Towards Customizability of a Symbolic-Execution-Based Program Verifier

Bachelor Project Description

Robin Sierra
Supervisors: Malte Schwerhoff, Vytautas Astrauskas, Peter Müller
ETH Zürich, Switzerland

24.3.2017

1 Motivation

Viper [1] is a verification infrastructure that is being developed at ETH Zurich that provides tools for encoding and automatically reasoning about properties of a wide variety of concurrent and heap-manipulating programs. A core concept of Viper is the notion of *permissions* to control access to *resources* such as fields (e.g. `this.val`), recursive predicates and magic wands, all of which can be used to control access to complex data structures and enforce an orderly modification thereof.

To achieve automated verification of the aforementioned properties, Viper includes Silicon [2], a verifier that is based on a technique called *symbolic execution*. Silicon internally manages the resources as sets of *chunks* that contain the permission to read/write the represented resource and a symbolic value, e.g. for the value of a heap location.

For each resource, Silicon defines its own rules for handling chunks. For example, permissions to fields are represented as values between 0 and 1, where 0 denotes no permission, 1 denotes full read/write permission and a value in between is a read permission. Consider two fields `x.f` and `y.f` with respective permission values p_x and p_y. If one knows that $p_x + p_y > 1$, one can conclude that $\mathtt{x} \neq \mathtt{y}$. For predicates, however, this is not generally true since there is no general upper bound for permissions to predicates. There also exist rules that are the same for both fields and predicates. Consider the locations denoted by `x.f` and `y.f` whose symbolic heap values are v and w. From learning that $\mathtt{x} = \mathtt{y}$ it follows that the heap location must be the same as well, i.e. $v = w$. The same is true for predicates: if one has predicate instances $pred(x)$ and $pred(y)$ and $\mathtt{x} = \mathtt{y}$, the heap values the predicate instances abstract over must also be equal.

Occasionally, one might want to change or adapt the rules for handling chunks or extend Silicon with new resources. This, however, is currently difficult to do, since each chunk and its handling is hard-coded into the verifier. Achieving extensibility and customizability of Silicon with respect to resources, chunks and their handling therefore is the goal of this bachelor's thesis.

2 Core Goals

The overall goal is to extend Silicon with a method to define customized resources in a more convenient way. This involves the following steps:

- Classify the currently supported kinds of resources by identifying their similarities, differences and in general customizable properties in both their representation as chunks and the operations they allow. Exploit the similarities and design a common way of representing and managing these resources. We expect that this will lead to a reduction of chunk types and unify their management rules.
- Develop an approach that provides users with the possibility to easily add custom resources to Silicon or change the handling of existing ones. Depending on the complexity and the needs the approach might be in the form of a config file or a plugin-like interface with a well-defined API.
- Evaluate the newly developed approach by integrating it into Silicon and basing Silicon's currently available resources on it. This involves refactoring the current chunks and their handling to use the new approach.
- Illustrate the remaining customization effort by changing the handling of existing chunks or adding support for new resources (and potentially chunks).
- Integrate gaining and losing permissions (chunks) into the new approach.

3 Extension Goals

- Investigate the feasibility for systematically deriving quantified chunks, i.e. chunks that represent sets of resources, from their respective non-quantified representation.
- Scan permission-logic-related publications, e.g. [3] or [4], for additional resources that could be added to Silicon via the newly developed approach.

References

[1] P. Müller and M. Schwerhoff and A. J. Summers: Viper: A Verification Infrastructure for Permission-Based Reasoning (VMCAI), 2016.

[2] M. Schwerhoff: Advancing Automated, Permission-Based Program Verification Using Symbolic Execution (PhD Thesis), 2016.

[3] A. J. Summers, P. Müller: Actor Services. In: Thiemann P. (eds) Programming Languages and Systems. ESOP 2016. Lecture Notes in Computer Science, vol 9632. Springer, Berlin, Heidelberg, 2016.

[4] M. Doko and V. Vafeiadis, Tackling Real-Life Relaxed Concurrency with FSL++ (ESOP), 2017.