Outline

1. Introduction
   - Goal
   - Non-interference
   - Preliminaries

2. Tracking Information Flow
   - Semantics
   - Properties
   - Example
   - Problem

3. Testing

4. Yes, but . . .

5. Conclusion
Goal

Any External Program

\( \langle f, g \rangle \)

Input Interface \( \sigma_h, \sigma_l \) \( \rightarrow \) \( \langle f, g \rangle \) \( \rightarrow \) Internal Store \( \langle f, g \rangle \) \( \rightarrow \) Output Interface

\( g(\sigma_h, \sigma_l) \)

\( f(\sigma_h, \sigma_l) \)
Semantics with Information Flow Monitoring

\[
\forall o \in \text{PublicOutput} : \\
\quad g'(\sigma_l)(o) = g(\sigma_h, \sigma_l)(o) \quad \lor \quad g'(\sigma_l)(o) = \bot
\]
NON-INTERFERENCE
Presentation of the concept of non-interference

- Introduced by Goguen and Meseguer
- Property of a program respecting secrets confidentiality

\[
\begin{array}{c|c}
\text{input stores} & h & l \\
\hline
\text{program} & \text{ } & \text{ } \\
\hline
\text{output stores} & h & l \\
\end{array}
\]
Introducing the concept of non-interference

- Introduced by Goguen and Meseguer
- Property of a program respecting secrets confidentiality

Input stores:

\[
\begin{array}{c}
\hline
h \\
\hline
l \\
\end{array}
\]

Program:

Output stores:

\[
\begin{array}{c}
\hline
h \\
\hline
l \\
\end{array}
\]
Introduce by Goguen and Meseguer

Property of a program respecting secrets confidentiality

Input stores $h \downarrow l$

Program:

Output stores $h \downarrow l$
NON-INTERFERENCE
Presentation of the concept of non-interference

- Introduced by Goguen and Meseguer
- Property of a program respecting secrets confidentiality

---

input stores $\begin{array}{c} h \\ l \end{array}$:

```
program :
```

output stores $\begin{array}{c} h \\ l \end{array}$:
Introduced by Goguen and Meseguer

Property of a program respecting secrets confidentiality

Input stores $h_l$

Program:

Output stores $h_l$
NON-INTERFERENCE
Presentation of the concept of non-interference

- Introduced by Goguen and Meseguer
- Property of a program respecting secrets confidentiality

input stores \( h \) : 
program : 
output stores \( h \) : 

NON-INTERFERENCE
Presentation of the concept of non-interference

- Introduced by Goguen and Meseguer
- Property of a program respecting secrets confidentiality

\[
\begin{array}{c}
\text{input stores } \begin{bmatrix} h \\ l \end{bmatrix} : \\
\text{program :} \\
\text{output stores } \begin{bmatrix} h \\ l \end{bmatrix} :
\end{array}
\]
Non-interference
Presentation of the concept of non-interference

- Introduced by Goguen and Meseguer
- Property of a program respecting secrets confidentiality

Input stores $\frac{h}{l}$:

Program:

Output stores $\frac{h}{l}$:
NON-INTERFERENCE
Presentation of the concept of non-interference

- Introduced by Goguen and Meseguer
- Property of a program respecting secrets confidentiality
NON-INTERFERENCE
Presentation of the concept of non-interference

- Introduced by Goguen and Meseguer
- Property of a program respecting secrets confidentiality

```
input stores \( \begin{array}{c} h \\ l \end{array} \) :

program :

output stores \( \begin{array}{c} h \\ l \end{array} \) :
```
INTRODUCTION

Presentation of the concept of non-interference

- Introduced by Goguen and Meseguer
- Property of a program respecting secrets confidentiality

```
input stores h : l

program :

output stores h : l
```
Introduce by Goguen and Meseguer

Property of a program respecting secrets confidentiality

```plaintext
input stores h l:

program:

output stores h l:
```
NON-INTERFERENCE
Presentation of the concept of non-interference

- Introduced by Goguen and Meseguer
- Property of a program respecting secrets confidentiality

Input stores $h \downarrow l$

Program:

Output stores $h \downarrow l$
Definition 1 (Sabelfeld & Myers)

\[ \forall s_1, s_2 \in S. s_1 =_L s_2 \Rightarrow [C]s_1 \approx_L [C]s_2 \]

- Weaknesses:
  - not fitted for monitoring
  - statically difficult

Example 2

```plaintext
x := 0 ; tmp := 0 ;
if test1(l) then tmp := h else skip end ;
if test2(l) then x := tmp else skip end ;
tmp := 0 ;
```
Non-interfering execution

