Comparing a Formal Proof

in Why3, Coq, Isabelle

Jean-Jacques Lévy
Inria Paris - Irif

VIP meeting 20-11-201

Motivation

- nice algorithms should have simple formal proofs
- to be fully published in articles or journals
- how to publish formal proofs ?
- algorithms on graphs = a good testbed (better than $\sqrt{2}$)
- formal proofs have to be checked by computer

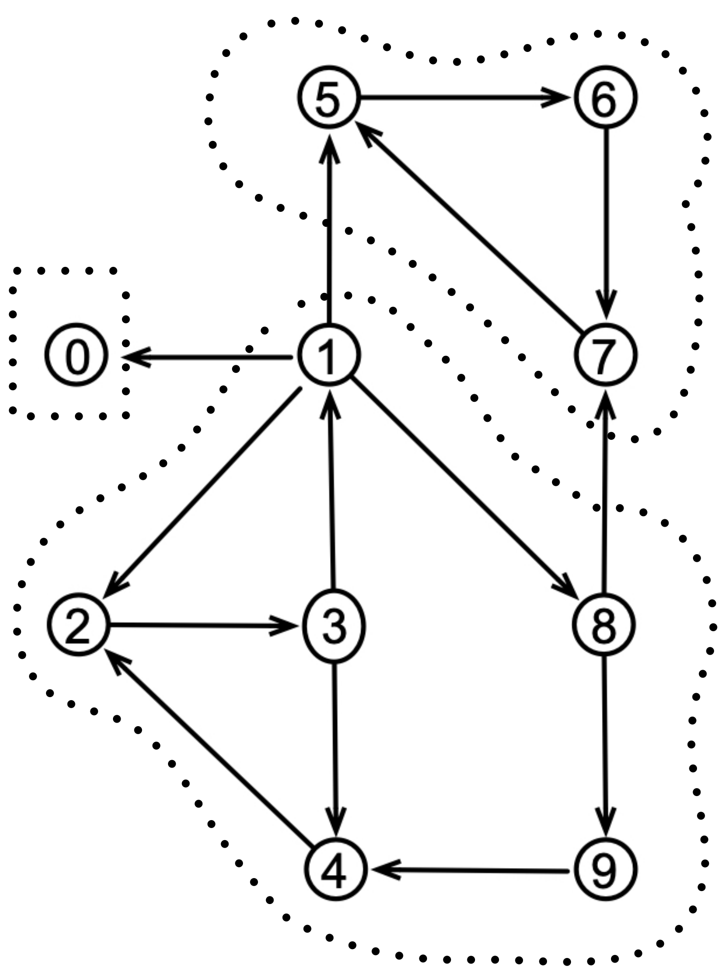
.. with Ran Chen, Cyril Cohen, Stephan Merz, Laurent Théry VSTTE 2017, CPP 2019 (?)

