ABSTRACT NON-INTERFERENCE

Parameterizing Non-Interference by Abstract Interpretation

Roberto Giacobazzi and Isabella Mastroeni

Dipartimento di Informatica
Università di Verona
Italy

POPL'04 - Venice, January 15th, 2004
The problem: Protect data confidentiality from erroneous/malicious attacks while data are processed

- Attack = disclosing properties of confidential data
- Access control by declaring data privileges!
The problem: Protect data confidentiality from erroneous/malicious attacks while data are processed

⇒ Access control methods do not put constraints on how the information is propagated!
The Problem

The problem: Protect data confidentiality from erroneous/malicious attacks while data are processed

Description of the problem:

- Typing of data (and variables) in *private* ($H$) and *public* ($L$);
- **Non-Interference**: to prevent the results of the computation from leaking even partial information about private inputs!
  - **Explicit flow**: caused by directly passing private data to a public variable: $l := 2 \times h$;
  - **Implicit flow**: arise from control structure of the program:
    ```
    while $h$ do $l := l + 1; h := h - 1.$
    ```
- We consider only *terminating* computations!
The Goal

IT IS ESSENTIAL TO KNOW HOW MUCH AN ATTACKER MAY LEARN FROM A PROGRAM!

Goal: Automatically generate certificates about secure information flows
- Design of accurate security polices
- Static program analysis & verification techniques (types, CFA, DFA, ...)

Abstract Non-interference - Parameterizing Non-Interference by Abstract Interpretation – p.3/15
IT IS ESSENTIAL TO KNOW HOW MUCH AN ATTACKER MAY LEARN FROM A PROGRAM!

Goal: Automatically generate certificates about secure information flows

State of the art: Standard non-interference is far too restrictive

- No sensitive information can be disclosed
- Any change upon confidential data has not to be revealed by public ones
- Rigid security policy: L can flow into H but H cannot flow into L
  [Denning and Denning ’77]
**The Goal**

**IT IS ESSENTIAL TO KNOW HOW MUCH AN ATTACKER MAY LEARN FROM A PROGRAM!**

- **Goal:** Automatically generate certificates about secure information flows
- **State of the art:** Standard non-interference is far too restrictive

**Question:** Is there a way to characterize what kind of information flows?
- Characterize the secrecy degree of a program
- $H$ can flow into $L$ unless a given property of $H$ is disclosed
- Weakening standard non-interference (a challenge in language-based security [Sabelfeld & Myers ’03])
while h do (l := l + 2; h := h - 1)
IDEA: Attackers as Abstract Interpretations

\[ \text{while } h \text{ do } \begin{cases} l := l + 2; & h := h - 1 \end{cases} \]

6. There is an (implicit/absolute) flow from \( h \) into \( l \)
6. The parity of \( l \) is not affected by any change of \( h \)
6. ... no information flow for \( parity \)!
IDEA: Attackers as Abstract Interpretations

while \( h \) do (\( l := l + 2; \ h := h - 1 \))

The idea: Abstract non-interference

- Attackers as program analyzers
  - Attackers can analyze I/O behaviour of public data
  - Attackers perform “static” program analyses

\[\rightarrow\] Abstract interpretation is a general method for specifying approximate semantics of programs [Cousot & Cousot ’77]

Attacker are abstract interpretations of program semantics
IDEA: Attackers as Abstract Interpretations

The idea: Abstract non-interference

Main results:
- Generalizing non-interference relatively to the attacker’s power
- Making non-interference parametric on the attacker’s point of view
- Checking abstract non-interference by abstract interpretation
- Systematic method for deriving attackers for programs by modifying abstractions
- Abstract Robust Declassification
Related works

Refining security policies by constraining attackers
Characterizing released information
Related works

Refining security policies by constraining attackers

Complexity:
- Security levels corresponding to how complex is attacking the program
  [Lowe ’02]
Refining security policies by constraining attackers

Complexity:
- Security levels corresponding to how complex is attacking the program
  [Lowe ’02]

Quantitative measure:
- An absolute (approximate) quantitative evaluation of information leakage (number of statistical tests to disclose properties)
  [Di Pierro et al. ’02]
Related works

Characterizing released information

Quantitative measure:
- Quantification of the information flowed by information theory
  [D. Clark et al. ’03]
Related works

Characterizing released information

Quantitative measure:
- Quantification of the information flowed by information theory
  [D. Clark et al. ’03]

Robust declassification:
- The observational capability of the attacker is characterized by equivalence relations, then the information released is identified and declassified.
  [Zdancewic and Myers ’01]
The concrete domain $< C, \leq, \wedge, \vee, \bot, T >$

