Monitoring Information Flow

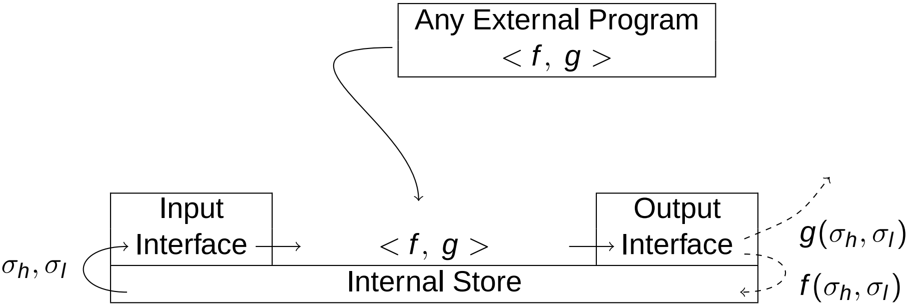
Gurvan Le Guernic

IRISA - Lande

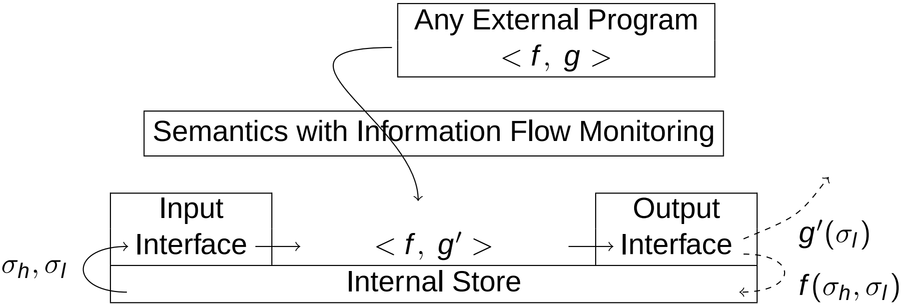
June 16, 2005 / ACI POTESTAT

Outline

Goal


Any External Program
< f, g >
Input
Interface
< f, g >
Output
Interface
Internal Store
σh, σl
g(σh, σl)
f(σh, σl)

Goal

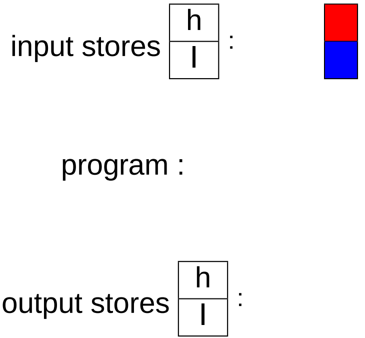


- $\forall o \in \text{PublicOutput}:$
  $$g'(\sigma_l)(o) = g(\sigma_h, \sigma_l)(o) \quad \vee \quad g'(\sigma_l)(o) = \bot$$

NON-INTERFERENCE

Presentation of the concept of non-interference

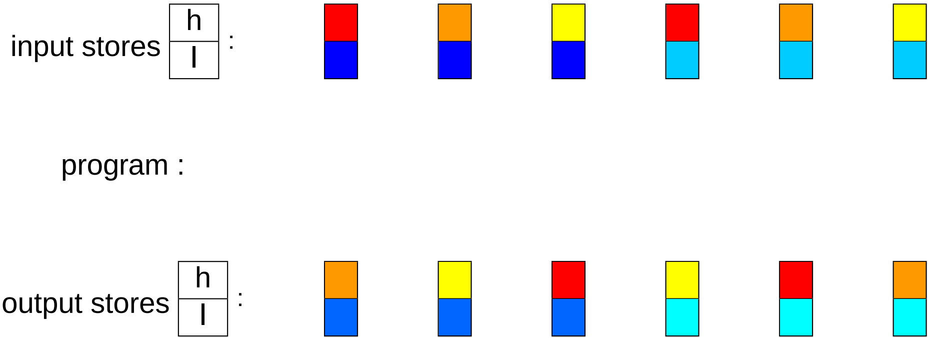
- Introduced by Goguen and Meseguer
- Property of a program respecting secrets confidentiality



NON-INTERFERENCE

Presentation of the concept of non-interference

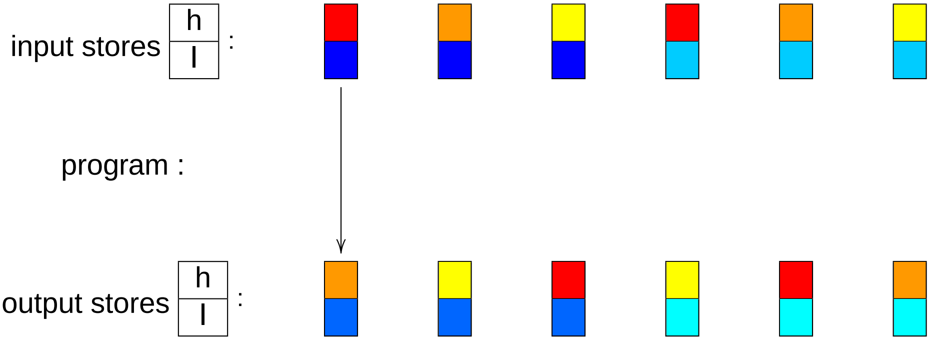
- Introduced by Goguen and Meseguer
- Property of a program respecting secrets confidentiality