`http://www-sop.inria.fr/marelle/Tarjan/contributions.html`

A one-pass linear-time algorithm

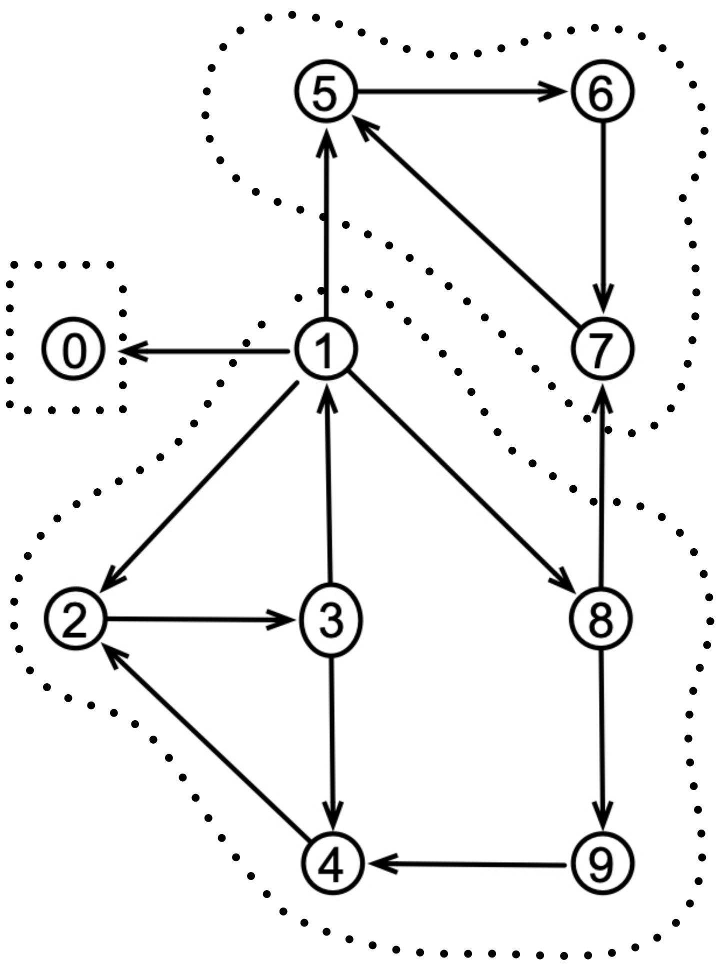
Tarjan, 1972

Strongly connected components

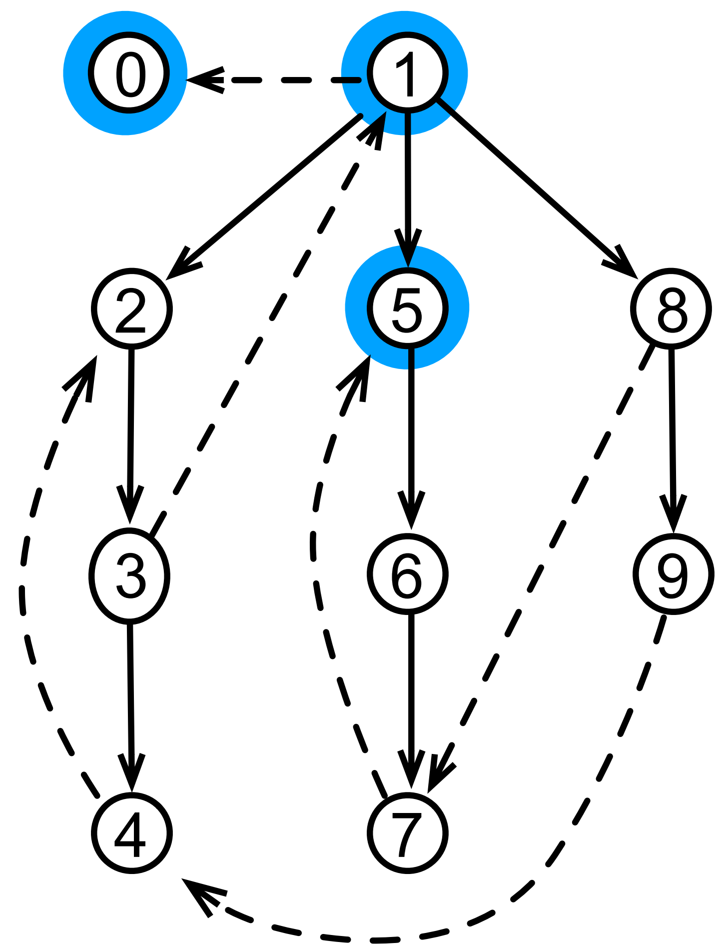


- maximum subsets of vertices connected by a path
- depth-first search algorithm pushing vertices on a stack in their order of visit
- computing oldest vertex reachable by at most one « cross-edge »
- when not strictly less than currently visited vertex, a new SCC is on top of current vertex in working stack
- the SCC is popped and algorithm continues

