

# Vérification formelle en Isabelle(HOL) d'un algorithme calculant les composantes fortement connexes d'un graphe

Vincent Trélat  
encadré par Stephan Merz

École Nationale Supérieure des Mines de Nancy  
Département Informatique

June 9, 2022

## ① Présentation et intérêt des méthodes formelles

Programmation défensive

Les méthodes formelles

Applications

## ② Introduction du projet

Définition

Motivation du problème

## ③ Vérification formelle en Isabelle

Présentation de l'algorithme

Structure de la preuve

## ④ Conclusion

Comment garantir le fonctionnement continu d'un logiciel dans des conditions imprévues ?

## Comment garantir le fonctionnement continu d'un logiciel dans des conditions imprévues ?

- Concepts de Software Engineering (design documents, design patterns, etc.)
- Review de code en équipe (GitHub, etc.)
- Séries de tests (unitaires, d'intégration, fonctionnels, etc.)

## Comment garantir le fonctionnement continu d'un logiciel dans des conditions imprévues ?

- Concepts de Software Engineering (design documents, design patterns, etc.)
- Review de code en équipe (GitHub, etc.)
- Séries de tests (unitaires, d'intégration, fonctionnels, etc.)

⇒ Responsabilité du programmeur !

*Probability of human error is considerably higher than that of machine error.*

*Kenneth Appel*

*Probability of human error is considerably higher than that of machine error.*

*Kenneth Appel*

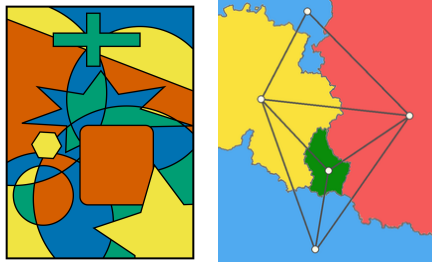


Figure: Illustrations du problème 4-COL

En méthodes formelles, on considère un programme comme une structure mathématique, ce qui permet de raisonner dessus de manière formelle.

- Rigueur
- Concepts de logique
- Validité par rapport à des spécifications
- Sémantique



## Preuve papier vs. preuve formelle

- L'utilisateur fournit la structure de la preuve
- Automatisation des preuves
- Différents outils informatiques : assistants à la preuve (Isabelle(HOL), Coq, Why3, B, etc.) et model checkers (TLC, etc.)

# Quelles applications ?

Éviter les “bugs” : applications aux systèmes critiques

Ex: Ariane 5, Entreprise Clearsy



Application au milieu ferroviaire : Certification du logiciel de pilotage automatique des lignes 1 et 14 de métro à Paris, bientôt la ligne 4

## ① Présentation et intérêt des méthodes formelles

Programmation défensive

Les méthodes formelles

Applications

## ② Introduction du projet

Définition

Motivation du problème

## ③ Vérification formelle en Isabelle

Présentation de l'algorithme

Structure de la preuve

## ④ Conclusion

# Composante fortement connexe

## Definition 1 (SCC)

Soient  $\mathcal{G} := (\mathcal{V}, \mathcal{E})$  un **graphe orienté** et  $\mathcal{C} \subseteq \mathcal{V}$ .  $\mathcal{C}$  est une SCC de  $\mathcal{G}$  si:

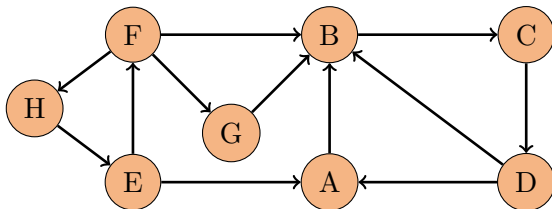
$$\forall x, y \in \mathcal{C}, (x \Rightarrow^* y) \wedge (y \Rightarrow^* x)$$

# Composante fortement connexe

## Definition 1 (SCC)

Soient  $\mathcal{G} := (\mathcal{V}, \mathcal{E})$  un **graphe orienté** et  $\mathcal{C} \subseteq \mathcal{V}$ .  $\mathcal{C}$  est une SCC de  $\mathcal{G}$  si:

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

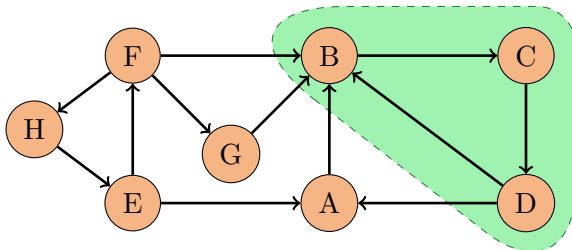


# Composante fortement connexe

## Definition 1 (SCC)

Soient  $\mathcal{G} := (\mathcal{V}, \mathcal{E})$  un **graphe orienté** et  $\mathcal{C} \subseteq \mathcal{V}$ .  $\mathcal{C}$  est une SCC de  $\mathcal{G}$  si:

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

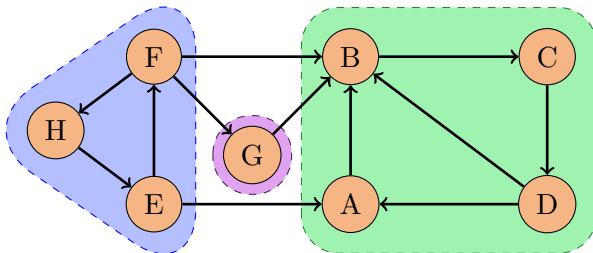


# Composante fortement connexe

## Definition 1 (SCC)

Soient  $\mathcal{G} := (\mathcal{V}, \mathcal{E})$  un **graphe orienté** et  $\mathcal{C} \subseteq \mathcal{V}$ .  $\mathcal{C}$  est une SCC de  $\mathcal{G}$  si:

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

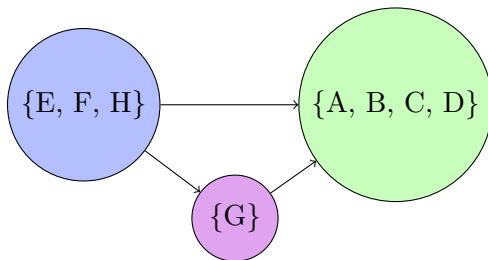


# Composante fortement connexe

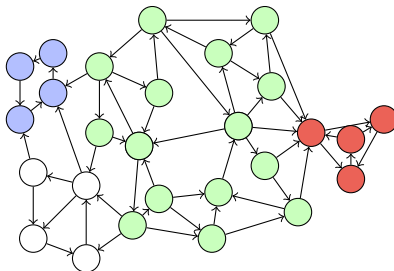
## Definition 1 (SCC)

Soient  $\mathcal{G} := (\mathcal{V}, \mathcal{E})$  un **graphe orienté** et  $\mathcal{C} \subseteq \mathcal{V}$ .  $\mathcal{C}$  est une SCC de  $\mathcal{G}$  si:

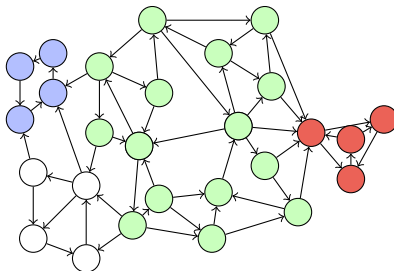
$$\forall x, y \in \mathcal{C}, (x \Rightarrow^* y) \wedge (y \Rightarrow^* x)$$







- Réseaux: interconnectivité et partage de données
- Model checking: recherche de contre-exemples



- Réseaux: interconnectivité et partage de données
- Model checking: recherche de contre-exemples

Algorithmes efficaces (ex: Tarjan)

- La vérification formelle de leur correction est utile
- Un autre défi : la parallélisation

## ① Présentation et intérêt des méthodes formelles

Programmation défensive

Les méthodes formelles

Applications

## ② Introduction du projet

Définition

Motivation du problème

## ③ Vérification formelle en Isabelle

Présentation de l'algorithme

Structure de la preuve

## ④ Conclusion

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

1 Initialize an empty set VISITED;

2 Initialize an empty set EXPLORED;

3 Initialize an empty stack R;

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

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**  $w \in \text{EXPLORED}$  **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                  $r := \text{R.pop}()$ ;

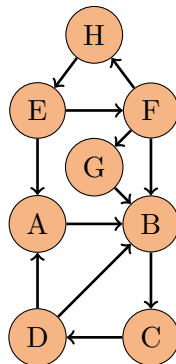
16                  $\text{UNITE}(\mathcal{S}, r, \text{R.top}())$ ;

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

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

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

20          $\text{R.pop}()$ ;



$R = []$

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

$\text{EXPLORED} = \{\}$

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

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

1 Initialize an empty set VISITED;

2 Initialize an empty set EXPLORED;

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**  $w \in \text{EXPLORED}$  **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                 |  $r := \text{R.pop}()$ ;

