Mechanized Verification of Graph-Manipulating Programs

Shengyi Wang[†] Qinxiang Cao[+] Aquinas Hobor[†]

National University of Singapore[†] Princeton University[+]

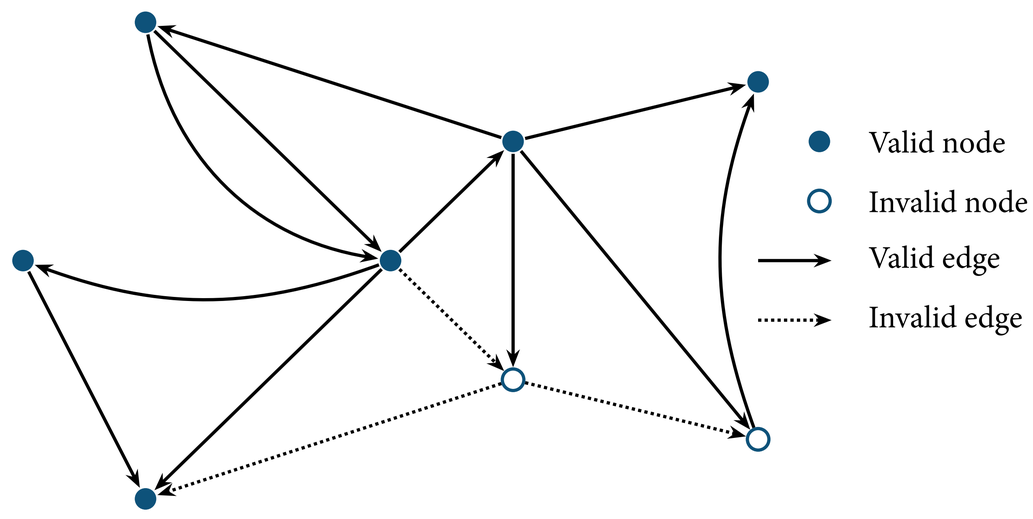
Introduction

We developed a general framework of graph theory powerful enough to support realistic verification. We build on CompCert—a verified compiler for a dialect of C, and VST—a series of machine-checked modules for verifying programs in that dialect. Using VST and our framework, we have verified the functional correctness of several classical graph-manipulating algorithms, including **graph marking**, **spanning tree**, **graph copy**, and **union-find**.

To enable the verification of full functional correctness of graph algorithms we need a way to reason about mathematical graphs. To allow such verification to be mechanized without undue pain we must take care to develop a modular and general-purpose framework for graphs.

The most basic kind of graph is PreGraph, out of which we build LabeledGraphs, and which in turn are used to build GeneralGraphs. Each kind has some lemmas and also inherits the lemmas of the previous kind. The dashed box represents a "plugin" system. These "plugins" can specify many different kinds of properties. Each property, in turn, can be used to prove many property-specific lemmas, all of which then apply to the instantiating GeneralGraph.

