Our Idea
Our Idea

Our Idea

Secret H
Public L

Public L
Secret H

Observer: $\rho$

Observable: $\phi$

Abstract Interpretation

Consider $C = \wp(\mathbb{Z})$: [Cousot & Cousot'77]

Abstract Interpretation

Consider $C = \wp(\mathbb{Z})$: [Cousot & Cousot'77]
Abstract Interpretation

Consider $C = \wp(\mathbb{Z})$: [Cousot & Cousot'77]

Abstract domain

Abstract Interpretation

Consider $C = \wp(\mathbb{Z})$: [Cousot & Cousot'77]

Abstract domain

$\alpha, \gamma$ monotone, $\alpha(x) \leq y \iff x \leq \gamma(y)$, $\alpha \gamma(y) = y$, $\gamma \alpha(x) \geq x$

$\gamma(x) = \bigvee \{ y \mid \alpha(y) \leq x \}$ $\text{def} \equiv \alpha^+(x)$ and $\alpha(x) = \bigwedge \{ y \mid x \leq \gamma(y) \}$ $\text{def} \equiv \gamma^-(x)$


Abstract Interpretation

Consider $C = \wp(\mathbb{Z})$: [Cousot & Cousot'77]

Abstract domain

$\gamma \alpha$ monotone, $\gamma \alpha(x) \geq x$, $\gamma \alpha(\gamma \alpha(x)) = \gamma \alpha(x)$

$\Rightarrow$ Upper closure operator.
Abstract Interpretation

Consider the complete lattice \( \langle C, \leq, \wedge, \vee, \bot, \top \rangle \), \( A_i \in uco(C) \)

Lattice of Abstract Domains \( \cong \) Lattice \( uco \)
\[
A \equiv \rho(C) \\
\langle uco(C), \subseteq, \sqcup, \sqcap, \lambda x. \top, \lambda x. x \rangle
\]

\( A_1 \subseteq A_2 \iff A_2 \subseteq A_1 \)
Abstract Interpretation

Consider the complete lattice \(< C, \leq, \wedge, \vee, \bot, \top >\), \(A_i \in uco(C)\)

Lattice of Abstract Domains \(\equiv\) Lattice \(uco\)

\[ A \models p(C) \]

\(< uco(C), \subseteq, \Pi, \cup, \lambda x. \top, \lambda x. x \>

\[ A_1 \subseteq A_2 \iff A_2 \subseteq A_1 \]

\[ \cap_i A_i = \mathcal{M}(\cup_i A_i) \]

\[ \cup_i A_i = \cap_i A_i \]
Abstract Interpretation

Consider the complete lattice \(< C, \leq, \land, \lor, \bot, \top >\), \(A_i \in uco(C)\)

Lattice of Abstract Domains \(\equiv\) Lattice \(uco\)
\[
A \equiv \rho(C)
\]
\[
\begin{align*}
A_1 &\subseteq A_2 \iff A_2 \subseteq A_1 \\
\land_i A_i &= M(\lor_i A_i) \\
\lor_i A_i &= \land_i A_i
\end{align*}
\]

Abstract domain completeness

Let \( < A, \alpha, \gamma, C > \) a galois insertion. [Cousot & Cousot '77,'79]

\( f : C \rightarrow C, f^\alpha = \alpha \circ f \circ \gamma : A \rightarrow A \) (b.c.a. of \( f \)) and \( \rho = \gamma \circ \alpha \)

\[ \rho \text{ correct for } f \]

\[ \top \rightarrow f(\chi) \rightarrow \alpha f(\chi) \rightarrow \alpha(\chi) \rightarrow \bot^\alpha \]

\[ \bot \rightarrow f(\chi) \rightarrow \alpha f(\chi) \rightarrow \alpha(\chi) \rightarrow \bot^\alpha \]

\[ \rho \text{ complete for } f \]

\[ ||| \]

\[ \rho f \rho = \rho f \]
DEFINING
ABSTRACT NON-INTERFERENCE

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

Standard Non-Interference
\[ \forall l : L, \forall h_1, h_2 : H. \lceil P \rceil(h_1, l)^L = \lceil P \rceil(h_2, l)^L \]
∀ \ell : L, ∀ h_1, h_2 : H. [P](h_1, \ell)^L = [P](h_2, \ell)^L

Standard Non-Interference

∀ \ell : L, ∀ h_1, h_2 : H. [P](h_1, \ell)^L = [P](h_2, \ell)^L

Standard Non-Interference
Standard Non-Interference

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


Abstract Non-Interference (Narrow)

\[ \rho, \eta \in Abs(\rho(V^L)) : [\eta]P(\rho) : \eta(l_1) = \eta(l_2) \Rightarrow \rho([\llbracket P \rrbracket (h_1, l_1)^L) = \rho([\llbracket P \rrbracket (h_2, l_2)^L) \]

