Verified Efficient Implementation of Gabow’s Strongly Connected Component Algorithm

Peter Lammich

TU München

July 2014
Motivation

- Verify algorithm that computes SCCs of a digraph
- Variants/Applications of algorithm
  - Enumerate SCCs
  - Emptiness check of Generalized Büchi-Automata
  - ...
- Re-use formalization between variants
- Generate efficiently executable code
Outline

1. Gabow’s SCC Algorithm
2. Isabelle/HOL Formalization
3. Performance Evaluation
Outline

1. Gabow’s SCC Algorithm
2. Isabelle/HOL Formalization
3. Performance Evaluation
Strongly Connected Components

- SCC is maximal set of mutually reachable nodes
Strongly Connected Components

- SCC is maximal set of mutually reachable nodes
Path-Based Algorithms

- Depth first search
- On back edge, collapse nodes of induced cycle
- Eventually, each node represents SCC
Path-Based Algorithm Example
Path-Based Algorithm Example
Path-Based Algorithm Example
Path-Based Algorithm Example
Path-Based Algorithm Example
Path-Based Algorithm Example
Path-Based Algorithm Example
Path-Based Algorithm Example
Path-Based Algorithm Example
Path-Based Algorithm Example
Path-Based Algorithm Example
Gabow’s Data Structure

- How to maintain collapsed nodes on stack?
- Use *boundary stack*
  - contains indexes of bounds between collapsed nodes
- Yields linear-time algorithm
Gabow’s Data Structure Example

DFS stack:
Boundary stack:
Gabow’s Data Structure Example

DFS stack: A
Boundary stack: 0
Gabow’s Data Structure Example

DFS stack: A B
Boundary stack: 0 1
Gabow’s Data Structure Example

DFS stack: A B
Boundary stack: 0
Gabow’s Data Structure Example

DFS stack: A B C
Boundary stack: 0 2
Gabow’s Data Structure Example

DFS stack: A B C
Boundary stack: 0
Gabow’s Data Structure Example

DFS stack: A B C D
Boundary stack: 0 4
Gabow’s Data Structure Example

DFS stack: A B C
Boundary stack: 0
Gabow's Data Structure Example

DFS stack: A B C E
Boundary stack: 0 4
Gabow’s Data Structure Example

DFS stack: A B C E F
Boundary stack: 0 4 5
Gabow’s Data Structure Example

DFS stack: A B C E F
Boundary stack: 0 4
Gabow’s Data Structure Example

DFS stack:  A  B  C
Boundary stack:  0
Gabow’s Data Structure Example

DFS stack:

Boundary stack:
Outline

1. Gabow’s SCC Algorithm
2. Isabelle/HOL Formalization
3. Performance Evaluation
Re-usable Formalization

- Goal: Formalize family of SCC-based algorithms
  - Enumerate SCCs
  - GBA emptiness check
  - ...

- Approach: Formalize "skeleton" SCC algorithm first
  - Just the node-contracting DFS, no output
  - Theorems for VCs (invariant preservation, ...)

- Stepwise refinement to executable code
- Reuse this formalization for actual algorithms
- Utilize existing Isabelle technologies
  - Collection Framework, Refinement Framework, Autoref tool
  - Code generator, locales
Re-usable Formalization

- Goal: Formalize family of SCC-based algorithms
  - Enumerate SCCs
  - GBA emptiness check
  - ...
- Approach: Formalize “skeleton” SCC algorithm first
  - Just the node-contracting DFS, no output
  - Theorems for VCs (invariant preservation, ...)
  - Stepwise refinement to executable code
Re-usable Formalization

- Goal: Formalize family of SCC-based algorithms
  - Enumerate SCCs
  - GBA emptiness check
  - ...

- Approach: Formalize “skeleton” SCC algorithm first
  - Just the node-contracting DFS, no output
  - Theorems for VCs (invariant preservation, . . .)
  - Stepwise refinement to executable code

- Reuse this formalization for actual algorithms
Re-usable Formalization

- Goal: Formalize family of SCC-based algorithms
  - Enumerate SCCs
  - GBA emptiness check
  - ...
- Approach: Formalize “skeleton” SCC algorithm first
  - Just the node-contracting DFS, no output
  - Theorems for VCs (invariant preservation, ...)
  - Stepwise refinement to executable code
- Reuse this formalization for actual algorithms
- Utilize existing Isabelle technologies
  - Collection Framework, Refinement Framework, Autoref tool
  - Code generator, locales
Design of the Formalization

Skeleton Specification
Design of the Formalization

Skeleton Specification \(\subseteq\) Abstract Skeleton Algorithm
Design of the Formalization

Skeleton Specification

\[ \subseteq \]

Abstract Skeleton Algorithm

\[ \subseteq \]

Gabow’s Implementation

\[ \subseteq \]
Design of the Formalization

- Skeleton Specification
  - Abstract Skeleton Algorithm
    - Gabow's Implementation
      - SML Code
Design of the Formalization

