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

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.)

$\Longrightarrow$ 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

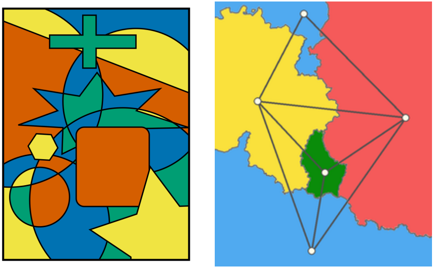


Figure: Illustrations du problème 4-COL

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

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:

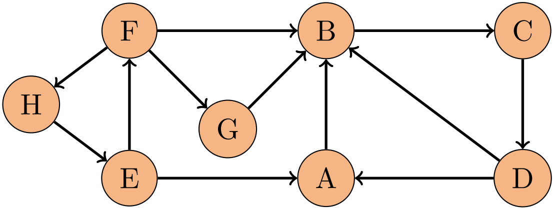
$$\forall x, y \in \mathcal{C}, (x \Rightarrow^* y) \wedge (y \Rightarrow^* x)$$

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:

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

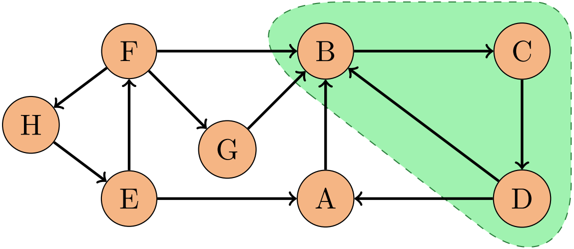


Composante fortement connexe

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:

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

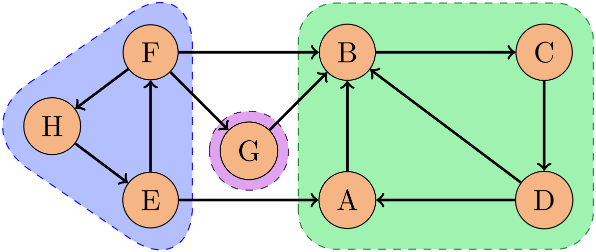


Composante fortement connexe

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:

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

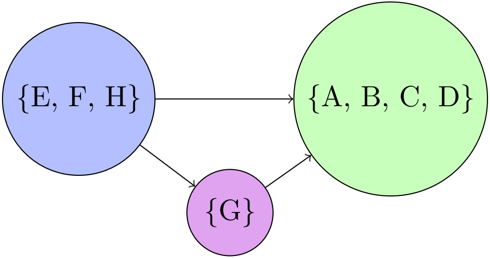


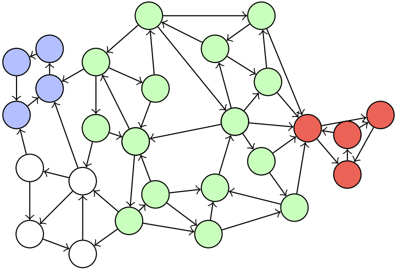
Composante fortement connexe

Definition 1 (SCC)

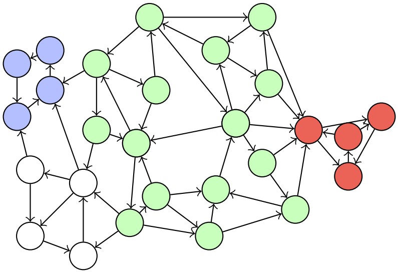
Soient $\mathcal{G} := (\mathcal{V}, \mathcal{E})$ un graphe orienté et $\mathcal{C} \subseteq \mathcal{V}$. $\mathcal{C}$ est une SCC de $\mathcal{G}$ si:

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





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

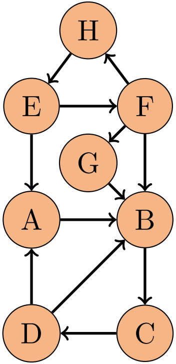


- Réseaux: interconnectivité et partage de données
- Model checking: recherche de contre-exemples

Algorithmes efficaces (ex: Tarjan)

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

```
Data: A graph G = (V, E), a starting node v0;
Initialize an empty set VISITED;
Initialize an empty set EXPLORED;
Initialize an empty stack R;
setBased(v0);
function setBased: v ∈ V → None
    VISITED := VISITED ∪ {v};
    R.push(v);
    foreach w ∈ POST(v) do
        if w ∈ EXPLORED then
            continue;
        else if w ∉ VISITED then
            setBased(w);
        else
            while S(v) ≠ S(w) do
                r := R.pop();
                UNITE(S, r, R.top());
    if v = R.top() then
        report SCC S(v);
        EXPLORED := EXPLORED ∪ S(v);
        R.pop();
```