NON-INTERFERENCE

Presentation of the concept of non-interference

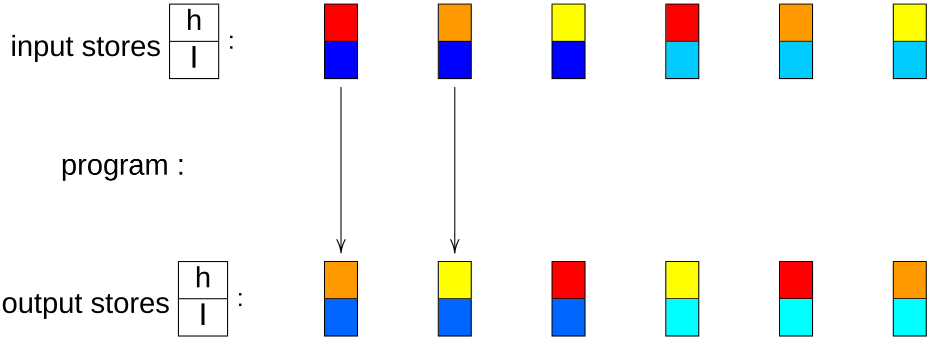
- Introduced by Goguen and Meseguer
- Property of a program respecting secrets confidentiality



NON-INTERFERENCE

Presentation of the concept of non-interference

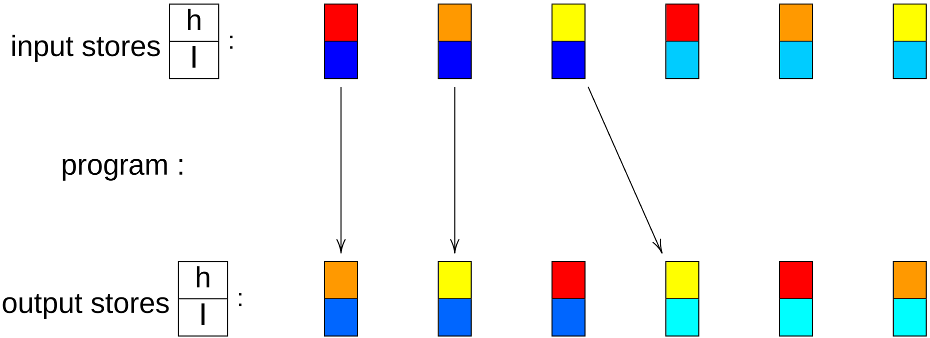
- Introduced by Goguen and Meseguer
- Property of a program respecting secrets confidentiality



NON-INTERFERENCE

Presentation of the concept of non-interference

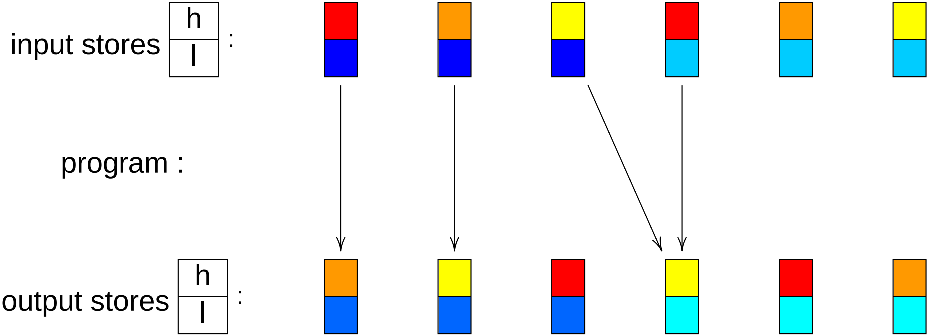
- Introduced by Goguen and Meseguer
- Property of a program respecting secrets confidentiality



NON-INTERFERENCE

Presentation of the concept of non-interference

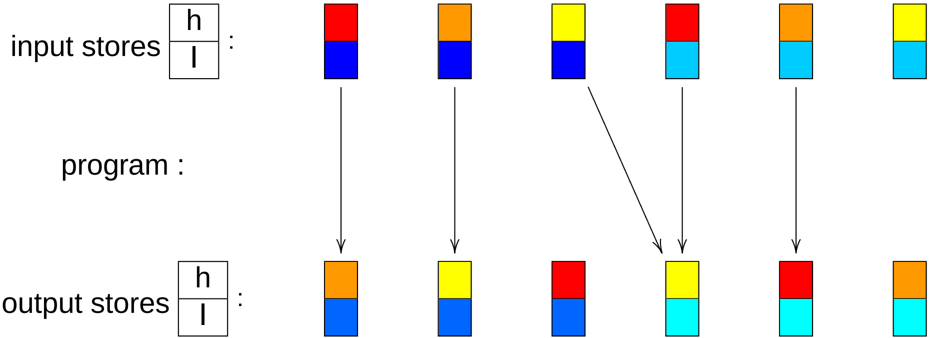
- Introduced by Goguen and Meseguer
- Property of a program respecting secrets confidentiality