Language-based Security: Abstract Non-Interference – p.8/32
Abstract Non-Interference (Narrow)

\[ \rho, \eta \in Abs(\rho(\forall L)) : [\eta]P(\rho) \colon \eta(l_1) = \eta(l_2) \Rightarrow \rho([P](h_1, l_1)^L) = \rho([P](h_2, l_2)^L) \]

Language-based Security: Abstract Non-Interference – p.8/32

\[ \rho, \eta \in Abs(\rho(\forall L)) : [\eta]P(\rho) : \eta(l_1) = \eta(l_2) \Rightarrow \rho([P](h_1, l_1)^L) = \rho([P](h_2, l_2)^L) \]
Abstract Non-Interference (Narrow)

\[ \rho, \eta \in \text{Abs}(\rho(V^L)) : [\eta]P(\rho) : \eta(l_1) = \eta(l_2) \Rightarrow \rho([P](h_1, l_1)^L) = \rho([P](h_2, l_2)^L) \]

Abstract Non-Interference (Narrow)

\[ \rho, \eta \in Abs(\rho(V_L)) : [\eta]P(\rho) \colon \eta(l_1) = \eta(l_2) \Rightarrow \rho([P](h_1, l_1)^L) = \rho([P](h_2, l_2)^L) \]

Abstract Non-Interference (ANI)

\[ \rho, \eta \in Abs(\rho(V_L)) : (\eta)P(\rho) : \eta(l_1) = \eta(l_2) \Rightarrow \rho([P](h_1, \eta(l_1))^L) = \rho([P](h_2, \eta(l_2))^L) \]
Abstract Non-Interference (ANI)

\[ \rho, \eta \in Abs(\wp(\wp^L)) : (\eta)P(\rho) : \\
\eta(l_1) = \eta(l_2) \Rightarrow \rho([P](h_1, \eta(l_1))^L) = \rho([P](h_2, \eta(l_2))^L) \]

Abstract Non-Interference (ANI)

\[\rho, \eta \in A b(s(\forall^L)) \Rightarrow (\eta) P(\rho) : \eta(l_1) = \eta(l_2) \Rightarrow \rho([P](h_1, \eta(l_1))^L) = \rho([P](h_2, \eta(l_2))^L)\]

Examples

**Example I:**

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

**Standard Non-Interference ≡ [id]P(id)**

- If `h = 0`, `l = 1` then `l = 1`
- If `h = 1`, `l = 1` then `l = 1 + 2n`
- If `h = n`, `l = 1` then `Par(l)` is odd

**EXAMPLE II:**

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

[\(\text{Par}\)\(P(Sign)\)]

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

**Example III:**

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

\[(id)P(Par)\]

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

---

**Declassified ANI via blocking**

\[ \rho, \eta \in Abs(\mathcal{V}^L), \phi \in Abs(\mathcal{V}^H); \ (\eta)P(\phi \sim \eta) : \eta(l_1) = \eta(l_2) \Rightarrow \rho([P](\phi(h_1), \eta(l_1))^L) = \rho([P](\phi(h_2), \eta(l_2))^L) \]

[Giacobazzi & Mastroeni '04]
Declassified ANI via blocking

\[ \rho, \eta \in \text{Abs}(g(V^L_1)), \phi \in \text{Abs}(g(V^H)): (\eta)P(\phi \sim \rho):
\]

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

[Giacobazzi & Mastroeni '04]

\[
\rho, \eta \in \text{Abs}(\phi(V_L)), \phi \in \text{Abs}(\phi(V_H)) : (\eta)P(\phi \sim \rho):
\eta(l_1) = \eta(l_2) \Rightarrow \rho([P](\phi(h_1), \eta(l_1))^L) = \rho([P](\phi(h_2), \eta(l_2))^L)
\]
Declassified ANI via blocking

\[ \rho, \eta \in \text{Abs}(\mathcal{V}^L), \phi \in \text{Abs}(\mathcal{V}^H); (\eta)P(\phi \rightarrow \rho): \eta(l_1) = \eta(l_2) \Rightarrow \rho([P](\phi(h_1), \eta(l_1))^L) = \rho([P](\phi(h_2), \eta(l_2))^L) \]