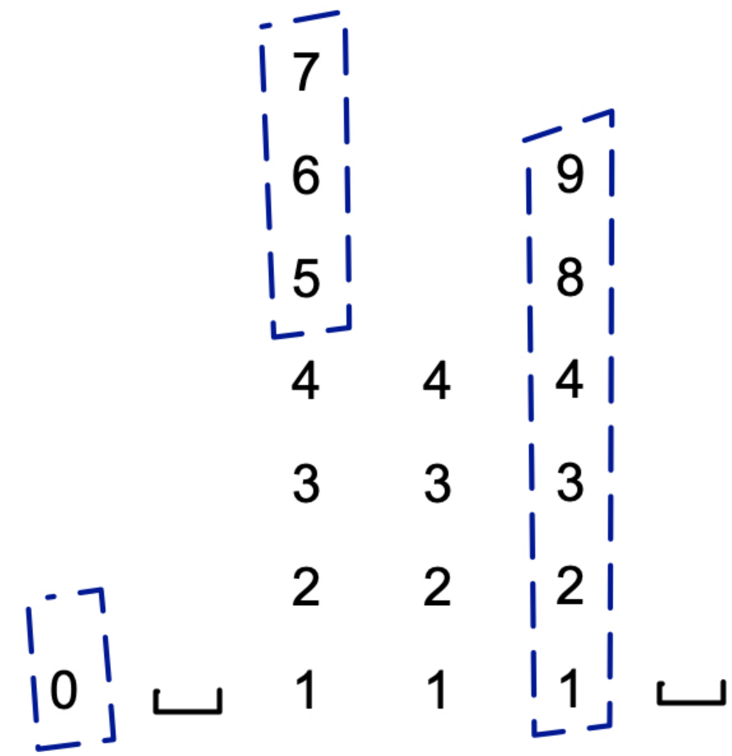
Strongly connected components



graph



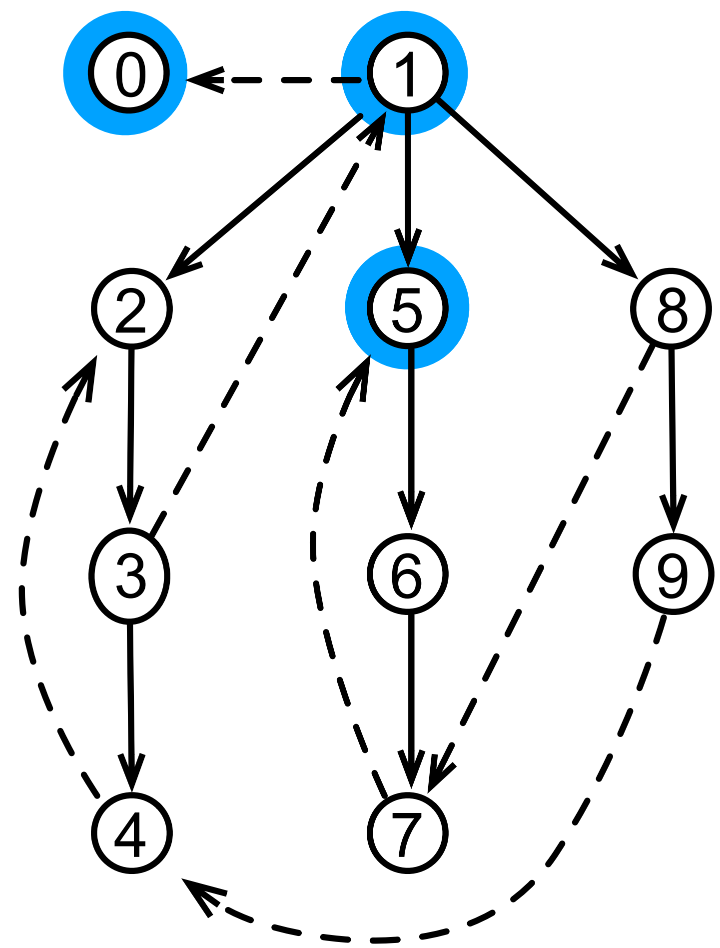
spanning forrest



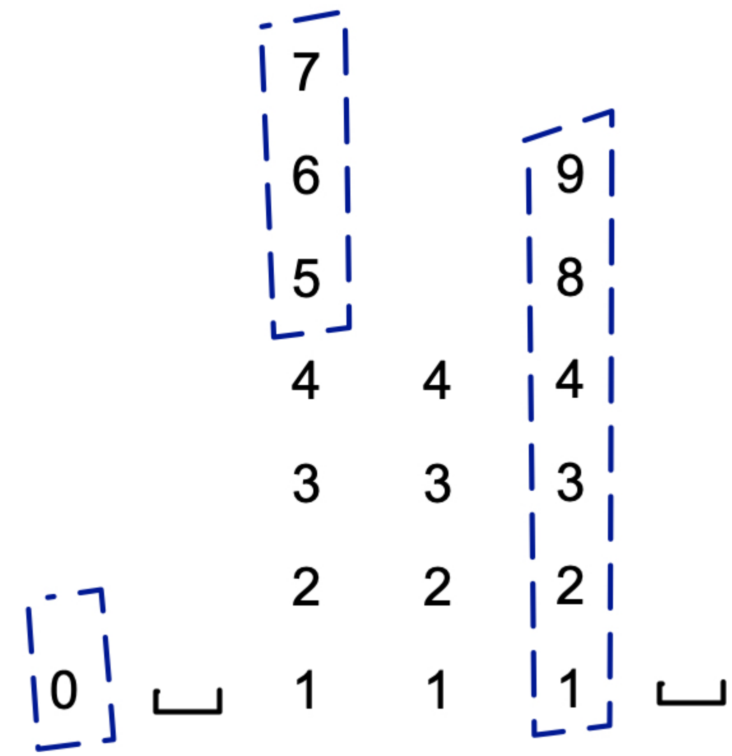
stack

Strongly connected components

0	$+\infty$
1	1
2	1
3	1
4	2