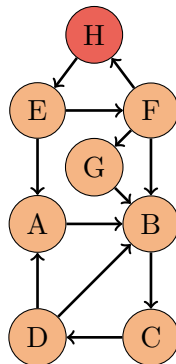
16                 | UNITE( $\mathcal{S}, r, \text{R.top}()$ );

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

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

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

20         R.pop();



$R = [H]$

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

$\text{EXPLORED} = \{\}$

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

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

1 Initialize an empty set VISITED;

2 Initialize an empty set EXPLORED;

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**  $w \in \text{EXPLORED}$  **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                 |  $r := \text{R.pop}()$ ;

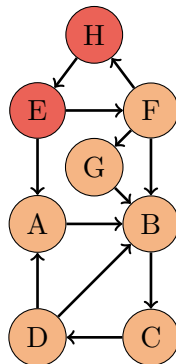
16                 |  $\text{UNITE}(\mathcal{S}, r, \text{R.top}())$ ;

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

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

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

20         |  $\text{R.pop}()$ ;



$R = [H, E]$

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

$\text{EXPLORED} = \{\}$

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

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

1 Initialize an empty set VISITED;

2 Initialize an empty set EXPLORED;

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**  $w \in \text{EXPLORED}$  **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                 |  $r := \text{R.pop}()$ ;

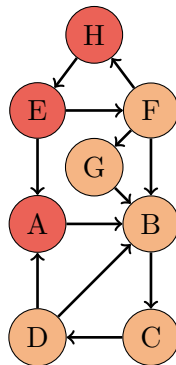
16                 | UNITE( $\mathcal{S}, r, \text{R.top}()$ );

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

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

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

20         R.pop();



$R = [H, E, A]$

VISITED = {H, E, A}

EXPLORED = {}

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

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

1 Initialize an empty set VISITED;

2 Initialize an empty set EXPLORED;

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**  $w \in \text{EXPLORED}$  **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                  $r := \text{R.pop}()$ ;

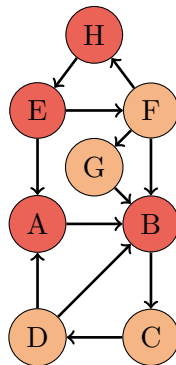
16                 UNITE( $\mathcal{S}, r, \text{R.top}()$ );

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

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

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

20         R.pop();



$R = [H, E, A, B]$

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

$\text{EXPLORED} = \{\}$

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



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

1 Initialize an empty set VISITED;

2 Initialize an empty set EXPLORED;

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**  $w \in \text{EXPLORED}$  **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                 |  $r := \text{R.pop}()$ ;

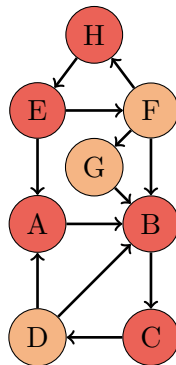
16                 |  $\text{UNITE}(\mathcal{S}, r, \text{R.top}())$ ;

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

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

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

20         R.pop();



$R = [H, E, A, B, C]$

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

$\text{EXPLORED} = \{\}$

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

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

1 Initialize an empty set VISITED;

2 Initialize an empty set EXPLORED;

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**  $w \in \text{EXPLORED}$  **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                 |  $r := \text{R.pop}()$ ;

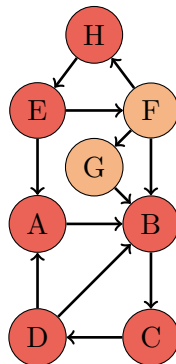
16                 |  $\text{UNITE}(\mathcal{S}, r, \text{R.top}())$ ;

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

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

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

20         R.pop();



$R = [H, E, A, B, C, D]$

VISITED = {H, E, A, B, C, D}

EXPLORED = {}

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

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

1 Initialize an empty set VISITED;

2 Initialize an empty set EXPLORED;

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**  $w \in \text{EXPLORED}$  **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                 |  $r := \text{R.pop}()$ ;

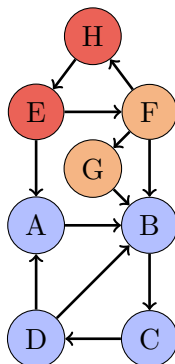
16                 |  $\text{UNITE}(\mathcal{S}, r, \text{R.top}())$ ;

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

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

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

20         R.pop();



$R = [H, E]$

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

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

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

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

1 Initialize an empty set VISITED;

2 Initialize an empty set EXPLORED;

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**  $w \in \text{EXPLORED}$  **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                 |  $r := \text{R.pop}()$ ;

