Typing Illegal Information Flow as Program Effects

Ana Almeida Matos[1] José Santos[2]

[1]Instituto de Telecomunicações, Lisbon, Portugal

[2]INRIA Sophia Antipolis Méditerrannée, Antibes, France

April 29, 2012

Problem

How can we?

- Reason about illegal programs:
  - Order illegal programs
- Express arbitrary relaxations of an information flow policy

Problem

How can we?

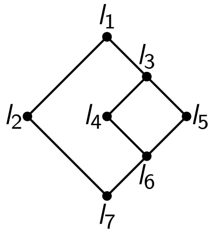
- Reason about illegal programs:
  - Order illegal programs
- Express arbitrary relaxations of an information flow policy

Approach

- Establish a base lattice $\Rightarrow$ strictest information flow policy
- Model illegal flows as kernels over the base lattice
- Assign to each program the strictest kernel that captures all its illegal flows

Flow Kernels

Original Lattice



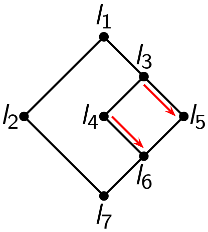
Kernels are computed iteratively

$$\Rsh_{\mathbf{k}} [\mathbf{l_1}, \mathbf{l_2}](l) = \begin{cases} \mathbf{k}(l \sqcap \mathbf{l_2}) & \text{if } l \preccurlyeq \mathbf{l_1} \\ \mathbf{k}(l) & \text{otherwise} \end{cases}$$

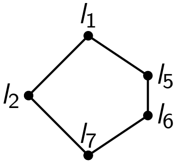
- **k:** original kernel
- $(\mathbf{l_1}, \mathbf{l_2})$**:** new flow

Flow Kernels

Original Lattice



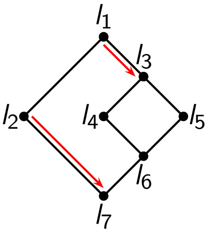
Illegal Flow: $l_3 \to l_5$



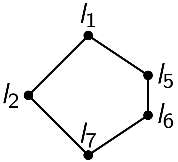
$l_3 \to l_5$

Flow Kernels

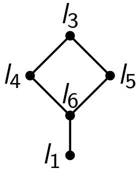
Original Lattice



Illegal Flow: $l_1 \rightarrow l_3$



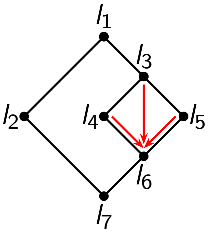
$l_3 \rightarrow l_5$



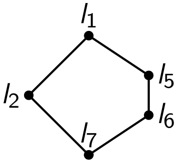
$l_1 \rightarrow l_3$

Flow Kernels

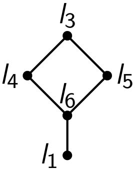
Original Lattice



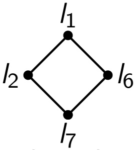
Illegal Flow: $l_3 \rightarrow l_6$



$l_3 \rightarrow l_5$



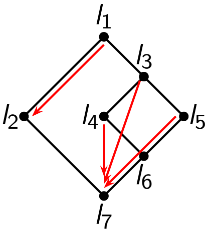
$l_1 \rightarrow l_3$



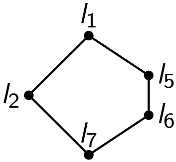
$l_3 \rightarrow l_6$

Flow Kernels

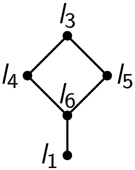
Original Lattice



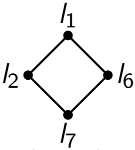
Illegal Flow: $l_1 \to l_2$



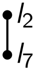
$l_3 \to l_5$



$l_1 \to l_3$



$l_3 \to l_6$



$l_1 \to l_2$

Language

Syntax

- **Expressions:** λ-Calculus $+$ Reference Creation $+$ Thread creation

Language

Syntax

- **Expressions:** λ-Calculus + Reference Creation + Thread creation

Model

- **Model:** $\langle P, S\rangle \rightarrow \langle P', S'\rangle$
- P: **initial** pool of expressions
- S: **initial** memory