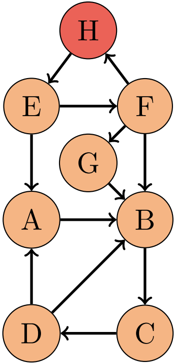
$$\mathrm{R} = []$$

$$\mathrm{VISITED} = \{\}$$

$$\mathrm{EXPLORED} = \{\}$$

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

```
Data: A graph G = (V, E), a starting node v0;
Initialize an empty set VISITED;
Initialize an empty set EXPLORED;
Initialize an empty stack R;
setBased(v0);
function setBased: v ∈ V → None
    VISITED := VISITED ∪ {v};
    R.push(v);
    foreach w ∈ POST(v) do
        if w ∈ EXPLORED then
            continue;
        else if w ∉ VISITED then
            setBased(w);
        else
            while S(v) ≠ S(w) do
                r := R.pop();
                UNITE(S, r, R.top());
    if v = R.top() then
        report SCC S(v);
        EXPLORED := EXPLORED ∪ S(v);
        R.pop();
```



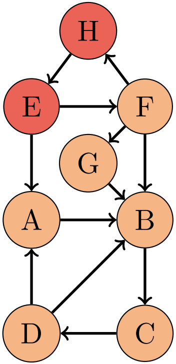
$$R = [H]$$

$$VISITED = \{H\}$$

$$EXPLORED = \{\}$$

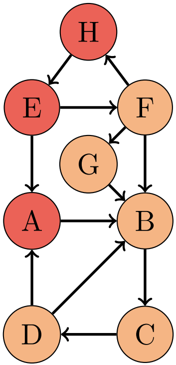
$$\mathcal{S} = \{A\} \cup \{B\} \cup \{C\} \cup \{D\} \cup \{E\} \cup \{F\} \cup \{G\} \cup \{H\}$$

Data: A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a starting node v_0;
Initialize an empty set VISITED;
Initialize an empty set EXPLORED;
Initialize an empty stack R;
setBased(v_0);
function *setBased*: $v \in \mathcal{V} \rightarrow$ *None*
  VISITED := VISITED $\cup \{v\}$;
  R.push(v);
  foreach $w \in$ *POST(v)* **do**
    if $w \in$ *EXPLORED* **then**
      continue;
    else if $w \notin$ *VISITED* **then**
      setBased(w);
    else
      while $\mathcal{S}(v) \neq \mathcal{S}(w)$ **do**
        $r :=$ R.pop();
        UNITE($\mathcal{S}$, r, R.top());
  if $v =$ *R.top()* **then**
    report SCC $\mathcal{S}(v)$;
    EXPLORED := EXPLORED $\cup\ \mathcal{S}(v)$;
    R.pop();



$$R = [H, E]$$
$$VISITED = \{H, E\}$$
$$EXPLORED = \{\}$$
$$\mathcal{S} = \{A\} \cup \{B\} \cup \{C\} \cup \{D\} \cup \{E\} \cup \{F\} \cup \{G\} \cup \{H\}$$

```
Data: A graph 𝒢 = (𝒱 , ℰ ), a starting node v0;
Initialize an empty set VISITED;
Initialize an empty set EXPLORED;
Initialize an empty stack R;
setBased(v0);
function setBased: v ∈ 𝒱 → None
    VISITED := VISITED ∪ {v};
    R.push(v);
    foreach w ∈ POST(v) do
        if w ∈ EXPLORED then
            continue;
        else if w ∉ VISITED then
            setBased(w);
        else
            while 𝒮(v) ≠ 𝒮(w) do
                r := R.pop();
                UNITE(𝒮, r, R.top());
    if v = R.top() then
        report SCC 𝒮(v);
        EXPLORED := EXPLORED ∪ 𝒮(v);
        R.pop();
```



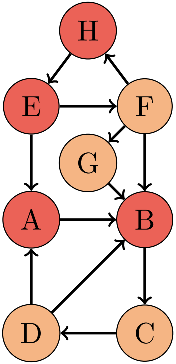
$$R = [H, E, A]$$

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

$$EXPLORED = \{\}$$

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

```
Data: A graph 𝒢 = (𝒱, ℰ), a starting node v0;
Initialize an empty set VISITED;
Initialize an empty set EXPLORED;
Initialize an empty stack R;
setBased(v0);
function setBased: v ∈ 𝒱 → None
    VISITED := VISITED ∪ {v};
    R.push(v);
    foreach w ∈ POST(v) do
        if w ∈ EXPLORED then
            continue;
        else if w ∉ VISITED then
            setBased(w);
        else
            while 𝒮(v) ≠ 𝒮(w) do
                r := R.pop();
                UNITE(𝒮, r, R.top());
    if v = R.top() then
        report SCC 𝒮(v);
        EXPLORED := EXPLORED ∪ 𝒮(v);
        R.pop();
```