**Main Goal**: being able to detect executions respecting the confidentiality of secret data independently from other executions

**Definition 3 (Non-interfering execution)**

\[ \forall s_1. \text{NIExec}(C, s_1) \equiv \forall s_2. s_1 =_L s_2 \Rightarrow [C]s_1 \approx_L [C]s_2 \]
Non-interfering execution

**Main Goal**: being able to detect executions respecting the confidentiality of secret data independently from other executions

**Definition 3 (Non-interfering execution)**

\[
\forall s_1. \text{NIExec}(C, s_1) \equiv \forall s_2. s_1 =_L s_2 \Rightarrow [C]s_1 \simeq_L [C]s_2
\]
Main Goal: being able to detect executions respecting the confidentiality of secret data independently from other executions

Definition 3 (Non-interfering execution)

\[ \forall s_1. \text{NIExec}(C, s_1) \equiv \forall s_2. s_1 =_L s_2 \Rightarrow [C]s_1 \approx_L [C]s_2 \]
Main Goal: being able to detect executions respecting the confidentiality of secret data independently from other executions

Definition 3 (Non-interfering execution)

\[ \forall s_1. \text{NIE}_{\text{Exec}}(C, s_1) \equiv \forall s_2. s_1 =_{L} s_2 \Rightarrow \llbracket C \rrbracket s_1 \approx_{L} \llbracket C \rrbracket s_2 \]
Main Goal: being able to detect executions respecting the confidentiality of secret data independently from other executions

Definition 3 (Non-interfering execution)

\[
\forall s_1. \text{NIExec}(C, s_1) \equiv \forall s_2. s_1 =_L s_2 \Rightarrow [C]s_1 \approx_L [C]s_2
\]
Some properties

**Fact 4 (Predicate Safe)**

\[ \forall s_1 \in S. \text{Safe}([C]s_1) \Rightarrow \text{NIExec}(C, s_1) \]

**Corollary 5 (Definition of low-equivalence is symmetric)**

\[ \forall s_1. \text{NIExec}(C, s_1) \Rightarrow (\forall s_2. s_2 =_L s_1 \Rightarrow \text{NIExec}(C, s_2)) \]

**Corollary 6**

\[ \forall s_1. \text{Safe}([C]s_1) \Rightarrow (\forall s_2. s_2 =_L s_1 \Rightarrow \text{NIExec}(C, s_2)) \]

**Benefit:** one execution may be sufficient to deduce a property of many executions
Some properties

Fact 4 (Predicate Safe)
\[ \forall s_1 \in S. \text{Safe}([C]s_1) \Rightarrow \text{NIExec}(C, s_1) \]

Corollary 5 (Definition of low-equivalence is symmetric)
\[ \forall s_1. \text{NIExec}(C, s_1) \Rightarrow (\forall s_2. s_2 =_L s_1 \Rightarrow \text{NIExec}(C, s_2)) \]

Corollary 6
\[ \forall s_1. \text{Safe}([C]s_1) \Rightarrow (\forall s_2. s_2 =_L s_1 \Rightarrow \text{NIExec}(C, s_2)) \]

Benefit: one execution may be sufficient to deduce a property of many executions
Some properties

Fact 4 (Predicate Safe)

\[ \forall s_1 \in S. \text{Safe}(\lbrack C \rbrack s_1) \Rightarrow \text{NIExec}(C, s_1) \]

Corollary 5 (Definition of low-equivalence is symmetric)

\[ \forall s_1. \text{NIExec}(C, s_1) \Rightarrow (\forall s_2. s_2 =_L s_1 \Rightarrow \text{NIExec}(C, s_2)) \]

Corollary 6

\[ \forall s_1. \text{Safe}(\lbrack C \rbrack s_1) \Rightarrow (\forall s_2. s_2 =_L s_1 \Rightarrow \text{NIExec}(C, s_2)) \]

**Benefit**: one execution may be sufficient to deduce a property of many executions
Language’s Grammar

\[ \begin{align*}
  v &::= c \\
  e &::= e_1 \ op \ e_2 \mid id \mid v \\
  S &::= \text{if } e \text{ then } S \text{ else } S \text{ end} \\
    &\mid \text{while } e \text{ do } S \text{ done} \\
    &\mid id := e \\
    &\mid \text{skip} \\
    &\mid S ; S
\end{align*} \]

*id* stands for any variable identifier (name)
general idea:

- data are tagged ($\bot \subseteq \top$)
  - $\bot$ (public) $\Rightarrow$ same value for any low-equivalent execution
  - $\top$ (secret) $\Rightarrow$ value may be different

- semantics updates tags
- Safe iff low outputs are tagged with $\bot$

when branching on a condition which is:

- low: execute the designated branch
- high: merge the result of executing the designated branch and analyzing the other one

Example 7

```plaintext
l := 0;
if h then skip else ? end;
```
general idea:
- data are tagged ($\bot \sqsubseteq \top$)
  - $\bot$ (public) $\Rightarrow$ same value for any low-equivalent execution
  - $\top$ (secret) $\Rightarrow$ value may be different

semantics updates tags
- Safe iff low outputs are tagged with $\bot$

when branching on a condition which is:
- low: execute the designated branch
- high: merge the result of executing the designated branch and analyzing the other one

Example 7
```plaintext
l := 0;
if h then skip else ? end;
```
General Description

- **general idea:**
  - data are tagged ($\bot \sqsubseteq \top$)
    - $\bot$ (public) $\Rightarrow$ same value for any low-equivalent execution
    - $\top$ (secret) $\Rightarrow$ value may be different
  - semantics updates tags
  - **Safe iff** low outputs are tagged with $\bot$

- when branching on a condition which is:
  - low: execute the designated branch
  - high: merge the result of executing the designated branch and analyzing the other one

**Example 7**

```lisp
l := 0;
if h then skip else ? end;
```
General Description

- general idea:
  - data are tagged \((\bot \subseteq \top)\)
    - \(\bot\) (public) \(\Rightarrow\) same value for any low-equivalent execution
    - \(\top\) (secret) \(\Rightarrow\) value may be different
  - semantics updates tags
  - \text{Safe if and only if} low outputs are tagged with \(\bot\)

- when branching on a condition which is:
  - low: execute the designated branch
  - high: merge the result of executing the designated branch and analyzing the other one

Example 7

```plaintext
l := 0;
if h then skip else ? end;
```
General Description

- general idea:
  - data are tagged ($\bot \sqsubseteq \top$)
    - $\bot$ (public) $\Rightarrow$ same value for any low-equivalent execution
    - $\top$ (secret) $\Rightarrow$ value may be different
  - semantics updates tags
  - Safe iff low outputs are tagged with $\bot$

- when branching on a condition which is:
  - low: execute the designated branch
  - high: merge the result of executing the designated branch and analyzing the other one

Example 7

```plaintext
l := 0;
if h then skip else ? end;
```
general idea:
- data are tagged (⊥ ⊑ ⊤)
  - ⊥ (public) ⇒ same value for any low-equivalent execution
  - ⊤ (secret) ⇒ value may be different
- semantics updates tags
- Safe iff low outputs are tagged with ⊥

when branching on a condition which is:
- low : execute the designated branch
- high : merge the result of executing the designated branch and analyzing the other one

Example 7

l := 0;
if h then skip else skip end;
General Description

- general idea:
  - data are tagged ($\bot \sqsubseteq \top$)
    - $\bot$ (public) $\Rightarrow$ same value for any low-equivalent execution
    - $\top$ (secret) $\Rightarrow$ value may be different
  - semantics updates tags
  - Safe iff low outputs are tagged with $\bot$

- when branching on a condition which is:
  - low: execute the designated branch
  - high: merge the result of executing the designated branch and analyzing the other one

Example 7

```
l := 0;
if h then skip else l := 1 end;
```
Semantics judgments

\[(\text{Id} \rightarrow \text{Value}); (\text{Id} \rightarrow \text{Tag}) \vdash \text{Expr} \downarrow \text{Value} : \text{Tag}\\
(\text{Id} \rightarrow \text{Value}); (\text{Id} \rightarrow \text{Tag}) \vdash \text{S} \downarrow (\text{Id} \rightarrow \text{Value}) : (\text{Id} \rightarrow \text{Tag}) : \mathcal{P}(\text{Id})\]

Example 8

\[
\text{if } h \text{ then}\\
\quad \text{l := true ;}\\
\text{if l then skip else x :=1}
\]
Semantics judgments

\[(\text{Id} \rightarrow \text{Value}); (\text{Id} \rightarrow \text{Tag}) \vdash \text{Expr} \downarrow \text{Value} : \text{Tag}\]

\[(\text{Id} \rightarrow \text{Value}); (\text{Id} \rightarrow \text{Tag}) \vdash S \downarrow (\text{Id} \rightarrow \text{Value}) : (\text{Id} \rightarrow \text{Tag}) : \mathcal{P}(\text{Id})\]

