Certifying Graph-Manipulating C Programs

Shengyi Wang[†] Qinxiang Cao[‡] Anshuman Mohan[†] Aquinas Hobor[†]
National University of Singapore[†] Shanghai Jiao Tong University[‡]

SPLASH Athens 2019

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**.

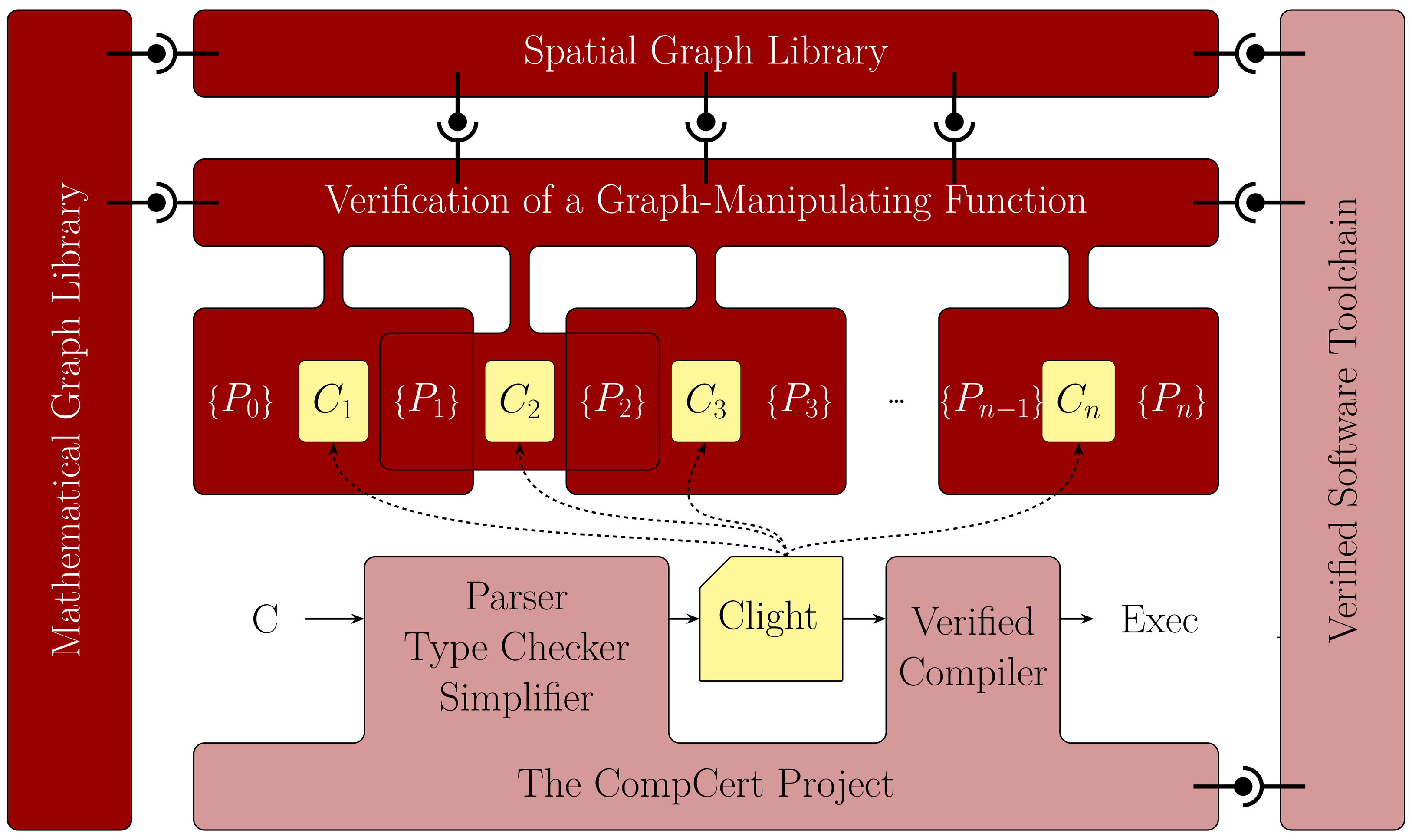
Challenge: These structures exhibit **deep intrinsic sharing** and have thus historically evaded analysis using traditional separation logic: the FRAME rule fails.