NON-INTERFERENCE

Presentation of the concept of non-interference

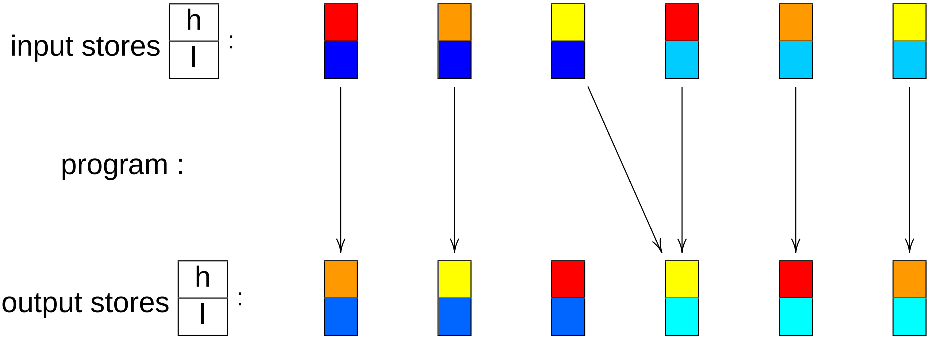
- Introduced by Goguen and Meseguer
- Property of a program respecting secrets confidentiality



NON-INTERFERENCE

Presentation of the concept of non-interference

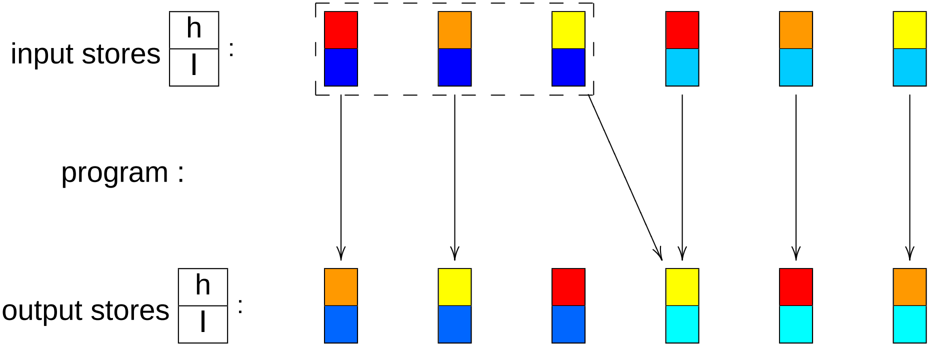
- Introduced by Goguen and Meseguer
- Property of a program respecting secrets confidentiality



NON-INTERFERENCE

Presentation of the concept of non-interference

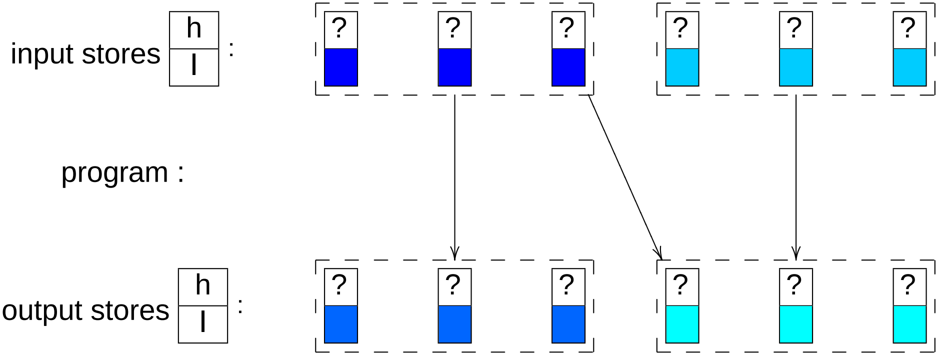
- Introduced by Goguen and Meseguer
- Property of a program respecting secrets confidentiality



NON-INTERFERENCE

Presentation of the concept of non-interference

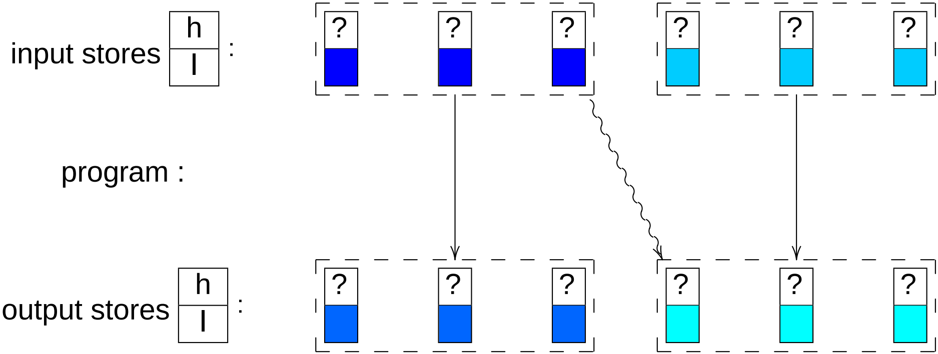
- Introduced by Goguen and Meseguer
- Property of a program respecting secrets confidentiality