Skeleton Specification ▶️ Actual Spec

Abstract Skeleton Algorithm

Gabow’s Implementation

SML Code
Design of the Formalization

Skeleton Specification \( \subseteq \) Abstract Skeleton Algorithm \( \subseteq \) Gabow’s Implementation \( \subseteq \) SML Code

Actual Spec \( \subseteq \) Actual Algo \( \subseteq \)

\text{re-use}
Design of the Formalization

Skeleton Specification \(\sqsubseteq\) Actual Spec

\[\begin{align*}
\text{Abstract Skeleton Algorithm} & \quad \sqsubseteq \quad \text{Actual Algo} \\
\text{Gabow's Implementation} & \quad \sqsubseteq \quad \text{Actual Impl} \\
\text{SML Code} & \quad \sqsubseteq 
\end{align*}\]
Design of the Formalization

Skeleton Specification
  ▼
  Abstract Skeleton Algorithm
    ▼
    Gabow’s Implementation
      ▼
      SML Code
  ▼
  Actual Spec
  ▼
  Actual Algo
    ▼
    Actual Impl
      ▼
      SML Code

re-use
Isabelle Refinement Framework

- Nondeterministic monadic programs

```
skeleton ≡ do {
  let D = {}; 
  r ← FOREACHi outer_invar V0 (λv0 D0. do {
    if v0∉D0 then do {
      let s = initial v0 D0;
      (p,D,pE) ← WHILEIT (invar v0 D0) (λ(p,D,pE). p ≠ [[]]) (λ(p,D,pE). do {
        (vo,(p,D,pE)) ← select_edge (p,D,pE);
        case vo of
          Some v ⇒
          if v ∈ ∪ set p then RETURN (collapse v (p,D,pE))
          else if v∉D then RETURN (push v (p,D,pE))
          else RETURN (p,D,pE)
        | None ⇒ RETURN (pop (p,D,pE))
      }) s;
      RETURN D 
    } else RETURN D0
  }) D;
  RETURN r }
```
Isabelle Refinement Framework

- Nondeterministic monadic programs
- Supports stepwise refinement
- Verification Condition Generator

```
lemma "skeleton_impl ≤ ↓oGS_rel skeleton"
  unfolding skeleton_impl_def skeleton_def
  by (refine_rcg skeleton_refines)
  (vc_solve (nopre) solve: asm_rl I_to_outer simp: skeleton_refine_simps)
```
Autoref-Tool and Collections Framework

- Automatic Refinement Tool (Autoref)
  - Parametricity-based approach to data refinement
  - Automatic synthesis of implementation from abstract program
- Isabelle Collection Framework
  - Efficient data structures (Array, Hash-Table, Bitvector, ...)
  - Generic Algorithm Library
  - Integrated with Autoref

schematic_lemma skeleton_code_aux:

"(RETURN ?skeleton_tr,skeleton_impl) ∈ ⟨oGSi_rel⟩nres_rel"

unfolding ... by autoref

export_code skeleton_tr in SML file "gabow.sml"
Re-use of Invariants

- Exploit locale mechanism to define extended invariants
- Set up VCG: Only preservation of extension needs to be proved

```plaintext
locale invar  -- "Invariants of Skeleton"
locale csc_c_invar_ext  -- "Additional invariants"
locale csc_c_invar = invar + csc_c_invar_ext  -- "Combined invariant"

lemma csc_c_invarI:
  assumes "invar s"
  assumes "invar s \implies csc_c_invar_ext (l,s)"
  shows "csc_c_invar (l,s)"
```
Re-use of Refinements

- Use basic operations in extended algorithm
- Re-use refinements for basic operations

```plaintext
compute_SCC ≜ ...
| None ⇒ do {
    (* No more outgoing edges from current node on path *)
    ASSERT (pE ∩ last p × UNIV = {});
    let V = last p;
    let (p,D,pE) = pop (p,D,pE);
    let l = V#l;
    RETURN (l,p,D,pE)
}
```

lemma compute_SCC_impl_refine: "compute_SCC_impl ⊆ ▼Id compute_SCC"

proof -

... 

show ?thesis
  unfolding compute_SCC_impl_def compute_SCC_def
  apply (refine_rcg ... pop_refine ...)
  by (vc_solve ...)

qed
Outline

1 Gabow’s SCC Algorithm
2 Isabelle/HOL Formalization
3 Performance Evaluation
Benchmark against Java Reference Implementation

Time in ms vs Number of edges for Poly/ML, MLton, Java, and Java*.
Conclusions

- Efficient, extensible formalization of Gabow’s Algorithm
  - Performance comparable to Java implementation ($\times 3 \ldots \times 4$)
  - Variants: Enumerate SCCs, emptiness check for GBA
- Used by the CAVA fully verified LTL model checker [CAV ‘13]
- Example of verified algorithm design in Isabelle/HOL
  - Using Collection/Refinement/Autoref framework [ITP ’10,’12,’13]
  - Refinement separates algorithmic ideas from implementation
  - Sharing of proofs between variants of the algorithm
Questions?
Remarks?