Semi-automatic proof of Strong connectivity

jean-jacques.levy@inria.fr

journées PPS, 12-10-2017
Plan

- motivation
- algorithm
- formal proof
- other systems
- conclusion

.. joint work (in progress) with Ran Chen [VSTTE 2017]

also cooperation with Cyril Cohen, Laurent Théry, Stephan Merz
Motivation

- nice algorithms \(\rightarrow\) **simple** formal proofs
- **fully** published in articles or journals
- how to publish formal proofs?
- formal proofs should be **exact** and **readable** (by human)
- mix automatic and interactive proofs
- first-order logic is **easy** to understand, but **not** expressive
- algorithms on graphs = a good testbed
One-pass linear-time algorithm

[tarjan 1972]
Depth-first-search

graph

spanning tree (forest)
The algorithm (1/3)

3 SCCs (strongly connected components)

3 vertices are their bases
The algorithm (2/3)

\[
\text{LOWLINK}(x) = \min \left( \{ \text{num}[x] \} \cup \{ \text{num}[y] \mid x \rightarrow^* y \right)
\]
\(\wedge x \text{ and } y \text{ are in same connected component} \}

\[
\text{LOWLINK}(x) = \min \left( \{ \text{num}[x] \} \cup \{ \text{num}[y] \mid x \rightarrow^* y \right)
\]
\(\wedge x \text{ and } y \text{ are in same connected component} \}

The algorithm (3/3)
The program

```ocaml
let rec printSCC (x: int) (s: stack int)
  (num: array int) (sn: ref int) =
    Stack.push x s;
  num[x] <- !sn; sn := !sn + 1;
  let low = ref num[x] in
  foreach y in (successors x) do
    let m = if num[y] = -1
      then printSCC y s num sn
      else num[y] in
    low := Math.min m !low
    done;
  if !low = num[x] then begin
    repeat
      let y = Stack.pop s in
      Printf.Printf "%d " y;
      num[y] <- max_int;
      if y = x then break;
    done;
    Printf.Printf "\n"
    low := max_int;
  end;
  return !low;
```

- print each component on a line
Proof in algorithms books (1/2)

- consider the spanning trees (forest)
- tree structure of strongly connected components
- 2-3 lemmas about ancestors in spanning trees

**Lemma 10.** Let \( v \) and \( w \) be vertices in \( G \) which lie in the same strongly connected component. Let \( F \) be a spanning forest of \( G \) generated by repeated depth-first search. Then \( v \) and \( w \) have a common ancestor in \( F \). Further, if \( u \) is the highest numbered common ancestor of \( v \) and \( w \), then \( u \) lies in the same strongly connected component as \( v \) and \( w \).

\[
LOWLINK(x) = \min \left( \{num[x]\} \cup \{num[y] \mid x \stackrel{*}{\Rightarrow} y \wedge x \text{ and } y \text{ are in same connected component}\} \right)
\]

**Lemma 12.** Let \( G \) be a directed graph with LOWLINK defined as above relative to some spanning forest \( F \) of \( G \) generated by depth-first search. Then \( v \) is the root of some strongly connected component of \( G \) if and only if \( LOWLINK(v) = v \).
Proof in algorithms book (2/2)

• give the program

• proof program

• that part of the proof is very informal
Our program (1/3)

let rec dfs1 x e =
    let n = e.sn in
    let (n1, e1) = dfs (successors x) (add_stack_incr x e) in
    let (s2, s3) = split x e1.stack in
    if n1 < n then (n1, e1) else
        (max_int(), {stack = s3; sccs = add (elements s2) e1.sccs;
                      sn = e1.sn; num = set_max_int s2 e1.num})

with dfs roots e = if is_empty roots then (max_int(), e) else
    let x = choose roots in
    let roots’ = remove x roots in
    let (n1, e1) = if e.num[x] ≠ -1 then (e.num[x], e) else dfs1 x e in
    let (n2, e2) = dfs roots’ e1 in (min n1 n2, e2)

let tarjan () =
    let e0 = {stack = Nil; sccs = empty; sn = 0; num = const (-1)} in
    let (_, e’) = dfs vertices e0 in e’.sccs

returns LOWLINK(x) and new environment
Formal proof using Why3
Plan of proof (1/2)

- define **reachability** in graphs and SCCs
- prove a few lemmas about positions in stacks (**ranks**)
- define **invariants** on environments
- give **pre-post conditions** for functions
- add a few intermediate **assertions** in function bodies

- avoid paths, prefer edges
Plan of proof (2/2)

• vertices have colors
  - white = unvisited
  - gray = being visited
  - black = visited