NON-INTERFERENCE

Presentation of the concept of non-interference

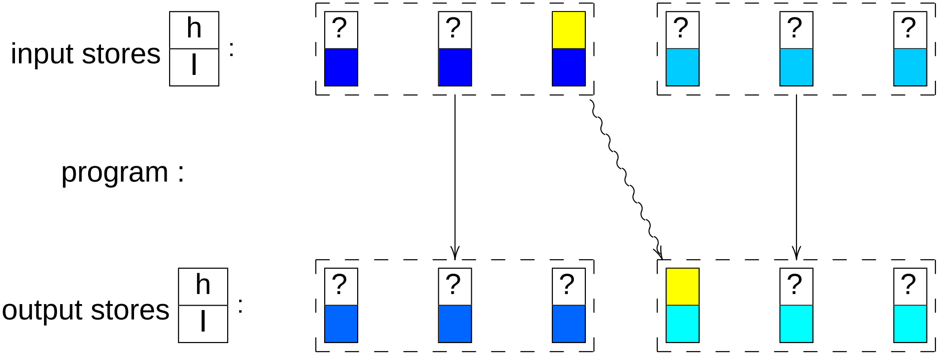
- Introduced by Goguen and Meseguer
- Property of a program respecting secrets confidentiality



NON-INTERFERENCE

Presentation of the concept of non-interference

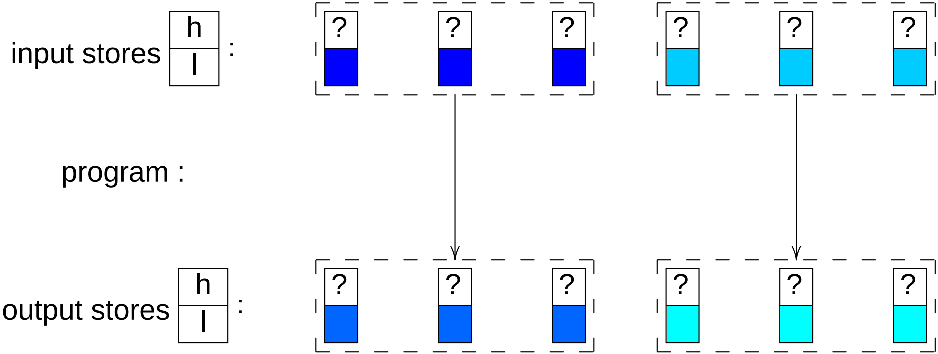
- Introduced by Goguen and Meseguer
- Property of a program respecting secrets confidentiality



NON-INTERFERENCE

Presentation of the concept of non-interference

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



NON-INTERFERENCE

Formalization of non-interference

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

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)

$$\forall s_1.\ \mathrm{NIExec}(C, s_1)\ \equiv\ \forall s_2.\ s_1 =_L s_2 \Rightarrow [\![C]\!]s_1 \approx_L [\![C]\!]s_2$$

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)

$$\forall s_1.\ \mathrm{NIExec}(C, s_1) \ \equiv\ \forall s_2.\ s_1 =_L s_2 \Rightarrow [\![C]\!]s_1 \approx_L [\![C]\!]s_2$$

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)

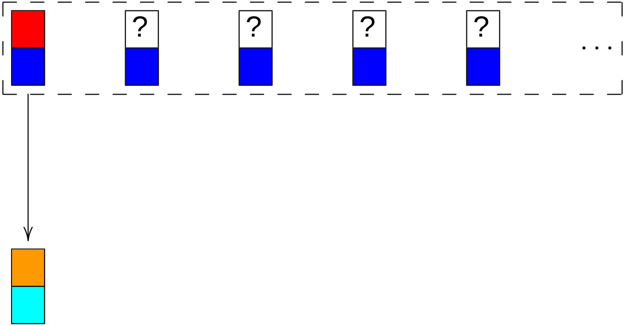
$$\forall s_1.\ \mathrm{NIExec}(C, s_1) \equiv \forall s_2.\ s_1 =_L s_2 \Rightarrow [\![C]\!]s_1 \approx_L [\![C]\!]s_2$$

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)

$$\forall s_1.\ \mathrm{NIExec}(C, s_1) \equiv \forall s_2.\ s_1 =_L s_2 \Rightarrow [\![C]\!]s_1 \approx_L [\![C]\!]s_2$$

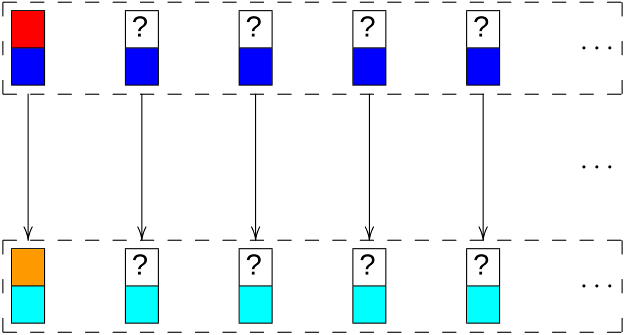


Non-interfering execution

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

Definition 3 (Non-interfering execution)

$$\forall s_1.\ \mathrm{NIExec}(C, s_1) \equiv \forall s_2.\ s_1 =_L s_2 \Rightarrow [\![C]\!]s_1 \approx_L [\![C]\!]s_2$$