Example

**EXAMPLE:**

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

\[ (id)P(Par) \]

\[ h = 2, \; l = 1 \; \rightarrow \; Par(l) = \text{even} \]
\[ h = 3, \; l = 1 \; \rightarrow \; Par(l) = \text{odd} \]
\[ h = n, \; l = 1 \; \rightarrow \; Par(l) = Par(n) \]

\[ (id)P(\text{Sign} \rightarrow Par) \]

\[ \text{Sign}(h) = +, \; l = 1 \; \rightarrow \; Par(l) = \text{I don't know} \]
\[ \text{Sign}(h) = -, \; l = 1 \; \rightarrow \; Par(l) = \text{I don't know} \]
Declassified ANI (via allowing)

\[ \rho, \eta \in \text{Abs}(\wp(V_L)), \phi \in \text{Abs}(\wp(V_H)): (\eta)P(\phi \Rightarrow \rho): \eta(l_1) = \eta(l_2) \text{ and } \phi(h_1) = \phi(h_2) \Rightarrow \rho([P](h_1, \eta(l_1))^L) = \rho([P](h_2, \eta(l_2))^L) \]


Declassified ANI (via allowing)

\[ \rho, \eta \in \text{Abs}(\wp(V_L)), \phi \in \text{Abs}(\wp(V_H)): (\eta)P(\phi \Rightarrow \rho): \eta(l_1) = \eta(l_2) \text{ and } \phi(h_1) = \phi(h_2) \Rightarrow \rho([P](h_1, \eta(l_1))^L) = \rho([P](h_2, \eta(l_2))^L) \]

Declassified ANI (via allowing)

\[
\rho, \eta \in \text{Abs}(\wp(\wp^L)), \phi \in \text{Abs}(\wp(\wp^H)) : (\eta)P(\phi \Rightarrow \rho) : \\
\eta(l_1) = \eta(l_2) \text{ and } \phi(h_1) = \phi(h_2) \Rightarrow \rho([P](h_1, \eta(l_1))^L) = \rho([P](h_2, \eta(l_2))^L)
\]

[Giacobazzi & Mastroeni '04]

Timed abstract non-interference

Standard denotational semantics

\[ [P] \]

\[ \alpha^D \]

Standard trace semantics

\[ \langle P \rangle \]

Trace semantics

\[ \text{Traces' length} = \text{TIME ELAPSED} \]
Timed abstract non-interference

Standard denotational semantics

Timed denotational semantics

Standard trace semantics

Traces’ length = Time elapsed

Stuttering removes time from traces!

TRACE SEMANTICS

2Z → 2Z → 2Z + 1 → 2Z

≠

2Z → 2Z + 1 → 2Z + 1 → 2Z

TRACE WITHOUT STUTTERING

2Z → 2Z + 1 → 2Z

= 

2Z → 2Z + 1 → 2Z

Timed abstract non-interference

Timed denotational semantics

Time counter = Time elapsed
Timed abstract non-interference

\[ \alpha_{\text{St}}^{\alpha_{\text{D}}} \]

\[ [P]^{+T} \]

\[ \langle P \rangle^{+T} \]

Standard denotational semantics

Standard trace semantics

Timed trace semantics

\[ \alpha_{\text{St}} \]

\[ \alpha_{\text{D}} \]

\[ \alpha_{\text{T}} \]

\[ \text{Timed denotational semantics} \]

\[ \Downarrow \]

\[ \text{Time counter} = \text{TIME ELAPSED} \]


---

Abstraction removes time from timed denotational semantics!

**Timed Semantics**

\[ Par([P][2, 4, 0])^{TL} = Par(6, 3) = \langle 2\mathbb{Z}, 2\mathbb{Z} + 1 \rangle \]

\[ Par([P][4, 4, 0])^{TL} = Par(8, 6) = \langle 2\mathbb{Z}, 2\mathbb{Z} \rangle \]

**Untimed Semantics**

\[ \Pi^T(\langle 2\mathbb{Z}, 2\mathbb{Z} + 1 \rangle) = \langle 2\mathbb{Z}, \mathbb{Z} \rangle \]

\[ \Pi^T(\langle 2\mathbb{Z}, 2\mathbb{Z} \rangle) = \langle 2\mathbb{Z}, \mathbb{Z} \rangle \]

CHARACTERIZING AND ENFORCING
ABSTRACT NON-INTERFERENCE

Deriving output attackers

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

Designing abstractions = designing attackers
Deriving output attackers

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

Designing abstractions = designing attackers

⇓

6 Characterize the most concrete ρ such that \((η)P(φ ⇝ \rho)\)
[The most powerful public observer]

Deriving output attackers

The following theorems hold:

6 Consider η ∈ Abs(℘(VL)):
We characterize the function \(λη. [η]P(id)\) whose result is
\(\bigcap \{ β \mid [η]P(β) \} \).
Deriving output attackers

The following theorems hold:

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

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

⇒ 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

⇓

6 Characterize the most concrete δ such that (δ)P(φ, δ)

[The most powerful canonical public observer]

