Mechanized Verification for Graph Algorithms

Shengyi Wang, Qinxiang Cao, Aquinas Hobor
Our goals

• Verify graph-manipulating programs

• All proofs mechanized

• Real code

• Techniques able to handle sizable examples
Graph-manipulating programs

• Heap represented graphs
  – Traditional challenge for verification

• Nontrivial algorithms with “real” specs
  – spanning tree
  – deep copy
  – union-find
  – sizable (~400-line) generational optimized garbage collector for certified compiler (in progress)
Mechanized proofs for real code

- All verification done in Coq

- Target language: CompCert Clight

- Hook into Verified Software Toolchain
Challenges

• Separation logic is a little tricky for graph-manipulating structures

• Real code is harder than toy code, sometimes in rather unexpected ways, e.g.
  – Garbage collectors break CompCert’s memory model (and type system) due to the typical uniform treatment of data and pointers
Challenges

• Graph algorithms are easier to specify relationally rather than functionally
  – No “issues” with termination (esp. in Coq)
  – Some algorithms’ “natural specifications” involve nondeterminism (e.g. union-find)
  – Some algorithms do not have easy/natural purely functional implementations (e.g. union-find)
Challenges

• Graph algorithms are easier to specify relationally rather than functionally
  – No “issues” with termination (esp. in Coq)
  – Some algorithms’ “natural specifications” involve nondeterminism (e.g. union-find)
  – Some algorithms do not have easy/natural purely functional implementations (e.g. union-find)
Challenges

• Graph algorithms are easier to specify relationally rather than functionally
  – No “issues” with termination (esp. in Coq)
  – Some algorithms’ “natural specifications” involve nondeterminism (e.g. union-find)
  – Some algorithms do not have easy/natural purely functional implementations (e.g. union-find)
Challenges

- Formal graph reasoning is surprisingly subtle, we’d like to reuse definitions, proofs, etc.
  - Reachability
  - Labels
  - Validity
  - Subgraphs

- We’d like generic graphs, and they should be general enough to handle real algorithms.
Some solutions

• Separation logic upgrades: “localization blocks”

```plaintext
22  // {graph(x, γ') ∧ γ(x) = (0, 1, r) ∧ mark1(γ, x, γ')} 
23  // □ {graph(1, γ')} 
24  i(8)  mark(1); 
25  // √ {∃γ''. graph(1, γ'') ∧ mark(γ', 1, γ'')}
26  // {∃γ''. graph(x, γ'') ∧ γ(x) = (0, 1, r) ∧ 
     {mark1(γ, x, γ') ∧ mark(γ', 1, γ'')}}
```
Localization is (upgraded) Ramification

\[
\text{Ramify-PQ (Program variables and quantifiers)}
\begin{align*}
\{L\} \ c \ \{\exists x. \ L_2\} & \quad \Rightarrow \quad G_1 \vdash L_1 \star [c] (\forall x. (L_2 \rightarrow \star G_2)) \\
\{G_1\} \ c \ \{\exists x. \ G_2\}
\end{align*}
\]

```plaintext
25 // ✓ \{\exists \gamma'' . \ \text{graph}(1, \gamma'') \land \text{mark}(\gamma', 1, \gamma'')\}
26 // \{\exists \gamma'' . \ \text{graph}(x, \gamma'') \land \gamma(x) = (0, 1, r) \land \text{mark1}(\gamma, x, \gamma') \land \text{mark}(\gamma', 1, \gamma'')\}
```
Localization is (upgraded) Ramification

\[
\text{RAMIFY-PQ (PROGRAM VARIABLES AND QUANTIFIERS)}
\begin{align*}
\{L\} \quad c \quad \{\exists x. \ L_2\} & \quad \frac{G_1 \vdash L_1 \star [c] (\forall x. \ (L_2 \rightarrow \star G_2))}{\{G_1\} \quad c \quad \{\exists x. \ G_2\}}
\end{align*}
\]

\[
\text{SOLVE RAMIFY-PQ}
\begin{align*}
G_1 \vdash L_1 \star F & \quad F \vdash \forall x. \ (L_2 \rightarrow \star G_2) & \quad F \text{ IGNORES} \\
G_1 \vdash L_1 \star [c] (\forall x. \ (L_2 \rightarrow \star G_2)) & \quad \text{ModVar}(c)
\end{align*}
\]
A little jig for modified variables

\[
\begin{align*}
15 & \quad \{ \text{graph}(x, \gamma) \land \gamma(x) = (0, l, r) \} \\
16 & \quad \{ x \mapsto 0, -, l, r \land \gamma(x) = (0, l, r) \} \\
17 & \quad l = x \rightarrow l; \\
18 & \quad r = x \rightarrow r; \\
19 & \quad x \rightarrow m = 1; \\
20 & \quad \{ x \mapsto 1, -, 1, r \land \gamma(x) = (0, 1, r) \land \exists \gamma'. \ mark1(\gamma, x, \gamma') \} \\
21 & \quad \{ \exists \gamma'. \ graph(x, \gamma') \land \gamma(x) = (0, 1, r) \land mark1(\gamma, x, \gamma') \}
\end{align*}
\]

