Sécurité du logiciel, deuxième cours

Flux d'information

Xavier Leroy

2022-03-17

Collège de France, chaire de sciences du logiciel
`xavier.leroy@college-de-france.fr`

Sécurité multi-niveaux et flux d'information

La sécurité multi-niveaux

Manipuler, dans un même système informatique, des données avec différents niveaux de confidentialité et d'intégrité.

Exemple : deux niveaux de confidentialité,

- secret (diffusion restreinte)
- public (diffusion libre)

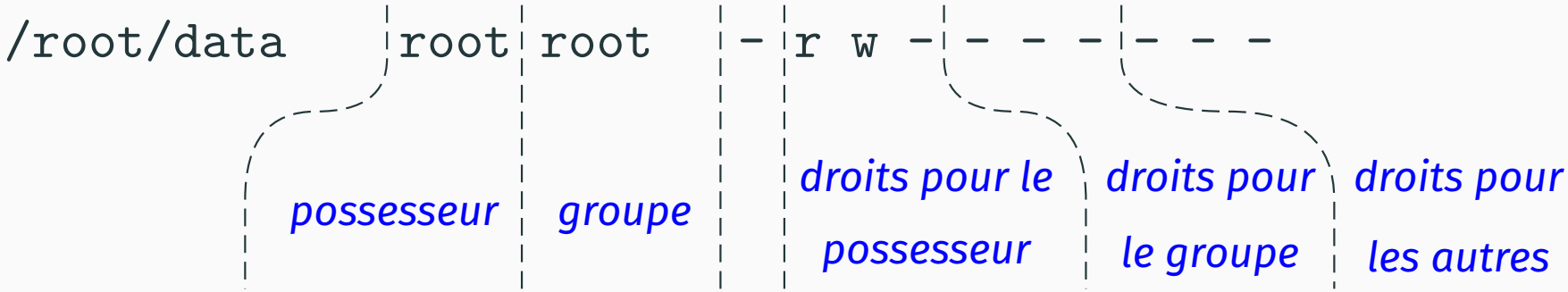
Exemple : deux niveaux d'intégrité,

- fiable (provenant du système)
- douteux (provenant des utilisateurs)

Utilisation du contrôle d'accès

Exemple : un fichier `/root/data` qui doit être secret et fiable.

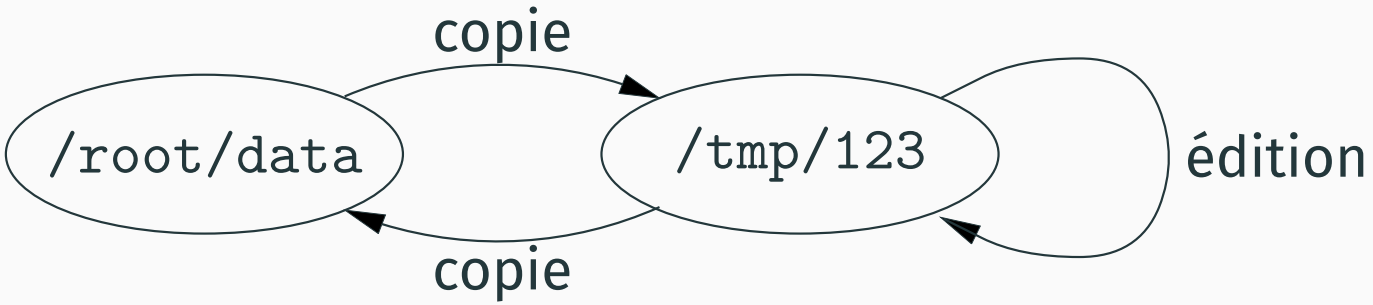
Lecture et écriture uniquement par l'administrateur `root`.



Insuffisance du contrôle d'accès

```
/root/data    root root    - r w - - - - - - - -
```

Édition du fichier via un fichier temporaire :



Si le fichier temporaire a été créé par l'attaquant, il peut lire son contenu et le modifier avant la copie dans `/root/data`.

Flux d'information

Contrôler non seulement les accès aux ressources
(données au repos)
mais aussi les flux d'information entre ces ressources
(données en transit).

Politique de confidentialité : les flux doivent toujours aller du moins secret vers le plus secret.

$$\text{public} \to \text{public} \quad \text{public} \to \text{secret}$$
$$\text{secret} \to \text{secret} \quad \text{secret} \not\to \text{public}$$

Politique d'intégrité : les flux doivent toujours aller du plus intègre vers le moins intègre.

$$\text{fiable} \to \text{fiable} \quad \text{fiable} \to \text{douteux}$$
$$\text{douteux} \to \text{douteux} \quad \text{douteux} \not\to \text{fiable}$$

Formaliser la confidentialité

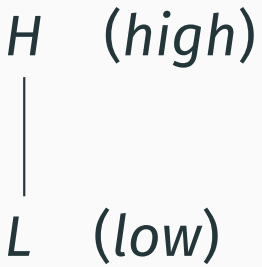
Un ordre partiel sur les niveaux de confidentialité

$$A \sqsubseteq B$$

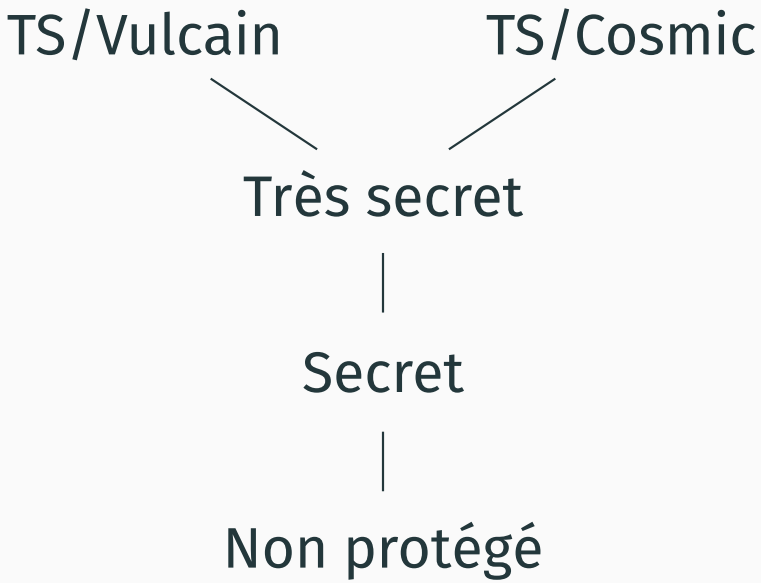
« B est aussi ou plus secret que A. »

« Une personne habilitée B peut accéder à une info classifiée A. »

Exemples : public/secret



secret défense en France



Politique de confidentialité de Bell-LaPadula

(D. E. Bell et L. J. LaPadula, *Secure Computer Systems : Mathematical Foundations*, 1973, MITRE Corporation.)

No read up : *(Simple Security Property)*

Un sujet au niveau ℓ ne peut lire que des objets de niveau $\ell' \sqsubseteq \ell$.

No write down : *(Star Security Property)*

Un sujet au niveau ℓ ne peut écrire que dans des objets de niveau $\ell' \sqsupseteq \ell$.

Variante : le point culminant (*high-water mark*)

Un sujet possède deux niveaux :

- R : niveau maximal de lecture, fixé.
- W : niveau minimal d'écriture, augmente au cours du temps.