Example 8

if h then
  l := true ;
else
  if l then skip else x :=1
The analysis

\[ (\text{Id} \rightarrow \forall \text{value}); (\text{Id} \rightarrow \text{Tag}) \vdash S^g = (\hat{D}, \hat{X}) \]

- \( \hat{D} = \mathcal{P}(\text{Id} \times \text{Id}) \)
  - over-approximation of the dependencies between initial and final values of variables
- \( \hat{X} = \mathcal{P}(\text{Id}) \)
  - over-approximation of the set of variables which may be assigned to

Gurvan Le Guernic

Monitoring Information Flow
## Rules (1)

\[
\sigma; \rho \vdash e \downarrow v : \text{true}
\]

\[
\sigma; \rho \vdash \text{id} := e \downarrow \sigma[\text{id} \mapsto v] : \rho[\text{id} \mapsto \text{true}] : \{\text{id}\}
\]

\[
\sigma; \rho \vdash e \downarrow v : \perp \quad \sigma; \rho \vdash S_v \downarrow \sigma_v : \rho_v : X
\]

\[
\rho_e = (X_{\text{if}} \times \{\top\}) \cup ((\mathbb{I} \text{id} - X_{\text{if}}) \times \{\perp\})
\]

\[
\sigma; \rho \vdash \text{if } e \text{ then } S_{\text{true}} \text{ else } S_{\text{false}} \text{ end} \downarrow \sigma_v : \rho_v \sqcup \rho_e : X
\]
### Rules (1)

\[
\sigma; \rho \vdash e \downarrow v : t^e
\]

\[
\sigma; \rho \vdash id := e \downarrow \sigma[id \mapsto v] : \rho[id \mapsto t^e] : \{id\}
\]

\[
\begin{align*}
\sigma; \rho \vdash e & \downarrow v : \bot \\
\sigma; \rho \vdash S_v & \downarrow \sigma_v : \rho_v : X
\end{align*}
\]

\[
\rho_e = (X_{if} \times \{\top\}) \cup ((\mathbb{I}d - X_{if}) \times \{\bot\})
\]

\[
\sigma; \rho \vdash \text{if } e \text{ then } S\text{true else } S\text{false end} \downarrow \sigma_v : \rho_v \amalg \rho_e : X
\]
\[ \sigma; \rho \vdash e \Downarrow v : \top \quad \sigma; \rho \vdash S_v \Downarrow \sigma_v : \rho_v : X_v \]

\[ \llbracket \sigma; \rho \vdash S_{\neg v} \rrbracket^g_G = (\hat{\mathcal{D}}, \hat{X}) \quad \rho_{\neg v} = \lambda x. \bigsqcup_{y \in \hat{\mathcal{D}}(x)} \rho(y) \]

\[ X_{if} = X_v \cup \hat{X} \quad \rho_e = (X_{if} \times \{\top\}) \cup ((\mathbb{I}d - X_{if}) \times \{\bot\}) \]

\[ \sigma; \rho \vdash \text{if } e \text{ then } S_{true} \text{ else } S_{false} \text{ end} \Downarrow \sigma_v : \rho_{v} \bowtie \rho_{\neg v} \bowtie \rho_e : X_{if} \]
\[\sigma; \rho \vdash e \downarrow v : T\]
\[\sigma; \rho \vdash S_v \downarrow \sigma_v : \rho_v : X_v\]
\[[\sigma; \rho \vdash S_{\neg v}]^g = (\hat{D}, \hat{X})\]
\[\rho_{\neg v} = \lambda x. \bigcup_{y \in \hat{D}(x)} \rho(y)\]
\[X_{if} = X_v \cup \hat{X}\]
\[\rho_e = (X_{if} \times \{\top\}) \cup ((\mathbb{I}d - X_{if}) \times \{\bot\})\]
\[\sigma; \rho \vdash \text{if } e \text{ then } S_{\text{true}} \text{ else } S_{\text{false}} \text{ end } \downarrow \sigma_v : \rho_v \parallel \rho_{\neg v} \parallel \rho_e : X_{if}\]
Rules (2)