5	5
6	5
7	5
8	4
9	4



spanning forrest



stack

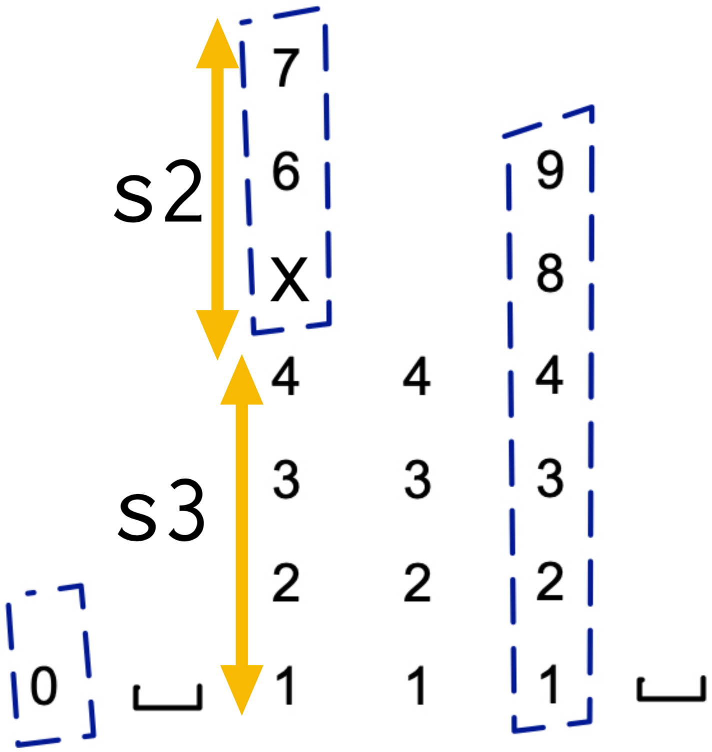
$$LOWLINK(x) = \min\{num[y] \mid x \overset{*}{\Longrightarrow} z \hookrightarrow y$$
$$\wedge\ x \text{ and } y \text{ are in the same connected component}\}$$

