Runtime Checking for Chalice

Student project for Heinz Hegi, supervised by Alex Summers

25. April 2012

Introduction

Program verification is a useful tool for designing correct software. Chalice is a specialised language and verifier for concurrent programs. It statically analyses programs and detects concurrency bugs such as deadlocks and race conditions. To do this, Chalice checks the contracts specified by the programmer and keeps track of field access permissions. Permissions are modeled as fractions between 0% and 100%. They can be handed from one method or thread to another at calls/returns or forks/joins. A thread always has full access to it's local objects, but has to transfer the corresponding permission to a Monitor when sharing an object. Data access is only allowed when the method/thread holds enough specific permission: 100% is needed in order to gain write access, whereas any fraction $> 0\%$ grants read access. All access rights and contracts are verified statically and do not affect the behavior of the program itself. Since there is no dynamic checking, the current C# code generator ignores the verification code and translates only the underlying program.

Goal

The goal of this project is to implement a runtime checker for Chalice. This can be used to get more information about where exactly and why the verification process failed, or to expand the language in order to handle constructs that can't easily be verified statically, such as iterative quantification: if the programmer wants to iterate through a list whose length is unknown at compile time, and wants to fork off a thread for every referenced object in the list, then the verifier doesn't have enough statical information to be useful. This is mainly an aliasing problem. In order to solve this, the programmer would need to state assumptions about the list's content, so that the compiler can infer whether or not the needed permissions can be given to the threads. At runtime, the list is given, and thus it's fairly easy to see if problems occur.

Tasks

The core task is to add to the existing Chalice-to-C# code generation an appropriate state representation and transformation for all the verification constructs, so that the runtime verification works properly and meaningful information about the verification process can be obtained Also, the existing code generation has to be updated in order to support all current Chalice language features. At this stage, predicates are immediately evaluated, so fold and unfold statements can be ignored.

Then, there are two possible extensions:

- An alternative handling of predicates, which distinguishes holding a predicate from holding it's contents. This mimics the static verifier's behavior.
- Handling of iterative quantification in the runtime checker as previously explained.