Un objet de niveau ℓ peut être lu si $\ell \sqsubseteq R$.
Dans ce cas, on fait $W \leftarrow W \sqcup \ell$.

Un objet de niveau ℓ peut être écrit si $W \sqsubseteq \ell$.

Formaliser l'intégrité

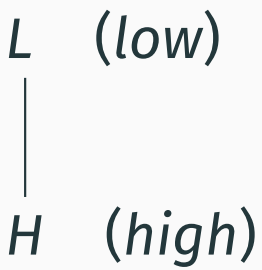
Un ordre partiel sur les niveaux d'intégrité.

$$A \sqsubseteq B$$

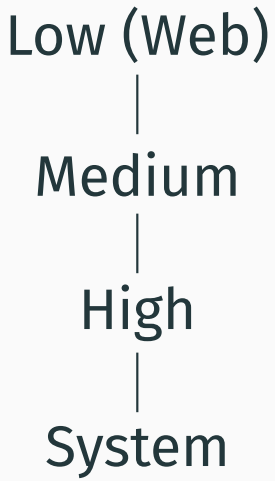
« *A* est plus fiable ou aussi fiable que *B*. »

« Une personne habilitée au niveau *A* peut modifier une donnée de niveau *B*. »

Exemples : Fiable/douteux



Windows (depuis Vista)

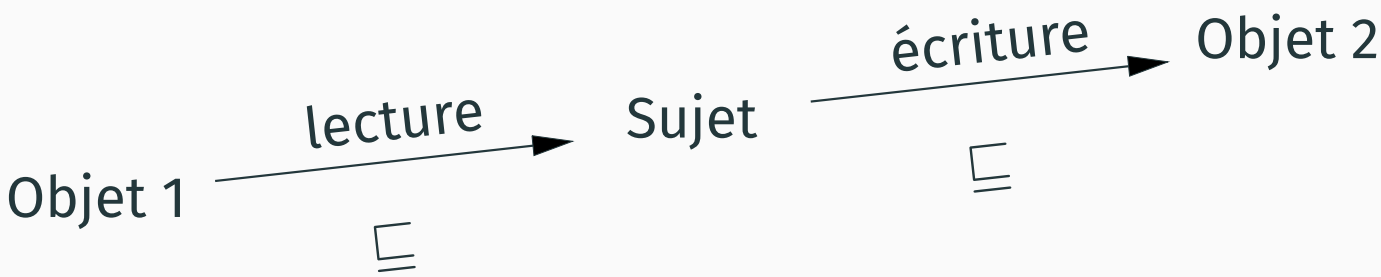


Politique d'intégrité de Biba

K. J. Biba, *Integrity Considerations for Secure Computer Systems*, 1975, MITRE Corporation.

No write down : Un sujet au niveau ℓ ne peut écrire que dans des objets de niveau $\ell' \sqsupseteq \ell$.

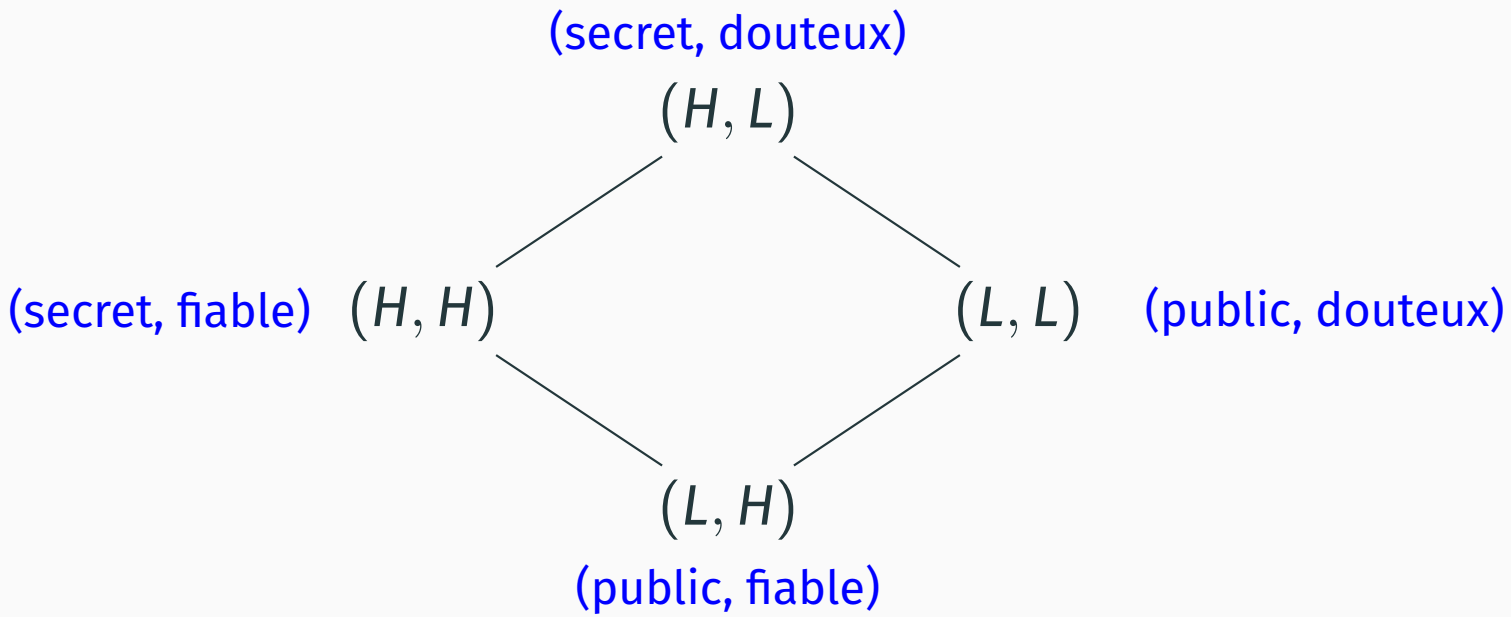
No read up : Un sujet au niveau ℓ ne peut lire que des objets de niveau $\ell' \sqsubseteq \ell$.



Combiner confidentialité et intégrité

Niveau = paire (niveau de confidentialité, niveau d'intégrité).

Ordre partiel = produit des ordres de confidentialité et d'intégrité.