Program

```
type vertex
constant vertices: set vertex
function successors vertex : set vertex
type env = {stack: list vertex;
            sccs: set (set vertex);
            sn: int; num: map vertex int}
```

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

Program




```
let rec dfs1 x e =
  let n0 = e.sn in
  let (n1, e1) = dfs (successors x)
        (add_stack_incr x e) in
  if n1 < n0 then (n1, e1) else
    let (s2, s3) = split x e1.stack in
    (+∞, {stack = s3;
       sccs = add (elements s2) e1.sccs;
       sn = e1.sn; num = set_infty s2 e1.num})
```

```
with dfs r e = if is_empty r then (+∞, e) else
  let x = choose r in
  let r' = remove x r in
  let (n1, e1) = if e.num[x] ≠ -1
     then (e.num[x], e) else dfs1 x e in
  let (n2, e2) = dfs r' e1 in (min n1 n2, e2)
```

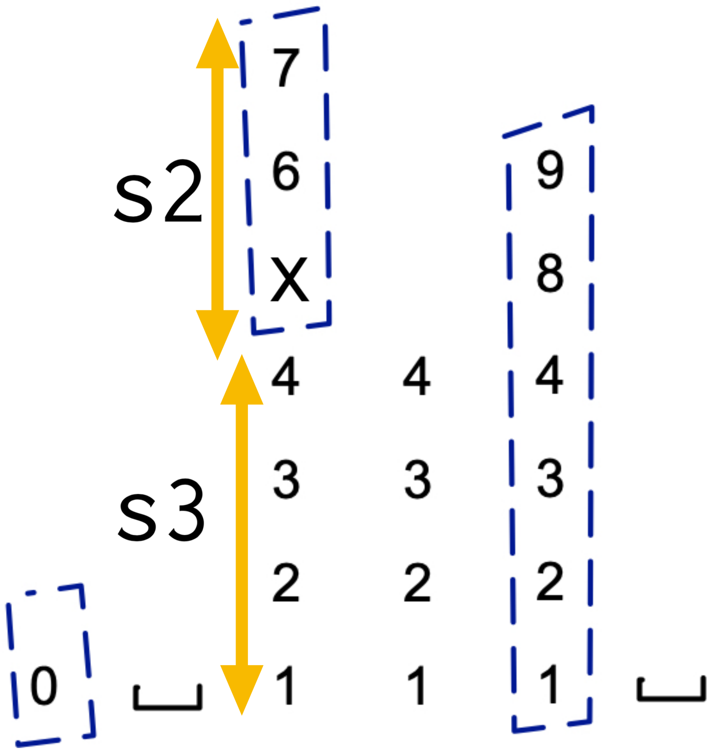
Proof

Program

```
type vertex
constant vertices: set vertex
function successors vertex : set vertex
type env = {ghost black: set vertex;
    ghost gray: set vertex;
    stack: list vertex; sccs: set (set vertex);
    sn: int; num: map vertex int}
```

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

Program




```
let rec dfs1 x e =
  let n0 = e.sn in
  let (n1, e1) = dfs (successors x)
            (add_stack_incr x e) in
  if n1 < n0 then (n1, add_black x e1) else
    let (s2, s3) = split x e1.stack in
    (+∞, {stack = s3;
       black = add x e1.black; gray = e.gray;
       sccs = add (elements s2) e1.sccs;
       sn = e1.sn; num = set_infty s2 e1.num})
```

```
let add_stack_incr x e =
  let n = e.sn in
  {black = e.black; gray = add x e.gray;
   stack = Cons x e.stack; sccs = e.sccs;
   sn = n+1; num = e.num[x ←n]}
```

```
let add_black x e =
 {black = add x e.black; gray = remove x e.gray;
  stack = e.stack; sccs = e.sccs;
  sn = e.sn; num = e.num}
```