⇒ This would provide a certificate for security.
Deriving canonical attackers

**Example:**

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

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

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

\[ \begin{align*}
  h &= 0, \ \pi(1) = 32^N & \rightarrow & \pi(1) = 32^N \\
  h &= 2, \ \pi(1) = 32^N & \rightarrow & \pi(1) = 32^N
\end{align*} \]

\[ \rightarrow \text{In the program } l \text{ is always multiplied by 2!} \]

Proving Abstract Non-Interference

Abstract Non-Interference is not defined *inductively* on the syntax
Proving Abstract Non-Interference

Abstract Non-Interference is not defined *inductively* on the syntax

\[ \downarrow \]

Use of Abstract Non-Interference hard in automatic program verification mechanisms.

---

Proving Abstract Non-Interference

Abstract Non-Interference is not defined *inductively* on the syntax

\[ \downarrow \]

Use of Abstract Non-Interference hard in automatic program verification mechanisms.

\[ \downarrow \]

Definition of *sound* proof systems for Abstract Non-Interference.

\[
\begin{align*}
[\eta]c_1(\rho), \ [\rho]c_2(\beta) & \quad (\eta)c_1(\gamma(\rho)), \ [\rho]c_2(\gamma(\beta)) \\
[\eta]c_1; c_2(\beta) & \quad (\eta)c_1; c_2(\gamma(\beta))
\end{align*}
\]
Observer vs Observable

Consider $\vdash (\eta)P(\phi \Rightarrow \square\rho)$: In order to preserve non-interference...
Observer vs Observable

Consider $\models (\eta)P(\phi \rightsquigarrow \rho)$: In order to preserve non-interference...

More abstract

$\text{uco}(\varphi(V^L))$

More concrete

$\text{uco}(\varphi(V^H))$

ANI: A completeness problem

Recall that [Joshi & Leino'00]

$\text{P is secure} \iff \text{HH ; P ; HH } \models \text{P ; HH}$
ANI: A completeness problem

Recall that [Joshi & Leino’00]

\[ P \text{ is } \textit{secure} \text{ iff } HH; P; HH \models P; HH \]

Let \( X = \langle X^H, X^L \rangle \Rightarrow H(X) \overset{\text{def}}{=} \langle T^H, X^L \rangle \in \text{uco}(\varphi(\mathbb{V})) \)

\[ HH; P; HH \models P; HH \]

\[ \downarrow \]

\[ H \circ [P] \circ H = H \circ [P] \]

\[ \Rightarrow \text{ A COMPLETENESS PROBLEM} \]
ANI: A completeness problem

Let \( X = \langle X^H, X^L \rangle \Rightarrow H(X) \overset{\text{def}}{=} \langle T^H, X^L \rangle \in \text{uco}(\varphi(V)) \)

\[
H \circ [P] \circ H = H \circ [P]
\]

Completeness = Non-Interference

\[\Downarrow\]

1. Transform \( H \) vs Core;
2. Transform \( H \) vs Shell. [Giacobazzi et al.’00]

Making Backward complete

\[
\rho_2 f \rho_1 = \rho_2 f
\]
Making Backward complete

\[ \rho_2 f \rho_1 = \rho_2 f \]

Completeness shells and cores

\[ R_f \overset{\text{def}}{=} \lambda \rho. M( \bigcup_{y \in \rho} \max(f^{-1}(\downarrow y))) \]
Completeness shells and cores

[Giacobazzi et al.'00]

P doesn't hold

A

P holds: Shell of $A$

\[
R_f^{\text{def}} = \lambda \rho. M( \bigcup_{y \in \rho} \max(f^{-1}(\downarrow y)))
\]

6 Absolute shell of $\rho$: $R_f(\rho) = gfp_\rho \lambda \varphi. \rho \cap R_f^B(\varphi)$;

6 Relative shell of $\eta$ relative to $\rho$: $R_f^\rho(\eta) = \eta \cap R_f(\rho)$.

Completeness shells and cores

[Giacobazzi et al.'00]

P doesn't hold

A

P holds: Shell of $A$

\[
C_f^{\text{def}} = \lambda \rho. \left\{ y \in C \mid \max(f^{-1}(\downarrow y)) \subseteq \rho \right\}
\]

Completeness shells and cores

\[ C_f \text{ def } = \lambda \rho. \left\{ y \in C \mid \max(f^{-1}(\downarrow y)) \subseteq \rho \right\} \]

6 Absolute core of \( \rho \): \( C_f(\rho) = \text{lt} \textbf{C}_f(\rho) \triangleq C_f^B(\rho) \);

6 Relative core of \( \rho \) relative to \( \eta \): \( C_f^n(\rho) = \rho \triangleq C_f(\eta) \).

ANI as completeness

Let \( \rho \in \text{uco}(\wp(V^L)) \Rightarrow \mathcal{H}_\rho(X) \text{ def } = (\top^\mathcal{H}, \rho(X^L)) \in \text{uco}(\wp(V)) \)

