



# Verification in HOL of an algorithm for computing SCCs

Vincent Trélat

École Nationale Supérieure des Mines de Nancy  
Département Informatique

January 14, 2022

## ① Introduction

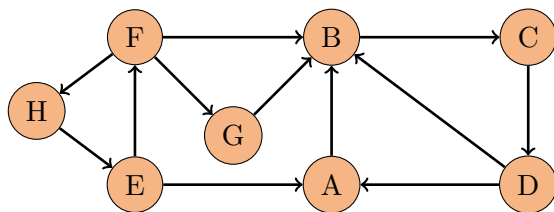
Definition

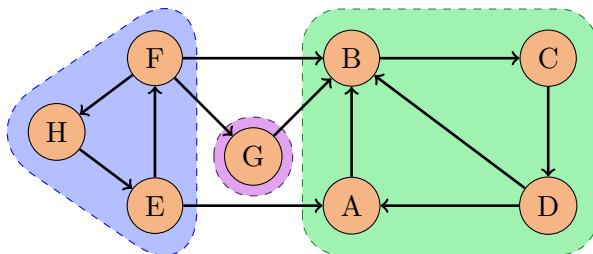
Motivation

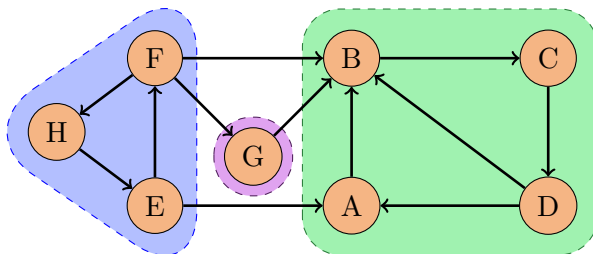
Example of the proof process

## ② Correctness of the SCC algorithm

Description of the algorithm



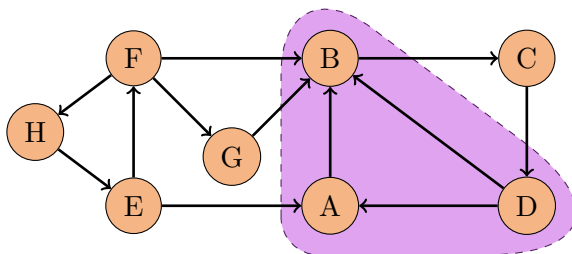




## Definition 1

Let  $\mathcal{G} := (\mathcal{V}, \mathcal{E})$  be a **directed graph** and  $\mathcal{C} \subseteq \mathcal{V}$ .  $\mathcal{C}$  is a SCC of  $\mathcal{G}$  if:

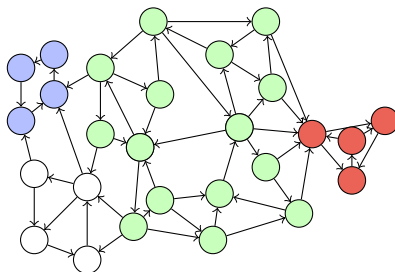
$$\forall x, y \in \mathcal{C}, (x \Rightarrow y) \wedge (y \Rightarrow x)$$



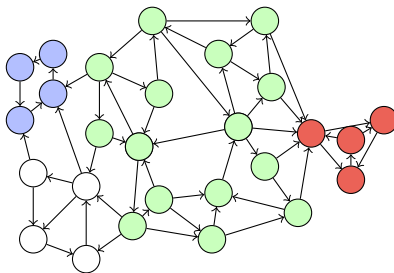
## Definition 1

Let  $\mathcal{G} := (\mathcal{V}, \mathcal{E})$  be a **directed graph** and  $\mathcal{C} \subseteq \mathcal{V}$ .  $\mathcal{C}$  is a SCC of  $\mathcal{G}$  if:

$$\forall x, y \in \mathcal{C}, (x \Rightarrow y) \wedge (y \Rightarrow x)$$

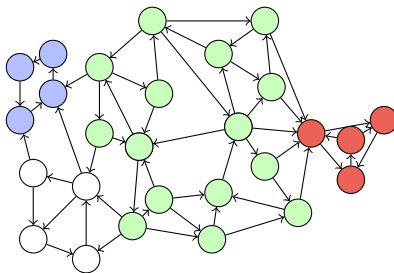


## Motivation



- Networks: connection and data sharing
- Model checking: counter-examples finding