Language

Syntax

- **Expressions:** λ-Calculus + Reference Creation + Thread creation

Model

- **Model:** $\langle P, S \rangle \to \langle P', S' \rangle$
- P': **final** pool of expressions
- S': **final** memory

IF Property

I-bissimulation

An **I-bissimulation** is a symmetric relation R on sets of threads such that, for all S_1, S_2, **if:**

- $P_1 R P_2$
- $\langle P_1, S_1 \rangle \rightarrow \langle P_1', S_1' \rangle$
- $S_1 =_I S_2$

Then there exists P_2' and S_2' st:

- $\langle P_2, S_2 \rangle \rightarrow^* \langle P_2', S_2' \rangle$
- $S_1' =_I S_2'$

IF Property

I-bissimulation

An **I-bissimulation** is a symmetric relation R on sets of threads such that, for all S_1, S_2, **if:**

- $P_1 R P_2$
- $\langle P_1, S_1 \rangle \rightarrow \langle P_1', S_1' \rangle$
- $S_1 =_I S_2$

Then there exists P_2' and S_2' st:

- $\langle P_2, S_2 \rangle \rightarrow^* \langle P_2', S_2' \rangle$
- $S_1' =_I S_2'$

$\approx_I$

The largest I-bissimulation.

Relaxing IF Settings

Original IF Setting

- **Lattice:** $\mathcal{L}$
- **Labeling:** Σ

Relaxing IF Settings

Original IF Setting

- **Lattice:** $\mathcal{L}$
- **Labeling:** Σ

$\xrightarrow{\textbf{k}}$

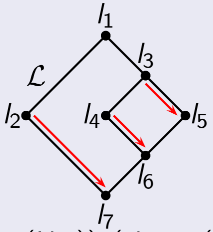
Relaxed IF Setting

- **Lattice:** $k(\mathcal{L})$
- **Labeling:** $k \circ \Sigma$

Relaxing IF Settings

Original IF Setting - Illegal

$$\Sigma = \left\{ \begin{array}{c} a \mapsto l_2, b \mapsto l_3 \\ c \mapsto l_5, d \mapsto l_7 \end{array} \right\}$$

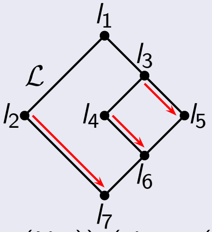


$$((c_{l_5} := (!b_{l_3})); (d_{l_7} := (!a_{l_2})))$$

Relaxing IF Settings

Original IF Setting - Illegal

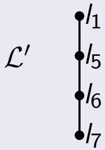
$$\Sigma = \left\{ \begin{array}{c} a \mapsto l_2, b \mapsto l_3 \\ c \mapsto l_5, d \mapsto l_7 \end{array} \right\}$$



$$((c_{l_5} := (!b_{l_3})); (d_{l_7} := (!a_{l_2})))$$

Relaxed IF Setting - Legal

$$\Sigma' = \left\{ \begin{array}{c} a \mapsto l_7, b \mapsto l_5 \\ c \mapsto l_5, d \mapsto l_5 \end{array} \right\}$$



$$((c_{l_5} := (!b_{l_5})); (d_{l_7} := (!a_{l_7})))$$

Information Flow Analysis

Checking Type System

$$\Gamma \vdash^{k}_{\mathcal{L},\Sigma} M : s, \tau$$

M is typable with **type** τ and **security effect** s in the **typing context** Γ with respect to the IF setting $\langle \mathcal{L}, \Sigma, k \rangle$.

Information Flow Analysis

Checking Type System

$$\Gamma \vdash^{k}_{\mathcal{L},\Sigma} M : s, \tau$$

M is typable with **type** τ and **security effect** s in the **typing context** Γ with respect to the IF setting $\langle \mathcal{L}, \Sigma, k \rangle$.