A PreGraph with invalid nodes and edges



Pointer Version of Union Find

```
1 struct Node { int rank; struct Node * parent; };
2
3 struct Node* find(struct Node* x) {
4 // {graph(g) ∧ vvalid(g, x)}
5     struct Node *p, *p0;
6     p = x -> parent;
7     if (p != x) {
8         p0 = find(p);
9 // {∃g', t. graph(g') ∧ uf_equiv(g, g') ∧ uf_root(g', p, t) ∧ p0 = t}
10 // {graph(g1) ∧ uf_equiv(g, g1) ∧ uf_root(g1, p, p0)}
11         p = p0;
12         x -> parent = p; }
13 // {g2 = redirect_parent(g1, x, p) ∧ graph(g2)) ∧ uf_equiv(g, g2) ∧ uf_root(g, x, p)}
14     return p; };
15 // {∃g', t. graph(g') ∧ uf_equiv(g, g') ∧ uf_root(g', x, t)}
16
17 void unionS(struct Node* x, struct Node* y) {
18 // {graph(g) ∧ vvalid(g, x)} ∧ vvalid(g, y)}
19     struct Node *xRoot, *yRoot; int xRank, yRank;
20     xRoot = find(x);
21 // {∃g', t. graph(g') ∧ uf_equiv(g, g') ∧ uf_root(g', x, t) ∧ xRoot = t}
22 // {graph(g1) ∧ vvalid(g1, y) ∧ uf_equiv(g, g1) ∧ uf_root(g1, x, xRoot)}
23     yRoot = find(y);
24 // {∃g', t. graph(g') ∧ uf_equiv(g1, g') ∧ uf_root(g', y, t) ∧ yRoot = t}
25 // {graph(g2) ∧ uf_equiv(g1, g2) ∧ uf_root(g2, x, yRoot)}
26     if (xRoot == yRoot) { return; }
27 // {graph(g2) ∧ uf_union(g, x, y, g2)}
28 // {∃g'. graph(g') ∧ uf_union(g, x, y, g')}
29     xRank = xRoot -> rank; yRank = yRoot -> rank;
30     if (xRank < yRank) { xRoot -> parent = yRoot;
31 // {g3 = redirect_parent(g2, xRoot, yRoot) ∧ graph(g3)) ∧ uf_union(g, x, y, g3)}
32     } else if (xRank > yRank) { yRoot -> parent = xRoot;
33 // {g3 = redirect_parent(g2, yRoot, xRoot) ∧ graph(g3)) ∧ uf_union(g, x, y, g3)}
34     } else {
35         yRoot -> parent = xRoot;
36         xRoot -> rank = xRank + 1; }};
37 // {g3 = redirect_parent(g2, yRoot, xRoot) ∧ graph(g3)) ∧ uf_union(g, x, y, g3)}
38 // {∃g'. graph(g') ∧ uf_union(g, x, y, g')}
```

$$\mathsf{graph}(g) \triangleq \exists l.\ \mathsf{NoDup}\ l \wedge \big(\forall v.\ v \in l \Leftrightarrow \mathsf{vvalid}(g, v)\big) \wedge \underset{v \in l}{\bigstar} v \mapsto \gamma(g, v)$$

$$\underset{\{e_1, e_2, \ldots, e_n\}}{\bigstar} \mathsf{P} \triangleq \mathsf{P}(e_1) * \mathsf{P}(e_2) * \cdots * \mathsf{P}(e_n)$$

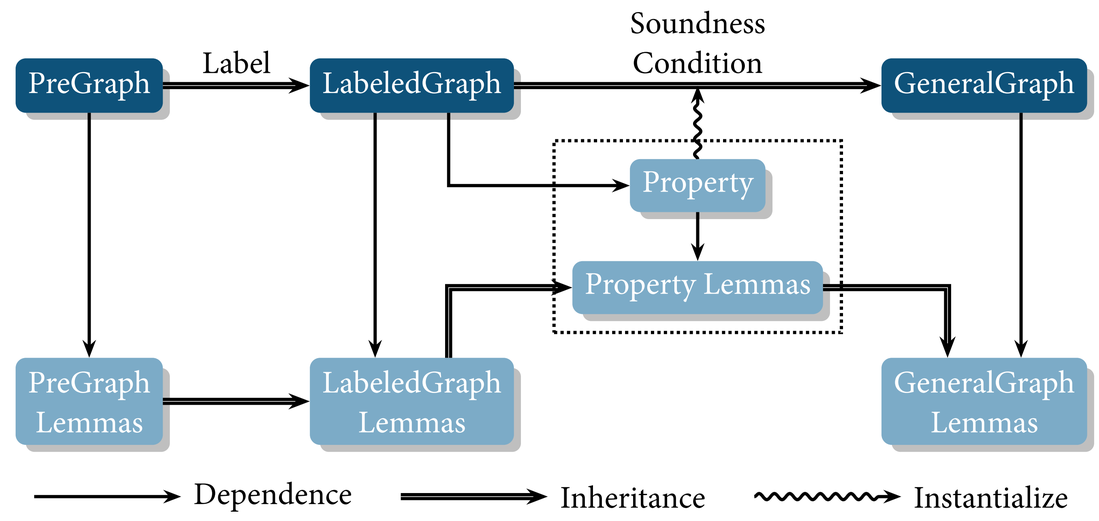
Component	Files	Spec (lines)	Proof (lines)
Union-Find	19	2,079	2,475
Graph Mark	10	986	1,255
Spanning Tree	4	808	1,850
Graph Copy	7	1,511	2,862
Graph Library	18	2,772	6,559
Extension of VST	32	1,967	3,588
Common Utilities	10	696	1,792
Total Development	100	10,819	20,381