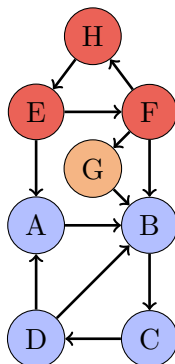
16                 |  $\text{UNITE}(\mathcal{S}, r, \text{R.top}())$ ;

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

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

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

20         R.pop();



$R = [H, E, F]$

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

EXPLORED = {A, B, C, D}

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

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

1 Initialize an empty set VISITED;

2 Initialize an empty set EXPLORED;

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**  $w \in \text{EXPLORED}$  **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                  $r := \text{R.pop}()$ ;

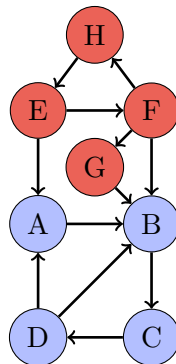
16                 UNITE( $\mathcal{S}, r, \text{R.top}()$ );

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

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

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

20         R.pop();



$R = [H, E, F, G]$

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

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

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

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

1 Initialize an empty set VISITED;

2 Initialize an empty set EXPLORED;

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**  $w \in \text{EXPLORED}$  **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                 |  $r := \text{R.pop}()$ ;

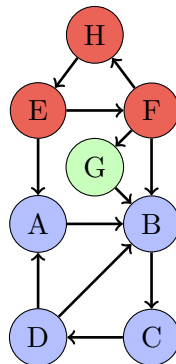
16                 |  $\text{UNITE}(\mathcal{S}, r, \text{R.top}())$ ;

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

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

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

20         R.pop();



$R = [H, E, F]$

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

EXPLORED = {A, B, C, D, G}

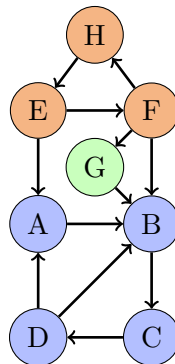
$\mathcal{S} = \{A, B, C, D\} \cup \{E\} \cup \{F\} \cup \{G\} \cup \{H\}$

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

```

1 Initialize an empty set VISITED;
2 Initialize an empty set EXPLORED;
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  $w \in \text{EXPLORED}$  then
10        $\mid$  continue;
11     else if  $w \notin \text{VISITED}$  then
12        $\mid$  setBased( $w$ );
13     else
14       while  $\mathcal{S}(v) \neq \mathcal{S}(w)$  do
15          $\mid$   $r := \text{R.pop}()$ ;
16          $\mid$  UNITE( $\mathcal{S}, r, \text{R.top}()$ );
17   if  $v = \text{R.top}()$  then
18     report SCC  $\mathcal{S}(v)$ ;
19     EXPLORED := EXPLORED  $\cup \mathcal{S}(v)$ ;
20     R.pop();

```



$R = []$

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

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

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

## ① Présentation et intérêt des méthodes formelles

Programmation défensive

Les méthodes formelles

Applications

## ② Introduction du projet

Définition

Motivation du problème

## ③ Vérification formelle en Isabelle

Présentation de l'algorithme

Structure de la preuve

## ④ Conclusion



- Modélisation et implémentation de l'algorithme
  - Définition de l'environnement
  - Fonctions `unite`, `dfs` et `dfss`
- Définition des invariants
  - définitions : `reachable` et `is_scc`
  - well-formed environment
  - pré-conditions et post-conditions sur `dfs` et `dfss`
- Écriture et preuve des lemmes
  - $\text{pre\_dfs} \implies \text{pre\_dfss}$
  - $\text{pre\_dfss} \implies \text{pre\_dfs}$
  - $\text{pre\_dfs} \implies \text{post\_dfs}$
  - $\text{pre\_dfss} \implies \text{post\_dfss}$
  - théorème final

## ① Présentation et intérêt des méthodes formelles

Programmation défensive

Les méthodes formelles

Applications

## ② Introduction du projet

Définition

Motivation du problème

## ③ Vérification formelle en Isabelle

Présentation de l'algorithme

Structure de la preuve

## ④ Conclusion

- Ce qui a été fait :
  - Modèle cohérent et stable
  - Preuve de tous les lemmes intermédiaires (ou presque)
  - Implémentation de l'algorithme : environ 10 lignes de code
  - Preuve : presque 2000 lignes de code
- Ce qui manque :
  - Quelques propriétés intermédiaires sont à démontrer
  - Théorème final
  - Montrer la terminaison