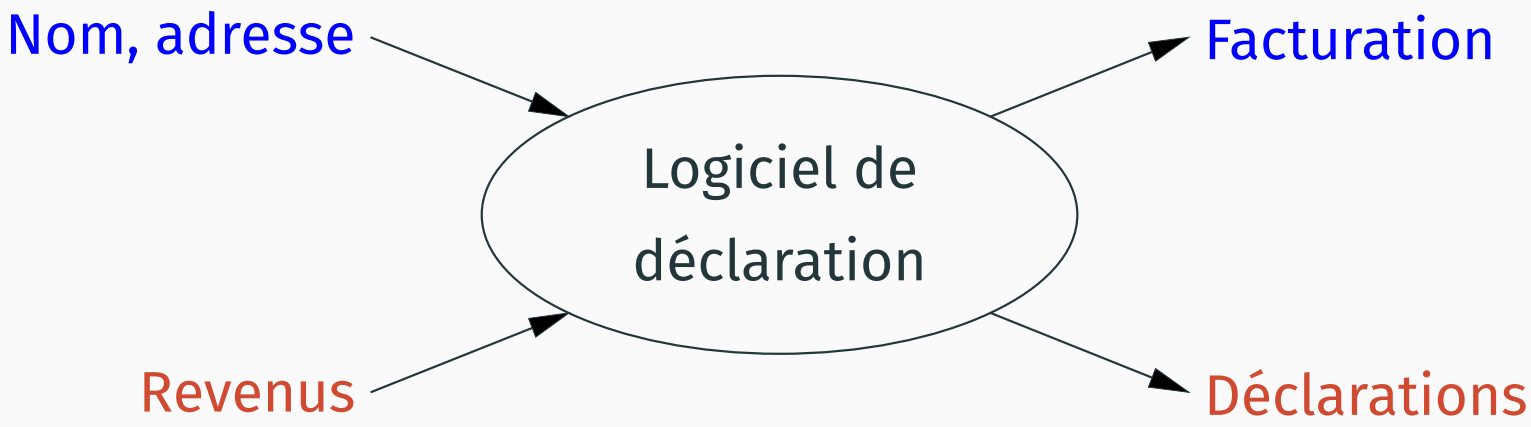


Flux d’information dans un programme

Sécurité multi-niveau dans un programme

Un même programme peut manipuler des données à plusieurs niveaux de confidentialité ou d'intégrité.

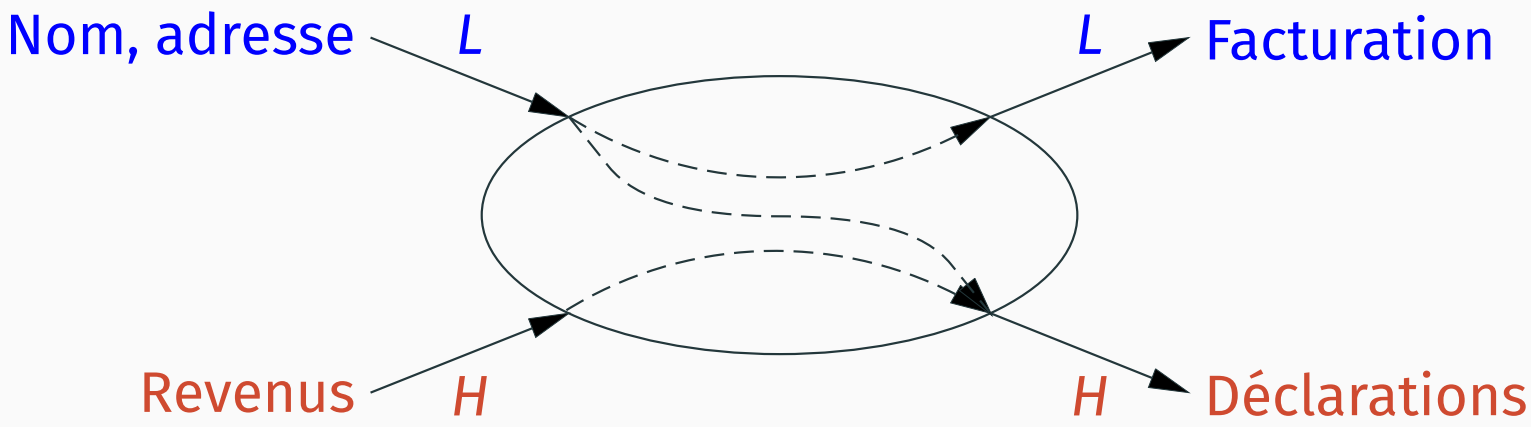
Exemple : un logiciel payant pour déclarer ses impôts.



L'information de facturation envoyée au fournisseur du logiciel ne doit révéler aucune information sur les revenus.

Bell-LaPadula pour un programme

Associer des niveaux de confidentialité aux entrées, aux sorties, et aux résultats intermédiaires du programme.



Vérifier que les flux d'information vont toujours «vers le haut» : une sortie de niveau ℓ ne dépend que d'entrées de niveau $\ell' \sqsubseteq \ell$.

Vérification dynamique des flux d'information

Remplacer chaque donnée par une paire (valeur, niveau).

Vérifier les niveaux à chaque opération du programme.

```
                              assert (x.level <= z.level);
z := (x + y) / 2      ⇒       assert (y.level <= z.level);
                              z.value := (x.value + y.value) / 2
```

Cela suffit à tracer les flux directs.

Le problème des flux indirects

Soit `x` une variable booléenne secrète (niveau *H*) et `y` une variable publique (de niveau *L*).

```
if x then y := true else y := false
```

Ce code a le même effet que `y := x`.

Et pourtant, seules des valeurs publiques (`true`, `false`) ont été affectées à `y` !

Vérification dynamique des flux indirects

On ajoute une variable *pc* de type «niveau» qui garde trace des informations révélées par les branchements conditionnels.

Cette variable est mise à jour lors des conditionnelles :

```
if x then ... else ...   ⇒   pc := max(pc, x.level);
                              if x.value then  ... else ...
```

Elle est prise en compte lors des affectations :

```
                             assert (x.level <= z.level);
z := (x + y) / 2   ⇒         assert (y.level <= z.level);
                             assert (pc <= z.level);
                             z.value := (x.value + y.value) / 2
```

Vérification dynamique des flux indirects

Cela permet bien de vérifier les flux indirects :

```
if x then y := true else y := false

   ⇒    pc := max(pc, x.level);
        if x.value
        then assert (pc <= y.level); y := true
        else assert (pc <= y.level); y := false
```

L'inflation des niveaux (*label creep*)

Le niveau du *pc* ne diminue jamais !

Un seul test sur une variable de niveau *H* force toute la suite du programme à travailler au niveau *H*.

```
    if x^H then y^H := true else y^H := false;
✘   z^L := false                          // rejeté car pc = H
```

On est tenté de remettre le *pc* à son niveau d'avant la conditionnelle :

```
if x then ... else ...   ⇒   pc1 := pc; pc := max(pc, x.level);
                             if x.value then  ... else ...;
                             pc := pc1
```

Mais ce serait incorrect…

L'inflation des niveaux (*label creep*)

L'absence d'affectation peut aussi créer un flux indirect !

```
y^L := false;
if x^H then y^L := true else skip;
C
```

Si le programme atteint le point C, sans avoir échoué sur l'assertion `pc <= y.level`, il connaît le secret « `x`^{H} est `false` », et doit donc exécuter C avec $pc = H$.

Typage statique des flux d'information

Vérification statique des flux d'information

Par analyse statique de type *dataflow* (Denning & Denning, 1973).