Security Effect - s

- $s.r$: reading effect
- $s.w$: writting effect
- $s.t$: testing effect

Information Flow Analysis

Checking Type System

$$\Gamma \vdash^{k}_{\mathcal{L},\Sigma} M : s, \tau$$

Informative Type System

$$\Gamma \vdash_{\mathcal{L},\Sigma} M : \langle s, s_d \rangle, \tau$$

s_d is the **declassification effect** of M.

Information Flow Analysis

Checking Type System

$$\Gamma \vdash^{k}_{\mathcal{L},\Sigma} M : s, \tau$$

Informative Type System

$$\Gamma \vdash_{\mathcal{L},\Sigma} M : \langle s, s_d \rangle, \tau$$

- Γ: **a map from variables to security levels**

Information Flow Analysis

Checking Type System

$$\Gamma \vdash^{k}_{\mathcal{L},\Sigma} M : s, \tau$$

Informative Type System

$$\Gamma \vdash_{\mathcal{L},\Sigma} M : \langle s, s_d \rangle, \tau$$

- Γ: **a map from variables to security levels**
- $\mathcal{L}$: **Lattice of security levels**

Information Flow Analysis

Checking Type System

$$\Gamma \vdash^{k}_{\mathcal{L},\Sigma} M : s, \tau$$

Informative Type System

$$\Gamma \vdash_{\mathcal{L},\Sigma} M : \langle s, s_d \rangle, \tau$$

- Γ: **a map from variables to security levels**
- $\mathcal{L}$: **lattice of security levels**
- Σ: **a map from references to security levels**

Information Flow Analysis

Checking Type System

$$\Gamma \vdash^{k}_{\mathcal{L},\Sigma} M : s, \tau$$

Informative Type System

$$\Gamma \vdash_{\mathcal{L},\Sigma} M : \langle s, s_d \rangle, \tau$$

- Γ: **a map from variables to security levels**
- $\mathcal{L}$: **lattice of security levels**
- Σ: **a map from references to security levels**
- s: **security effect**

Information Flow Analysis

Checking Type System

$$\Gamma \vdash^{k}_{\mathcal{L},\Sigma} M : s, \tau$$

- k: **parametrizing kernel**

Informative Type System

$$\Gamma \vdash_{\mathcal{L},\Sigma} M : \langle s, s_d \rangle, \tau$$

- s_d: **declassification effect**

- Γ: **a map from variables to security levels**
- $\mathcal{L}$: **lattice of security levels**
- Σ: **a map from references to security levels**
- s: **security effect**

Information Flow Analysis

Checking Type System - Assign Rule

$$\frac{\begin{array}{ll} \Gamma \vdash^{k}_{\mathcal{L},\Sigma} M : s_1, \theta \text{ ref}_l & k(s_1.t) \sqsubseteq k(s_2.w) \\ \Gamma \vdash^{k}_{\mathcal{L},\Sigma} N : s_2, \theta & k(s_1.r), k(s_2.r) \sqsubseteq k(l) \end{array}}{\Gamma \vdash^{k}_{\mathcal{L},\Sigma} M := N : s_1 \sqcup s_2 \sqcup s_l, \text{unit}}$$

Where: $s_l = \langle \bot, k(l), \bot \rangle$

Information Flow Analysis

Checking Type System - Assign Rule

$$\frac{\begin{array}{ll}\Gamma \vdash^{k}_{\mathcal{L},\Sigma} M : s_1, \theta \text{ ref}_l & k(s_1.t) \sqsubseteq k(s_2.w) \\ \Gamma \vdash^{k}_{\mathcal{L},\Sigma} N : s_2, \theta & k(s_1.r), k(s_2.r) \sqsubseteq k(l)\end{array}}{\Gamma \vdash^{k}_{\mathcal{L},\Sigma} M := N : s_1 \sqcup s_2 \sqcup s_l, \text{unit}}$$

