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.

---

### Pointer Version of Union Find

```c
struct Node { int rank; struct Node * parent; };  2
struct Node* find(struct Node* x) {  4
  if (graph(x) -> valid(g, x))  5
    p = x -> parent;  6
  if (p != x) {  7
    p0 = find(p);  8
    /\  // g0 = uf_equ(g, g0)', uf_root(g0, p, t) != p0  9
    /\  // g0 = uf_equ(g, g0)', uf_root(g0, p, p0)  10
    p = p0;  11
    x -> parent = p;  12
  }  13
  return p;  14
}  15
```

### Array Version of Union Find

```c
struct subset { int parent; int rank; };  2
int find(struct subset s[], int i) {  3
  if (graph(s, i) -> valid(g, i))  4
    return i;  5
  int p = s[i].parent;  6
  if (p != i) {  7
    s[p].parent = p;  8
    if (graph(s, p) -> valid(g, i))  9
      return find(s, p);  10
    /\  // g0 = uf_equ(g, g0)', uf_root(g0, p, t) != p0  11
    /\  // g0 = uf_equ(g, g0)', uf_root(g0, p, p0)  12
    p = p0;  13
    s[i].parent = p;  14
    return p;  15
  }  16
}  17
```

---

### Coq Goal Snippet after Line 8

```coq
H2 : uf_equ ig g'  
H3 : uf_root g' pa root  
=============================================  
sema 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')  
(Sseqnue (Sset_p (etempvar_p0 (t.ptr (Tstruct_node noatt)))))  
MORE_COMMANDS POSTCONDITION  
```

---

**Component:**
- Files: 19
- Spec (lines): 2,079
- Proof (lines): 2,475

**Graph Mark:**
- Files: 10
- Spec (lines): 986
- Proof (lines): 1,255

**Spanning Tree:**
- Files: 4
- Spec (lines): 808
- Proof (lines): 1,850

**Graph Copy:**
- Files: 7
- Spec (lines): 1,511
- Proof (lines): 2,862

**Graph Library:**
- Files: 18
- Spec (lines): 2,772
- Proof (lines): 6,559

**Extension of VST:**
- Files: 32
- Spec (lines): 1,967
- Proof (lines): 3,588

**Common Utilities:**
- Files: 10
- Spec (lines): 696
- Proof (lines): 1,792

**Total Development:**
- Files: 100
- Spec (lines): 10,819
- Proof (lines): 20,381