Sous forme d'un système de types (Volpano, Irvine, Smith, 1996) :

$\vdash a : \ell$ la valeur de l'expression a est de niveau ℓ

$pc \vdash c : *$ la commande c est sûre dans un contexte de niveau pc

IMP : un petit langage impératif à contrôle structuré

Expressions arithmétiques :

$a ::= \mathtt{x}^{\ell}$	variables avec leur niveau ℓ
$\mid 0 \mid 1 \mid \ldots$	constantes
$\mid a_1 + a_2 \mid a_1 \times a_2 \mid \ldots$	opérations

Expressions booléennes :

$b ::= a_1 \leq a_2 \mid \ldots$	comparaisons
$\mid b_1 \texttt{ and } b_2 \mid \texttt{not } b \mid \ldots$	connecteurs

Commandes :

$c ::= \texttt{skip}$	commande vide
$\mid \mathtt{x}^{\ell} := a$	affectation
$\mid c_1; c_2$	séquence
$\mid \texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2$	conditionnelle
$\mid \texttt{while } b \texttt{ do } c$	boucle

Les règles de typage des flux d'information en IMP

Expressions arithmétiques ou booléennes :

$$\frac{\ell' \sqsubseteq \ell \text{ pour toute variable } x^{\ell'} \text{ apparaissant dans } a}{\vdash a : \ell}$$

$$\frac{\ell' \sqsubseteq \ell \text{ pour toute variable } x^{\ell'} \text{ apparaissant dans } b}{\vdash b : \ell}$$

Les règles de typage des flux d'information en IMP

$$pc \vdash \texttt{skip} : *$$

$$\frac{\vdash a : \ell' \qquad \ell' \sqsubseteq \ell \qquad pc \sqsubseteq \ell}{pc \vdash x^{\ell} := a : *}$$

$$\frac{pc \vdash c_1 : * \qquad pc \vdash c_2 : *}{pc \vdash c_1; c_2 : *}$$

$$\frac{\vdash b : \ell \qquad pc \sqcup \ell \vdash c_1 : * \qquad pc \sqcup \ell \vdash c_2 : *}{pc \vdash \texttt{if}\ b\ \texttt{then}\ c_1\ \texttt{else}\ c_2 : *}$$

$$\frac{\vdash b : \ell \qquad pc \sqcup \ell \vdash c : *}{pc \vdash \texttt{while}\ b\ \texttt{do}\ c}$$

(Contrôle des flux directs) (Contrôle des flux indirects)

Exemple de typage

Contrôle des flux indirects :

$$(\texttt{if } \texttt{x}^{H} = 0 \texttt{ then } \texttt{y}^{\ell} := 0 \texttt{ else skip}); \texttt{z}^{L} := 1$$

Dérivation de typage :

$$\dfrac{\dfrac{\vdash \texttt{x}^{H} = 0 : H \qquad \dfrac{\vdash 0 : L \quad L \sqsubseteq \ell \quad H \sqsubseteq \ell}{H \vdash \texttt{y}^{\ell} := 0 : *} \qquad H \vdash \texttt{skip} : *}{L \vdash \texttt{if } \texttt{x}^{H} = 0 \texttt{ then } \texttt{y}^{\ell} := 0 \texttt{ else skip} : *} \qquad \dfrac{\vdash 1 : L \quad L \sqsubseteq L \quad L \sqsubseteq L}{L \vdash \texttt{z}^{L} := 1 : *}}{L \vdash (\texttt{if } \texttt{x}^{H} = 0 \texttt{ then } \texttt{y}^{\ell} := 0 \texttt{ else skip}); \texttt{z}^{L} := 1 : *}$$

Le programme est donc accepté si $\ell = H$, rejeté si $\ell = L$.

Pas d'inflation des niveaux car on vérifie les 2 branches
(au contraire de la vérification dynamique qui n'en vérifie qu'une).

La propriété de non-interférence

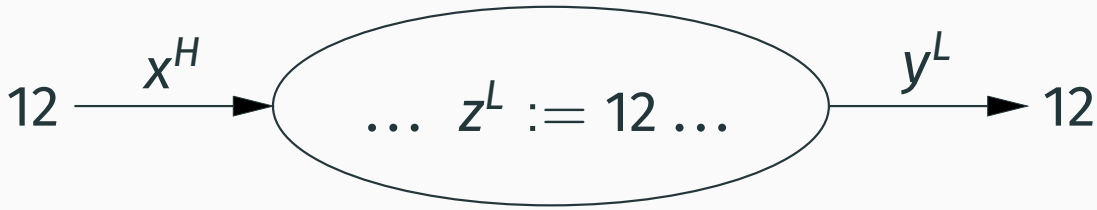
Une caractérisation sémantique du bon contrôle des flux d'information.

> *Les valeurs des sorties de niveau ℓ ne doivent pas dépendre des valeurs des entrées de niveau $\ell' \sqsupset \ell$.*

La propriété de non-interférence

La non-interférence n'est pas une propriété d'une seule exécution du programme.

Exemple : dans l'exécution ci-dessous, y-a-t'il interférence entre la sortie y^H et l'entrée x^L ?



La propriété de non-interférence

La non-interférence est une hyper-propriété qui porte sur deux exécutions du même programme.

La sémantique naturelle d'IMP

Un prédicat $c/s \Downarrow s'$

«la commande c démarrée dans l'état s termine dans l'état s'».

$$\texttt{skip}/s \Downarrow s \qquad\qquad x := a/s \Downarrow s[x \leftarrow [\![a]\!]\, s]$$

$$\frac{c_1/s \Downarrow s' \qquad c_2/s' \Downarrow s''}{c_1; c_2/s \Downarrow s''} \qquad\qquad \frac{\begin{array}{c} c_1/s \Downarrow s' \text{ si } [\![b]\!]\, s = \texttt{true} \\ c_2/s \Downarrow s' \text{ si } [\![b]\!]\, s = \texttt{false} \end{array}}{\texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2/s \Downarrow s'}$$

$$\frac{[\![b]\!]\, s = \texttt{false}}{\texttt{while } b \texttt{ do } c/s \Downarrow s} \qquad\qquad \frac{[\![b]\!]\, s = \texttt{true} \qquad c/s \Downarrow s' \qquad \texttt{while } b \texttt{ do } c/s' \Downarrow s''}{\texttt{while } b \texttt{ do } c/s \Downarrow s''}$$

Formalisation de la non-interférence

(On se ramène à deux niveaux, L et H.)

On définit l'égalité entre deux états mémoire sur les variables L :

$$s_1 \overset{L}{\approx} s_2 \quad \overset{def}{=} \quad \forall x^L,\ s_1(x^L) = s_2(x^L)$$

Une expression de niveau L a la même valeur dans deux états $\overset{L}{\approx}$:

$$[\![a]\!]\, s_1 = [\![a]\!]\, s_2 \qquad \text{si} \ \vdash a : L \ \text{ et } \ s_1 \overset{L}{\approx} s_2$$
$$[\![b]\!]\, s_1 = [\![b]\!]\, s_2 \qquad \text{si} \ \vdash b : L \ \text{ et } \ s_1 \overset{L}{\approx} s_2$$

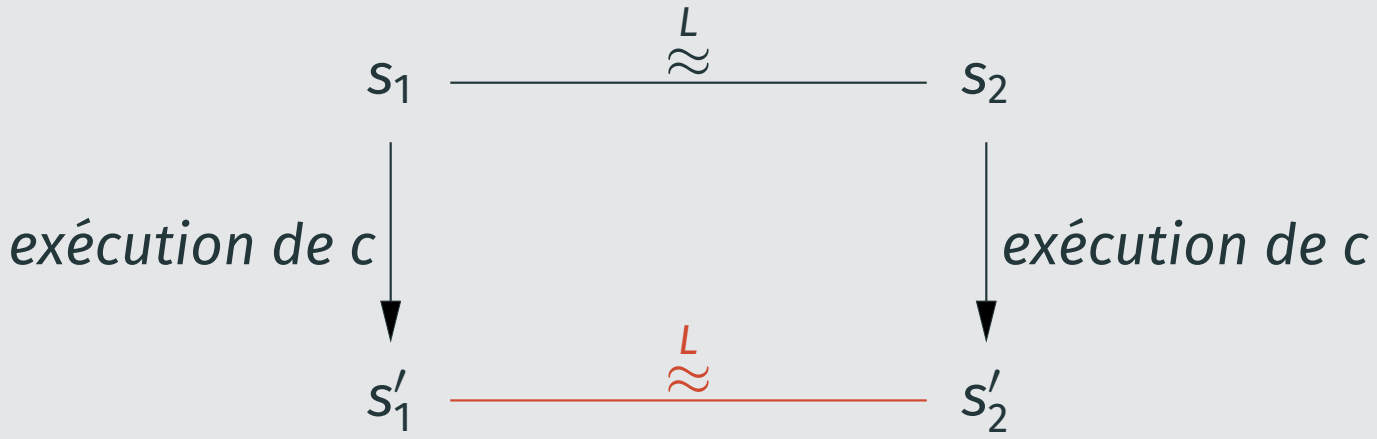
Une commande au niveau H ne modifie pas les variables L :

$$\text{si} \ \ H \vdash c : * \ \text{ et } \ c/s \Downarrow s' \ \text{ alors } \ s \overset{L}{\approx} s'$$