Lattice of abstract domains $\equiv \text{Abs}(C)$

$< \text{Abs}(C), \subseteq, \cap, \cup, T, C >$

$A_1 \subseteq A_2 \iff A_2 \subseteq A_1$ (A_1 more precise than A_2)
“One group of users [...] is noninterfering with another group of users if what the first group does [...] has no effect on what the second group of users can see” [Goguen & Meseguer '82]

Standard non-interference

\[ \forall l : L, \forall h_1, h_2 : H. \llbracket P \rrbracket (h_1, l)^L = \llbracket P \rrbracket (h_2, l)^L \]
Standard non-interference

\[ \forall l : L, \forall h_1, h_2 : H. \llbracket P \rrbracket (h_1, l)^L = \llbracket P \rrbracket (h_2, l)^L \]

**Example:**

\[
\textbf{while } h \textbf{ do } (l := l + 2; \ h := h - 1).\]
Standard non-interference

∀l : L, ∀h₁, h₂ : H. \[ [P](h₁, l)^L = [P](h₂, l)^L \]

**Example:**

**while** h **do** (l := l + 2; h := h - 1).

- \( h = 0, \ l = 1 \) \(\rightarrow\) \( l = 1 \)
- \( h = 1, \ l = 1 \) \(\rightarrow\) \( l = 3 \)
- \( h = n, \ l = 1 \) \(\rightarrow\) \( l = 1 + 2n \)
Standard non-interference

\[ \forall l : L, \forall h_1, h_2 : H. \llbracket P \rrbracket (h_1, l)^L = \llbracket P \rrbracket (h_2, l)^L \]

**Example:**

```
while h do (l := l + 2; h := h - 1).
```

- \( h = 0, \ l = 1 \) \( \leadsto \) \( l = 1 \)
- \( h = 1, \ l = 1 \) \( \leadsto \) \( l = 3 \)
- \( h = n, \ l = 1 \) \( \leadsto \) \( l = 1 + 2n \)

If \( l \) is unchanged then \( h \) is 0!

\( \leadsto \) There is an information flow from \( h \) into \( l \).
Standard non-interference

∀ l : L, ∀ h₁, h₂ : H. \( \llbracket P \rrbracket (h₁, l)^L = \llbracket P \rrbracket (h₂, l)^L \)

Example:

```plaintext
while h do (l := l + 2; h := h - 1).
```

<table>
<thead>
<tr>
<th>h</th>
<th>l</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>l = 1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>l = 3</td>
</tr>
<tr>
<td>n</td>
<td>1</td>
<td>l = 1 + 2n</td>
</tr>
</tbody>
</table>

If l is unchanged then h is 0!
⇒ There is an information flow from h into l.

⇒ Note that if the input l is even/odd then the output l is even/odd!
Abstracting non-interference

Standard non-interference
\[ \forall l \in L, \forall h_1, h_2 \in H. \ [P](h_1, l)^L = [P](h_2, l)^L \]

Consider \( \alpha, \eta \in Abs(\varphi(\forall^L)) \):
Abstracting non-interference I

Standard non-interference
$$\forall l : L, \forall h_1, h_2 : H. \, \mathcal{P}(h_1, l)^L = \mathcal{P}(h_2, l)^L$$

Consider $$\alpha, \eta \in \text{Abs}(\wp(\mathbb{V}^L))$$:

**Narrow (abstract) non-interference** $$[\eta] \mathcal{P}(\alpha)$$:
$$\eta(l_1) = \eta(l_2) \Rightarrow \alpha(\mathcal{P}(h_1, l_1)^L) = \alpha(\mathcal{P}(h_2, l_2)^L)$$

- No change of $$H$$ values and $$\eta$$-equivalent $$L$$ values may affect the $$\alpha$$ abstraction of $$L$$ outputs.
- Possible deceptive interference due to $$\eta$$-undistinguished $$L$$ values!
- The more $$\eta$$ is precise the less deceptive interference appears.
Abstracting non-interference

Standard non-interference
\[ \forall l : L, \forall h_1, h_2 : H. \; \sem{P}(h_1, l)^L = \sem{P}(h_2, l)^L \]

Consider \( \alpha, \eta \in \text{Abs}(\wp(\mathbb{N}^L)) \):

**Narrow (abstract) non-interference** \([\eta]P(\alpha)\):
\[ \eta(l_1) = \eta(l_2) \Rightarrow \alpha(\sem{P}(h_1, l_1)^L) = \alpha(\sem{P}(h_2, l_2)^L) \]

**EXAMPLE:** \([\text{id}]P(\text{Par})\)

\[
P = \text{while } h \text{ do } (l := l + 2; \ h := h - 1).
\]

\[
\begin{align*}
h &= 0, \ l = 1 & \Rightarrow & \ Par(l) = \text{odd} \\
h &= 1, \ l = 1 & \Rightarrow & \ Par(l) = \text{odd} \\
h &= n, \ l = 1 & \Rightarrow & \ Par(l) = \text{odd}
\end{align*}
\]
Abstracting non-interference I