Some properties

Fact 4 (Predicate Safe)

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

Corollary 5 (Definition of low-equivalence is symmetric)

$$\forall s_1.\ \mathrm{NIExec}(C, s_1) \Rightarrow (\forall s_2.\ s_2 =_L s_1 \Rightarrow \mathrm{NIExec}(C, s_2))$$

Corollary 6

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

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

Some properties

Fact 4 (Predicate Safe)

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

Corollary 5 (Definition of low-equivalence is symmetric)

$$\forall s_1.\ \mathrm{NIExec}(C, s_1) \Rightarrow (\forall s_2.\ s_2 =_L s_1 \Rightarrow \mathrm{NIExec}(C, s_2))$$

Corollary 6

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

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

Some properties

Fact 4 (Predicate Safe)

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

Corollary 5 (Definition of low-equivalence is symmetric)

$$\forall s_1.\ \mathrm{NIExec}(C, s_1) \Rightarrow (\forall s_2.\ s_2 =_L s_1 \Rightarrow \mathrm{NIExec}(C, s_2))$$

Corollary 6

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

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

Language's Grammar

$$
\begin{array}{rcl}
v & ::= & \mathrm{c} \\
e & ::= & e_1 \ \mathrm{op} \ e_2 \quad | \quad id \quad | \quad v \\
S & ::= & \mathrm{if}\ e\ \mathrm{then}\ S\ \mathrm{else}\ S\ \mathrm{end} \\
 & | & \mathrm{while}\ e\ \mathrm{do}\ S\ \mathrm{done} \\
 & | & id \ := \ e \\
 & | & \mathrm{skip} \\
 & | & S\ ;\ S
\end{array}
$$

id stands for any variable identifier (name)

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 ? 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 ? 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 ? 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 ? 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 ? 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 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

$$(\mathbb{I}d \rightarrow \mathbb{V}alue);(\mathbb{I}d \rightarrow \mathbb{T}ag) \vdash \mathbb{E}xpr \Downarrow \mathbb{V}alue : \mathbb{T}ag$$

$$(\mathbb{I}d \rightarrow \mathbb{V}alue);(\mathbb{I}d \rightarrow \mathbb{T}ag) \vdash \mathbb{S} \Downarrow (\mathbb{I}d \rightarrow \mathbb{V}alue) : (\mathbb{I}d \rightarrow \mathbb{T}ag) : \mathcal{P}(\mathbb{I}d)$$

Example 8

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

Semantics judgments

$$(\mathbb{I}d \rightarrow \mathbb{V}alue); (\mathbb{I}d \rightarrow \mathbb{T}ag) \vdash \mathbb{E}xpr \Downarrow \mathbb{V}alue : \mathbb{T}ag$$

$$(\mathbb{I}d \rightarrow \mathbb{V}alue); (\mathbb{I}d \rightarrow \mathbb{T}ag) \vdash \mathbb{S} \Downarrow (\mathbb{I}d \rightarrow \mathbb{V}alue) : (\mathbb{I}d \rightarrow \mathbb{T}ag) : \mathcal{P}(\mathbb{I}d)$$

Example 8

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

The analysis

$$[\![(\mathbb{I}d \rightarrow \mathbb{V}alue);(\mathbb{I}d \rightarrow \mathbb{T}ag) \vdash \mathbb{S}]\!]^{\sharp\mathcal{G}} = (\widehat{\mathrm{D}}, \widehat{\mathrm{X}})$$

- $\widehat{\mathrm{D}} = \mathcal{P}(\mathbb{I}d \times \mathbb{I}d)$
  - over-approximation of the dependencies between initial and final values of variables
- $\widehat{\mathrm{X}} = \mathcal{P}(\mathbb{I}d)$
  - over-approximation of the set of variables which *may* be assigned to

Rules (1)

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

$$\frac{\sigma;\rho \vdash e \Downarrow v : \bot \qquad \sigma;\rho \vdash S_v \Downarrow \sigma_v : \rho_v : X \qquad \rho_e = (X_{if} \times \{\top\}) \cup ((\mathbb{I}d - X_{if}) \times \{\bot\})}{\sigma;\rho \vdash \text{if } e \text{ then } S_{true} \text{ else } S_{false} \text{ end} \Downarrow \sigma_v : \rho_v \amalg \rho_e : X}$$

Rules (1)

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

$$\frac{\sigma;\rho \vdash e \Downarrow v : \bot \qquad \sigma;\rho \vdash S_v \Downarrow \sigma_v : \rho_v : X \qquad \rho_e = (X_{if} \times \{\top\}) \cup ((\mathbb{I}d - X_{if}) \times \{\bot\})}{\sigma;\rho \vdash \text{if } e \text{ then } S_{true} \text{ else } S_{false} \text{ end} \Downarrow \sigma_v : \rho_v \amalg \rho_e : X}$$