Démonstration de la non-interférence

Théorème

Si $pc \vdash c : * \; et \; s_1 \overset{L}{\approx} s_2 \; et \; c/s_1 \Downarrow s'_1 \; et \; c/s_2 \Downarrow s'_2$, *alors* $s'_1 \overset{L}{\approx} s'_2$.

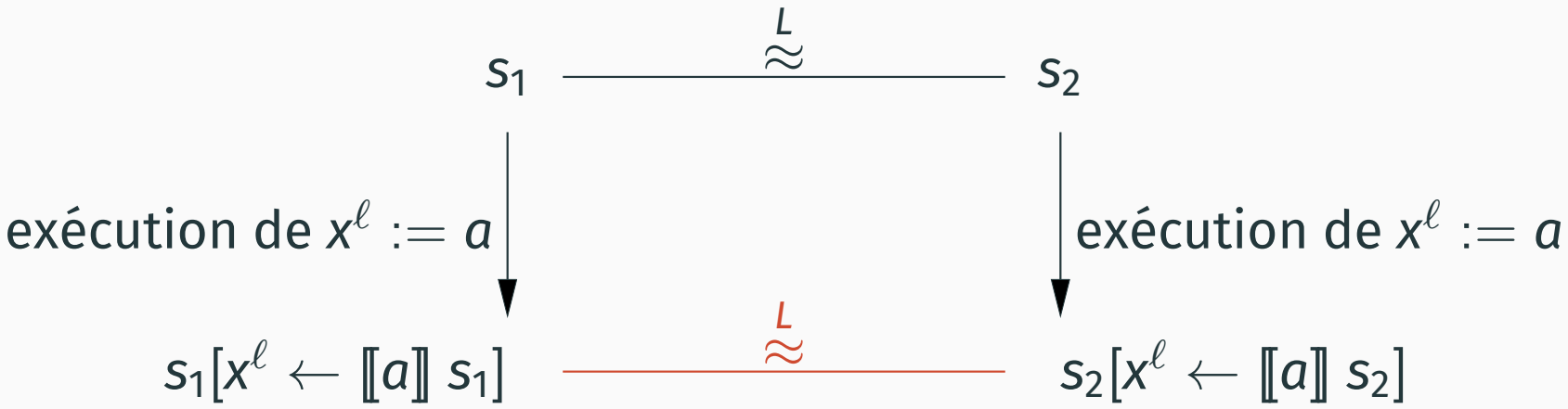


Démonstration.

Par récurrence sur la dérivation de $c/s_1 \Downarrow s'_1$ et par cas sur c. □

Démonstration de la non-interférence

Cas de l'affectation $x^\ell := a$:

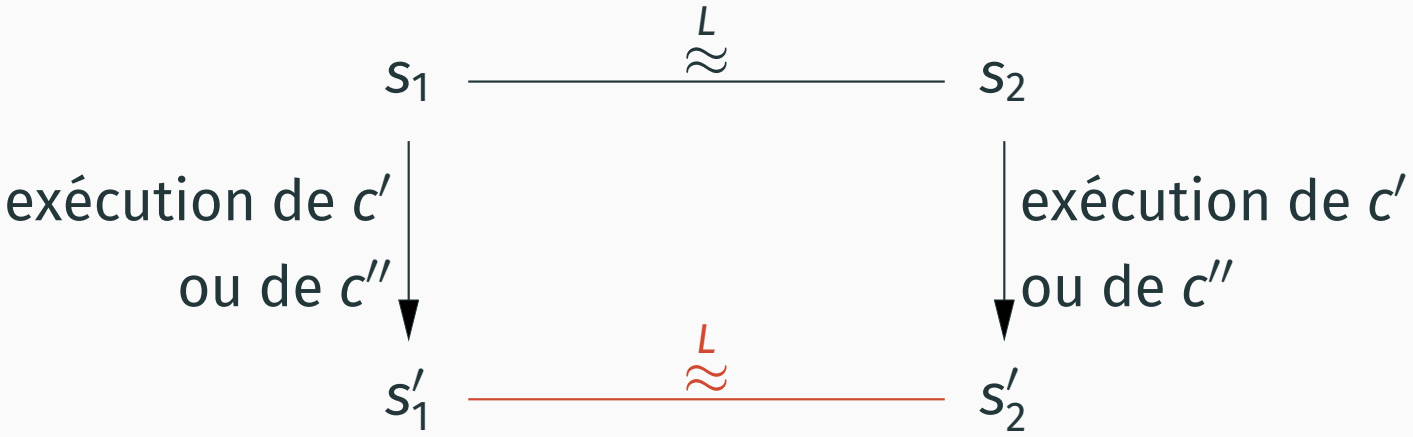


Si ℓ est L, par hypothèse de typage, $\vdash a : L$, donc $[\![a]\!]\, s_1 = [\![a]\!]\, s_2$ et $\overset{L}{\approx}$ est préservée.

Si ℓ est H, aucune variable de niveau L n'est modifiée et $\overset{L}{\approx}$ est préservée.

Démonstration de la non-interférence

Cas de la conditionnelle `if` b `then` c' `else` c'' :



Si $\vdash b : L$, on a $[\![b]\!]\, s_1 = [\![b]\!]\, s_2$, donc les deux exécutions suivent la même branche c' ou c''. Par hyp. de rec. on a $s'_1 \overset{L}{\approx} s'_2$.

Si $\vdash b : H$, les deux exécutions peuvent suivre des branches différentes. Mais par hyp. de typage on a $H \vdash c' : *$ et $H \vdash c'' : *$. D'où $s_1 \overset{L}{\approx} s'_1$ et $s_2 \overset{L}{\approx} s'_2$ et finalement $s'_1 \overset{L}{\approx} s'_2$.

La terminaison comme flux d'information

Le système de types et le critère de non-interférence précédents sont insensibles à la terminaison (*termination insensitive*).

Cependant, un programme peut terminer ou pas suivant la valeur d'un secret :

$$\texttt{if } \mathtt{s}^H \texttt{ < 0 then skip else diverge}$$

où `diverge` est la boucle infinie `while true do skip done`.

Ce programme fait ≪fuiter≫ le bit de signe de $\mathtt{s}^H$.

On considère généralement qu'observer la terminaison ou la divergence transmet au plus un bit d'information à l'attaquant.