• invariant on environment

vertex in stack reaches all vertices with higher rank
Invariants

def type env = {ghost blacks: set vertex; ghost grays: set vertex;
stack: list vertex; sccs: set (set vertex);
sn: int; num: map vertex int}
let rec dfs1 x e =

requires {mem x vertices} (* R1 *)
requires {access_to e.grays x} (* R2 *)
requires {not mem x (union e.blacks e.grays)} (* R3 *)

\[
\begin{align*}
\text{e.sccs} & \subseteq \text{e'.sccs} \\
\text{e.blacks} & \subseteq \text{e'.blacks} \\
\text{e.grays} & = \text{e'.grays}
\end{align*}
\]
let n = e.sn in
let (n1, e1) =
  dfs' (successors x) (add_stack_incr x e) in
let (s2, s3) = split x e1.stack in

if n1 < n then begin
  (n1, add_blacks x e1) end
else begin
  (max_int(), {blacks = add x e1.blacks; grays = e.grays;
    stack = s3; sccs = add (elements s2) e1.sccs;
    sn = e1.sn; num = set_max_int s2 e1.num}) end

[ http://jeanjacqueslevy.net/why3/graph/abs/scct/1-7/scc.html ]
Assertions

assert {forall y. in_same_scc y x -> lmem y s2};

- proof by contradiction: \( \exists y, \ in\_same\_scc\ y\ x \land y \notin s2 \)
- \( \exists x' y', \ reachable\ x\ x' \land edge\ x'\ y' \land reachable\ y'\ y \land x' \in s2 \land y' \notin s2 \)
- 3 cases:

  [1] \( y' \) is white
     \( x' = x \) then \( y' \in successors\ x \rightarrow y' \) is black
     \( x' \neq x \) then \( x' \) is black \( \rightarrow \) \( \neg \) no\_black\_to\_white \( b_1 g_1 \)

  [2] \( y' \in e1.sccs \) then \( in\_same\_scc\ y'\ x \rightarrow x \) is black

  [3] \( y' \in s3 \rightarrow rank\ y'\ s1 < rank\ x\ s1 \rightarrow e1.num[y'] < e1.num[x] = e.num[x] = n \)
     \( x' = x \) then \( y' \in successors\ x \rightarrow n1 \leq e1.num[y'] \)
     \( x' \neq x \) then \( xedge\_to\ s1\ (Cons\ x\ s3)\ y' \)
## Proof stats

<table>
<thead>
<tr>
<th>prover</th>
<th>Alt-Ergo</th>
<th>CVC3</th>
<th>CVC4</th>
<th>Coq</th>
<th>E-prover</th>
<th>Spass</th>
<th>Yices</th>
<th>Z3</th>
<th>all</th>
<th>#VC</th>
<th>#PO</th>
</tr>
</thead>
<tbody>
<tr>
<td>38 lemmas split</td>
<td>2.35</td>
<td>0.23</td>
<td>5.79</td>
<td>0.66</td>
<td>0.75</td>
<td>0.21</td>
<td>9.99</td>
<td>77</td>
<td>38</td>
<td></td>
<td></td>
</tr>
<tr>
<td>add_stack_incr</td>
<td>0.01</td>
<td>0.2</td>
<td></td>
<td>0.01</td>
<td>0.01</td>
<td>0.02</td>
<td>0</td>
<td>6</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>add_blacks</td>
<td>0.01</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>set_max_int</td>
<td>0.02</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dfs1</td>
<td>53.52</td>
<td>12.88</td>
<td>36.39</td>
<td>3.06</td>
<td>28.06</td>
<td>9.01</td>
<td>142.92</td>
<td>218</td>
<td>24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dfs</td>
<td>4.6</td>
<td>0.23</td>
<td>11.63</td>
<td></td>
<td>0.31</td>
<td>16.77</td>
<td>51</td>
<td>35</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tarjan</td>
<td>0.44</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.44</td>
<td>16</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>total</td>
<td>61.04</td>
<td>13.54</td>
<td>53.81</td>
<td>3.06</td>
<td>28.72</td>
<td>0.75</td>
<td>0.21</td>
<td>9.32</td>
<td>170.45</td>
<td>371</td>
<td>112</td>
</tr>
</tbody>
</table>

[ http://jeanjacqueslevy.net/why3/graph/abs/scct/1-7/scc.html ]
Other systems
Coq / ssreflect

[cyril cohen, laurent théry, J JL]

- port in 1 week
- graphs and finite sets already in mathematical components
- problems with termination (hacky & higher-order)
- 920 lines

[http://github.com/CohenCyril/tarjan]
Isabelle / HOL

[stephan merz]

• port in 1 month
• use many strategies (metis, blast, sledgehammer)
• still problems with proving termination
• 31 pages

F*

[kenji maillard, catalin hritcu]

- start discuss with them
- Z3 single automatic prover
- ??
Future work

- library for formal proofs on graphs
- other graph algorithms
- **beyond** graphs …
- teaching formal methods on **test cases**
- **imperative** programs

[http://jeanjacqueslevy.net/why3]