- Networks: connection and data sharing
- Model checking: counter-examples finding

Efficient algorithms (ex: Tarjan)

- Formal verification of correctness is worthwhile
- Parallelization is another challenge

## Isabelle / HOL

- Generic proof assistant
- Formalisation of mathematical proofs
- Higher-Order Logic theorem proving environment
- Powerful proof tools and language (Isar)
- Mutual induction, recursion and datatypes, complex pattern matching

## Example

Simple proofs on a basic data structure

## (Type definition)

```
datatype 'a list = Empty | Cons 'a "'a list"
```

- Generic / polymorphic and static type
- Implicit constructor definition
- Recursive structure giving an induction principle for that type

## (Function definition)

```
fun concat :: "'a list ⇒ 'a list ⇒ 'a list" where  
  "concat Empty xs = xs"  
| "concat (Cons x xs) ys = Cons x (concat xs ys)"
```

```
fun rev :: "'a list ⇒ 'a list" where  
  "rev Empty = Empty"  
| "rev (Cons x xs) = concat (rev xs) (Cons x Empty)"
```

(Theorem statement)

```
theorem rev_rev [simp]: "rev (rev x) = x"
```

## (Theorem statement)

```
theorem rev_rev [simp]: "rev (rev x) = x"  
  apply (induction x)  
  apply auto
```

### (Theorem statement)

```
theorem rev_rev [simp]: "rev (rev x) = x"  
  apply (induction x)  
  apply auto
```

### (Subgoal)

```
⋀ x1 x.  
  rev (rev x) = x  $\implies$   
  rev (concat (rev x) (Cons x1 Empty)) = Cons x1 x
```



## (Adding a first lemma)

```
lemma rev_concat [simp]:  
  "rev (concat xs ys) = concat (rev ys) (rev xs)"  
  apply (induction xs)  
  apply auto
```

## (Adding a first lemma)

```
lemma rev_concat [simp]:  
  "rev (concat xs ys) = concat (rev ys) (rev xs)"  
  apply (induction xs)  
  apply auto
```

## (Subgoals)

1.  $\text{rev } ys = \text{concat } (\text{rev } ys) \text{ Empty}$
2.  $\bigwedge x1 \ xs.$   
 $\text{rev } (\text{concat } xs \ ys) = \text{concat } (\text{rev } ys) (\text{rev } xs) \implies$   
 $\text{rev } (\text{concat } (\text{Cons } x1 \ xs) \ ys) = \text{concat } (\text{rev } ys) (\text{rev } (\text{Cons } x1 \ xs))$

### (Adding a second lemma)

```
lemma concat_empty [simp]: "concat xs Empty = xs"  
  apply (induction xs)  
  apply auto
```

### (Adding a third lemma: associative property)

```
lemma concat_assoc [simp]: "concat (concat xs ys) zs =  
concat xs (concat ys zs)"  
  apply (induction xs)  
  apply auto
```

```
theorem rev_rev [simp]: "rev (rev x) = x"  
  apply (induction x)  
  apply auto
```

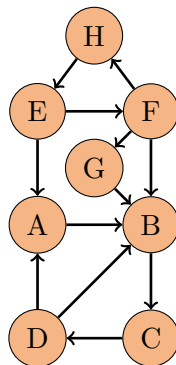
No subgoals!