La terminaison comme flux d'information

(Askarov, Hunt, Sabelfeld et Sands, *Termination-Insensitive Noninterference Leaks More Than Just a Bit*, ESORICS 2008.)

Ce n'est plus vrai si le programme peut communiquer sur un canal public :

```
i^L := 0;
while true do
    output i^L;
    if i^L = s^H then diverge else skip;
    i^L := i^L + 1
done
```

La valeur du secret $\mathtt{s}^H$ est le dernier entier envoyé par le programme avant de diverger.

Ce type d'attaque fait ≪fuiter≫ k bits en temps $O(2^k)$.

Parallélisme et terminaison

Si le langage permet aussi l'exécution en parallèle, on fait facilement ≪fuiter≫ k bits en temps $O(k)$:

$$\begin{array}{l} \texttt{if } \mathrm{s}^H \texttt{ land } 1 = 0 \\ \texttt{then diverge} \\ \texttt{else skip;} \\ \texttt{output } 0 \end{array} \;\Big\|\; \ldots \;\Big\|\; \begin{array}{l} \texttt{if } \mathrm{s}^H \texttt{ land } 2^{k-1} = 0 \\ \texttt{then diverge} \\ \texttt{else skip;} \\ \texttt{output } (k-1) \end{array}$$

Garantir l'équi-terminaison

On peut renforcer la règle de typage de la boucle :

$$\frac{\vdash b : L \qquad L \vdash c : *}{L \vdash \texttt{while}\ b\ \texttt{do}\ c}$$

Cela garantit que

- La condition de la boucle ne dépend pas de variables H.
  (Pas de $\texttt{while}\ x^H = 0\ \texttt{do skip done}$.)
- Une conditionnelle de niveau H ne contient pas de boucles.
  (Pas de $\texttt{if}\ x^H = 0\ \texttt{then diverge else skip}$.)

Non-interférence sensible à la terminaison

Deux exécutions d'un programme c dans deux états mémoires reliés par $\overset{L}{\approx}$ terminent toutes deux, ou divergent toutes deux.

Théorème

Si $pc \vdash c : * \text{ et } s_1 \overset{L}{\approx} s_2 \text{ et } c/s_1 \Downarrow s_1'$,

alors il existe s_2' *tel que* $c/s_2 \Downarrow s_2' \text{ et } s_1' \overset{L}{\approx} s_2'$.

Ordre supérieur : fonctions comme valeurs

On part d'un système de types usuel et on ajoute des niveaux sur les types :

$$\begin{array}{rll} \tau ::= & \texttt{int}^{\ell} \mid \texttt{bool}^{\ell} & \text{types de base} \\ & \mid (\sigma \to \tau)^{\ell} & \text{fonctions} \\ & \mid \texttt{list}(\tau)^{\ell} & \text{listes} \end{array}$$

L'ordre $\sqsubseteq$ entre niveaux induit une relation de sous-typage :

$$\frac{\ell \sqsubseteq \ell'}{\texttt{int}^{\ell} <: \texttt{int}^{\ell'}} \qquad \frac{\sigma' <: \sigma \quad \tau <: \tau' \quad \ell \sqsubseteq \ell'}{(\sigma \to \tau)^{\ell} <: (\sigma' \to \tau')^{\ell'}}$$

Règles de typage dans le cas fonctionnel pur

$$\Gamma \vdash n : \texttt{int}^{\ell} \qquad \frac{\Gamma \vdash e : \sigma \quad \sigma <: \tau}{\Gamma \vdash e : \tau}$$

$$\frac{\Gamma, x : \sigma \vdash e : \tau}{\Gamma \vdash \lambda x.e : (\sigma \to \tau)^{\ell}} \qquad \frac{\Gamma \vdash e_1 : (\sigma \to \tau)^{\ell} \quad \Gamma \vdash e_2 : \sigma \quad \ell \sqsubseteq \mathit{Label}(\tau)}{\Gamma \vdash e_1\ e_2 : \tau}$$

$$\frac{\Gamma \vdash e_1 : \texttt{bool}^{\ell} \quad \Gamma \vdash e_2 : \tau \quad \Gamma \vdash e_3 : \tau \quad \ell \sqsubseteq \mathit{Label}(\tau)}{\Gamma \vdash \texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3 : \tau}$$

Avec $\mathit{Label}(\texttt{int}^{\ell}) = \ell$, $\mathit{Label}((\sigma \to \tau)^{\ell}) = \ell$, etc.

Ajout de l'état mutable

(F. Pottier, V. Simonet, *Information flow inference for ML*, 2002, 2003.)

$$
\begin{array}{lrll}
\text{Types :} & \tau ::= & \texttt{int}^{\ell} \mid \texttt{bool}^{\ell} & \text{types de base} \\
& \mid & (\sigma \xrightarrow{pc} \tau)^{\ell} & \text{fonctions} \\
& \mid & \texttt{list}(\tau)^{\ell} & \text{listes} \\
& \mid & \texttt{ref}(\tau)^{\ell} & \text{références mutables}
\end{array}
$$

Il faut maintenant garder trace des flux indirects en ajoutant un niveau *pc* au prédicat de typage

$$pc, \Gamma \vdash e : \tau$$

mais aussi comme effet latent dans les types de fonctions

$$(\sigma \xrightarrow{pc} \tau)^{\ell}$$

Règles de typage avec état mutable

$$\frac{pc, \Gamma \vdash e_1 : \texttt{ref}(\tau)^{\ell} \qquad pc, \Gamma \vdash e_2 : \tau \qquad \ell \sqcup pc \sqsubseteq \mathit{Label}(\tau)}{pc, \Gamma \vdash e_1 := e_2 : \texttt{unit}}$$

$$\frac{pc, \Gamma \vdash e_1 : \texttt{bool}^{\ell} \qquad pc \sqcup \ell, \Gamma \vdash e_2 : \tau \qquad pc \sqcup \ell, \Gamma \vdash e_3 : \tau \qquad \ell \sqsubseteq \mathit{Label}(\tau)}{pc, \Gamma \vdash \texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 : \tau}$$

$$\frac{pc', \Gamma, x : \sigma \vdash e : \tau}{pc, \Gamma \vdash \lambda x.e : (\sigma \xrightarrow{pc'} \tau)^{\ell}}$$

$$\frac{\Gamma \vdash e_1 : (\sigma \xrightarrow{pc'} \tau)^{\ell} \qquad \Gamma \vdash e_2 : \sigma \qquad \ell \sqsubseteq \mathit{Label}(\tau) \qquad pc \sqsubseteq pc'}{pc, \Gamma \vdash e_1\ e_2 : \tau}$$

Logiques de programmes et auto-composition

Limitations des systèmes de types pour les flux d'information

Les systèmes de types pour les flux d'information sont parfois trop stricts et rejettent des programmes qui ne contiennent pas de flux dangereux et satisfont la propriété de non-interférence.

Exemples : (`s` est au niveau *H* et `x` au niveau *L*)

```
x := s; x := 0
```

```
x := x + s;  ...;  x := x - s
```

```
assert (s >= 0);
x := s;
while x > 0 do ... ; x := x - 1 done
```

Dans ces 3 programmes, la valeur finale de `x` ne dépend pas de la valeur initiale de `s`.