Rules (2)

$$\frac{\sigma;\rho \vdash e \Downarrow v : \top \qquad \sigma;\rho \vdash S_v \Downarrow \sigma_v : \rho_v : X_v \\ [\![\sigma;\rho \vdash S_{\neg v}]\!]^{\sharp\mathcal{G}} = (\widehat{\mathrm{D}},\widehat{\mathrm{X}}) \qquad \rho_{\neg v} = \lambda x. \bigsqcup_{y \in \widehat{\mathrm{D}}(x)} \rho(y) \\ X_{if} = X_v \cup \widehat{\mathrm{X}} \qquad \rho_e = (X_{if} \times \{\top\}) \cup ((\mathbb{I}d - X_{if}) \times \{\bot\})}{\sigma;\rho \vdash \text{if } e \text{ then } S_{true} \text{ else } S_{false} \text{ end} \Downarrow \sigma_v : \rho_v \amalg \rho_{\neg v} \amalg \rho_e : X_{if}}$$

Rules (2)

$$\frac{\sigma;\rho \vdash e \Downarrow v : \top \qquad \sigma;\rho \vdash S_v \Downarrow \sigma_v : \rho_v : X_v \qquad [\![\sigma;\rho \vdash S_{\neg v}]\!]^{\sharp\mathcal{G}} = (\widehat{\mathrm{D}},\widehat{\mathrm{X}}) \qquad \rho_{\neg v} = \lambda x.\bigsqcup_{y \in \widehat{\mathrm{D}}(x)} \rho(y) \qquad X_{if} = X_v \cup \widehat{\mathrm{X}} \qquad \rho_e = (X_{if} \times \{\top\}) \cup ((\mathbb{I}d - X_{if}) \times \{\bot\})}{\sigma;\rho \vdash \text{if } e \text{ then } S_{true} \text{ else } S_{false} \text{ end} \Downarrow \sigma_v : \rho_v \amalg \rho_{\neg v} \amalg \rho_e : X_{if}}$$

Rules (2)

$$\frac{\sigma;\rho \vdash e \Downarrow v : \top \qquad \sigma;\rho \vdash S_v \Downarrow \sigma_v : \rho_v : X_v \qquad [\![\sigma;\rho \vdash S_{\neg v}]\!]^{\sharp\mathcal{G}} = (\widehat{D},\widehat{X}) \qquad \rho_{\neg v} = \lambda x.\bigsqcup_{y\in\widehat{D}(x)} \rho(y) \qquad X_{if} = X_v \cup \widehat{X} \qquad \rho_e = (X_{if} \times \{\top\}) \cup ((\mathbb{I}d - X_{if}) \times \{\bot\})}{\sigma;\rho \vdash \text{if } e \text{ then } S_{true} \text{ else } S_{false} \text{ end} \Downarrow \sigma_v : \rho_v \amalg \rho_{\neg v} \amalg \rho_e : X_{if}}$$

Rules (2)

$$\frac{\begin{array}{c}\sigma;\rho \vdash e \Downarrow v : \top \qquad \sigma;\rho \vdash S_v \Downarrow \sigma_v : \rho_v : X_v \\ [\![\sigma;\rho \vdash S_{\neg v}]\!]^{\sharp\mathcal{G}} = (\widehat{\mathrm{D}}, \widehat{\mathrm{X}}) \qquad \rho_{\neg v} = \lambda x. \bigsqcup_{y \in \widehat{\mathrm{D}}(x)} \rho(y) \\ X_{if} = X_v \cup \widehat{\mathrm{X}} \qquad \rho_e = (X_{if} \times \{\top\}) \cup ((\mathbb{I}d - X_{if}) \times \{\bot\})\end{array}}{\sigma;\rho \vdash \text{if } e \text{ then } S_{true} \text{ else } S_{false} \text{ end} \Downarrow \sigma_v : \rho_v \amalg \rho_{\neg v} \amalg \rho_e : X_{if}}$$

Rules (2)

$$\frac{\begin{array}{cc}\sigma;\rho \vdash e \Downarrow v : \top & \sigma;\rho \vdash S_v \Downarrow \sigma_v : \rho_v : X_v \\ [\![\sigma;\rho \vdash S_{\neg v}]\!]^{\sharp\mathcal{G}} = (\widehat{\mathrm{D}},\widehat{\mathrm{X}}) & \rho_{\neg v} = \lambda x. \bigsqcup_{y \in \widehat{\mathrm{D}}(x)} \rho(y) \\ X_{if} = X_v \cup \widehat{\mathrm{X}} & \rho_e = (X_{if} \times \{\top\}) \cup ((\mathbb{I}d - X_{if}) \times \{\bot\})\end{array}}{\sigma;\rho \vdash \text{if } e \text{ then } S_{true} \text{ else } S_{false} \text{ end} \Downarrow \sigma_v : \rho_v \amalg \rho_{\neg v} \amalg \rho_e : X_{if}}$$