\[
\sigma; \rho \vdash e \downarrow v : \top \\
\sigma; \rho \vdash S_v \downarrow \sigma_v : \rho_v : X_v \\
\begin{align*}
\llbracket \sigma; \rho \vdash S_{\neg v} \rrbracket^g &= (\hat{D}, \hat{X}) \\
\rho_{\neg v} &= \lambda x. \bigsqcup_{y \in \hat{D}(x)} \rho(y) \\
X_{if} &= X_v \cup X \\
\rho_e &= (X_{if} \times \{\top\}) \cup ((\Id - X_{if}) \times \{\bot\})
\end{align*}
\]

\[
\sigma; \rho \vdash \text{if } e \text{ then } S_{\text{true}} \text{ else } S_{\text{false}} \text{ end} \downarrow \sigma_v : \rho_v \parallel \rho_{\neg v} \parallel \rho_e : X_{if}
\]
Rules (2)

\[
\sigma; \rho \vdash e \downarrow v : \top \quad \sigma; \rho \vdash S_v \downarrow \sigma_v : \rho_v : X_v \\
\llbracket \sigma; \rho \vdash S_{\neg v} \rrbracket^g = (\hat{D}, \hat{X}) \quad \rho_{\neg v} = \lambda x. \bigsqcup_{y \in \hat{D}(x)} \rho(y) \\
X_{if} = X_v \cup \hat{X} \quad \rho_e = (X_{if} \times \{\top\}) \cup ((\mathbb{I}d - X_{if}) \times \{\bot\}) \\
\sigma; \rho \vdash \text{if } e \text{ then } S_{true} \text{ else } S_{false} \text{ end} \downarrow \sigma_v : \rho_v \sqcup \rho_{\neg v} \sqcup \rho_e : X_{if}
\]
Rules (2)

\[
\begin{align*}
\sigma; \rho \vdash e \Downarrow v : \top & \quad \sigma; \rho \vdash S_v \Downarrow \sigma_v : \rho_v : X_v \\
\llbracket \sigma; \rho \vdash S_{\neg v} \rrbracket^g & = (\hat{D}, \hat{X}) \\
\rho_{\neg v} & = \lambda x. \bigsqcup_{y \in \hat{D}(x)} \rho(y) \\
X_{if} & = X_v \cup \hat{X} \\
\rho_e & = (X_{if} \times \{\top\}) \cup ((\mathbb{I}d - X_{if}) \times \{\bot\})
\end{align*}
\]

\[
\sigma; \rho \vdash \text{if } e \text{ then } S_{\text{true}} \text{ else } S_{\text{false}} \text{ end} \Downarrow \sigma_v : \rho_v \uplus \rho_{\neg v} \uplus \rho_e : X_{if}
\]
Rules (2)

\[
\sigma; \rho \vdash e \Downarrow v : \top \quad \sigma; \rho \vdash S_v \Downarrow \sigma_v : \rho_v : X_v
\]

\[
\llbracket \sigma; \rho \vdash S_{\neg v} \rrbracket ^g = (\hat{D}, \hat{X}) \quad \rho_{\neg v} = \lambda x. \bigsqcup_{y \in \hat{D}(x)} \rho(y)
\]

\[
X_{if} = X_v \cup \hat{X} \quad \rho_e = (X_{if} \times \{\top\}) \cup ((\text{Id} - X_{if}) \times \{\bot\})
\]

\[
\sigma; \rho \vdash \text{if } e \text{ then } S_{\text{true}} \text{ else } S_{\text{false}} \text{ end} \Downarrow \sigma_v : \rho_v \parallel \rho_{\neg v} \parallel \rho_e : X_{if}
\]
\[ \sigma; \rho \vdash e \Downarrow v : \top \quad \sigma; \rho \vdash S_v \Downarrow \sigma_v : \rho_v : X_v \]

\[ \llbracket \sigma; \rho \vdash S_{\neg v} \rrbracket_g = (\hat{D}, \hat{X}) \]

\[ \rho_{\neg v} = \lambda x. \bigcup_{y \in \hat{D}(x)} \rho(y) \]

\[ X_{if} = X_v \cup \hat{X} \quad \rho_e = (X_{if} \times \{\top\}) \cup ((\mathbb{I}d - X_{if}) \times \{\bot\}) \]

\[ \sigma; \rho \vdash \text{if } e \text{ then } S_{\text{true}} \text{ else } S_{\text{false}} \text{ end} \Downarrow \sigma_v : \rho_v \parallel \rho_{\neg v} \parallel \rho_e : X_{if} \]
\sigma; \rho \vdash e \Downarrow v : \top \quad \sigma; \rho \vdash S_v \Downarrow \sigma_v : \rho_v : \mathcal{X}_v

\begin{align*}
\llbracket \sigma; \rho \vdash S_{\neg v} \rrbracket^g &= (\hat{D}, \hat{X}) \\
\rho_{\neg v} &= \lambda x. \bigcup_{y \in \hat{D}(x)} \rho(y) \\
\mathcal{X}_{if} &= \mathcal{X}_v \cup \hat{X} \\
\rho_e &= (\mathcal{X}_{if} \times \{\top\}) \cup ((\mathbb{I}d - \mathcal{X}_{if}) \times \{\bot\})
\end{align*}