Invariant

(1) consistent colors

(2) consistent numbering

(3) vertices pairwise **distinct** in stack

(4) no edge from black to white

(5) in stack any vertex reaches any **higher** vertex

(6) in stack any vertex reaches a **gray lower** vertex

(7) the sccs field is the set of **black** SCCs

Why3 Proof

Pre/Post-conditions

```
let rec dfs1 x e  =
(* pre-condition *)
requires{mem x vertices}
requires{∀y. mem y e.gray → reachable y x}
requires{not mem x (union e.black e.gray)}
requires{wf_env e} (* I *)


(* post-condition *)
returns{(_, e') → wf_env e' ∧ subenv e e'}
returns{(_, e') → mem x e'.black}
returns{(n, e') → n ≤ e'.num[x]}
returns{(n, e') → n = +∞ ∨ num_of_reachable n x e'}
returns{(n, e') → ∀y. xedge_to e'.stack e.stack y
                     → n ≤ e'.num[y]}
```

LOWLINK

Assertions

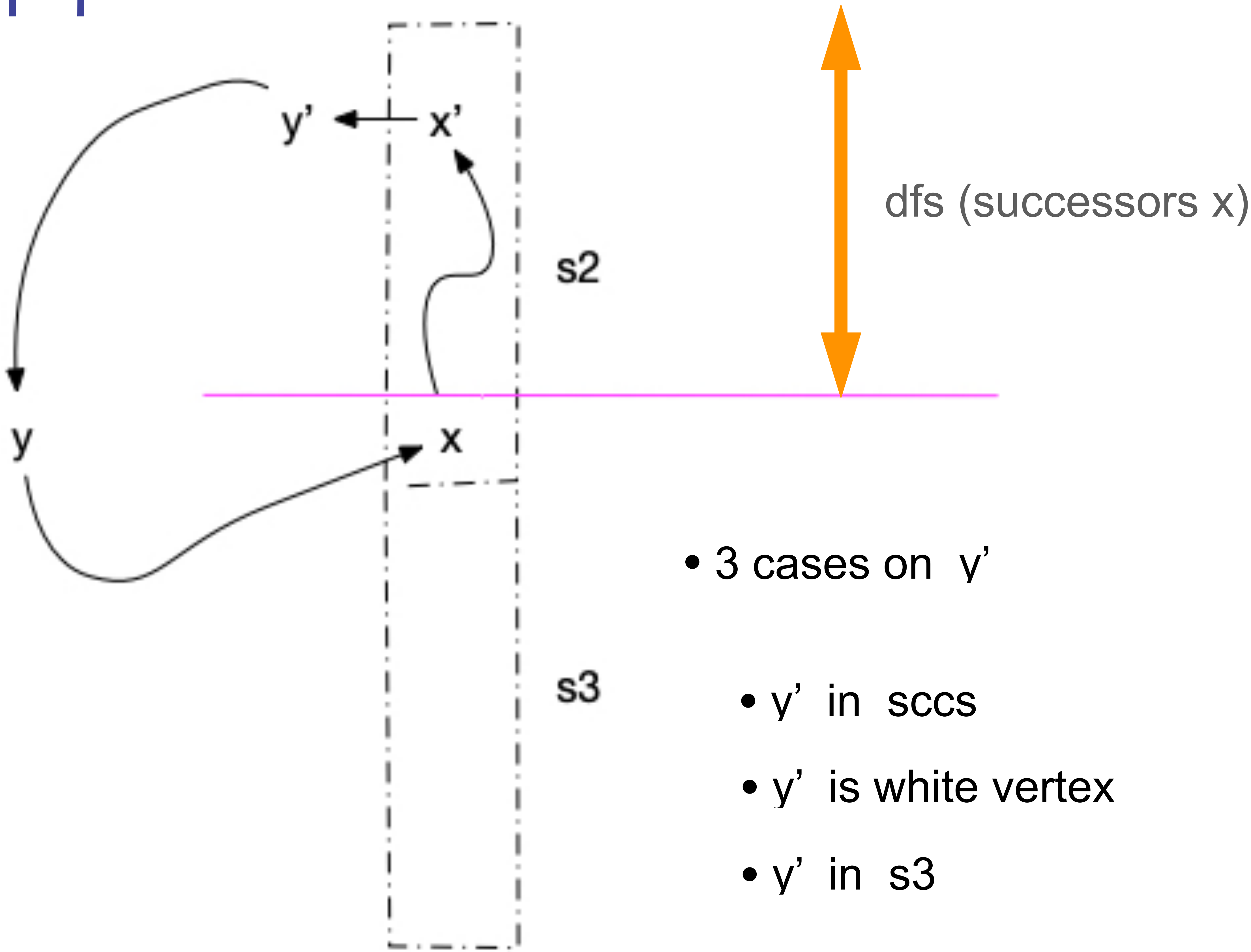
```
let n0 = e.sn in
let (n1, e1) =
  dfs (successors x) (add_stack_incr x e) in
if n1 < n0 then begin
  assert{∃y. y ≠ x ∧ precedes x y e1.stack ∧
     reachable x y};
  assert{∃y. y ≠ x ∧ mem y e1.gray ∧
     e1.num[y] < e1.num[x] ∧ in_same_scc x y};
  (n1, add_black x e1) end
```