Array Version of Union Find

```
1 struct subset { int parent; int rank; };
2
3 int find(struct subset s[], int i) {
4 // {graph(g, s) ∧ vvalid(g, i)}
5     int p0 = 0;
6     int p = s[i].parent;
7     if (p != i) {
8         p0 = find(s, p);
9 // {∃g', t. graph(g', s) ∧ uf_equiv(g, g') ∧ uf_root(g', p, t) ∧ p0 = t}
10 // {graph(g1, s) ∧ uf_equiv(g, g1) ∧ uf_root(g1, p, p0)}
11         p = p0;
12         s[i].parent = p; }
13 // {g2 = redirect_parent(g1, i, p) ∧ graph(g2, s)) ∧ uf_equiv(g, g2) ∧ uf_root(g, i, p)}
14     return p; }
15 // {∃g', t. graph(g', s) ∧ uf_equiv(g, g') ∧ uf_root(g', x, t)}
```

$$\mathsf{graph}(g, \mathrm{x}) \triangleq \exists n.\ \big(\forall v.\ 0 \leqslant v < n \Leftrightarrow \mathsf{vvalid}(g, v)\big) \wedge \big(n \leqslant \mathrm{MaxInt}/8\big) \wedge \mathsf{array\_at}\big(\mathrm{x}, \mathsf{map}\ (\lambda v.\ \gamma(g, v))\ [0, 1, 2, \ldots, n]\big)$$

$$\begin{aligned}
\mathsf{uf_root}(g, x, t) &\triangleq g \vDash x \rightsquigarrow t \wedge \forall y.\ g \vDash t \rightsquigarrow y \Rightarrow y = t \\
\mathsf{uf_equiv}(g_1, g_2) &\triangleq \big(\forall x.\ \mathsf{vvalid}(g_1, x) \Leftrightarrow \mathsf{vvalid}(g_2, x)\big) \wedge \\
&\quad \forall x, r_1, r_2.\ \mathsf{uf_root}(g_1, x, r_1) \Rightarrow \mathsf{uf_root}(g_2, x, r_2) \Rightarrow r_1 = r_2 \\
\mathsf{uf_union}(g_1, v_1, v_2, g_2) &\triangleq \forall s_1, s_2.\ v_1 \in s_1 \Rightarrow v_2 \in s_2 \Rightarrow s_1 \subset g_1 \Rightarrow s_2 \subset g_1 \Rightarrow s_1 \cup s_2 \subset g_2 \wedge \\
&\quad (\forall s.\ s \neq s_1 \Rightarrow s \neq s_2 \Rightarrow s \subset g_1 \Rightarrow s \subset g_2) \wedge \\
&\quad (\forall s.\ s \subset g_2 \Rightarrow s = (s_1 \cup s_2) \vee s \subset g1) \\
s \subset g &\triangleq s = \varnothing \vee \exists r.\ r \in s \wedge \big(\forall v.\ v \in s \Leftrightarrow \mathsf{uf_root}(g, v, r)\big)
\end{aligned}$$



Coq Goal Snippet after Line 8

```
H2 : uf_equiv g g'
H3 : uf_root g' pa root
============================================================
semax Delta0 (PROP ( ) LOCAL
 (temp _p0 (pointer_val_val root); temp _p (pointer_val_val pa);
  temp _x (pointer_val_val x))  SEP (graph sh g'))
 (Ssequence (Sset _p (Etempvar _p0 (tptr (Tstruct _Node noattr))))
 MORE_COMMANDS) POSTCONDITION
```