6 Narrow abstract non-interference: \( \mathcal{H}_\rho \circ [P] \circ \mathcal{H}_\eta = \mathcal{H}_\rho \circ [P] \);

6 Abstract non-interference: \( \mathcal{H}_\rho \circ [P]^\eta,\phi \circ \mathcal{H}_\eta = \mathcal{H}_\rho \circ [P]^\eta,\phi \)
ANI as completeness

Let $\rho \in uco(\wp(V^L)) \Rightarrow H_\rho(X) \overset{\text{def}}{=} (\top^H, \rho(X^L)) \in uco(\wp(V))$

6 Narrow abstract non-interference: $H_\rho \circ [P] \circ H_\eta = H_\rho \circ [P]$;

6 Abstract non-interference: $H_\rho \circ [P]^{\eta, \phi} \circ H_\eta = H_\rho \circ [P]^{\eta, \phi}$

↓

6 Public observer as completeness core: $C_{[P]^{\eta, \phi}}(H) = (\eta)[P](\phi \sim id)$

PRIVATE OBSERVABLE AS COMPLETENESS SHELL: $(\eta)P[R_{[P]^{\eta, id}}(H_\eta) \Rightarrow \rho)$
ANI as completeness

Public observer as completeness core: $C_{\mathcal{P}^\eta,\Phi}^{\mathcal{H}}(\mathcal{H}) = (\eta)[\mathcal{P}](\Phi \sim [id])$

Private observable as completeness shell: $(\eta)\mathcal{P}^{\mathcal{R}^{\mathcal{H},\rho} \mathcal{P}^\eta} \mathcal{H}_\eta \Rightarrow \rho$

Adjoining attackers and declassification

$$id \sqsubseteq (\eta)[\mathcal{P}](id \sim [id]) \Leftrightarrow \mathcal{P}(\cap_{L \in \eta} \mathcal{M}(\Pi_{\eta, id}[L])) \sqsubseteq T$$

Declassification

[Banerjee, Giacobazzi and Mastroeni '07]

By exploiting the strong relation between completeness and non-interference we can obtain the following results:

- Model declassification as a forward completeness problem for the weakest precondition semantics;
- Derive counterexamples to a given declassification policy;
- Refine a given declassification policy;

$$p \overset{\text{def}}{=} \begin{cases} \text{if}(d \leq x + y \leq d + d_x + d_y \land -d_y \leq x - y \leq d_x) \text{ then} & \\
\text{if}(x \geq 0 \land x \leq d) \text{ then } x_L := d; & \\
\text{if}(x > d \land x \leq d_x) \text{ then } x_L := x; & \\
\text{if}(x > d_x \land x \leq d_x + d) \text{ then } x_L := d_x; & \\
\text{if}(y \geq 0 \land y \leq d) \text{ then } y_L := d; & \\
\text{if}(y > d \land y \leq d_y) \text{ then } y_L := y; & \\
\text{if}(y > d_y \land y \leq d_y + d) \text{ then } y_L := d_y; & \\
\end{cases}$$
By exploiting the strong relation between completeness and non-interference we can obtain the following results:

- Model declassification as a forward completeness problem for the weakest precondition semantics;
- Derive counterexamples to a given declassification policy;
- Refine a given declassification policy;

We can model declassification as a model checking problem (see the relation with robust declassification)
Generalized abstract non-interference

NON-INTERFERENCE

Corresponds to asking that the behavior of the chosen relevant aspects of the computation be invariant with respect to what an attacker may observe.

αOBS: Specifies the semantics of the computations relevant for interference (observation abstraction);
Generalized abstract non-interference

Non-interference

Corresponds to asking that the behavior of the chosen relevant aspects of the computation be invariant with respect to what an attacker may observe.

\( \alpha_{\text{OBS}} \): Specifies the semantics of the computations relevant for interference (observation abstraction);

\( \alpha_{\text{INT}} \): Specifies the maximum amount of information that an attacker may observe concerning a computation (interference abstraction);

\( \alpha_{\text{ATT}} \): Characterizes what the model of the attacker can observe about the system behavior (attacker abstraction).
Generalized abstract non-interference

**NON-INTERFERENCE**

Corresponds to asking that the behavior of the chosen relevant aspects of the computation be invariant with respect to what an attacker may observe.

