Modular Verification of Finite Blocking via Obligations

Problem Description

Christian Klauser
`klauserc@student.ethz.ch`

June 3, 2014

1 Background

Verifiers for *Chalice*, an experimental programming language designed for specifying and verifying concurrent programs, today prevent certain cases of infinite blocking, for instance deadlocks with monitors. It is still possible, though, to construct programs that might block indefinitely but pass verification today, especially when non-terminating threads are involved.

A promising way to tackle this problem is the idea that whenever a thread could cause another to block (e.g., by holding a lock) it get an *obligation* to allow other to continue at some point. Such obligations would be checked per method in isolation and could in certain cases be passed between methods, a bit like permissions in Chalice today.

2 Core

The goal of this Mater's thesis is to devise a way to implement checking of obligations for finite blocking in *Silver*, an intermediate language for permission-based reasoning, and then to deliver an implementation that serves as a proof of concept. This implementation spans Silver itself and *Silicon*, a verifier for Silver that is based on symbolic execution.

At it's core, the thesis involves

- coming up with a model of obligations in the intermediate language Silver
- implementing the necessary extensions to Silver in the Silicon verifier backend
- extending the existing lock-order mechanism to become a wait-order that involves all operations that can cause a thread to block, not just monitor-locks
- implementing a proof-of-concept extension to Chalice that uses the new obligation model to ensure finite blocking for some constructs

3 Extensions

Depending on the progress of the main task, several extensions to the project are possible.

- Implement verification of additional blocking operations. Re-entrant locks would be one example.
- Implement verification of other operations that benefit from coming with obligations. The obligation to free memory after an allocation would be one example.
- Extend obligations to be able to use ranking domains other than integers to measure the ”amount of obligation” a thread has on a resource.
- Investigate the relationship between obligations and permissions.
- Extend the proof-of-concept implementation of obligations in Chalice to include more of the existing blocking language constructs.