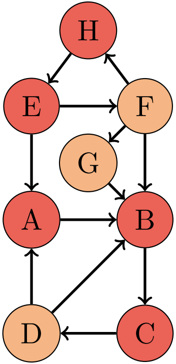
$$\mathrm{R} = [\mathrm{H}, \mathrm{E}, \mathrm{A}, \mathrm{B}]$$

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

$$\mathrm{EXPLORED} = \{\}$$

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

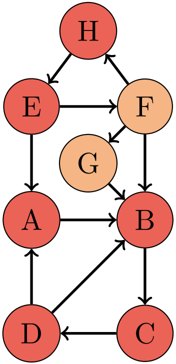
Data: A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a starting node v_0;
Initialize an empty set VISITED;
Initialize an empty set EXPLORED;
Initialize an empty stack R;
setBased(v_0);
function *setBased*: $v \in \mathcal{V} \rightarrow None$
    VISITED := VISITED $\cup \{v\}$;
    R.push(v);
    foreach $w \in POST(v)$ **do**
        if $w \in EXPLORED$ **then**
            continue;
        else if $w \notin VISITED$ **then**
            setBased(w);
        else
            while $\mathcal{S}(v) \neq \mathcal{S}(w)$ **do**
                $r :=$ R.pop();
                UNITE($\mathcal{S}, r$, R.top());
    if $v = R.top()$ **then**
        report SCC $\mathcal{S}(v)$;
        EXPLORED := EXPLORED $\cup\ \mathcal{S}(v)$;
        R.pop();



$$R = [H, E, A, B, C]$$
$$VISITED = \{H, E, A, B, C\}$$
$$EXPLORED = \{\}$$

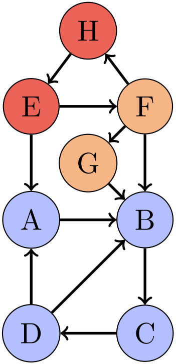
$\mathcal{S} = \{A\} \cup \{B\} \cup \{C\} \cup \{D\} \cup \{E\} \cup \{F\} \cup \{G\} \cup \{H\}$

Data: A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a starting node v_0;
Initialize an empty set `VISITED`;
Initialize an empty set `EXPLORED`;
Initialize an empty stack `R`;
setBased(v_0);
function *setBased:* $v \in \mathcal{V} \rightarrow$ *None*
    `VISITED` := `VISITED` $\cup \{v\}$;
    `R.push`(v);
    foreach $w \in$ *POST(v)* **do**
        if $w \in$ *EXPLORED* **then**
            continue;
        else if $w \notin$ *VISITED* **then**
            setBased(w);
        else
            while $\mathcal{S}(v) \neq \mathcal{S}(w)$ **do**
                r := `R.pop()`;
                UNITE($\mathcal{S}$, r, `R.top()`);
    if v = *R.top()* **then**
        report SCC $\mathcal{S}(v)$;
        `EXPLORED` := `EXPLORED` $\cup\ \mathcal{S}(v)$;
        `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 𝒢 = (𝒱 , ℰ ), a starting node v0;
Initialize an empty set VISITED;
Initialize an empty set EXPLORED;
Initialize an empty stack R;
setBased(v0);
function setBased: v ∈ 𝒱 → None
    VISITED := VISITED ∪ {v};
    R.push(v);
    foreach w ∈ POST(v) do
        if w ∈ EXPLORED then
            continue;
        else if w ∉ VISITED then
            setBased(w);
        else
            while 𝒮(v) ≠ 𝒮(w) do
                r := R.pop();
                UNITE(𝒮, r, R.top());
    if v = R.top() then
        report SCC 𝒮(v);
        EXPLORED := EXPLORED ∪ 𝒮(v);
        R.pop();
```



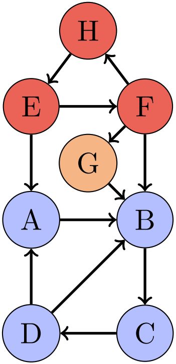
$$\mathrm{R} = [\mathrm{H}, \mathrm{E}]$$

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

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

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