Assertions

```
else
  let (s2, s3) = split x e1.stack in
  assert{is_last x s2 ∧ s3 = e.stack ∧
     subset (elements s2) (add x e1.black)};
  assert{is_subscc (elements s2)};
  assert{∀y. in_same_scc y x → lmem y s2};
  assert{is_scc (elements s2)};
  assert{inter e.gray (elements s2) == empty};
  (+∞, {black = add x e1.black; gray = e.gray;
     stack = s3; sccs = add (elements s2) e1.sccs;
     sn = e1.sn; num = set_infty s2 e1.num})
```

Coq

Coq proof



- 3 cases on y'
  - y' in sccs
  - y' is white vertex
  - y' in s3

Assertions

provers	Alt-Ergo	CVC4	E-prover	Z3	#VC	#PO
49 lemmas	1.91	26.11	3.33		70	49
split	0.09	0.16			6	6
add_stack_incr	0.01				1	1
add_black	0.02				1	1
set_infty	0.03				1	1
dfs1	77.89	150.2	19.99	13.67	79	20
dfs	4.71	3.52		0.26	58	25
tarjan	0.85				15	5
total	85.51	179.99	23.32	13.93	231	108

Table 1. Performance results of the provers (in seconds, on a 3.3 GHz Intel Core i5 processor). Total time is 341.15 seconds. The two last columns contain the numbers of verification conditions and proof obligations. Notice that there may be several VCs per proof obligation.

+ 2 Coq proofs (16 loc + 141 loc)

Coq Proof

Functions

```
Record env := Env {black : {set V};
                   stack : seq V;
                   sccs : {set {set V}};
                   sn : nat;  num : {ffun V ➔ nat}}.
```

```
Definition dfs1  dfs  x  e :=
 let: (n1, e1) :=
  dfs [set y  in  successors  x] (add_stack  x  e) in
  if  n1 < sn e  then  (n1,  add_black  x  e1)
  else (infty,  add_sccs  x  e1).
```

```
Definition dfs  dfs1  dfs'  r  e :=
 if  [pick x in r]  isn't  Some x  then  (infty, e)
 else  let r' := r :\ x in
         let: (n1, e1) :=
           if num  e  x != 0 then (num  e  x, e)  else  dfs1  x  e in
         let: (n2, e2) := dfs'  r'  e1   in   (minn  n1  n2,  e2).
```

Functions

```
Fixpoint tarjan_rec  n :=
 if n  is  n1.+1  then
  dfs  (dfs1 (tarjan_rec  n1))  (tarjan_rec  n1)
 else fun  r  e  =>  (infty,  e).
```

```
Let N := #|V| * #|V|.+1 + #|V|.
Definition tarjan := sccs (tarjan_rec  N  setT  e0).2.
```