Standard non-interference
\[ \forall l : L, \forall h_1, h_2 : H. \ [P](h_1, l)^L = [P](h_2, l)^L \]

Consider \( \alpha, \eta \in \text{Abs}(\varphi(\forall l^L)) \):

**Narrow (abstract) non-interference** \([\eta]P(\alpha)\):
\[ \eta(l_1) = \eta(l_2) \Rightarrow \alpha([P](h_1, l_1)^L) = \alpha([P](h_2, l_2)^L) \]

**EXAMPLE:** \([id]P(Par)\)

```plaintext
P= while h do (l := l + 2; h := h - 1).

h = 0, l = 1 \Rightarrow Par(l) = \text{odd}

h = 1, l = 1 \Rightarrow Par(l) = \text{odd}

h = n, l = 1 \Rightarrow Par(l) = \text{odd}
```

If \( l \) is odd/even then, independently from \( h \), after the execution \( l \) is odd/even!
\[ \sim \Rightarrow \text{There is not an information flow from } h \text{ into the parity of } l. \]
Abstracting non-interference I

Standard non-interference
\[ \forall l : L, \forall h_1, h_2 : H. \ [P](h_1, l)^L = [P](h_2, l)^L \]

Consider \( \alpha, \eta \in Abs(\varphi(\forall^L)) \):

**Narrow (abstract) non-interference** \( [\eta]P(\alpha) \):
\[ \eta(l_1) = \eta(l_2) \Rightarrow \alpha([P](h_1, l_1)^L) = \alpha([P](h_2, l_2)^L) \]

**Example II**: \([Par]P(\text{Sign})\)

\[ P = l := 2 * l * h^2. \]
\[ h = -3, l = -2 \ (Par(-2) = \text{even}) \Rightarrow Sign(l) = - \]
\[ h = 1, l = -4 \ (Par(-4) = \text{even}) \Rightarrow Sign(l) = - \]
Abstracting non-interference I

Standard non-interference
\[ \forall l : L, \forall h_1, h_2 : H. \, [P](h_1, l)^L = [P](h_2, l)^L \]

Consider \( \alpha, \eta \in \text{Abs}(\varphi(\mathbb{V}^L)) \):

**Narrow (abstract) non-interference** \([\eta]P(\alpha)\):
\[ \eta(l_1) = \eta(l_2) \Rightarrow \alpha([P](h_1, l_1)^L) = \alpha([P](h_2, l_2)^L) \]

**Example II:** \([Par]P(\text{Sign})\)

\[ P = l := 2 \times l \times h^2. \]

<table>
<thead>
<tr>
<th>( h )</th>
<th>( l )</th>
<th>( Par(\text{even}) )</th>
<th>( Sign(\text{calculated}) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>( Par(4) = \text{even} )</td>
<td>( Sign(4) = + )</td>
</tr>
<tr>
<td>1</td>
<td>-4</td>
<td>( Par(-4) = \text{even} )</td>
<td>( Sign(-4) = - )</td>
</tr>
</tbody>
</table>

The sign of the output \( l \) depends on the sign of the input \( l \!
\sim\sim \text{There is a DECEPTIVE FLOW!} \]
Abstracting non-interference I

Standard non-interference

\[ \forall l : \mathbb{L}, \forall h_1, h_2 : \mathbb{H}. \quad \llbracket P \rrbracket (h_1, l)^L = \llbracket P \rrbracket (h_2, l)^L \]

Consider \( \alpha, \eta \in \text{Abs}(\varphi(\mathbb{V}^L)) \):

**Narrow (abstract) non-interference** \([\eta] P(\alpha)\):

\[ \eta(l_1) = \eta(l_2) \Rightarrow \alpha(\llbracket P \rrbracket (h_1, l_1)^L) = \alpha(\llbracket P \rrbracket (h_2, l_2)^L) \]

**Example II**: \([Par] P(\text{Sign})\)

\[ P = l := 2 \times l \times h^2. \]

\[ h = 1, \quad l = 4 \quad \text{(Par}(4) = \text{even}) \quad \sim \quad \text{Sign}(l) = + \]

\[ h = 1, \quad l = -4 \quad \text{(Par}(-4) = \text{even}) \quad \sim \quad \text{Sign}(l) = - \]

The sign of the output \( l \) depends on the sign of the input \( l \)!

\[ \sim \Rightarrow \text{There is a DECEPTIVE FLOW!} \]

\[ \Rightarrow \text{We compute the semantics on the concrete value of the input } l! \]
Consider $\alpha, \eta \in \text{Abs}(\wp(\mathbb{N}^L))$:

**Narrow (abstract) non-interference** $[\eta]P(\alpha)$:

$\eta(l_1) = \eta(l_2) \Rightarrow \alpha(\llbracket P \rrbracket(h_1, l_1)^L) = \alpha(\llbracket P \rrbracket(h_2, l_2)^L)$
Consider $\alpha, \eta \in \text{Abs}(\varphi(\forall^L))$:

**Narrow (abstract) non-interference** $[\eta]P(\alpha)$:

$\eta(l_1) = \eta(l_2) \Rightarrow \alpha([P](h_1, l_1)^L) = \alpha([P](h_2, l_2)^L)$

**Abstracting non-interference** $(\eta)P(\alpha)$:

$\eta(l_1) = \eta(l_2) \Rightarrow \alpha([P](h_1, \eta(l_1))^L) = \alpha([P](h_2, \eta(l_2))^L)$

- No change of $H$ values may affect the $\alpha$ abstraction of $L$ outputs.
- No deceptive interference due to $L$ data
Abstracting non-interference II

Consider $\alpha, \eta \in \text{Abs}(\varphi(\mathbb{V}^L))$:

**Narrow (abstract) non-interference** $[\eta]P(\alpha)$:

$$\eta(l_1) = \eta(l_2) \Rightarrow \alpha([P](h_1, l_1)^L) = \alpha([P](h_2, l_2)^L)$$

**Abstracting non-interference** $(\eta)P(\alpha)$:

$$\eta(l_1) = \eta(l_2) \Rightarrow \alpha([P](h_1, \eta(l_1))^L) = \alpha([P](h_2, \eta(l_2))^L)$$

**Example:** $(\text{Par})P(\text{Sign})$

$$P = l := 2 * l * h^2.$$  

- $h = -3, \text{Par}(l) = \text{even} \leadsto \text{Sign}(l) = \text{I don’t know}$
- $h = 1, \text{Par}(l) = \text{even} \leadsto \text{Sign}(l) = \text{I don’t know}$
Abstracting non-interference II

Consider \( \alpha, \eta \in \text{Abs}(\wp(\mathcal{W}^L)) \):

**Narrow (abstract) non-interference** \([\eta]P(\alpha)\):
\[
\eta(l_1) = \eta(l_2) \Rightarrow \alpha([P](h_1, l_1)^L) = \alpha([P](h_2, l_2)^L)
\]

**Abstracting non-interference** \((\eta)P(\alpha)\):
\[
\eta(l_1) = \eta(l_2) \Rightarrow \alpha([P](h_1, \eta(l_1))^L) = \alpha([P](h_2, \eta(l_2))^L)
\]

**Example**: \((Par)P(\text{Sign})\)

\[
P = l := 2 \times l \times h^2.
\]

\( h = -3, \ Par(l) = \text{even} \leadsto \ Sign(l) = I \ don't \ know \)

\( h = 1, \ Par(l) = \text{even} \leadsto \ Sign(l) = I \ don't \ know \)

\( \leadsto \) There is not an information flow from \( h \) into the sign of \( l \).
Consider $\alpha, \eta \in \text{Abs}(\varphi(\mathbb{V}^L))$:

**Narrow (abstract) non-interference** $[\eta]P(\alpha)$:
$\eta(l_1) = \eta(l_2) \Rightarrow \alpha([P](h_1, l_1)^L) = \alpha([P](h_2, l_2)^L)$

**Abstracting non-interference** $(\eta)P(\alpha)$:
$\eta(l_1) = \eta(l_2) \Rightarrow \alpha([P](h_1, \eta(l_1))^L) = \alpha([P](h_2, \eta(l_2))^L)$

**Example II**: $(\text{id})P(\text{Par})$

$$P = l := l \ast h^2.$$

- $h = 2, \ l = 1 \leadsto \text{Par}(l) = \text{even}$
- $h = 3, \ l = 1 \leadsto \text{Par}(l) = \text{odd}$
- $h = n, \ l = 1 \leadsto \text{Par}(l) = \text{Par}(n)$
Abstracting non-interference II

Consider $\alpha, \eta \in \text{Abs}(\varphi(\mathcal{V}^L))$:

**Narrow (abstract) non-interference** $[\eta]\mathcal{P}(\alpha)$:

$\eta(l_1) = \eta(l_2) \Rightarrow \alpha([\mathcal{P}](h_1, l_1)^L) = \alpha([\mathcal{P}](h_2, l_2)^L)$

**Abstracting non-interference** $(\eta)\mathcal{P}(\alpha)$:

$\eta(l_1) = \eta(l_2) \Rightarrow \alpha([\mathcal{P}](h_1, \eta(l_1))^L) = \alpha([\mathcal{P}](h_2, \eta(l_2))^L)$