```
Data: A graph 𝒢 = (𝒱, ℰ), a starting node v0;
Initialize an empty set VISITED;
Initialize an empty set EXPLORED;
Initialize an empty stack R;
setBased(v0);
function setBased: v ∈ 𝒱 → None
    VISITED := VISITED ∪ {v};
    R.push(v);
    foreach w ∈ POST(v) do
        if w ∈ EXPLORED then
            continue;
        else if w ∉ VISITED then
            setBased(w);
        else
            while 𝒮(v) ≠ 𝒮(w) do
                r := R.pop();
                UNITE(𝒮, r, R.top());
    if v = R.top() then
        report SCC 𝒮(v);
        EXPLORED := EXPLORED ∪ 𝒮(v);
        R.pop();
```



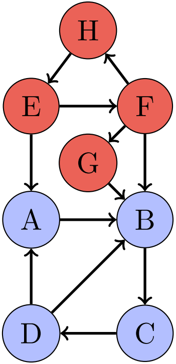
$$\mathrm{R} = [\mathrm{H}, \mathrm{E}, \mathrm{F}]$$

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

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

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

```
Data: A graph G = (V, E), a starting node v0;
Initialize an empty set VISITED;
Initialize an empty set EXPLORED;
Initialize an empty stack R;
setBased(v0);
function setBased: v ∈ V → None
    VISITED := VISITED ∪ {v};
    R.push(v);
    foreach w ∈ POST(v) do
        if w ∈ EXPLORED then
            continue;
        else if w ∉ VISITED then
            setBased(w);
        else
            while S(v) ≠ S(w) do
                r := R.pop();
                UNITE(S, r, R.top());
    if v = R.top() then
        report SCC S(v);
        EXPLORED := EXPLORED ∪ S(v);
        R.pop();
```



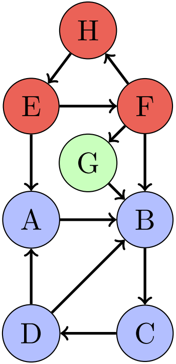
$$R = [H, E, F, G]$$

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

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

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

```
Data: A graph G = (V , E ), a starting node v0;
Initialize an empty set VISITED;
Initialize an empty set EXPLORED;
Initialize an empty stack R;
setBased(v0);
function setBased: v ∈ V → None
    VISITED := VISITED ∪ {v};
    R.push(v);
    foreach w ∈ POST(v) do
        if w ∈ EXPLORED then
            continue;
        else if w ∉ VISITED then
            setBased(w);
        else
            while S(v) ≠ S(w) do
                r := R.pop();
                UNITE(S, r, R.top());
    if v = R.top() then
        report SCC S(v);
        EXPLORED := EXPLORED ∪ S(v);
        R.pop();
```



$$\mathrm{R} = [\mathrm{H}, \mathrm{E}, \mathrm{F}]$$

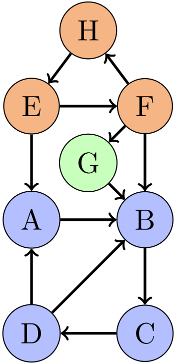
$$\mathrm{VISITED} = \{\mathrm{H}, \mathrm{E}, \mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}, \mathrm{F}, \mathrm{G}\}$$

$$\mathrm{EXPLORED} = \{\mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}, \mathrm{G}\}$$

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

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

Initialize an empty set `VISITED`;
Initialize an empty set `EXPLORED`;
Initialize an empty stack `R`;
setBased(v_0);
function *setBased*: $v \in \mathcal{V} \rightarrow$ *None*
    `VISITED` $:=$ `VISITED` $\cup \{v\}$;
    `R.push`(v);
    foreach $w \in$ *POST(v)* **do**
        if $w \in$ *EXPLORED* **then**
            continue;
        else if $w \notin$ *VISITED* **then**
            setBased(w);
        else
            while $\mathcal{S}(v) \neq \mathcal{S}(w)$ **do**
                $r :=$ `R.pop()`;
                `UNITE`($\mathcal{S}, r,$ `R.top()`);
    if $v =$ *R.top()* **then**
        report SCC $\mathcal{S}(v)$;
        `EXPLORED` $:=$ `EXPLORED` $\cup\, \mathcal{S}(v)$;
        `R.pop()`;



$$\mathrm{R} = []$$

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

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

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

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
  - `pre_dfs` $\Longrightarrow$ `pre_dfss`
  - `pre_dfss` $\Longrightarrow$ `pre_dfs`
  - `pre_dfs` $\Longrightarrow$ `post_dfs`
  - `pre_dfss` $\Longrightarrow$ `post_dfss`
  - théorème final

- Ce qui a été fait :
  - Modèle cohérent et stable
  - Preuve de tous les lemmes intermédiaires (ou presque)

- Ce qui manque :
  - Quelques propriétés intermédiaires sont à démontrer
  - Théorème final
  - Montrer la terminaison