Proof

```
Definition dfs_correct
    (dfs : {set V} ➔ env ➔ nat * env)  r  e  :=
    pre_dfs  r  e  ➔
    let (n, e’) := dfs  r  e  in  post_dfs  r  e  e’  n.
```

```
Definition dfs1_correct
    (dfs1 : V ➔ env ➔ nat * env)  x  e :=
    (x ∈ white e)  ➔  pre_dfs  [set x]  e ➔
    let (n, e’) :=  dfs1  x  e  in  post_dfs  [set  x]  e  e’  n.
```

Proof

```
Lemma dfs_is_correct  dfs1'  dfs'  (r : {set V})  e :
  (∀x, x ∈ r ➔ dfs1_correct  dfs1'  x  e) ➔
  (∀x, x ∈ r ➔ ∀e1,  white  e1  \subset  white  e ➔
        dfs_correct  dfs'  (r  :\  x)  e1) ➔
  dfs_correct (dfs  dfs1'  dfs') r   e.
```

```
Lemma dfs1_is_correct  dfs'  (x : V)  e :
  (dfs_correct  dfs'  [set y | edge x y]  (add_stack  x  e)) ➔
  dfs1_correct  (dfs1  dfs') x  e.
```

```
Theorem  tarjan_rec_terminates  n  r  e :
n ≥ #|white  e| * #|V|.+1  + #|r|   ➔
dfs_correct  (tarjan_rec  n)  r  e.
```

Isabelle/HOL Proof

Proof

```
function (domintros) dfs1  and  dfs  where
 dfs1 x e =
  (let (n1, e1) = dfs (successors x) (add_stack_incr x e) in
   if  n1 < int (sn e)  then  (n1, add_black x e1)
   else (let (l, r) = split_list x (stack e1)  in
      (+∞, (| black = insert x (black e1), gray = gray e,
            stack = r, sn = sn e1, sccs = insert (set l) (sccs e1),
            num = set_infty l (num e1) |) )))

and
dfs roots e =
  (if roots = {} then (+∞, e)
   else (let x = SOME x . x ∈ roots;
         res1 = (if num e x ≠ -1 then (num e x, e) else dfs1 x e);
         res2 = dfs (roots - {x}) (snd res1)
    in   (min (fst res1) (fst res2),  snd res2) ))
```

Proof

definition colored_num **where** colored_num e $\equiv$

$\forall v \in$ colored e. $v \in$ vertices $\wedge$ num e v $\neq -1$

theorem dfs1_dfs_termination :

[x $\in$ vertices - colored e; colored_num e] $\Longrightarrow$ dfs1_dfs_dom (Inl(x, e))

[r $\subseteq$ vertices; colored_num e] $\Longrightarrow$ dfs1_dfs_dom (Inr(r, e))

theorem dfs_partial_correct:

[dfs1_dfs_dom (Inl(x, e)); dfs1_pre x e] $\Longrightarrow$ dfs1_post x e (dfs1 x e)

[dfs1_dfs_dom (Inr(r, e)); dfs_pre r e] $\Longrightarrow$ dfs_post r e (dfs r e)

theorem dfs_correct:

dfs1_pre x e $\Longrightarrow$ dfs1_post x e (dfs1 x e)

dfs_pre r e $\Longrightarrow$ dfs_post roots e (dfs r e)

Conclusion

Why3 - Coq - Isabelle

	why3	coq	isabelle/HOL
expressivity	-	++	+
readability	+++	-	+
stability	-	+++	+
ease of use	-	-	-
automation	++	-	+
partial correctness	+++	- -	-
code extraction	++	+	-
trusted base	-	+++	+++
# lines auto	392	0	? (314ui)
# lines manual	157	1535	1690

… other systems ?

`http://www-sop.inria.fr/marelle/Tarjan/contributions.html`

Todo list

- proof of implementation
- other algorithms (biconnected, planarity, minimum spanning tree)
- teaching