**Example II:** $(\text{id})\mathcal{P}(\text{Par})$

$$
\mathcal{P} = \text{l := l} * \text{h}^2.
$$

- $h = 2$, $l = 1 \leadsto \text{Par}(l) = \text{even}$
- $h = 3$, $l = 1 \leadsto \text{Par}(l) = \text{odd}$
- $h = n$, $l = 1 \leadsto \text{Par}(l) = \text{Par}(n)$

$\leadsto$ The parity of $h$ is flowing into $l$!
Consider $\alpha, \eta \in \text{Abs}(\wp(\mathcal{V}^L))$: 

**Narrow (abstract) non-interference** $[\eta]P(\alpha)$: 
$\eta(l_1) = \eta(l_2) \Rightarrow \alpha([P](h_1, l_1)^L)) = \alpha([P](h_2, l_2)^L)$

**Abstracting non-interference** $(\eta)P(\alpha)$: 
$\eta(l_1) = \eta(l_2) \Rightarrow \alpha([P](h_1, \eta(l_1))^L)) = \alpha([P](h_2, \eta(l_2))^L)$

**Example II:** $(\text{id})P(\text{Par})$

\[
P = \ l := l \ast h^2.
\]

$h = 2, \ l = 1 \leadsto \ Par(l) = \text{even}$

$h = 3, \ l = 1 \leadsto \ Par(l) = \text{odd}$

$h = n, \ l = 1 \leadsto \ Par(l) = Par(n)$

$\leadsto$ The parity of $h$ is flowing into $l$!

$\Rightarrow$ We are looking for flows from any possible property of $h$ into $l$!
Consider $\alpha, \eta \in \text{Abs} (\wp (\mathbb{V}_L))$ and $\phi \in \text{Abs} (\wp (\mathbb{V}_H))$:

**Abstracting non-interference** $(\eta) \text{P}(\alpha)$:

$\eta (l_1) = \eta (l_2) \Rightarrow \alpha (\llbracket \text{P} \rrbracket (h_1, \eta (l_1))^L) = \alpha (\llbracket \text{P} \rrbracket (h_2, \eta (l_2))^L)$
Abstracting non-interference III

Consider $\alpha, \eta \in \text{Abs}(\phi(\mathbb{V}^L))$ and $\phi \in \text{Abs}(\phi(\mathbb{V}^H))$:

**Abstracting non-interference** $(\eta)\mathcal{P}(\alpha)$:

$$\eta(l_1) = \eta(l_2) \Rightarrow \alpha(\llbracket \mathcal{P} \rrbracket(h_1, \eta(l_1))^L) = \alpha(\llbracket \mathcal{P} \rrbracket(h_2, \eta(l_2))^L)$$

**Abstract non-interference** $(\eta)\mathcal{P}(\phi \sim \llbracket \alpha \rrbracket)$:

$$\eta(l_1) = \eta(l_2) \Rightarrow \alpha(\llbracket \mathcal{P} \rrbracket(\phi(h_1), \eta(l_1))^L) = \alpha(\llbracket \mathcal{P} \rrbracket(\phi(h_2), \eta(l_2))^L)$$

No change of $\phi$-equivalent $H$ values may affect the $\alpha$ abstraction of $L$ outputs.

No deceptive interference due to $L$ data;

$\phi$ does not flow into what $\alpha$ can see on the output
Abstracting non-interference III

Consider $\alpha, \eta \in \text{Abs}(\wp(\mathbb{V}_L))$ and $\phi \in \text{Abs}(\wp(\mathbb{V}_H))$:

**Abstracting non-interference** $(\eta)P(\alpha)$:
$$\eta(l_1) = \eta(l_2) \Rightarrow \alpha([P](h_1, \eta(l_1))^L) = \alpha([P](h_2, \eta(l_2))^L)$$

**Abstract non-interference** $(\eta)P(\phi \sim [\alpha])$:
$$\eta(l_1) = \eta(l_2) \Rightarrow \alpha([P](\phi(h_1), \eta(l_1))^L) = \alpha([P](\phi(h_2), \eta(l_2))^L)$$

**Example**: $(\text{id})P(\text{Sign} \sim [\text{Par}])$

$$P = l := l \times h^2.$$  

$\text{Sign}(h) = +, \ l = 1 \sim \text{Par}(l) = l \text{ don’t know}$

$\text{Sign}(h) = -, \ l = 1 \sim \text{Par}(l) = l \text{ don’t know}$
Consider $\alpha, \eta \in \text{Abs}(\wp(\mathbb{V}^L))$ and $\phi \in \text{Abs}(\wp(\mathbb{V}^H))$:

**Abstracting non-interference** $(\eta)P(\alpha)$:

$$\eta(l_1) = \eta(l_2) \Rightarrow \alpha([P](h_1, \eta(l_1))^L) = \alpha([P](h_2, \eta(l_2))^L)$$