- $\alpha_{\text{OBS}}$: Specifies the semantics of the computations relevant for interference (*observation abstraction*);
- $\alpha_{\text{INT}}$: Specifies the maximum amount of information that an attacker may observe concerning a computation (*interference abstraction*);
- $\alpha_{\text{ATT}}$: Characterizes what the model of the attacker can observe about the system behavior (*attacker abstraction*).

\[
\alpha_{\text{ATT}} \circ \alpha_{\text{OBS}}([P]) = \alpha_{\text{ATT}} \circ \alpha_{\text{INT}} \circ \alpha_{\text{OBS}}([P]).
\]

**⇒** We characterize the minimal abstraction of $\alpha_{\text{ATT}}$ that guarantees GANI.
Generalized abstract non-interference

**NON-INTERFERENCE**

Corresponds to asking that the behavior of the chosen relevant aspects of the computation be invariant with respect to what an attacker may observe.

- $\alpha_{OBS}$: Specifies the semantics of the computations relevant for interference (*observation abstraction*);
- $\alpha_{INT}$: Specifies the maximum amount of information that an attacker may observe concerning a computation (*interference abstraction*);
- $\alpha_{ATT}$: Characterizes what the model of the attacker can observe about the system behavior (*attacker abstraction*).

$$\text{SNNI} = \alpha_T \circ \alpha_{\text{low}} \circ \text{id}(\llbracket P \rrbracket) = \alpha_T \circ \alpha_{\text{low}} \circ \alpha_L \circ \text{id}(\llbracket P \rrbracket).$$

Generalized abstract non-interference

**NON-INTERFERENCE**

Corresponds to asking that the behavior of the chosen relevant aspects of the computation be invariant with respect to what an attacker may observe.

- $\alpha_{OBS}$: Specifies the semantics of the computations relevant for interference (*observation abstraction*);
- $\alpha_{INT}$: Specifies the maximum amount of information that an attacker may observe concerning a computation (*interference abstraction*);
- $\alpha_{ATT}$: Characterizes what the model of the attacker can observe about the system behavior (*attacker abstraction*).

$$\text{BNDC} = \alpha_B \circ \alpha_L \circ \text{id}(\llbracket P \| IT \rrbracket) = \alpha_B \circ \alpha_L \circ \alpha_{\text{sec}} \circ \text{id}(\llbracket P \| IT \rrbracket).$$
Generalized abstract non-interference

Corresponds to asking that the behavior of the chosen relevant aspects of the computation be invariant with respect to what an attacker may observe.

- $\alpha_{OBS}$: Specifies the semantics of the computations relevant for interference (observation abstraction);
- $\alpha_{INT}$: Specifies the maximum amount of information that an attacker may observe concerning a computation (interference abstraction);
- $\alpha_{ATT}$: Characterizes what the model of the attacker can observe about the system behavior (attacker abstraction).

$n$-interference for Timed Automata = $\alpha_{low} \circ \alpha_n([P]) = \alpha_{low} \circ \alpha_L \circ \alpha_n([P])$.

PER model of ANI

[Hunt & Mastroeni ’05]

Partitioning Closure

[Ranzato and Tapparo ’04]
\( \Pi(\eta) \) is the most concrete partitioning closure containing \( \eta! \)

\[
[\eta] P(\rho) \iff [P] : All \times Rel^\eta \rightarrow All \times Rel^\rho \\
[\Pi(\eta)] P(\Pi(\rho))
\]
**PER model of ANI**

\[ \Pi(\eta) \] is the most concrete partitioning closure containing \( \eta \).

\[
\begin{align*}
[\eta] P(\rho) \quad \text{iff} \quad [P] : \text{All} \times \text{Ref}^1 &\rightarrow \text{All} \times \text{Ref}^p \\
\text{iff} \quad [\Pi(\eta)] P(\Pi(\rho))
\end{align*}
\]

\[
\downarrow
\]

\[
[R] P(S) \text{ sse } [P] : \text{All} \times R \rightarrow \text{All} \times S
\]

**ANI vs Robust Declassification**

\[ \text{Zdancewic & Myers '01} \]

The PER model of Abstract Non-interference on the maximal trace semantics is equivalent to the security property introduced for Robust Declassification!
ANI vs Robust Declassification

[Zdancewic & Myers ’01]

6 Let us recall that Declassificazione robusta transform the attacker observational capability in order to derive what the program releases:

\[
\forall \sigma, \sigma' \in \Sigma . \langle \sigma, \sigma' \rangle \in S[\approx] \iff \text{Obs}_\sigma(S, \approx) \equiv \text{Obs}_{\sigma'}(S, \approx)
\]

6 [Banerjee, Giacobazzi and Mastroeni ‘07] This is a backward completeness problem!