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

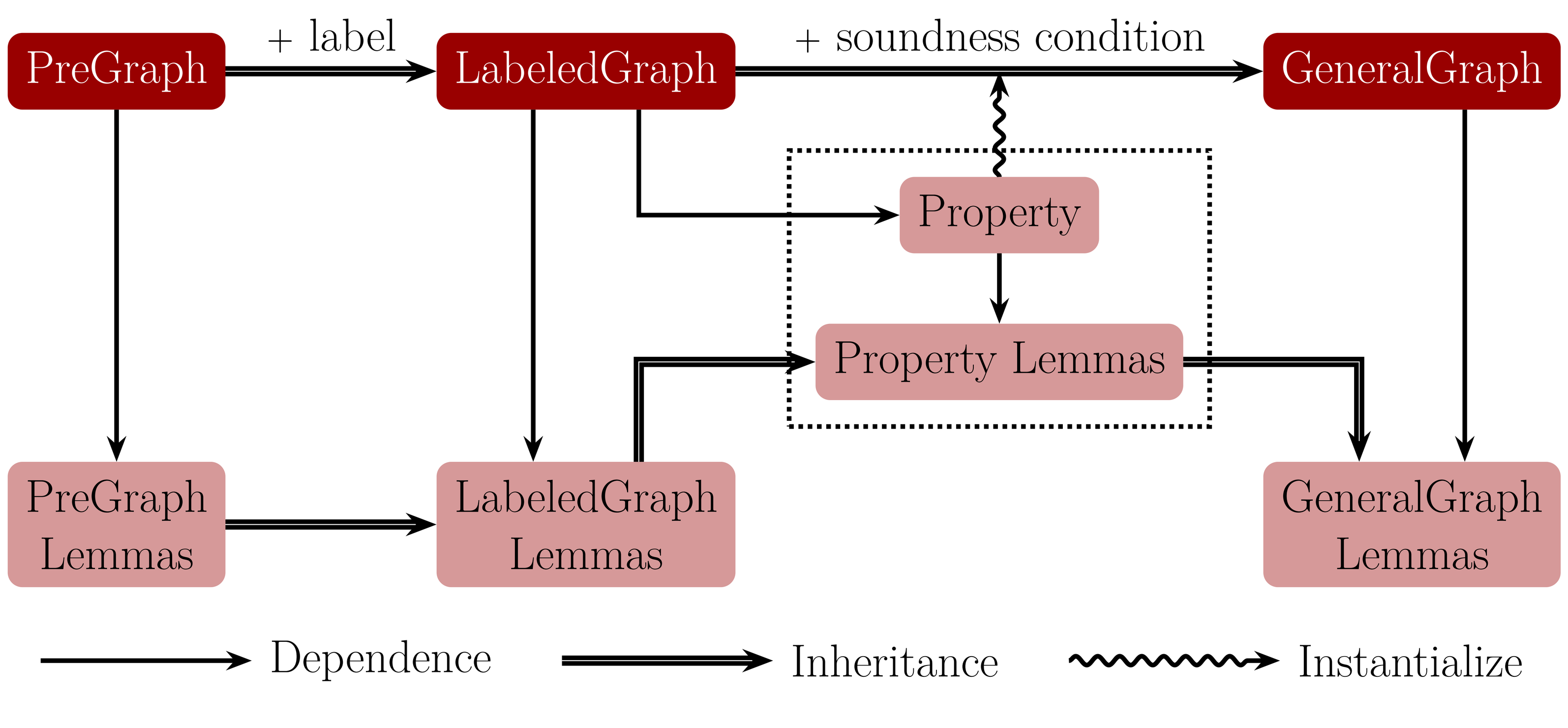
- **General and Lightweight:** We integrate our work with the CompCert and Verified Software Toolchain projects with minimal reengineering on their side.
- **Powerful:** We certify six graph-manipulating C programs. Our flagship example is a 400-line generational garbage collector. Our proofs are entirely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



Our key addition to separation logic. The rule below is coderivable from the FRAME rule.