**Abstract non-interference** $(\eta)P(\phi \leadsto \alpha)$:

$$\eta(l_1) = \eta(l_2) \Rightarrow \alpha([P](\phi(h_1), \eta(l_1))^L) = \alpha([P](\phi(h_2), \eta(l_2))^L)$$

**Example**: $(\text{id})P(\text{Sign} \leadsto \text{Par})$

$$P = \begin{array}{c}
l := l * h^2.
\end{array}$$

$$\text{Sign}(h) = +, \ l = 1 \leadsto \text{Par}(l) = I \text{ don’t know}$$

$$\text{Sign}(h) = -, \ l = 1 \leadsto \text{Par}(l) = I \text{ don’t know}$$

$\leadsto$ There is not an information flow from the sign of $h$ into the parity of $l$.  

**Abstract Non-interference** - Parameterizing Non-Interference by Abstract Interpretation – p.10/15
Basic properties

6 \[ [\eta] P(\top) \]

6 \[ [\eta] P(\alpha) \Rightarrow \forall \beta \subseteq \eta. [\beta] P(\alpha) \]

6 \[ [\eta] P(\alpha) \Rightarrow \forall \beta \supseteq \alpha. [\eta] P(\beta) \]

6 \[ \forall i. [\eta] P(\alpha_i) \Rightarrow [\eta] P(\bigcap_{i \in I} \alpha_i) \]
Basic properties

\[ [\eta] P(\top) \]

\[ [\eta] P(\alpha) \Rightarrow \forall \beta \subseteq \eta. [\beta] P(\alpha) \]

\[ [\eta] P(\alpha) \Rightarrow \forall \beta \supseteq \alpha. [\eta] P(\beta) \]

\[ \forall i. [\eta] P(\alpha_i) \Rightarrow [\eta] P(\bigcap_{i \in I} \alpha_i) \]

\[ (\eta) P(\phi \rightsquigarrow \top) \]

\[ (\eta) P(\phi \rightsquigarrow \alpha) \Rightarrow \forall \beta \supseteq \alpha. (\eta) P(\phi \rightsquigarrow \beta) \]

\[ \forall i. (\eta) P(\phi \rightsquigarrow \alpha_i) \Rightarrow (\eta) P(\phi \rightsquigarrow \bigcap_{i \in I} \alpha_i) \]
Basic properties

\[ \eta \] P(⊤) 

\[ \eta \] P(\alpha) \Rightarrow \forall \beta \subseteq \eta. [\beta] P(\alpha) 

\[ \eta \] P(\alpha) \Rightarrow \forall \beta \supseteq \alpha. [\eta] P(\beta) 

\forall i. [\eta] P(\alpha_i) \Rightarrow [\eta] P(\bigcap_{i \in I} \alpha_i) 

(\eta) P(\phi \sim id) 

(\eta) P(\phi \sim \alpha) \Rightarrow \forall \beta \supseteq \alpha. (\eta) P(\phi \sim \beta) 

\forall i. (\eta) P(\phi \sim \alpha_i) \Rightarrow (\eta) P(\phi \sim \bigcap_{i \in I} \alpha_i) 

Standard non-interference: [id] P(id) = (id) P(id \sim id)
Basic properties

\[ \eta \mapsto \top \]

\[ \eta \mapsto \alpha \Rightarrow \forall \beta \subseteq \eta \cdot [\beta] \mapsto \alpha \]

\[ \eta \mapsto \alpha \Rightarrow \forall \beta \supseteq \alpha \cdot [\eta] \mapsto \beta \]

\[ \forall i \cdot [\eta] \mapsto \alpha_i \Rightarrow [\eta] \mapsto \bigcap_{i \in I} \alpha_i \]

\[ (\eta) \mapsto \phi \sim [\top] \]

\[ (\eta) \mapsto \phi \sim [\alpha] \Rightarrow \forall \beta \supseteq \alpha \cdot (\eta) \mapsto \phi \sim [\beta] \]

\[ \forall i \cdot (\eta) \mapsto \phi \sim [\alpha_i] \Rightarrow (\eta) \mapsto \phi \sim [\bigcap_{i \in I} \alpha_i] \]

\[ [\text{id}] \mapsto \text{id} \Rightarrow (\eta) \mapsto \text{id} \sim [\alpha] \Rightarrow (\eta) \mapsto \phi \sim [\alpha] \]
Basic properties

\[ [\eta] P(\top) \]

\[ [\eta] P(\alpha) \Rightarrow \forall \beta \subseteq \eta. [\beta] P(\alpha) \]

\[ [\eta] P(\alpha) \Rightarrow \forall \beta \supseteq \alpha. [\eta] P(\beta) \]

\[ \forall i. [\eta] P(\alpha_i) \Rightarrow [\eta] P(\bigsqcup_{i \in I} \alpha_i) \]

\[ (\eta) P(\phi \sim[\top]) \]

\[ (\eta) P(\phi \sim[\alpha]) \Rightarrow \forall \beta \supseteq \alpha. (\eta) P(\phi \sim[\beta]) \]

\[ \forall i. (\eta) P(\phi \sim[\alpha_i]) \Rightarrow (\eta) P(\phi \sim[\bigsqcup_{i \in I} \alpha_i]) \]

\[ [\eta] P(\alpha) \Rightarrow (\eta) P(\text{id} \sim[\alpha]) \Rightarrow (\eta) P(\phi \sim[\alpha]) \]
Basic properties