\sigma; \rho \vdash \text{if } e \text{ then } S_{\text{true}} \text{ else } S_{\text{false}} \text{ end} \Downarrow \sigma_v : \rho_v \boxplus \rho_{\neg v} \boxplus \rho_e : \mathcal{X}_{if}
\[\begin{align*}
\sigma; \rho \vdash e \Downarrow v : \top & \quad \sigma; \rho \vdash S_v \Downarrow \sigma_v : \rho_v : X_v \\
\sev{\sigma; \rho \vdash S_{\neg v}}^g = (\widehat{D}, \widehat{X}) & \quad \rho_{\neg v} = \lambda x. \bigcup_{y \in \widehat{D}(x)} \rho(y) \\
X_{if} = X_v \cup \widehat{X} & \quad \rho_e = (X_{if} \times \{\top\}) \cup ((\Pi d - X_{if}) \times \{\bot\}) \\
\sigma; \rho \vdash \text{if } e \text{ then } S_{\text{true}} \text{ else } S_{\text{false}} \text{ end} \Downarrow \sigma_v : \rho_v \Pi \rho_{\neg v} \Pi \rho_e : X_{if}
\end{align*}\]
Hypothesis 1

“$[σ; ρ \vdash S]^{♯G}$ is not a too bad information flow analysis”

Theorem 9

For any command $C$, “total” value store $σ_1$ and $σ_2$, and “well-tagged” tag store $ρ$, such that:

1. $\llbracket C \rrbracket_{σ_2, ρ}^V \neq \bot$
2. $\text{Safe}(\llbracket C \rrbracket_{σ_1, ρ}^T)$

if $σ_1 =_{L_i} σ_2$ then $\llbracket C \rrbracket_{σ_1, ρ}^V =_{L_o} \llbracket C \rrbracket_{σ_2, ρ}^V$
Acceptability

$(\hat{D}, \hat{X})$ is an acceptable result if:

$$(\hat{D}, \hat{X}) \models (\sigma, \rho \vdash S)$$

- A syntactic analyzer
  - simple
  - quite efficient

$$\llbracket \sigma; \rho \vdash C \rrbracket^g = (\hat{D}, \hat{X})$$

- $\hat{X}$: set of all identifiers assigned to
- $\hat{D}$: $\forall x \in \hat{X}, \hat{D}(x) = Id$ and $\forall y \notin \hat{X}, \hat{D}(y) = \{y\}$
Example 10

\[
x := 0;
\text{if } l \text{ then}
\quad \text{if } h \text{ then } x := 1 \text{ else skip end}
\text{else skip end}
\]

<table>
<thead>
<tr>
<th></th>
<th>(\sigma(h))</th>
<th>(\sigma(l))</th>
<th>True</th>
<th>False</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>False</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>(\sigma(h))</th>
<th>(\sigma(l))</th>
<th>True</th>
<th>False</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>(\top)</td>
<td>(\bot)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>False</td>
<td>(\top)</td>
<td>(\bot)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[\text{TAB.: } [P]_\sigma,\rho(x)\]
### Limitations

**Fact 11 (Safe is not NIExec)**

\[
\forall s_1 \in S. \text{Safe}([C]s_1) \not\Rightarrow (\forall s_2 \in S. \ s_2 =_{L} s_1 \Rightarrow \text{Safe}([C]s_2))
\]

### Example 12

```
x := 0;
if h then
  if l then x := 1 else skip end
else skip end
```

<table>
<thead>
<tr>
<th>(\sigma(h))</th>
<th>(\sigma(l))</th>
<th>True</th>
<th>False</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>False</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(\sigma(h))</th>
<th>(\sigma(l))</th>
<th>True</th>
<th>False</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>(\top)</td>
<td>(\bot)</td>
<td></td>
</tr>
<tr>
<td>False</td>
<td>(\top)</td>
<td>(\top)</td>
<td>(\top)</td>
</tr>
</tbody>
</table>