LOCALIZE

$$\frac{G_1 \vdash L_1 \star R \qquad \{L_1\}\ c\ \{\exists x.\ L_2\} \qquad R \vdash \forall x.\ (L_2 \mathrel{-\!\!\star} G_2)}{\{G_1\}\ c\ \{\exists x.\ G_2\}}\ \mathsf{FreeVar}(R) \cap \mathsf{ModVar}(c) = \varnothing$$

Some statistics relating to our development.

Component	Files	Size (loc)	Definitions	Theorems
Common Utilities	10	3,578	44	289
Math Graph Library	20	10,585	216	581
Spatial Graph Library	3	2,328	59	110
Integration into VST	11	2,783	17	172
Marking (graph and DAG)	6	775	9	20
Spanning Tree	5	2,723	17	92
Union-Find (heap and array)	18	3,193	107	135
Garbage Collector	16	13,858	235	712
Total Development	89	39,823	704	2,111

Annotated Proof Sketch for `find`

```
struct Node { unsigned int rank;
              struct Node * parent; }
//{uf_graph(γ) ∧ x ∈ V(γ)}
struct Node* find(struct Node* x) {
  struct Node *p;
  //{uf_graph(γ) ∧ x ∈ V(γ) ∧
  //  ∃r, pa. γ(x) = (r, pa) ∧ pa ∈ V(γ)}
  //↘{x ↦ r, pa ∧ x ∈ V(γ) ∧
  //   γ(x) = (r, pa) ∧ pa ∈ V(γ)}
↯ p = x -> parent;
  //↙{x ↦ r, pa ∧ p = pa ∧ x ∈ V(γ) ∧
  //   γ(x) = (r, pa) ∧ pa ∈ V(γ)}
  //{uf_graph(γ) ∧ p = pa ∧ x ∈ V(γ) ∧
  //  γ(x) = (r, pa) ∧ pa ∈ V(γ)}
  if (p != x) {
    //{uf_graph(γ) ∧ p = pa ∧ pa ≠ x ∧
    //  x ∈ V(γ) ∧ γ(x) = (r, pa) ∧ pa ∈ V(γ)}
    p = find(p);
    //{∃γ', rt. uf_graph(γ') ∧ p = rt ∧ pa ≠ x ∧ x ∈ V(γ) ∧
    //  findS(γ, pa, γ') ∧ uf_root(γ', pa, rt) ∧ γ(x) = (r, pa)}
    //↘{x ↦ r, pa ∧ p = rt ∧ pa ≠ x ∧ findS(γ, pa, γ') ∧
    //   uf_root(γ', pa, rt) ∧ x ∈ V(γ) ∧ γ(x) = (r, pa)}
↯   x -> parent = p;
    //↙{x ↦ r, rt ∧ p = rt ∧ pa ≠ x ∧ findS(γ, pa, γ') ∧
    //   uf_root(γ', pa, rt) ∧ x ∈ V(γ) ∧ γ(x) = (r, pa)}
    //{∃γ''. uf_graph(γ'') ∧ findS(γ, pa, γ'') ∧
    //  uf_root(γ'', x, rt) ∧ p = rt}
  } return p;
} //{∃γ'', rt. uf_graph(γ'') ∧ findS(γ, x, γ'') ∧
  //  uf_root(γ'', x, rt) ∧ ret = rt}
```

$$\mathsf{uf\_graph}(x, \gamma) \triangleq \bigstar_{v \in V(\gamma)} v \mapsto \gamma(v)$$

$$uf\_root(\gamma, x, rt) \triangleq x \overset{\gamma}{\leadsto}{}^{\star}\ rt\ \wedge \forall rt'.\ rt \overset{\gamma}{\leadsto}{}^{\star}\ rt' \Rightarrow rt = rt'$$

$$\begin{aligned} findS(\gamma, x, \gamma') \triangleq\ & \big(\forall v.\ v \in V(\gamma) \Leftrightarrow v \in V(\gamma')\big) \wedge \\ & \big(\forall v.\ v \in V(\gamma) \Rightarrow \gamma(v).rank = \gamma'(v).rank\big) \wedge \\ & \big(\forall r, r'.\ uf\_root(\gamma, v, r) \Rightarrow uf\_root(\gamma', v, r') \Rightarrow r = r'\big) \wedge \\ & \big(\gamma \setminus \{v \in \gamma \mid x \overset{\gamma}{\leadsto}{}^{\star}\ v\} \cong \gamma' \setminus \{v \in \gamma \mid x \overset{\gamma}{\leadsto}{}^{\star}\ v\}\big) \end{aligned}$$