1. \( [\eta]P(\top) \)
2. \( [\eta]P(\alpha) \Rightarrow \forall \beta \sqsubseteq \eta \cdot [\beta]P(\alpha) \)
3. \( [\eta]P(\alpha) \Rightarrow \forall \beta \sqsupseteq \alpha \cdot [\eta]P(\beta) \)
4. \( \forall i \cdot [\eta]P(\alpha_i) \Rightarrow [\eta]P(\bigcap_{i \in I} \alpha_i) \)
5. \( (\eta)P(\phi \sim \top) \)
6. \( (\eta)P(\phi \sim \alpha) \Rightarrow \forall \beta \sqsupseteq \alpha \cdot (\eta)P(\phi \sim \beta) \)
7. \( \forall i \cdot (\eta)P(\phi \sim \alpha_i) \Rightarrow (\eta)P(\phi \sim \bigcap_{i \in I} \alpha_i) \)

\[ [\text{id}]P(\text{id}) \nRightarrow [\eta]P(\alpha) \text{ due to deceptive flows} \]
Abstract interpretation provides advanced methods for designing abstractions (refinement, simplification, compression ...)

[Giacobazzi & Ranzato ’97]

Designing abstractions = designing attackers
Abstract interpretation provides advanced methods for designing abstractions (refinement, simplification, compression ... ) [Giacobazzi & Ranzato '97]

Designing abstractions = designing attackers

Characterize the most concrete $\alpha$ such that $(\eta)P(\phi \leadsto \|\alpha)$

[The most powerful output attacker]
Deriving output attackers

The following theorems hold:

Consider \( \eta \in \text{Abs}(\mathcal{G}(\forall^L)) \):

We characterize the function \( \lambda \eta. [\eta][P](\text{id}) \) whose result is

\[
\cap \{ \beta \mid [\eta]P(\beta) \}.
\]
The following theorems hold:

6. Consider $\eta \in \text{Abs}(\varphi(\mathbb{V}^L))$:
   We characterize the function $\lambda \eta. [\eta][P](\text{id})$ whose result is
   \[ \Pi \{ \beta \mid \eta P(\beta) \} \].

6. Consider $\eta \in \text{Abs}(\varphi(\mathbb{V}^L))$ and $\phi \in \text{Abs}(\varphi(\mathbb{V}^H))$:
   We characterize the function $\lambda \eta. (\eta)[P](\phi \sim \text{id})$ whose result is
   \[ \Pi \{ \beta \mid (\eta)P(\phi \sim \beta) \} \].
Deriving output attackers

The following theorems hold:

1. Consider \( \eta \in Abs(\varphi(\mathbb{V}^L)) \):
   We characterize the function \( \lambda \eta. [\eta][P](id) \) whose result is
   \[
   \bigcap \{ \beta \mid [\eta]P(\beta) \}.
   \]

2. Consider \( \eta \in Abs(\varphi(\mathbb{V}^L)) \) and \( \phi \in Abs(\varphi(\mathbb{V}^H)) \):
   We characterize the function \( \lambda \eta. (\eta)[P](\phi \sim id) \) whose result is
   \[
   \bigcap \{ \beta \mid (\eta)P(\phi \sim \beta) \}.
   \]

\( \Rightarrow \) This would provide a certificate for security with a fixed input observation.
Deriving canonical attackers

Abstract interpretation provides advanced methods for designing abstractions (refinement, simplification, compression ...) [Giacobazzi & Ranzato ’97]

Transforming abstractions = transforming attackers
Deriving canonical attackers

Abstract interpretation provides advanced methods for designing abstractions (refinement, simplification, compression ...) \([\text{Giacobazzi & Ranzato '97}]\)