Vérifier la non-interférence avec une logique de programmes

Pour un programme c donné, on voudrait vérifier directement la propriété de non-interférence

$$s_1 \overset{L}{\approx} s_2 \ \wedge \ c/s_1 \Downarrow s_1' \ \wedge \ c/s_2 \Downarrow s_2' \implies s_1' \overset{L}{\approx} s_2'$$

en s'aidant d'une logique de programmes appropriée.

(Voir aussi : mon cours de 2020–2021 sur les logiques de programmes.)

La logique de Hoare (rappel)

Des règles pour le prédicat

$$\{\, P \,\}\ c\ \{\, Q \,\}$$

qui signifie

$$\forall s, s',\quad P(s) \wedge c/s \Downarrow s' \Rightarrow Q(s')$$

Les préconditions P et postconditions Q sont des prédicats sur les états mémoires s.

La logique de Hoare relationnelle

Des règles pour le prédicat

$$\{\, P \,\}\ c_1 \mid c_2\ \{\, Q \,\}$$

qui signifie

$$\forall s_1, s_2, s_1', s_2',\quad P(s_1, s_2) \wedge c_1/s_1 \Downarrow s_1' \wedge c_2/s_2 \Downarrow s_2' \Rightarrow Q(s_1', s_2')$$

Les préconditions P et postconditions Q sont des relations entre deux états mémoires s_1, s_2.

Application à la non-interférence : le programme c satisfait la condition de non-interférence si et seulement si

$$\{\, \overset{L}{\approx} \,\}\ c \mid c\ \{\, \overset{L}{\approx} \,\}$$

Quelques règles «diagonales»

(D. A. Naumann, *37 years of relational Hoare logic*, 2020)

$$\{\, Q[x_1 \leftarrow a_1, x_2 \leftarrow a_2] \,\}\ x_1 := a_1 \mid x_2 := a_2\ \{\, Q \,\}$$

$$\frac{\{\, P \,\}\ c_1 \mid c_2\ \{\, Q \,\} \qquad \{\, Q \,\}\ c_1' \mid c_2'\ \{\, R \,\}}{\{\, P \,\}\ c_1; c_1' \mid c_2; c_2'\ \{\, R \,\}}$$

$$\frac{\begin{array}{cc}\{\, P \wedge b_1 \wedge b_2 \,\}\ c_1 \mid c_2\ \{\, Q \,\} & \{\, P \wedge \neg b_1 \wedge \neg b_2 \,\}\ c_1' \mid c_2'\ \{\, Q \,\} \\ \{\, P \wedge b_1 \wedge \neg b_2 \,\}\ c_1 \mid c_2'\ \{\, Q \,\} & \{\, P \wedge \neg b_1 \wedge b_2 \,\}\ c_1' \mid c_2\ \{\, Q \,\}\end{array}}{\{\, P \,\}\ \texttt{if}\ b_1\ \texttt{then}\ c_1\ \texttt{else}\ c_1' \mid \texttt{if}\ b_2\ \texttt{then}\ c_2\ \texttt{else}\ c_2'\ \{\, Q \,\}}$$

$$\frac{\begin{array}{c}Q \Rightarrow b_1 = b_2 \vee (b_1 \wedge L) \vee (b_2 \wedge R) \\ \{\, Q \wedge b_1 \wedge b_2 \wedge \neg L \wedge \neg R \,\}\ c_1 \mid c_2\ \{\, P \,\} \\ \{\, Q \wedge b_1 \wedge L \,\}\ c_1 \mid \texttt{skip}\ \{\, Q \,\} \qquad \{\, Q \wedge b_2 \wedge R \,\}\ \texttt{skip} \mid c_2\ \{\, Q \,\}\end{array}}{\{\, P \,\}\ \texttt{while}\ b_1\ \texttt{do}\ c_1\ \texttt{done} \mid \texttt{while}\ b_2\ \texttt{do}\ c_2\ \texttt{done}\ \{\, P \wedge \neg b_1 \wedge \neg b_2 \,\}}$$

Quelques règles ≪gauches≫ (+ règles ≪droites≫ symétriques)

$$\{\, Q[x_1 \leftarrow a_1] \,\}\; x_1 := a_1 \mid \texttt{skip}\; \{\, Q \,\}$$

$$\frac{\{\, P \,\}\; c_1 \mid \texttt{skip}\; \{\, Q \,\} \qquad \{\, Q \,\}\; c_1' \mid c_2'\; \{\, R \,\}}{\{\, P \,\}\; c_1; c_1' \mid c_2'\; \{\, R \,\}}$$

$$\frac{\{\, P \wedge b_1 \,\}\; c_1 \mid c_2\; \{\, Q \,\} \qquad \{\, P \wedge \neg b_1 \,\}\; c_1' \mid c_2\; \{\, Q \,\}}{\{\, P \,\}\; \texttt{if}\; b_1\; \texttt{then}\; c_1\; \texttt{else}\; c_1' \mid c_2\; \{\, Q \,\}}$$

$$\frac{\begin{array}{c}\{\, P \,\}\; \texttt{while}\; b_1\; \texttt{do}\; c_1\; \texttt{done} \mid c_2\; \{\, Q \,\} \\ \{\, Q \,\}\; \texttt{while}\; b_1\; \texttt{do}\; c_1\; \texttt{done} \mid c_2'\; \{\, R \,\} \qquad Q \wedge \neg b_1 \Rightarrow R\end{array}}{\{\, P \,\}\; \texttt{while}\; b_1\; \texttt{do}\; c_1\; \texttt{done} \mid c_2; c_2'\; \{\, R \,\}}$$

Réduction à la logique de Hoare usuelle

(N. Francez, *Product properties and their verification*, 1983)

Si les variables V_1 utilisées par c_1 sont disjointes des variables V_2 utilisées par c_2, alors

$$\{\,\bar{P}\,\}\ c_1; c_2\ \{\,\bar{Q}\,\} \quad \text{implique} \quad \{\,P\,\}\ c_1 \mid c_2\ \{\,Q\,\}$$

où $\bar{P}$ est le prédicat obtenu à partir de la relation P par

$$\bar{P}(s) \stackrel{def}{=} P(s_{|V_1}, s_{|V_2})$$

Idée de la démonstration : si $c_1/s_1 \Downarrow s_1'$ et $c_2/s_2 \Downarrow s_2'$, on se ramène au cas $Dom(s_i) \subseteq V_i$ et $Dom(s_i') \subseteq V_i$, et alors

$$\frac{c_1/(s_1 \uplus s_2) \Downarrow (s_1' \uplus s_2) \qquad c_2/(s_1' \uplus s_2) \Downarrow (s_1' \uplus s_2')}{c_1; c_2/(s_1 \uplus s_2) \Downarrow (s_1' \uplus s_2')}$$

Auto-composition

(G. Barthe, P. D'Argenio, T. Rezk, *Secure information flow by self-composition*, 2004)

Pour montrer la propriété de non-interférence pour un programme c, il suffit donc de prendre deux copies de c où les variables sont renommées :

$$c_1 = c\{x \leftarrow x_1 \mid x \in \mathit{Vars}(c)\} \qquad c_2 = c\{x \leftarrow x_2 \mid x \in \mathit{Vars}(c)\}$$

et de montrer en logique de Hoare usuelle

$$\{\, L \,\}\ c_1; c_2\ \{\, L \,\}$$

où L est l'assertion «les variables de niveau L sont égales» :

$$L = \bigwedge \{x_1 = x_2 \mid x \in \mathit{Vars}(c), x \text{ de niveau } L\}$$