- Uh oh... \( l \), \( r \), and \( x \) are modified in the localization block...
A little jig for modified variables

4  /// \{L_2\}
5  /// √ \{∃x, y. \ x = x \ ∧ \ y = y \ ∧ \ [x ↦ x][y ↦ y]L_2\}
6  /// \{∃x, y. \ x = x \ ∧ \ y = y \ ∧ \ [x ↦ x][y ↦ y]G_2\}
7  /// \{G_2\}
A little jig for modified variables

4  //  \{L_2\}
5  //  \sqrt{\exists x, y. \ x = x \land y = y \land [x \mapsto x][y \mapsto y]L_2}\}
6  //  \{\exists x, y. \ x = x \land y = y \land [x \mapsto x][y \mapsto y]G_2\}
7  //  \{G_2\}

\[ F \triangleq \forall x, y. [x \mapsto x][y \mapsto y](L_2 \rightarrow G_2) \]
A little jig for modified variables

\[ F \triangleq \forall x, y. \ [x \mapsto x][y \mapsto y](L_2 \rightarrow \ast G_2) \]

\[ G_1 \vdash L_1 \ast F \mid A \vdash B \ast \forall x, y. \ [x \mapsto x][y \mapsto y](L_2 \rightarrow \ast G_2) \]
A little jig for modified variables

\[
\begin{align*}
4 & \quad \{L_2\} \\
5 & \quad \{\exists x, y. \ x = x \land y = y \land [x \mapsto x][y \mapsto y]L_2\} \\
6 & \quad \{\exists x, y. \ x = x \land y = y \land [x \mapsto x][y \mapsto y]G_2\} \\
7 & \quad \{G_2\}
\end{align*}
\]

\[F \triangleq \forall x, y. \ [x \mapsto x][y \mapsto y](L_2 \rightarrow \star G_2)\]

\[G_1 \vdash L_1 \star F \quad | \quad A \vdash B \star \forall x, y. \ [x \mapsto x][y \mapsto y](L_2 \rightarrow \star G_2)\]

\[F \vdash (\forall x, y. \ [x \mapsto x][y \mapsto y](L_2 \rightarrow \star G_2)) \vdash \]

\[L_2 \rightarrow \star (\exists x, y. \ x = x \land y = y \land [x \mapsto x][y \mapsto y]L_2) \rightarrow \star G_2\]
Some other solutions:

a sound "graph" predicate in SL

\[
\text{graph}(x, \gamma) \iff x \rightarrow \gamma(x) \star \left( \bigcup_{n \in \text{neighbors}(\gamma, x)} \text{graph}(\gamma, n) \right)
\]
Some other solutions:
a sound “graph” predicate in SL

$$\text{graph}(x, \gamma) \iff x \mapsto \gamma(x) \star \left( \bigcup_{n \in \text{neighbors}(\gamma, x)} \text{graph}(\gamma, n) \right)$$

$$\text{graph}(x, \gamma) \triangleq \bigstar_{v \in \text{reach}(\gamma, x)} v \mapsto \gamma(v)$$
Some other solutions: a sound “graph” predicate in SL

\[ \text{graph}(x, \gamma) \iff x \mapsto \gamma(x) \uplus \left( \bigcup_{n \in \text{neighbors}(\gamma, x)} \text{graph}(\gamma, n) \right) \]

\[ \text{graph}(x, \gamma) \triangleq \bigstar_{v \in \text{reach}(\gamma, x)} v \mapsto \gamma(v) \]

\[ \forall x, y. \left( \text{P}(x) \uplus \text{P}(y) \vdash (\text{P}(x) \land x = y) \lor (\text{P}(x) \star \text{P}(y)) \right) \]
Modular mechanized proof engineering
Some other solutions:
A powerful and general graph library
Some other solutions: mechanizing localization blocks in VST
Some future work

• Increase modularity
  – Once you’ve done one union-find proof, have you done them all?

• Overlaid data structures
  – Common case; can we make them easier?

• Increase confidence of scalability
  – Garbage collector for a “real” client
  – Lots of “undefined operations”
  – Bonus: found a significant performance bug