Example:

- \langle t, h, p, q, r \rangle \leftrightarrow \langle t, h, p, q, r \rangle
- \langle 0, h, q, q, 0 \rangle \leftrightarrow \langle 1, h, q, q, 1 \rangle
- \langle 0, h, q, q, 1 \rangle \leftrightarrow \langle 1, h, q, q, 0 \rangle
- \langle 0, h, q, 0 \rangle \leftrightarrow \langle 1, h, p, q, 0 \rangle \quad p \neq q
- \langle 0, h, q, 1 \rangle \leftrightarrow \langle 1, h, p, q, 1 \rangle \quad p \neq q

The public variables are \( t, q, r \), hence the partition induced by \( \mathcal{H} \) is:

\[
\langle t, h, p, q, r \rangle \equiv \langle t', h', p', q', r' \rangle \quad \text{iff} \quad t = t' \land q = q' \land r = r'
\]
ANI vs Robust Declassification

[Zdancewic & Myers '01]

Let us recall that **Declassificazione robusta** transform the attacker observational capability in order to derive what the program releases:

$$\forall \sigma, \sigma' \in \sum. \langle \sigma, \sigma' \rangle \in S[\approx] \iff \text{Obs}_\sigma(S, \approx) \equiv \text{Obs}_{\sigma'}(S, \approx)$$

**Example:**

- $\langle t, h, p, q, r \rangle \rightarrow \langle t, h, p, q, r \rangle$
- $\langle 0, h, q, q, 0 \rangle \rightarrow \langle 1, h, q, q, 1 \rangle$
- $\langle 0, h, q, q, 1 \rangle \rightarrow \langle 1, h, q, q, 0 \rangle$
- $\langle 0, h, p, q, 0 \rangle \rightarrow \langle 1, h, p, q, 0 \rangle \quad p \neq q$
- $\langle 0, h, p, q, 1 \rangle \rightarrow \langle 1, h, p, q, 1 \rangle \quad p \neq q$

ANI vs Enforcing Robust Declassification

[Myers et al. '04]

**Language** = **IMP** + **declassify(e)** + {•}

$P[a]$ is the program $P$ under the attack $a$!

$$P[•] \text{ è ROBUSTO se}$$

$$\forall s_1, s_2 \in \sum. \forall a, a' : \left[ [ [P[a]](s_1)^L = [P[a]](s_2)^L \Rightarrow [P[a']]^L(s_1)^L = [P[a']]^L(s_2)^L \right]$$

If $a$ controls only public inputs then it is ANI!

**Example:**

$$P \overset{\text{def}}{=} [\cdot], l := 1 + \text{declassify}(h \ mod\ 3)$$

$$P \text{ satisfies robust declassification!}$$
ANI vs Enforcing Robust Declassification

[Myers et al. ’04]

Language=IMP+\textit{declassify}(e)+[\bullet]

$P[\alpha]$ is the program $P$ under the attack $\alpha$!

$\Downarrow$

$P[\bullet]$ è ROBUSTO se

$\forall s_1, s_2 \in \Sigma. \forall a, a' : \llbracket P[\alpha] \rrbracket(s_1) = \llbracket P[\alpha] \rrbracket(s_2) \Rightarrow \llbracket P[a'] \rrbracket(s_1) = \llbracket P[a'] \rrbracket(s_2)$

If $\alpha$ controls only public inputs then it is ANI!

\textbf{Example:}

$P \overset{\text{def}}{=} [\bullet]_1 : l \assign l + \textit{declassify}(h \mod 3)$

$P$ satisfies \textit{robust declassification}!

$\Downarrow$

Declassified ANI shows that $P$ satisfies \textit{robust declassification} declassifying the MINIMAL amount of information!

Language-based Security: Abstract Non-Interference – p.29/32

ANI vs Enforcing Robust Declassification

[Myers et al. ’04]

Language=IMP+\textit{declassify}(e)+[\bullet]

$P[\alpha]$ is the program $P$ under the attack $\alpha$!

$\Downarrow$

$P[\bullet]$ è ROBUSTO se

$\forall s_1, s_2 \in \Sigma. \forall a, a' : \llbracket P[\alpha] \rrbracket(s_1) = \llbracket P[\alpha] \rrbracket(s_2) \Rightarrow \llbracket P[a'] \rrbracket(s_1) = \llbracket P[a'] \rrbracket(s_2)$

If $\alpha$ controls only public inputs then it is ANI!

\textbf{Example:}

Consider

$P = l : l + (h \mod 3)$

$\Downarrow$

The MAXIMAL amount of information released is

$\phi = \{T, 3\mathbb{Z}, 3\mathbb{Z} + 1, 3\mathbb{Z} + 2, \emptyset\}$. 