## A sequential set-based SCC algorithm

**Data:** A graph  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ , a starting node  $v_0$ ;

```

1 Initialize an empty set DEAD;
2 Initialize an empty set VISITED;
3 Initialize an empty stack R;
4 setBased( $v_0$ );
5 function setBased:  $v \in \mathcal{V} \rightarrow \text{None}$ 
6     VISITED := VISITED  $\cup \{v\}$ ;
7     R.push( $v$ );
8     foreach  $w \in \text{POST}(v)$  do
9         if  $w \in \text{DEAD}$  then
10             continue;
11         else if  $w \notin \text{VISITED}$  then
12             setBased( $w$ );
13         else
14             while  $\mathcal{S}(v) \neq \mathcal{S}(w)$  do
15                  $r := \text{R.pop}()$ ;
16                 UNITE( $\mathcal{S}, r, \text{R.top}()$ );
17
18 if  $v = \text{R.top}()$  then
19     report SCC  $\mathcal{S}(v)$ ;
20     DEAD := DEAD  $\cup \mathcal{S}(v)$ ;
21     R.pop();
  
```



$R = \{\}$   
 $VISITED = \{\}$   
 $DEAD = \{\}$

$\mathcal{S} = \{A\} \cup \{B\} \cup \{C\} \cup \{D\} \cup \{E\} \cup \{F\} \cup \{G\} \cup \{H\}$

---

Data: A graph  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ , a starting node  $v_0$ ;

Initialize an empty set DEAD;

2 Initialize an empty set VISITED;

3 Initialize an empty stack R;

4  $\text{setBased}(v_0)$ ;

5 **function**  $\text{setBased}: v \in \mathcal{V} \rightarrow \text{None}$

6     VISITED := VISITED  $\cup \{v\}$ ;

7     R.push( $v$ );

8     **foreach**  $w \in \text{POST}(v)$  **do**

9         **if**  $w \in \text{DEAD}$  **then**

10             continue;

11         **else if**  $w \notin \text{VISITED}$  **then**

12              $\text{setBased}(w)$ ;

13         **else**

14             **while**  $\mathcal{S}(v) \neq \mathcal{S}(w)$  **do**

15                  $r := \text{R.pop}()$ ;

16                  $\text{UNITE}(\mathcal{S}, r, \text{R.top}())$ ;

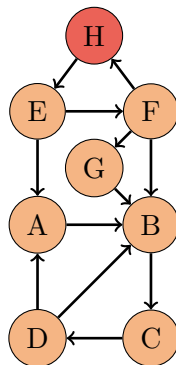
17     **if**  $v = \text{R.top}()$  **then**

18         **report** SCC  $\mathcal{S}(v)$ ;

19          $\text{DEAD} := \text{DEAD} \cup \mathcal{S}(v)$ ;

20          $\text{R.pop}()$ ;

---



$R = \{H\}$

$\text{VISITED} = \{H\}$

$\text{DEAD} = \{\}$

$\mathcal{S} = \{A\} \cup \{B\} \cup \{C\} \cup \{D\} \cup \{E\} \cup \{F\} \cup \{G\} \cup \{H\}$

---

Data: A graph  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ , a starting node  $v_0$ ;

Initialize an empty set DEAD;

2 Initialize an empty set VISITED;

3 Initialize an empty stack R;

4  $\text{setBased}(v_0)$ ;

5 **function**  $\text{setBased}: v \in \mathcal{V} \rightarrow \text{None}$

6     VISITED := VISITED  $\cup \{v\}$ ;

7     R.push( $v$ );

8     **foreach**  $w \in \text{POST}(v)$  **do**

9         **if**  $w \in \text{DEAD}$  **then**

10             continue;

11         **else if**  $w \notin \text{VISITED}$  **then**

12              $\text{setBased}(w)$ ;

13         **else**

14             **while**  $\mathcal{S}(v) \neq \mathcal{S}(w)$  **do**

15                  $r := \text{R.pop}()$ ;

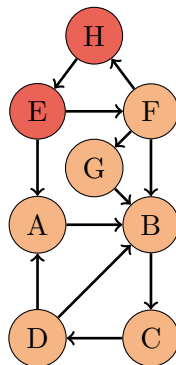
16                  $\text{UNITE}(\mathcal{S}, r, \text{R.top}())$ ;

17         **if**  $v = \text{R.top}()$  **then**

18             **report** SCC  $\mathcal{S}(v)$ ;