**TAB.:** \([P]^V_{\sigma,\rho}(x)\)  
**TAB.:** \([P]^T_{\sigma,\rho}(x)\)
Fact 11 (Safe is not NIExec)

\[\forall s_1 \in S. \text{Safe}\left(\left[\begin{array}{c} C \end{array}\right] s_1\right) \not\Rightarrow \left(\forall s_2 \in S. s_2 =_L s_1 \Rightarrow \text{Safe}\left(\left[\begin{array}{c} C \end{array}\right] s_2\right)\right)\]

Example 12

\[x := 0;\]
\[\text{if } h \text{ then}\]
\[\quad \text{if } l \text{ then } x := 1 \text{ else skip end}\]
\[\text{else skip end}\]

\[
\begin{array}{c|c|c}
\sigma(h) & \sigma(l) & \text{True} \\
\hline
\text{True} & \text{True} & 1 \\
\text{False} & \text{True} & 0 \\
\text{True} & \text{False} & 0 \\
\text{False} & \text{False} & 0 \\
\end{array}
\]

\[
\begin{array}{c|c|c}
\sigma(h) & \sigma(l) & \text{True} \\
\hline
\text{True} & \text{T} & \top \\
\text{False} & \text{T} & \bot \\
\end{array}
\]

**Tab.:** \[P\]_{\sigma,\rho}(x)
A protocol for testing a set of executions starting in one class of low-equivalent inputs:

- while ( 
  - run one execution
  - Safe → exit YES
  - low outputs different from previous executions → exit NO
)
A protocol for testing a set of executions starting in one class of low-equivalent inputs:

- while (arbitrary limit not reached)
  - run one execution
  - Safe → exit YES
  - low outputs different from previous executions → exit NO
A protocol for testing a set of executions starting in one class of low-equivalent inputs:

- while (arbitrary limit not reached [ or all paths done])
  - run one execution
  - Safe → exit YES
  - low outputs different from previous executions → exit NO
A protocol for testing a set of executions starting in one class of low-equivalent inputs:

- while (arbitrary limit not reached [or all paths done])
  - run one execution
  - Safe $\rightarrow$ exit YES
  - low outputs different from previous executions $\rightarrow$ exit NO

Efficiency increased if:

- statements branching on high conditions buried deeper in the program
- executions picked up take different branches of statements branching on high conditions
Testing protocol

A protocol for testing a set of executions starting in one class of low-equivalent inputs:

- while (arbitrary limit not reached [ or all paths done])
  - run one execution
  - Safe → exit YES
  - low outputs different from previous executions → exit NO

Efficiency increased if:

- statements branching on high conditions buried deeper in the program
- executions picked up take different branches of statements branching on high conditions
A protocol for testing a set of executions starting in one class of low-equivalent inputs:

- while (arbitrary limit not reached [or all paths done])
  - run one execution
  - Safe → exit YES
  - low outputs different from previous executions → exit NO

Efficiency increased if:

- statements branching on high conditions buried deeper in the program
- executions picked up take different branches of statements branching on high conditions
Yes, but . . .

Partial evaluation & information flow analysis

- **statically**:  
  - infinitely many low-equivalent classes  
  - difficult to know which “residual programs” can be encountered

- **dynamically**:  
  - requires “smart” partial evaluation and IF analysis

Example 13

```
while (c_L > 0)  
f(l l l l l l)  
c_L = 3
```

- dynamic analysis: l l l l h l
- type system: h h h h h l
- information flow logic: l l l l h l (Clark & Hankin & Hunt)
Yes, but . . .

Partial evaluation & information flow analysis

- statically:
  - infinitely many low-equivalent classes
  - difficult to know which “residual programs” can be encountered

- dynamically:
  - requires “smart” partial evaluation and IF analysis

Example 13

```
while (cL > 0)  cL = 3
f(   [h l l l l l])
```

dynamic analysis: [l l l h l]

type system: [h h h h h l]

information flow logic: [l l l h l] ([Clark & Hankin & Hunt])

Gurvan Le Guernic  Monitoring Information Flow
Yes, but . . .