Transforming abstractions = transforming attackers

\[ \begin{align*}
\text{Characterize the most concrete } \delta \text{ such that } (\delta) \mathcal{P}(\phi \sim \| \delta) \\
\text{[The most powerful } \textit{canonical} \text{ attacker]}
\end{align*} \]

\[ \Rightarrow \text{ This would provide a certificate for security.} \]
Deriving canonical attackers

\[ \lambda X. [X][P](\text{id}) \text{ is monotone on } Abs(\varphi(\{V\})). \]

\[ [\alpha]P(\alpha) \iff \alpha = [\alpha][P](\text{id}). \]

\[ lfp(\lambda X. [X][P](\text{id})) \text{ is the most concrete secure attacker for } P \text{ for narrow abstract non-interference.} \]
Deriving canonical attackers

\[ \lambda X. \ [X][P](\text{id}) \text{ is monotone on } \text{Abs}(\wp(\mathcal{V}^L)). \]

\[ [\alpha]P(\alpha) \iff \alpha = [\alpha][P](\text{id}). \]

\[ \text{lfp}(\lambda X. \ [X][P](\text{id})) \text{ is the most concrete secure attacker for } P \text{ for narrow abstract non-interference.} \]

\[ (\alpha)P(\phi \leadsto [\alpha]) \iff \alpha = (\alpha)[P](\phi \leadsto [\text{id}]) \]

\[ \lambda X. \ (X)[P](\phi \leadsto [\text{id}] \sqcup X) \text{ is extensive on } \text{Abs}(\wp(\mathcal{V}^L)). \]

\[ \text{fix}(\lambda X. \ (X)[P](\phi \leadsto [\text{id}] \sqcup X) \text{ is a secure attacker for } P \text{ for abstract non-interference.} \]
Deriving canonical attackers

Example:

\[ P = \text{while } h \text{ do } (l := l \times 2; \ h := h - 1) \]

.... we derive a secure attacker \( \pi = \gamma (\{ n2^N \mid n \in 2^N + 1 \} \cup \{0\}) \):

\[ (\pi)[P](id \sim \pi) \]

\( h = 0, \ \pi(l) = 32^N \sim \pi(l) = 32^N \)

\( h = 2, \ \pi(l) = 32^N \sim \pi(l) = 32^N \)

\( \sim \) In the program \( l \) is always multiplied by 2!
Abstract robust declassification

Consider a program $P$ and its finite computations.

A passive attacker may be able to learn some information by observing the system but, by assumption, that information leakage is allowed by the security policy.

[Zdancewic and Myers 2001]
Abstract robust declassification

Consider a program \( P \) and its finite computations.

A passive attacker may be able to learn some information by observing the system but, by assumption, that information leakage is allowed by the security policy.

[Zdancewic and Myers 2001]

We want to characterize the most concrete *flow-irredundant* property such that

\[
(\eta)P(\phi \leadsto \alpha)
\]

[The maximal amount of information disclosed]

\( \Rightarrow \) This would provide a certificate for disclosed secrets.
Abstract robust declassification

Consider the program

\[ P = l := l + (h \mod 3) \]

The transition system is such that \( < h, l > \rightarrow < h, l + (h \mod 3) > \).

Consider \( \eta(\varnothing(\mathbb{Z})) = \{\mathbb{Z}, [2, 4], [5, 8], \{5\}, \emptyset\} \) and \( \alpha = \text{id} \).
Abstract robust declassification

Consider the program

\[ P = l := l + (h \mod 3) \]

The transition system is such that \( < h, l > \mapsto < h, l + (h \mod 3) > \).

Consider \( \eta(\varphi(\mathbb{Z})) = \{\mathbb{Z}, [2, 4], [5, 8], \{5\}, \emptyset\} \) and \( \alpha = \text{id} \).

The flow is revealed when \( h_1 \) and \( h_2 \) differs in the values \( h_1 \mod 3 \) and \( h_2 \mod 3 \)

\[ \Rightarrow \phi = \gamma(\{3\mathbb{Z}, 3\mathbb{Z} + 1, 3\mathbb{Z} + 2\}) \]

is the maximal amount of information disclosed!
Discussion

We map security of programs into the lattice of abstract interpretations:
- systematic methods for designing attackers and certificates
- security degrees compared in the lattice
- checking abstract non-interference by static program analysis
We map security of programs into the lattice of abstract interpretations:

- systematic methods for designing attackers and certificates
- security degrees compared in the lattice
- checking abstract non-interference by static program analysis

Abstract non-interference is a semantics property

- the method is language independent (as any abstract interpretation)
- refined semantics may refine security:
  covert channels (termination, non-determinism, synchronization, probabilistic, etc...) is a matter of semantics!
Discussion

We map security of programs into the lattice of abstract interpretations:
- systematic methods for designing attackers and certificates
- security degrees compared in the lattice
- checking abstract non-interference by static program analysis

Abstract non-interference is a semantics property
- the method is language independent (as any abstract interpretation)
- refined semantics may refine security:
  covert channels (termination, non-determinism, synchronization, probabilistic, etc...) is a matter of semantics!

How far is any practical application?
- program slicing may help in checking program secrecy!
- the common abstraction which is not disclosed for all program slices will not be disclosed by the whole program...