frama C
Software Analyzers

Frama-C Training Session
Tips and Tricks in Software Verification

Virgile Prevosto

CEA List

October 22nd, 2010

frama C
Software Analyzers

outline

Verification Environment

Initial Specification

Intermediate Contracts

Loop Invariants

Auxiliary Annotations

frama C
Software Analyzers

Verification Environment

Initial Specification

Intermediate Contracts

Loop Invariants

Auxiliary Annotations

Identifying the verification tasks

Kind of properties

- Safety (absence of runtime error)
- Functional properties (Pre/Post)
- Others (Information flow, reachability)

Kind of code

- Complete application
- Library, which entry points?
- Need to provide stubs?

Decide which tool to use

Jessie	Value analysis
✔ Arbitrary functional properties	✘ Properties that can be encoded in the abstraction used
✔ Modular	✘ Complete program
✘ Write formal specification	✔ Just need the code
✘ Needs user interaction	✔ Automated

frama C
Software Analyzers

Verification Environment

Initial Specification

Intermediate Contracts

Loop Invariants

Auxiliary Annotations

From informal to formal specification

Example

Function `sort` should take an array of int and its length and sort it in place. It does not return anything.

High level vs low level specification

- ACSL is essentially a low-level specification language
- Needs more link with modelization languages
- A good goal for future research projects?

Using predicate and logic functions

- Use them for notions that come up frequently in the development.
- Pay attention to axiomatics: nothing prevents you to write inconsistencies
  - Proof-reading
  - Quick check: write `lemma` that are obviously false and see if they can be proved.
  - Complete check: write (in Coq, Isabelle) a model of the axiomatics.
- Inductive definitions provide syntactic safeguards
- Pay attention to generality:

```
sorted_whole(int* a,integer length) or
sorted_slice(int* a, integer first, integer last)
```

Contract of Entry Points

Primary role

- Main input of the verification process
- Must reflect the informal specification
- Should not be modified just to suit the verification tasks

What to put in it

- Functional properties
- Safety requirements (`\valid`)

frama C
Software Analyzers

Verification Environment

Initial Specification

Intermediate Contracts

Loop Invariants

Auxiliary Annotations

Contracts for internal functions

- Specification is here only to help the whole proof
- `requires` can be over-specified (if called only in particular contexts)
- `ensures` can be under-specified (if result does not impact directly the properties of interest in main functions)
- Pay attention to `assigns`
- Specification can evolve according to needs of provers

frama C
Software Analyzers

Verification Environment

Initial Specification

Intermediate Contracts

Loop Invariants

Auxiliary Annotations

Traversing Loops

- The only thing known on the state after the loop is what is in the invariants. Make them strong enough
- But not too strong (or it won't be possible to prove them inductively).
- Each assigned location should appear somewhere in an invariant
- Use `loop assigns` give a precise idea of what has been assigned so far.

frama C
Software Analyzers

Verification Environment

Initial Specification

Intermediate Contracts

Loop Invariants

Auxiliary Annotations

Helping automated provers

- Should be introduced carefully, when it appears that provers definitely go in the wrong direction on a given property
- Completely tied to a particular verification task
- `assert`: Give a cut point on a particular program state
- `lemma`: Some general property that the provers can not discover by themselves.

Introducing Proof Assistant

- When everything else fails
- Coq, Isabelle, PVS, are not that scary: we need only a tiny portion of the underlying theory.
- When used to them, helps understanding where the issue lies.
- Plans for an interactive theorem prover à la Caveat.