19              $\text{DEAD} := \text{DEAD} \cup \mathcal{S}(v)$ ;

20              $\text{R.pop}()$ ;



$R = \{H, E\}$

$\text{VISITED} = \{H, E\}$

$\text{DEAD} = \{\}$

$\mathcal{S} = \{A\} \cup \{B\} \cup \{C\} \cup \{D\} \cup \{E\} \cup \{F\} \cup \{G\} \cup \{H\}$



Data: A graph  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ , a starting node  $v_0$ ;

Initialize an empty set DEAD;

Initialize an empty set VISITED;

Initialize an empty stack R;

setBased( $v_0$ );

**function** setBased:  $v \in \mathcal{V} \rightarrow \text{None}$

    VISITED := VISITED  $\cup \{v\}$ ;

    R.push( $v$ );

**foreach**  $w \in \text{POST}(v)$  **do**

**if**  $w \in \text{DEAD}$  **then**

            continue;

**else if**  $w \notin \text{VISITED}$  **then**

            setBased( $w$ );

**else**

**while**  $\mathcal{S}(v) \neq \mathcal{S}(w)$  **do**

$r := \text{R.pop}()$ ;

                UNITE( $\mathcal{S}, r, \text{R.top}()$ );

**while**  $\mathcal{S}(v) \neq \mathcal{S}(w)$  **do**

$r := \text{R.pop}()$ ;

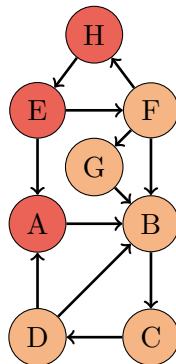
                UNITE( $\mathcal{S}, r, \text{R.top}()$ );

**if**  $v = \text{R.top}()$  **then**

**report** SCC  $\mathcal{S}(v)$ ;

        DEAD := DEAD  $\cup \mathcal{S}(v)$ ;

        R.pop();



$R = \{H, E, A\}$

$\text{VISITED} = \{H, E, A\}$

$\text{DEAD} = \{\}$

$\mathcal{S} = \{A\} \cup \{B\} \cup \{C\} \cup \{D\} \cup \{E\} \cup \{F\} \cup \{G\} \cup \{H\}$

Data: A graph  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ , a starting node  $v_0$ ;

Initialize an empty set DEAD;

2 Initialize an empty set VISITED;

3 Initialize an empty stack R;

4  $\text{setBased}(v_0)$ ;

5 **function**  $\text{setBased}: v \in \mathcal{V} \rightarrow \text{None}$

6     VISITED := VISITED  $\cup \{v\}$ ;

7     R.push( $v$ );

8     **foreach**  $w \in \text{POST}(v)$  **do**

9         **if**  $w \in \text{DEAD}$  **then**

10             continue;

11         **else if**  $w \notin \text{VISITED}$  **then**

12              $\text{setBased}(w)$ ;

13         **else**

14             **while**  $\mathcal{S}(v) \neq \mathcal{S}(w)$  **do**

15                  $r := \text{R.pop}()$ ;

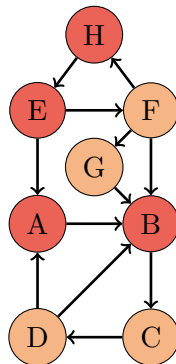
16                  $\text{UNITE}(\mathcal{S}, r, \text{R.top}())$ ;

17         **if**  $v = \text{R.top}()$  **then**

18             **report SCC**  $\mathcal{S}(v)$ ;

19              $\text{DEAD} := \text{DEAD} \cup \mathcal{S}(v)$ ;

20              $\text{R.pop}()$ ;



$R = \{H, E, A, B\}$

$\text{VISITED} = \{H, E, A, B\}$

$\text{DEAD} = \{\}$

$\mathcal{S} = \{A\} \cup \{B\} \cup \{C\} \cup \{D\} \cup \{E\} \cup \{F\} \cup \{G\} \cup \{H\}$

