Automatic Inference of Quantified Permissions by Abstract Interpretation

Master Thesis Project Description

Seraiah Walter

Supervisors: Alexander J. Summers, Caterina Urban
ETH Zürich

23.03.2016

1. INTRODUCTION

Software Verification is a research area in which the software engineering community has been working for a couple of decades. The goal of proving a program to be correct is probably the dream of many software companies. Sadly, this does not come for free and researchers still have to solve some challenges. Nevertheless, the introduction of separation logic and other permission-based logics recently led to a new boost in the development of verification technologies. These logics make it easier to reason about parts of the memory independently. On the other hand, they introduce the need of specifying permissions to those memory locations, and this can be a burdensome overhead. A second big influence on verification research was the introduction of intermediate infrastructures such as Boogie and Why. They made it possible for researchers and developers to focus their work on a single level of abstraction, without having to consider the whole pipeline from high-level source code, over abstract representations, to eventually solving decision problems for logical formulas using SMT-solvers.

This is where Viper comes into play. It offers such an infrastructure for permission-based logics such as separation logic, which facilitate reasoning for modern-day concurrent programs. Viper is structured into different parts. At the front-end it allows for the development of compilers from any programming language to Vipers intermediate language Silver. At the back-end it allows for the development of different efficient verifiers. In between it allows for analyzing the program through abstract interpretation. This way of reasoning, which is orthogonal to what the verifiers do, greatly enriches the possibilities of the infrastructure. For example new specifications can be inferred or existing ones can be strengthened. This dimension is what we want to extend in our project.

2. CORE GOALS

The introduction of permissions significantly increases annotation overhead. Our goal is to reduce such overhead in order to make program verification more practical. A program is only allowed to read or write to a memory location if it has sufficient permissions at the corresponding program point. Hence, in Viper a method has to be annotated with permissions to all memory locations that are access inside the method. Our goal is to automatically infer these permissions.

Inferring permissions gets more complicated if we consider complex data structures in which case we not only have to handle single memory locations, but also sets of memory locations and these sets might potentially be unbounded. In Viper one has the possibility to define permissions by quantifying over these sets. A precondition which requires access to an array from index 0 to 10 would then look like this:

```
forall q:Int :: q>=0 && q<=10 ==> acc(array[q].val)
```

We will follow this pattern and try to infer those permissions through the help of abstract interpretation. In particular we will address the following core goals.

- **Arrays**: In a first phase, we are going to analyze simple arrays. The challenge here is to track all the indices at which the array is accessed.
  We try to achieve this by a backward "collecting" analysis where we collect all the indices and then combine it with the results from a forward abstract interpretation analysis of the variables. We think that this will give us precise enough information about which locations of the array get accessed.
- **Sets, Trees and Graphs**: In the next phase, we also consider more complex data structures. Here the way we access a location might, instead of an integer index, also be a reference or any other domain. A challenge we have to face is how to identify these data structures.
  We first develop a pattern for each data structure and then try to map the data structure at hand with one of our patterns. If possible we want to generalize our patterns to actually handle any kind of data structure.
- **Soundness vs. Precision**: Our analysis has to be sound. If not we might miss some locations to which we need access and the verifier will complain. But our main goal is to stay as precise as possible. This will lead us to require only the minimum amount of permissions needed. Abstract interpretations usually rely on operations such as widening which lose precision. Therefore we plan to combine abstract interpretation with an algorithm which does not lose precision, for example with weakest precondition rules, in order to keep as much precision as possible.
- **Implementation**: After we have the theoretical foundations, we will take it one step further and implement it in Sample to test how well our approach works in practice.
- **Evaluation**: We are going to look for, encode and verify challenging program examples and explore the possibilities and limitations of our approach. We can test the precision of the approach by comparing existing specifications with the inferred ones.

3. POSSIBLE EXTENSIONS

There are various ways to extend the project. All of them will help us to improve our analysis.

- **Heap-Dependent Accesses:** Following is a simple example where the index of the next array-access depends on the array itself:

  ```
  int index    = array[0];
  array[index] = 10;
  ```

  We see that the analysis gets more complicated since we now also have to start reasoning about what values the data structure itself contains.
- **Fractional-Permissions**: We adapt our analysis to also handle fractional permission, i.e. distinguish between read and write accesses. Hence we not only need to track the access itself but also the corresponding amount.
- **Functional Specification**: Either we extend our analysis or introduce a new one to infer also functional specifications, like the sortedness of an array.
- **Nested Quantifiers**: This is currently not supported in Viper, but we still try to explore the impact of allowing nested quantifications over permissions on our analysis.