to specify graph properties. However, there is no direct possibility to describe higher-order graph properties in the Viper language. One important property is reachability. Reachability could be simulated in first-order logic but it requires some expertise as it is tedious and error-prone to simulate it. Simulation requires manually writing axioms, which is dangerous, as mistakes in an axiom may cause unsoundness. Hence, it is not only cumbersome, but also dangerous to verify graph algorithms deductively without enough expertise.

The goal of this project is to design a language extension of the Viper language and a corresponding front-end that allows to encode topological and memory safety properties of graph-manipulating programs without losing the amount of automation Viper has. Effectively, one should be able to write graph algorithms, specify higher-order graph properties and verify them without the need of encoding the graph axioms and rewriting graph properties as constraints in first-order logic.

To structure the project, we have split it into different modules and classified the prerequisite modules as core goals and the others as extensional goals.

2 Core Goals

The main idea of this project is to implement a front-end for Viper that applies existing expertise for automatic verification of imperative graph algorithms. The core modules are:

1. **Learning**

The starting phase consists of learning to use Viper and understanding how the Viper works. Additionally, we need to learn about the existing expertise in automated verification of graph algorithms.

2. **Design the front-end architecture**

   Afterwards, we will choose and examine key examples from an existing pool of graph algorithms. This way, we will elaborate the user requirements for the front-end language. Subsequently, we will design a language extension of the Viper language and a corresponding front-end such that these requirements are fulfilled. The front-end should meet the following requirements:

   - **soundness**: This requirement is crucial, because with an unsound verifier, we cannot prove the absence of errors in a program.
   - **error backtracking/ usability**: The front-end should be able to show where the point of failure in the front-end code was, if Viper could not verify.
   - **extensibility**: It should be possible to extend it with new features without changing the whole project.
   - **testability**: It should be possible to extensively test it to eliminate major bugs.

3. **Implementation and testing**

   We will implement the designed front-end from the previous step and test it extensively. As this module takes a lot of time, we have partitioned into two smaller submodules, namely:

   - **Minimal implementation and testing** First, we will only implement the must-have features of our front-end and test them.
   - **Full implementation and testing** Afterwards, we will implement and test the advanced features that are, however, not a prerequisite of the extensional goals.

3 Extensional Goals

After having completed the core goals, we can focus on extending the front-end to push its limitations forward. For that, we need to do the following:

1. **Encode and verify new examples in the front-end language**

   To see the limitations of our front-end, we will examine new examples of graph algorithms and we will attempt to encode and verify them in the front-end language.

2. **Obtain new expertise for the cases that cannot be handled**

   We will further examine the examples that are not possible to be verified in the front-end language. If we are able to find solutions for these problems, we will extend the front-end to support them.

3. **Comparative Analysis**

   Finally, we will compare the capabilities of our front-end with the capabilities of other tools.

1 Schedule

The project starts on 12 March 2018 and lasts six months. In the following, the modules of the project and their corresponding expected duration and due dates are listed:

Module	**Number of Weeks**	**Deadline**
Learning	2.5	29 March 2018
Design the front-end architecture	2	12 April 2018
Initial presentation	1	19 April 2018
Minimal implementation and testing	4	17 May 2018
Full implementation and testing	4	14 June 2018
Extensional goals	6	26 July 2018
Evaluation	4	23 August 2018
Writing thesis	2	6 September 2018
Final presentation	1	12 September 2018