Exemple de vérification par auto-composition

Pour le programme $x := x + s; x := x - s$

$$\{ x_1 = x_2 \} \Rightarrow$$
$$\{ (x_1 + s_1) - s_1 = (x_2 + s_2) - s_2 \}$$
$$x_1 := x_1 + s_1;$$
$$\{ x_1 - s_1 = (x_2 + s_2) - s_2 \}$$
$$x_1 := x_1 - s_1;$$
$$\{ x_1 = (x_2 + s_2) - s_2 \}$$
$$x_2 := x_2 + s_2;$$
$$\{ x_1 = x_2 - s_2 \}$$
$$x_2 := x_2 - s_2;$$
$$\{ x_1 = x_2 \}$$

Exemple de vérification par auto-composition

Pour $\texttt{assert}(s \geq 0); x := s; \texttt{while}\ x > 0\ \texttt{do}\ x := x - 1\ \texttt{done}$

$$
\begin{array}{ll}
 & \{ x_1 = x_2 \} \Rightarrow \\
 & \{ T \} \\
\texttt{assert}(s_1 \geq 0); & \{ s_1 \geq 0 \} \\
x_1 := s_1; & \{ x_1 \geq 0 \} \\
\texttt{while}\ x_1 > 0\ \texttt{do} & \{ x_1 > 0 \} \\
\quad x_1 := x_1 - 1 & \{ x_1 \geq 0 \} \\
\texttt{done}; & \{ x_1 = 0 \} \\
\texttt{assert}(s_2 \geq 0); & \{ x_1 = 0 \wedge s_2 \geq 0 \} \\
x_2 := s_2; & \{ x_1 = 0 \wedge x_2 \geq 0 \} \\
\texttt{while}\ x_2 > 0\ \texttt{do} & \{ x_1 = 0 \wedge x_2 > 0 \} \\
\quad x_2 := x_2 - 1 & \{ x_1 = 0 \wedge x_2 \geq 0 \} \\
\texttt{done}; & \{ x_1 = 0 \wedge x_2 = 0 \} \\
 & \Rightarrow \{ x_1 = x_2 \}
\end{array}
$$

Déclassification et approbation

Un document déclassifié

Attachment 1

[redacted]

Work Scope

1. Introduction

Raytheon seeks to sponsor fundamental research in order to [redacted]

Déclassification

Diminuer volontairement le niveau de confidentialité de certaines données ou résultats de calculs.

Exemple : vérification d'un mot de passe.

```
let checkpwd (input: string^H) (hashed_password: string^H)
             : bool^L =
  let res : bool^H = (hash(input) = hashed_password) in
  declassify(res)
```

Quelques techniques :

- Caviardage manuel + coup de tampon / signature crypto
- Révéler ≤ 1 bit du secret (comme dans `checkpwd`)
- Chiffrer ou hacher le secret

Approbation (*endorsement*)

Augmenter volontairement le niveau d'intégrité de certaines données.

Exemple : validation d'un code postal entré sur une page Web.

```
let checkzip (input: string^L) : string^H =
  if DB.search zip_database input = Found
  then endorse(input)
  else raise "bad ZIP code"
```

Quelques techniques :

- Vérification manuelle + coup de tampon / signature crypto.
- Croisement avec des données fiables (cf. `checkzip`).

Fonctions de déclassification ?

Il est risqué de présenter la déclassification sous forme de fonctions $H \to L$ utilisables un nombre arbitraire de fois.

Exemple : avec $\texttt{checkpwd} : \texttt{string}^H \to \texttt{string}^H \to \texttt{bool}^L$, on peut faire ≪fuiter≫ tous les bits d'un secret s^H.

```
for b^H in bits(s^H) do
    let c^H = if b^H then "1" else "0" in
    let z^L = checkpwd c^H (hash("1")) in
    output(z^L)
done
```

Fonctions de déclassification ?

Il est risqué de présenter la déclassification sous forme de fonctions $H \to L$ utilisables un nombre arbitraire de fois.

Exemple : avec une fonction de chiffrement
$\texttt{encrypt} : \texttt{key}^H \to \texttt{string}^H \to \texttt{string}^L$.

```
for b^H in bits(s^H) do
    let c^H = if b^H then "X" else "" in
    let z^L = enc k^H c^H in
    output(z^L)
done
```

Tous les bits ≪fuitent≫ si le chiffrement préserve la longueur du texte clair, ou juste s'il est déterministe.

Politiques de déclassification globales

(Li & Zdancewicz, *Downgrading Policies and Relaxed Noninterference*, 2015.)

Un ensemble de fonctions F_i qui peuvent être appliquées aux entrées secrètes du programme pour produire des résultats déclassifiés. Par exemples, pour le programme suivant :

```
let checkpwd (input: string^H) (hashed: string^H) =
  hash(input) = hashed
```

Une fonction de déclassification : $F(i, h) = (\texttt{hash}(i) = h)$.

La valeur $F(\texttt{input}, \texttt{hashed}) = (\texttt{hash}(\texttt{input}) = \texttt{hashed})$
est déclassifiée et utilisable au niveau L.

Toute autre comparaison de hashes n'est pas déclassifiée.

Déclassification et non-interférence

Relaxation du critère de non-interférence : les sorties de niveau L ne dépendent que des entrées de niveau L et des valeurs déclassifiées, c.à.d. les valeurs des F_i appliquées aux entrées H.

En logique de Hoare relationnelle :

$$\{ \overset{L}{\approx} \wedge \mathcal{D} \}\ c \mid c\ \{ \overset{L}{\approx} \}$$

où $\mathcal{D}$ exprime l'égalité des valeurs déclassifiées dans les 2 états :

$$\mathcal{D}(s_1, s_2) \overset{def}{=} \bigwedge_i F_i(s_1^H) = F_i(s_2^H)$$

Point d’étape

Bilan sur la sécurité multi-niveaux et les flux d'information

Les systèmes multi-niveaux ≪sécurité défense≫ tels qu'envisagés par Bell et LaPadula sont peu nombreux…

… mais des systèmes comme Android, iOS et Windows intègrent des politiques d'intégrité dans le style de Biba…

… et des problèmes similaires de confidentialité et d'intégrité se présentent dans bien d'autres contextes, à commencer par les pages Web.

La notion de flux d'information est cruciale pour assurer la confidentialité et l'intégrité des données.

Bilan sur l'analyse des flux d'information

L'analyse des flux d'information (par typage ou par logiques de programmes) est très stricte…

… mais efficace pour identifier les endroits du code où une déclassification ou une approbation se produit

… et a d'autres utilisations,
p.ex. pour le ≪temps constant≫ ($\rightarrow$ cours du 31 mars).

Pour aller plus loin

Typage des flux d'information pour des ≪vrais≫ langages :
fonctions, objets, exceptions, parallélisme, non-déterminisme, …

(Voir p.ex. JIF, ≪Java + Information Flow≫ par Myers *et al*, `https://www.cs.cornell.edu/jif/`.)

Prise en compte d'autres canaux d'information :
temps d'exécution ($\rightarrow$ cours du 31 mars), consommation électrique, émissions électromagnétiques, …

Raisonner sur la quantité d'information qui ≪fuite≫ :
théorie de l'information, modèles bayésiens, …

(Voir le livre de Alvim *et al*, *The Science of Quantitative Information Flow*, Springer, 2020.)