Overview

We present a formalization of graph theory in Coq and a library of techniques that together mechanically verify real C programs that manipulate heap-represented graphs.

Challenge: These structures exhibit deep intrinsic sharing and have thus historically evaded analysis using traditional separation logic: the FRAMe rule fails.

Solution: First, we create a modular setup for reasoning about abstract mathematical graphs. Second, we add a new LOCALIZER rule to separation logic, thus supporting existential quantifiers in postconditions and smoothly handling modified program variables.

Our techniques are:

- General and Lightweight: We integrate our work with the CompCert and Verified Software Toolschain projects with minimal reengineering on their side.
- Powerful: We certify six graph-manipulating C programs. Our flagship example is a 400-line generational garbage collector. Our proofs are entirely machine-checked in Coq.

Workflow and Key Components

A sketch of how we verify C programs. Note where we integrate with other projects.

The structure of our Mathematical Graph Library. The soundness condition is entirely customizable. Lemmas and properties can be composed, and are automatically inherited.

Verification of Garbage Collector

We verify a generational garbage collector for the CertiCoq Project. It is 400 lines long, and is based on the OCaml GC: 12 generations, variable-sized blocks, and runtime disambiguation of boxed/unboxed fields.

We identify two areas where ANSL C semantics are too weak to certify OCaml-style GCs:

- Double-bordered pointer comparisons:
  ```c
  int is_from(value * from_start, value * from_limit, value * v) { return (from_start <= v && v < from_limit); }
  ```
- A classic OCaml trick for runtime disambiguation of fields:
  ```c
  int test_int_or_ptr (value *v) { return (int)(((intnat)v)&1); }
  ```

Both tests, although undefined in C, are compatible with the CompCert compiler. Below we present a visualization of the theorems involved in the proof.