Developing Certified Program Verifiers with a Proof Assistant

Adam Chipala, University of California, Berkeley

Why Certified Verifiers?

- Proof-Carrying Code
  - User
  - Program
  - Developer

Proof-Carrying Verifiers

- User
- Program
- Developer

The expressivity of traditional foundational PCC requires that these proofs can be very large and expensive to check!

Why Certified Verifiers?

- Coq and Extraction
  - Advantages for Programming
    - Dependent types ensure total correctness.
    - Includes an ML-style module system that can mix program and proof module elements.
    - Easy OCaml integration
  - Advantages for Proving
    - Combine goal-directed proof search with standard programming.
    - Take advantage of mechanisms for organizing and automating direct and tactic-based proofs.

Coq and Extraction

Platform: The Coq Proof Assistant, which includes a rich dependently-typed functional programming language.

Implementation Technique: Monads for Composing Decision Procedures

OnceSoundness : forcall (abs state) : Prop
  (forcall (conc state) : Prop)
  check_soundness (src	
  dst)

typedef: forcall (abs state) : Prop
  (forcall (conc state) : Prop)
  (forall (conc state),
    compatible conc abs
    -> hasType (eval conc e) t)

Definition do_write:forall (src	
  dst) : absState,
  (forall (conc state),
    compatible conc abs
    -> hasType (eval conc e) t)

Implementation Technique: Functors for Lowering Abstractions

Use a dependently-typed language to make these dependencies explicit.

Memory Safety from the Ground Up

- Weak Update Type System
  - Use a simplified type system based on partial type assignments to memory cells.
  - Simple Flags
  - Track dependencies between condition/flags and registers/memory.
  - Stack Types
    - Augment type system with types to track stack and calling conventions.

Type System

- Use Cartesian abstraction assigning types to registers.

Fixed Code

- Enforce immutable code.

- Reduce
  - Compile to simplified RISC.

Abstract Interpretation

- Ensures that the semantics of the low-level language are preserved under the type system.

The Final Product

- What
  - A certified memory safety verifier for x86 machine code programs compiled by a fictitious certifying compiler for a language featuring algebraic datatypes.

- Trusted Base
  - 2000 lines of Coq formalizing bitvectors and x86 semantics.
  - OCaml code for parsing x86 binaries to ASTs.
  - Coq checking and extraction.
  - OCaml compiler.

- Reusable Pieces
  - 10,000-line Coq utility library.
  - 7000 lines of Coq for a library of functions spanning 8 levels of abstraction.

- Verifier-Specific Code
  - One 600-line Coq file describing a type system declaratively.

- Extracted Code
  - 5000 lines of OCaml representing bitvectors with native words and mathematical integers with infinite-precision integers, thanks to a custom extraction optimization.

x86 Semantics