Data: A graph  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ , a starting node  $v_0$ ;

Initialize an empty set DEAD;

2 Initialize an empty set VISITED;

3 Initialize an empty stack R;

4  $\text{setBased}(v_0)$ ;

5 **function**  $\text{setBased}: v \in \mathcal{V} \rightarrow \text{None}$

6     VISITED := VISITED  $\cup \{v\}$ ;

7     R.push( $v$ );

8     **foreach**  $w \in \text{POST}(v)$  **do**

9         **if**  $w \in \text{DEAD}$  **then**

10             continue;

11         **else if**  $w \notin \text{VISITED}$  **then**

12              $\text{setBased}(w)$ ;

13         **else**

14             **while**  $\mathcal{S}(v) \neq \mathcal{S}(w)$  **do**

15                  $r := \text{R.pop}()$ ;

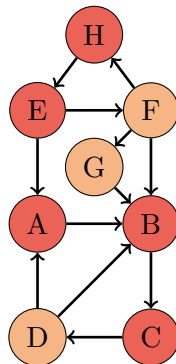
16                  $\text{UNITE}(\mathcal{S}, r, \text{R.top}())$ ;

17             **if**  $v = \text{R.top}()$  **then**

18                 **report** SCC  $\mathcal{S}(v)$ ;

19                  $\text{DEAD} := \text{DEAD} \cup \mathcal{S}(v)$ ;

20                  $\text{R.pop}()$ ;



$R = \{H, E, A, B, C\}$

$\text{VISITED} = \{H, E, A, B, C\}$

$\text{DEAD} = \{\}$

$\mathcal{S} = \{A\} \cup \{B\} \cup \{C\} \cup \{D\} \cup \{E\} \cup \{F\} \cup \{G\} \cup \{H\}$

---

Data: A graph  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ , a starting node  $v_0$ ;

Initialize an empty set DEAD;

2 Initialize an empty set VISITED;

3 Initialize an empty stack R;

4  $\text{setBased}(v_0)$ ;

5 **function**  $\text{setBased}: v \in \mathcal{V} \rightarrow \text{None}$

6     VISITED := VISITED  $\cup \{v\}$ ;

7     R.push( $v$ );

8     **foreach**  $w \in \text{POST}(v)$  **do**

9         **if**  $w \in \text{DEAD}$  **then**

10             continue;

11         **else if**  $w \notin \text{VISITED}$  **then**

12              $\text{setBased}(w)$ ;

13         **else**

14             **while**  $\mathcal{S}(v) \neq \mathcal{S}(w)$  **do**

15                  $r := \text{R.pop}()$ ;

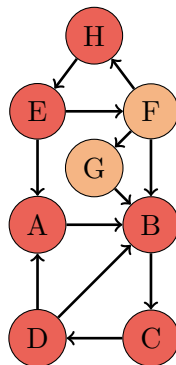
16                  $\text{UNITE}(\mathcal{S}, r, \text{R.top}())$ ;

17         **if**  $v = \text{R.top}()$  **then**

18             **report** SCC  $\mathcal{S}(v)$ ;

19              $\text{DEAD} := \text{DEAD} \cup \mathcal{S}(v)$ ;

20              $\text{R.pop}()$ ;



$R = \{H, E, A, B, C, D\}$

$\text{VISITED} = \{H, E, A, B, C, D\}$

$\text{DEAD} = \{\}$

$\mathcal{S} = \{A\} \cup \{B\} \cup \{C\} \cup \{D\} \cup \{E\} \cup \{F\} \cup \{G\} \cup \{H\}$

---

Data: A graph  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ , a starting node  $v_0$ ;

Initialize an empty set DEAD;

2 Initialize an empty set VISITED;

3 Initialize an empty stack R;

4  $\text{setBased}(v_0)$ ;

5 **function**  $\text{setBased}: v \in \mathcal{V} \rightarrow \text{None}$

6     VISITED := VISITED  $\cup \{v\}$ ;

7     R.push( $v$ );

8     **foreach**  $w \in \text{POST}(v)$  **do**

9         **if**  $w \in \text{DEAD}$  **then**

10             continue;