Where: $s_l = \langle \bot, k(l), \bot \rangle$

Informative Type System - Assign Rule

$$\frac{\Gamma \vdash_{\mathcal{L},\Sigma} M : \langle s_1, s_1^d \rangle, \theta \text{ ref}_l \quad \Gamma \vdash_{\mathcal{L},\Sigma} N : \langle s_2, s_2^d \rangle, \theta}{\Gamma \vdash_{\mathcal{L},\Sigma} M := N : \langle s, s_d \rangle, \text{unit}}$$

Where:

$s_d = s_1^d \curlyvee s_2^d \curlyvee \Lsh \{(s_1.t, s_2.w), (s_1.r, l), (s_2.r, l)\}$

$s = s_1 \sqcup s_2 \sqcup \langle \bot, l, \bot \rangle$

Information Flow Analysis

Soundness

$$\Gamma \vdash_{\mathcal{L},\Sigma} M : \langle s, s_d \rangle, \tau$$
$$\Downarrow$$
$$\Gamma \vdash^{s_d}_{\mathcal{L},\Sigma} M : s', \tau$$

Information Flow Analysis

Soundness

$$\Gamma \vdash_{\mathcal{L},\Sigma} M : \langle s, s_d \rangle, \tau$$
$$\Downarrow$$
$$\Gamma \vdash^{s_d}_{\mathcal{L},\Sigma} M : s', \tau$$

Optimality

$$\Gamma \vdash^{k}_{\mathcal{L},\Sigma} M : s_1, \tau \quad \textbf{and} \quad \Gamma \vdash_{\mathcal{L},\Sigma} M : \langle s_2, s_d^2 \rangle, \tau$$
$$\Downarrow$$
$$k \sqsubseteq s_d^2$$

Flow Relations as IF Setting Relaxations

Ingredients

- **Set of Principals: Pri**
- **Security levels:** subsets of **Pri**
- **Security lattice:** $\langle \mathcal{P}(\mathbf{Pri}), \supseteq \rangle$
- **Flow Relations:** binary relations on **Pri**
  - $(A, B) \in F$: information may flow from principal A to principal B

Flow Relations as IF Setting Relaxations

Ingredients

- **Set of Principals: Pri**
- **Security levels:** subsets of **Pri**
- **Security lattice:** $\langle \mathcal{P}(\mathbf{Pri}), \supseteq \rangle$
- **Flow Relations:** binary relations on **Pri**

Remark

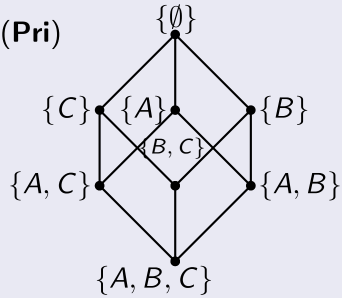
Flow Relations correspond to the co-additive kernels on Pri