Rules (2)

$$\frac{\begin{array}{c}\sigma;\rho \vdash e \Downarrow v : \top \qquad \sigma;\rho \vdash S_v \Downarrow \sigma_v : \rho_v : X_v \\ [\![\sigma;\rho \vdash S_{\neg v}]\!]^{\sharp\mathcal{G}} = (\widehat{\mathrm{D}}, \widehat{\mathrm{X}}) \qquad \rho_{\neg v} = \lambda x. \bigsqcup_{y \in \widehat{\mathrm{D}}(x)} \rho(y) \\ X_{if} = X_v \cup \widehat{\mathrm{X}} \qquad \rho_e = (X_{if} \times \{\top\}) \cup ((\mathbb{I}d - X_{if}) \times \{\bot\})\end{array}}{\sigma;\rho \vdash \text{if } e \text{ then } S_{true} \text{ else } S_{false} \text{ end} \Downarrow \sigma_v : \rho_v \amalg \rho_{\neg v} \amalg \rho_e : X_{if}}$$

Rules (2)

$$\frac{\begin{array}{c}\sigma;\rho \vdash e \Downarrow v : \top \qquad \sigma;\rho \vdash S_v \Downarrow \sigma_v : \rho_v : X_v \\ [\![\sigma;\rho \vdash S_{\neg v}]\!]^{\sharp\mathcal{G}} = (\widehat{\mathrm{D}}, \widehat{\mathrm{X}}) \qquad \rho_{\neg v} = \lambda x.\bigsqcup_{y \in \widehat{\mathrm{D}}(x)} \rho(y) \\ X_{if} = X_v \cup \widehat{\mathrm{X}} \qquad \rho_e = (X_{if} \times \{\top\}) \cup ((\mathbb{I}d - X_{if}) \times \{\bot\})\end{array}}{\sigma;\rho \vdash \text{if } e \text{ then } S_{true} \text{ else } S_{false} \text{ end} \Downarrow \sigma_v : \rho_v \amalg \rho_{\neg v} \amalg \rho_e : X_{if}}$$

Rules (2)

$$\frac{\sigma;\rho \vdash e \Downarrow v : \top \qquad \sigma;\rho \vdash S_v \Downarrow \sigma_v : \rho_v : X_v \\ [\![\sigma;\rho \vdash S_{\neg v}]\!]^{\sharp\mathcal{G}} = (\widehat{\mathrm{D}}, \widehat{\mathrm{X}}) \qquad \rho_{\neg v} = \lambda x. \bigsqcup_{y \in \widehat{\mathrm{D}}(x)} \rho(y) \\ X_{if} = X_v \cup \widehat{\mathrm{X}} \qquad \rho_e = (X_{if} \times \{\top\}) \cup ((\mathbb{I}d - X_{if}) \times \{\bot\})}{\sigma;\rho \vdash \text{if } e \text{ then } S_{true} \text{ else } S_{false} \text{ end} \Downarrow \sigma_v : \rho_v \amalg \rho_{\neg v} \amalg \rho_e : X_{if}}$$

Rules (2)

$$\frac{\begin{array}{c}\sigma;\rho \vdash e \Downarrow v : \top \qquad \sigma;\rho \vdash S_v \Downarrow \sigma_v : \rho_v : X_v \\ [\![\sigma;\rho \vdash S_{\neg v}]\!]^{\sharp\mathcal{G}} = (\widehat{\mathrm{D}},\widehat{\mathrm{X}}) \qquad \rho_{\neg v} = \lambda x. \bigsqcup_{y \in \widehat{\mathrm{D}}(x)} \rho(y) \\ X_{if} = X_v \cup \widehat{\mathrm{X}} \qquad \rho_e = (X_{if} \times \{\top\}) \cup ((\mathbb{I}d - X_{if}) \times \{\bot\})\end{array}}{\sigma;\rho \vdash \text{if } e \text{ then } S_{true} \text{ else } S_{false} \text{ end} \Downarrow \sigma_v : \rho_v \amalg \rho_{\neg v} \amalg \rho_e : X_{if}}$$

Properties of the semantics

Hypothesis 1

"$[\![\sigma ; \rho \vdash S]\!]^{\sharp \mathcal{G}}$ 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. $[\![C]\!]^{\mathbb{V}}_{\sigma_2,\rho} \neq \bot$
2. $\mathrm{Safe}([\![C]\!]^{\mathbb{T}}_{\sigma_1,\rho})$

if $\sigma_1 =_{L_i} \sigma_2$ *then* $[\![C]\!]^{\mathbb{V}}_{\sigma_1,\rho} =_{L_o} [\![C]\!]^{\mathbb{V}}_{\sigma_2,\rho}$

Acceptability

$(\widehat{\mathrm{D}}, \widehat{\mathrm{X}})$ is an *acceptable* result if :

