Summary

Formal Methods for Imperative Languages Such as C

Objective: Develop a Framework in Coq capable of reasoning about realistic programs and the complex data structures they manipulate.

Research Contributions

- Extension of Coq separation logic reasoning to larger programs with more complex data structures.
- A collection of library definitions and tactics to support reasoning about separation logic.
- Development of a powerful simplification tactic for major proof steps.

Creation of a library of useful predicates, functions and tactics—Deep model gives greater control over the design of tactics.

Key design decisions:

- Tradeoffs between performance and automation.
- Development of a powerful simplification tactic.

Simplification tactic executed after every major proof step.

- Based on term rewriting (with contextual rewriting) concepts.
- Automates reasoning about associativity, commutativity and other simple properties.
- Design decisions in creating canonical form addressed.

Proof steps involve the following:

- (1) Hoare rules to break up if-then-else, while and statement sequences.
- (2) Merging two states at the end of an if-then-else.

Statistical professor code statistics:

- Main proof size: ~220 lines
- Invariant size: ~52 lines
- Code size: ~30 lines

Coq data structure invariant:

Contains all of the important properties in about 30 lines.

Next, we add on two assertions that guarantee that both the assignment_stmt_v2 and assignment_stmt_v3 are well-formed. We use a (A -> B) as an abbreviation for

Result summary:

The work on the examples so far indicate that the approach is proving successful.

Properties:

- Soundness of invariants
- Termination or completeness proofs not provided

Resulting code:

```
Proof size: main
Invariant size: main
Coq verification time: main
```

Verifying the small tree traversal program

The main function, pre_order, takes as input a pointer to a tree structure and generates a list which represents a traversal of the tree.

Procedure excerpt:

Goals: Heape triples of form \((\text{pre}_\text{simms}(\text{post}_\text{simms}))\)

- (1)向前 propagation over unit statements
- (2) Applying Hoare rules to break up if-then-else, while and statement sequences
- (3) Merging two states at the end of an if-then-else
- (4) Proving one state implies another state

Proof steps involve the following:

- (1) Hoare rules to break up if-then-else, while and statement sequences
- (2) Merging two states at the end of an if-then-else
- (3) Proving one state implies another state

Theorem loopInvariant : {{afterInitAssigns}}loop{{afterWhile,NoResult}}.

The following constructs come from Bedrock/YNot/Charge!:

- Higher order separation * operator
- find takes an address in a tree and a tree functional
- AbsAll clauses assert for each list that the pointers in the second field of each node in the list pointed into the tree.

Verifying the DPLL algorithm

Efficient SAT solving algorithm for CNF expressions such as:

\[ (A \lor \neg A) \land (B \lor \neg B) \]

**DPLL algorithm:**

1. Choose a variable and assign a value
2. Perform unit propagation to find additional assignments by the choices already made
3. Backtrack and change choices when a contradiction is found

Next, we add on two assertions that guarantee that both the assignment_stmt_v2 and assignment_stmt_v3 are well-formed. We use a (A -> B) as an abbreviation for

**Property classes**

- soundness of invariants
- termination or completeness proofs not provided

**Result summary**

The work on the examples so far indicate that the approach is proving successful.

**Proof size**:

main

**Invariant size**:

main

**Coq verification time**:

main

**Statistics**:

- Code size: ~300 lines
- Invariant size: ~100 lines
- Proof time: ~5 minutes
- Main proof size: ~220 lines

**Status**:

Top level proof complete: Many lemmas need to be proven