11         **else if**  $w \notin \text{VISITED}$  **then**

12              $\text{setBased}(w)$ ;

13         **else**

14             **while**  $\mathcal{S}(v) \neq \mathcal{S}(w)$  **do**

15                  $r := \text{R.pop}()$ ;

16                  $\text{UNITE}(\mathcal{S}, r, \text{R.top}())$ ;

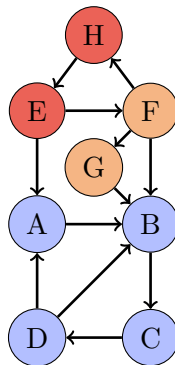
17             **if**  $v = \text{R.top}()$  **then**

18                 **report** SCC  $\mathcal{S}(v)$ ;

19                 DEAD := DEAD  $\cup \mathcal{S}(v)$ ;

20                 R.pop();

---



$R = \{H, E\}$

$\text{VISITED} = \{H, E, A, B, C, D\}$

$\text{DEAD} = \{A, B, C, D\}$

$S = \{A, B, C, D\} \cup \{E\} \cup \{F\} \cup \{G\} \cup \{H\}$

---

Data: A graph  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ , a starting node  $v_0$ ;

Initialize an empty set DEAD;

2 Initialize an empty set VISITED;

3 Initialize an empty stack R;

4  $\text{setBased}(v_0)$ ;

5 **function**  $\text{setBased}: v \in \mathcal{V} \rightarrow \text{None}$

6     VISITED := VISITED  $\cup \{v\}$ ;

7     R.push( $v$ );

8     **foreach**  $w \in \text{POST}(v)$  **do**

9         **if**  $w \in \text{DEAD}$  **then**

10             continue;

11         **else if**  $w \notin \text{VISITED}$  **then**

12              $\text{setBased}(w)$ ;

13         **else**

14             **while**  $\mathcal{S}(v) \neq \mathcal{S}(w)$  **do**

15                  $r := \text{R.pop}()$ ;

16                  $\text{UNITE}(\mathcal{S}, r, \text{R.top}())$ ;

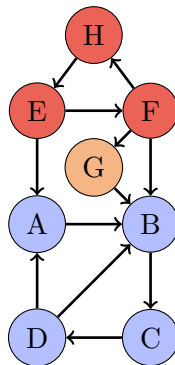
17             **if**  $v = \text{R.top}()$  **then**

18                 **report** SCC  $\mathcal{S}(v)$ ;

19                 DEAD := DEAD  $\cup \mathcal{S}(v)$ ;

20                 R.pop();

---



$R = \{H, E, F\}$

$\text{VISITED} = \{H, E, A, B, C, D, F\}$

$\text{DEAD} = \{A, B, C, D\}$

$S = \{A, B, C, D\} \cup \{E\} \cup \{F\} \cup \{G\} \cup \{H\}$

---

Data: A graph  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ , a starting node  $v_0$ ;

Initialize an empty set DEAD;

2 Initialize an empty set VISITED;

3 Initialize an empty stack R;

4  $\text{setBased}(v_0)$ ;

5 **function**  $\text{setBased}: v \in \mathcal{V} \rightarrow \text{None}$

6     VISITED := VISITED  $\cup \{v\}$ ;

7     R.push( $v$ );

8     **foreach**  $w \in \text{POST}(v)$  **do**

9         **if**  $w \in \text{DEAD}$  **then**

10             continue;

11         **else if**  $w \notin \text{VISITED}$  **then**

12              $\text{setBased}(w)$ ;

13         **else**

14             **while**  $\mathcal{S}(v) \neq \mathcal{S}(w)$  **do**

15                  $r := \text{R.pop}()$ ;

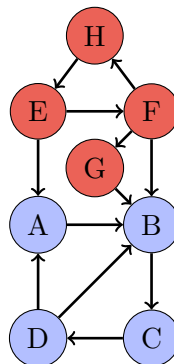
16                  $\text{UNITE}(\mathcal{S}, r, \text{R.top}())$ ;