Partial evaluation & information flow analysis

- statically:
  - infinitely many low-equivalent classes
  - difficult to know which “residual programs” can be encountered

- dynamically:
  - requires “smart” partial evaluation and IF analysis

Example 13

```
while (c_L > 0)
  f(h l l l l l)
```

- dynamic analysis: l l l h l
- type system: h h h h h l
- information flow logic: l l l h l ([Clark & Hankin & Hunt])

Gurvan Le Guernic
Monitoring Information Flow
Partial evaluation & information flow analysis

- statically:
  - infinitely many low-equivalent classes
  - difficult to know which “residual programs” can be encountered

- dynamically:
  - requires “smart” partial evaluation and IF analysis

**Example 13**

```
while (c_L > 0)  
f([h l l l l])  
c_L = 2
```

- dynamic analysis: [l l l h l]
- type system: [h h h h h l]
- information flow logic: [l l l h l] ([Clark & Hankin & Hunt])
Partial evaluation & information flow analysis

- statically:
  - infinitely many low-equivalent classes
  - difficult to know which “residual programs” can be encountered

- dynamically:
  - requires “smart” partial evaluation and IF analysis

Example 13

```
while (c_L > 0)  c_L = 1
f(l l h l)
```

- dynamic analysis: l l h l
- type system: h h h h l
- information flow logic: l l h l

(Clark & Hankin & Hunt)
Yes, but . . .

Partial evaluation & information flow analysis

- **statically:**
  - infinitely many low-equivalent classes
  - difficult to know which “residual programs” can be encountered

- **dynamically:**
  - requires “smart” partial evaluation and IF analysis

**Example 13**

\[
\text{while } (c_L > 0) \quad \quad c_L = 0
\]

- dynamic analysis: \[
\begin{array}{c}
 h \quad l \quad l \quad l \quad l \\
 l \quad l \\
 f(l \ l \ l \ l \ h \ l)
\end{array}
\]

- type system: \[
\begin{array}{c}
 h \quad h \quad h \quad h \quad h \quad l
\end{array}
\]

- information flow logic: \[
\begin{array}{c}
 l \quad l \quad l \quad h \quad l
\end{array}
\]

([Clark & Hankin & Hunt])

Gurvan Le Guernic

Monitoring Information Flow
Partial evaluation & information flow analysis

- **statically:**
  - infinitely many low-equivalent classes
  - difficult to know which “residual programs” can be encountered

- **dynamically:**
  - requires “smart” partial evaluation and IF analysis

### Example 13

```
while (cL > 0)
  f(l h l l l)
```

- `cL = 3`
- `f(l h l l l)`
- `f(l l h l l)`
- `f(l l l h l)`

- **dynamic analysis:** `l l l l h l`
- **type system:** `h h h h l`
- **information flow logic:** `l l l h l` ([Clark & Hankin & Hunt])

---

**Gurvan Le Guernic**

**Monitoring Information Flow**
Partial evaluation & information flow analysis

- **statically**:
  - infinitely many low-equivalent classes
  - difficult to know which “residual programs” can be encountered

- **dynamically**:
  - requires “smart” partial evaluation and IF analysis

---

Example 13

<table>
<thead>
<tr>
<th>h</th>
<th>l</th>
<th>l</th>
<th>l</th>
<th>l</th>
<th>l</th>
</tr>
</thead>
<tbody>
<tr>
<td>while ( (c_L &gt; 0) )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f(</td>
<td>←</td>
<td>←</td>
<td>←</td>
<td>←</td>
<td>←</td>
</tr>
</tbody>
</table>

\[ c_L = 3 \]

\[ f( l | h | l | l | l | l ) \]

\[ f( l | l | h | l | l | l ) \]

\[ f( l | l | l | h | l | l ) \]

- **dynamic analysis**: l l l l h l
- **type system**: h h h h h l
- **information flow logic**: l l l l h l ([Clark & Hankin & Hunt])
Introduction

Tracking

Testing

But

Conclusion

Yes, but . . .

Partial evaluation & information flow analysis

- statically:
  - infinitely many low-equivalent classes
  - difficult to know which “residual programs” can be encountered

- dynamically:
  - requires “smart” partial evaluation and IF analysis

Example 13

while (c_L > 0)

\[
\begin{align*}
  &f(h, l, l, l, l) \\
  &c_L = 3
\end{align*}
\]

- dynamic analysis: l l l h l
- type system: h h h h h l
- information flow logic: l l l h l ([Clark & Hankin & Hunt])

Monitoring Information Flow
Conclusion

- A non-interference definition with a reduced scope:
  - non-interfering execution
- A “smart” semantics
- A predicate for detecting non-interfering executions

⇒ Possible to detect the “safe” behavior of a set of executions from only one of those executions
A non-interference definition with a reduced scope:
- non-interfering execution
- A “smart” semantics
- A predicate for detecting non-interfering executions

⇒ Possible to detect the “safe” behavior of a set of executions from only one of those executions