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

- A syntactic analyzer
  - simple
  - quite efficient
- $[\![\sigma; \rho \vdash C]\!]^{\sharp\mathcal{G}} = (\widehat{\mathrm{D}}, \widehat{\mathrm{X}})$
  - $\widehat{\mathrm{X}}$: set of all identifiers assigned to
  - $\widehat{\mathrm{D}}$: $\forall x \in \widehat{\mathrm{X}},\ \widehat{\mathrm{D}}(x) = \mathbb{I}d$ and $\forall y \notin \widehat{\mathrm{X}},\ \widehat{\mathrm{D}}(y) = \{y\}$

Example

Example 10

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

σ(h) \ σ(l)	True	False
True	1	0
False	0	0

TAB.: $[\![P]\!]^{\mathbb{V}}_{\sigma,\rho}$(x)

σ(h) \ σ(l)	True	False
True	$\top$	$\bot$
False	$\top$	$\bot$

TAB.: $[\![P]\!]^{\mathbb{T}}_{\sigma,\rho}$(x)

Limitations

Fact 11 (Safe is not NIExec)

$$\forall s_1 \in S.\ \mathrm{Safe}([\![C]\!]s_1) \not\Rightarrow (\forall s_2 \in S.\ s_2 =_L s_1 \Rightarrow \mathrm{Safe}([\![C]\!]s_2))$$

Example 12

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

σ(h) \ σ(l)	True	False
True	1	0
False	0	0

TAB.: $[\![P]\!]^{\mathbb{V}}_{\sigma,\rho}$(x)

σ(h) \ σ(l)	True	False
True	$\top$	$\bot$
False	$\top$	$\top$

TAB.: $[\![P]\!]^{\mathbb{T}}_{\sigma,\rho}$(x)

Limitations

Fact 11 (Safe is not NIExec)

$$\forall s_1 \in S.\ \mathrm{Safe}([\![C]\!]s_1) \not\Rightarrow (\forall s_2 \in S.\ s_2 =_L s_1 \Rightarrow \mathrm{Safe}([\![C]\!]s_2))$$

Example 12

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

σ(h) \ σ(l)	True	False
True	1	0
False	0	0

TAB.: $[\![P]\!]^{\mathbb{V}}_{\sigma,\rho}$(x)

σ(h) \ σ(l)	True	False
True	$\top$	$\bot$
False	$\top$	$\top$

TAB.: $[\![P]\!]^{\mathbb{T}}_{\sigma,\rho}$(x)

Testing protocol

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

- while ( )
  - run one execution
  - Safe $\longrightarrow$ exit YES
  - low outputs different from previous executions $\longrightarrow$ exit NO

Testing protocol

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 $\longrightarrow$ exit YES
  - low outputs different from previous executions $\longrightarrow$ exit NO

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 $\longrightarrow$ exit YES
  - low outputs different from previous executions $\longrightarrow$ exit NO

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
  - $\mathrm{Safe} \longrightarrow$ exit YES
  - low outputs different from previous executions $\longrightarrow$ 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 $\longrightarrow$ exit YES
  - low outputs different from previous executions $\longrightarrow$ 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
  - $\mathrm{Safe} \longrightarrow$ exit YES
  - low outputs different from previous executions $\longrightarrow$ 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

h	l	l	l	l

while ($c_L > 0$) $c_L = 3$

f( )

- dynamic analysis :

l	l	l	h	l

- type system :

h	h	h	h	l

- information flow logic :

l	l	l	h	l

([Clark & Hankin & Hunt])

Yes, but . . .

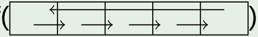
Partial evaluation & information flow analysis

- 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

h	l	l	l	l

while ($c_L > 0$)

f()

$c_L = 3$

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

h	l	l	l	l

while ($c_L > 0$)

f(| h | l | l | l | l |)

$c_L = 3$

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

h	l	l	l	l

while ($c_L > 0$)

f(| l | h | l | l | l |)

$c_L = 2$

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

h	l	l	l	l

while ($c_L > 0$)

f(| l | l | h | l | l |)

$c_L = 1$

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

h	l	l	l	l

while ($c_L > 0$)

f(| l | l | l | h | l |)

$c_L = 0$

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

Yes, but . . .

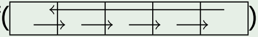
Partial evaluation & information flow analysis

- 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

h	l	l	l	l

while ($c_L > 0$)

f()

$c_L = 3$

f(l | h | l | l | l)

f(l | l | h | l | l)

f(l | l | l | h | l)

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

Yes, but . . .

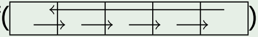
Partial evaluation & information flow analysis

- 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

h	l	l	l	l

while ($c_L > 0$)

f()

$c_L = 3$

f(| l | h | l | l | l |)

f(| l | l | h | l | l |)

f(| l | l | l | h | l |)

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

Yes, but . . .

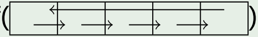
Partial evaluation & information flow analysis

- 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

| h | l | l | l | l |

while ($c_L > 0$)

f()

$c_L = 3$

f(| l | h | l | l | l |)

f(| l | l | h | l | l |)

f(| l | l | l | h | l |)

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

Conclusion

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

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

Conclusion

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

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

Monitoring Information Flow

Gurvan Le Guernic

IRISA - Lande

June 16, 2005 / ACI POTESTAT