Flow Relations as IF Setting Relaxations

Original IF Setting

$$\mathbf{Pri} = \{A, B, C\}$$

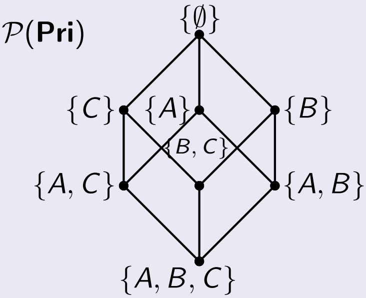
$\mathcal{P}(\mathbf{Pri})$



Flow Relations as IF Setting Relaxations

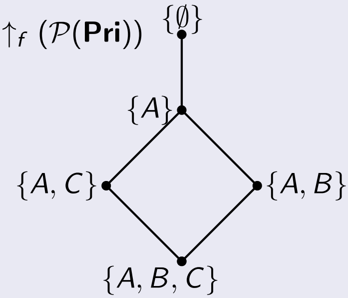
Original IF Setting

$$\mathbf{Pri} = \{A, B, C\}$$



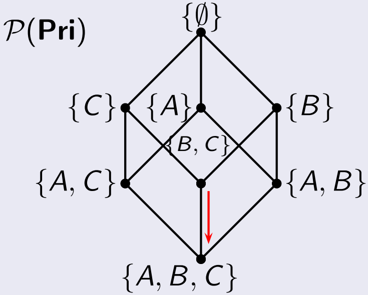
Relaxed IF Setting

$$\mathbf{f} = \{(B, A), (C, A)\}$$



Flow Relations as IF Setting Relaxations

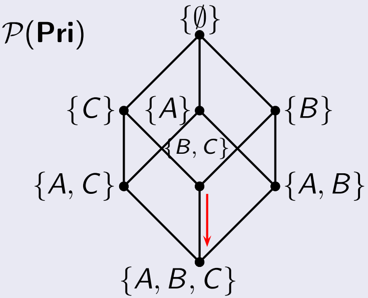
IF Setting



$$(b_{\{A\}} := (!a_{\{B,C\}}))$$

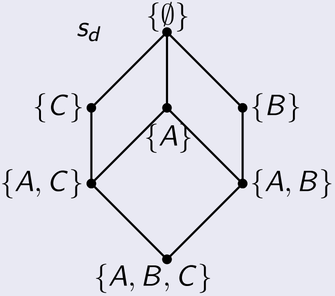
Flow Relations as IF Setting Relaxations

IF Setting



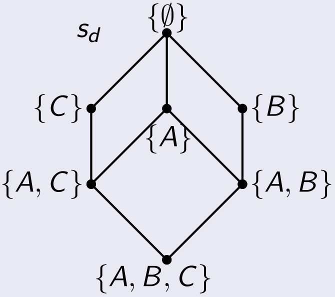
$$(b_{\{A\}} := (!a_{\{B,C\}}))$$

Declassification Effect

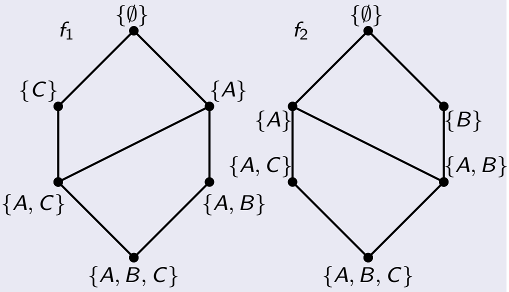


Flow Relations as IF Setting Relaxations

Declassification Effect



Flow Relations below s_d



No optimality result for flow relations!

- $f_1, f_2 \preccurlyeq s_d$ **and** $f_1 \not\preccurlyeq f_2$ **and** $f_2 \not\preccurlyeq f_1$

Permissivity Contexts as Kernels

Ingredients

- Model the **permissivity context** under which a program executes as a kernel
- The permissivity context $\Rightarrow$ Relaxation of the original IF setting
- Permissivity contexts are allowed to change dynamically

Permissivity Contexts as Kernels

Ingredients

- Model the **permissivity context** under which a program executes as a kernel
- The permissivity context $\Rightarrow$ Relaxation of the original IF setting
- Permissivity contexts are allowed to change dynamically

Goal

Only the threads that respect **all** the permissivity contexts that were allowed during the program execution are allowed to terminate.

Permissivity Contexts as Kernels

Approach

- The current permissivity context - k_A - to configurations
- Add a mapping from thread names to their declassification effect - D - to configurations
- When the permissivity context changes remove the threads that are not compliant with it

Configurations

$$\langle P, S, D, k_A \rangle$$

Changing the permissivity context

$$\langle P, S, D, k_A \rangle \rightarrow \langle P', S, D, k_F \rangle$$

Future Work

Study new program constructs to **dynamically** interact with the permissivity context:

- Check if an expression is compliant with the current permissivity context
- Test the current permissivity context
- Kernels as values...

Thank You!

Questions...