17             **if**  $v = \text{R.top}()$  **then**

18                 **report SCC**  $\mathcal{S}(v)$ ;

19                  $\text{DEAD} := \text{DEAD} \cup \mathcal{S}(v)$ ;

20                  $\text{R.pop}()$ ;



$$R = \{H, E, F, G\}$$

$$\text{VISITED} = \{H, E, A, B, C, D, F, G\}$$

$$\text{DEAD} = \{A, B, C, D\}$$

$$S = \{A, B, C, D\} \cup \{E\} \cup \{F\} \cup \{G\} \cup \{H\}$$

---

Data: A graph  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ , a starting node  $v_0$ ;

Initialize an empty set DEAD;

2 Initialize an empty set VISITED;

3 Initialize an empty stack R;

4  $\text{setBased}(v_0)$ ;

5 **function**  $\text{setBased}: v \in \mathcal{V} \rightarrow \text{None}$

6     VISITED := VISITED  $\cup \{v\}$ ;

7     R.push( $v$ );

8     **foreach**  $w \in \text{POST}(v)$  **do**

9         **if**  $w \in \text{DEAD}$  **then**

10             continue;

11         **else if**  $w \notin \text{VISITED}$  **then**

12              $\text{setBased}(w)$ ;

13         **else**

14             **while**  $\mathcal{S}(v) \neq \mathcal{S}(w)$  **do**

15                  $r := \text{R.pop}()$ ;

16                  $\text{UNITE}(\mathcal{S}, r, \text{R.top}())$ ;

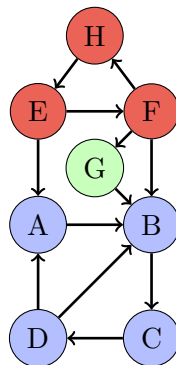
17             **if**  $v = \text{R.top}()$  **then**

18                 **report SCC**  $\mathcal{S}(v)$ ;

19                 DEAD := DEAD  $\cup \mathcal{S}(v)$ ;

20                 R.pop();

---



$R = \{H, E, F\}$

$\text{VISITED} = \{H, E, A, B, C, D, F, G\}$

$\text{DEAD} = \{A, B, C, D, G\}$

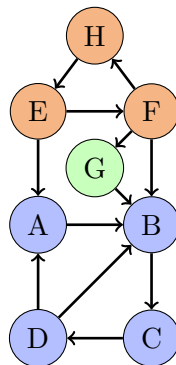
$\mathcal{S} = \{A, B, C, D\} \cup \{E\} \cup \{F\} \cup \{G\} \cup \{H\}$



**Data:** A graph  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ , a starting node  $v_0$ ;

```

1 Initialize an empty set DEAD;
2 Initialize an empty set VISITED;
3 Initialize an empty stack R;
4 setBased( $v_0$ );
5 function setBased:  $v \in \mathcal{V} \rightarrow \text{None}$ 
6     VISITED := VISITED  $\cup \{v\}$ ;
7     R.push( $v$ );
8     foreach  $w \in \text{POST}(v)$  do
9         if  $w \in \text{DEAD}$  then
10              $\text{continue}$ ;
11         else if  $w \notin \text{VISITED}$  then
12             setBased( $w$ );
13         else
14             while  $\mathcal{S}(v) \neq \mathcal{S}(w)$  do
15                  $r := \text{R.pop}()$ ;
16                 UNITE( $\mathcal{S}, r, \text{R.top}()$ );
17
18 if  $v = \text{R.top}()$  then
19     report SCC  $\mathcal{S}(v)$ ;
20     DEAD := DEAD  $\cup \mathcal{S}(v)$ ;
21     R.pop();
  
```



$$R = \{\}$$

$$\text{VISITED} = \{H, E, A, B, C, D, F, G\}$$

$$\text{DEAD} = \{A, B, C, D, G, E, F, H\}$$

$$\mathcal{S} = \{A, B, C, D\} \cup \{E, F, H\} \cup \{G\}$$