Language-based Security: Abstract Non-Interference – p.29/32
ANI vs Delimited release

[Sabelfeld & Myers ’04]

Language=\text{IMP} + \text{declassify}(e)

s_1 \approx_E s_2 \text{ iff } \forall e \in E. \llbracket e \rrbracket(s_1) = \llbracket e \rrbracket(s_2)

\Rightarrow

P \text{ satisfies Delimited Release, } E = \{ e \mid \text{declassify}(e) \text{ in } P \}

\forall s_1, s_2 \in \Sigma. s_1^L = s_2^L \land s_1 \approx_E s_2 \Rightarrow \llbracket P \rrbracket(s_1)^L = \llbracket P \rrbracket(s_2)^L

\text{Example:}

The program P satisfies delimited release while P’ doesn’t:

$$P \overset{\text{def}}{=} \text{if } \text{declassify}(h \geq k) \text{ then } (h := h - k; l := l + k) \text{ else nil}$$

$$\phi_k = (\forall h, \{ h \mid h \geq k \}, \{ h \mid h < k \}, \varnothing)$$

$$\phi = \bigcap_k \phi_k = id$$

Language-based Security: Abstract Non-Interference – p.30/32

ANI vs Delimited release

[Sabelfeld & Myers ’04]

Language=\text{IMP} + \text{declassify}(e)

s_1 \approx_E s_2 \text{ iff } \forall e \in E. \llbracket e \rrbracket(s_1) = \llbracket e \rrbracket(s_2)

\Rightarrow

P \text{ satisfies Delimited Release, } E = \{ e \mid \text{declassify}(e) \text{ in } P \}

\forall s_1, s_2 \in \Sigma. s_1^L = s_2^L \land s_1 \approx_E s_2 \Rightarrow \llbracket P \rrbracket(s_1)^L = \llbracket P \rrbracket(s_2)^L

\text{Example:}

The program P satisfies delimited release while P’ doesn’t:

$$P' \overset{\text{def}}{=} \begin{cases} 
1 := 0; \\
\text{while } n \geq 0 \text{ do} \quad k := 2^{n+1} \\
\quad \text{if } \text{declassify}(h \geq k) \text{ then } (h := h - k; l := l + k) \text{ else nil} \\
n := n - 1 
\end{cases}$$

$$\phi = id$$

Language-based Security: Abstract Non-Interference – p.30/32
ANI vs Relaxed non-interference

[Li & Zdancewic ’05]

Language=λ-calculus (no explicit declassification)

\[ P \text{ satisfies RELAXED NON-INTERFERENCE, if } \]
\[ P \equiv f(n_1 \sigma_1)(n_2 \sigma_2) \ldots (n_k \sigma_k) \]

**Example:**

\[ P \overset{\text{def}}{=} \begin{cases} \begin{array}{l} x := \text{hash}(\text{sec}); y := x \ mod \ 2^{64}; \\ \text{if } y = \text{in then out }= 1 \text{ else out }:= 0; \end{array} \end{cases} \]

\[ \phi_{\text{in}} = \left\{ \begin{array}{l} \forall \text{sec} \mid \text{hash}(\text{sec}) \ mod \ 2^{64} = \text{in} \\ \exists \text{sec} \mid \text{hash}(\text{sec}) \ mod \ 2^{64} \neq \text{in} \end{array} \right\}, \emptyset \]

\[ \phi = \bigcap_{\text{in}} \phi_{\text{in}} = \{\forall \text{sec} \mid \text{hash}(\text{sec}) \ mod \ 2^{64} = \text{in} \} \cup \{2^{64} \mathbb{Z} + n \mid 0 \leq n < 2^{64}\} \]

ANI vs Relaxed non-interference

[Li & Zdancewic ’05]

Language=λ-calculus (no explicit declassification)

\[ P \text{ satisfies RELAXED NON-INTERFERENCE, if } \]
\[ P \equiv f(n_1 \sigma_1)(n_2 \sigma_2) \ldots (n_k \sigma_k) \]

**Example:**

Password check:

\[ P = \lambda \text{in}. \begin{cases} \begin{array}{l} \text{if in }= \sigma_{\text{pw}} \text{ then out }:= 1 \text{ else out }:= 0 \end{array} \end{cases} \]

satisfies relaxed non-interference being equivalent to:

\[ \lambda x. \lambda y : \mathbb{Z} \rightarrow \mathbb{Z}. (\text{if } g(x) \text{ then out }:= 1 \text{ else out }:= 0) \text{in}((\lambda x. \lambda y. x = y)\sigma_{\text{pw}}) \]

⇒ As far as Declassified ANI is concerned the program is not secure : \( \phi = \text{id} \).
Conclusions

WHAT HAVE WE DONE AND WHAT HAVE WE STILL TO DO?

Language-based Security: Abstract Non-Interference – p.32/32