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 ANSI C semantics are too weak to certify OCaml-style GCs:

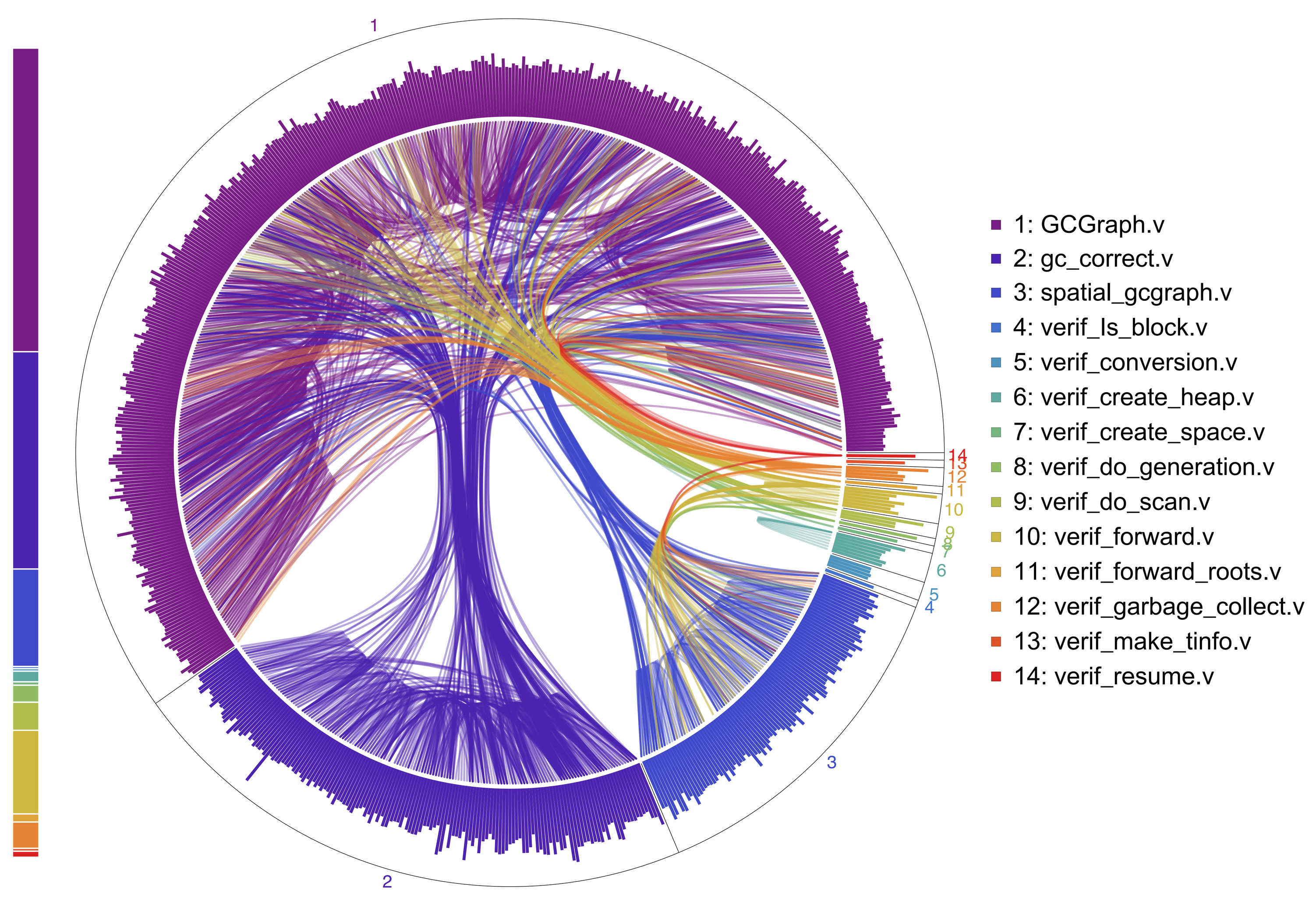
- Double-bounded pointer comparisons:

```
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:

```
int test_int_or_ptr (value x) { return (int)(((intnat)x)&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.



LaTeX TikZposter