Facets of Information Flow Control

Marco Vassena
Complex Software System

Sensitive Data
Complex Software System

Sensitive Data

Devices
Complex Software System

Sensitive Data

Devices

Outputs
Modern software contains many 3rd party components!
Modern software contains many 3rd party components!

App Components

Mutually Distrusting
Modern software contains many 3rd party components!
Modern software contains many 3rd party components!
Modern software contains many 3rd party components!
Modern software contains many 3rd party components!

Data **confidentiality** and **integrity** is at stake
Example

Sign up

Username

Password

Join
Example

Sign up

- Username
- Password
- Join

Untrusted Library

strengthOf(pwd : String)
db.log(pwd)
return STRONG
Example

Sign up

Username
Password
Join

Untrusted Library

strengthOf(pwd : String)
db.log(pwd)
return STRONG

Attacker Controlled Database
Example

Sign up

Username
Password
Join

Untrusted Library

strengthOf(pwd : String)
db.log(pwd)
return STRONG

Attacker Controlled Database

Password leak!
Access Control?

Restrict **access** to sensitive data in untrusted components

**Sign up**

- Username
- Password

**Untrusted Library**

```java
strengthOf(pwd : String)
db.log(pwd)
return STRONG
```

**Attacker Controlled Database**
Access Control?

Restrict **access** to sensitive data in untrusted components

Sign up

- Username
- Password
- Join

Untrusted Library

```
strength0f(pwd : String)
        db.log(pwd)
        return STRONG
```

Attacker Controlled Database
**Access Control?**

Restrict *access* to sensitive data in untrusted components

---

**Untrusted Library**

```scala
strengthOf(pwd : String) 
db.log(pwd) 
return STRONG
```

- **Sign up**
  - Username
  - Password
  - Join

- **Legitimate need to access the password**

---

**Attacker Controlled Database**
Access Control?

Restrict **access** to sensitive data in untrusted components

Sign up

Username

Password

Join

Untrusted Library

`strength0f(pwd : String)`

`db.log(pwd)`

return STRONG

Legitimate need to access the password

Attacker Controlled Database

This is the leak!
Access Control?
Restrict **access** to sensitive data in untrusted components

Sign up

- **Username**
- **Password**
- **Join**

**Untrusted Library**

- `strengthOf(pwd : String)`
- `db.log(pwd)`
- `return STRONG`

**Attacker Controlled Database**

*This is the leak!*
Information Flow Control

Do not restrict data access, restrict **where** data can flow!

- **Untrusted Library**
  ```scala
  strengthOf(pwd : String)
  db.log(pwd)
  return STRONG
  ```

- **Attacker Controlled Database**

**Sign up**
- Username
- Password
- Join
Information Flow Control

Do not restrict data access, restrict where data can flow!

Sign up
- Username
- Password
- Join

Untrusted Library
- strengthOf(pwd : String)
- db.log(pwd)
- return STRONG

Attacker Controlled Database

Track data flows across program components
Information Flow Control

Do not restrict data access, restrict **where** data can flow!

Sign up

- Username
- Password
- Join

Untrusted Library

- `strengthOf(pwd : String)`
- `db.log(pwd)`
- `return STRONG`

Detect and suppress information leakage

Attacker Controlled Database

Track data flows across program components
Facets of Language-based IFC

Associate data with security levels to track data flows in programs
Facets of Language-based IFC

Associate data with security levels to track data flows in programs

“Public” and “Secret”
Facets of Language-based IFC

Associate data with **security levels** to track data flows in programs

"Public" and "Secret"

- Static
- Hybrid
- Dynamic

Tracking
Facets of Language-based IFC

Associate data with security levels to track data flows in programs

Tracking

Static  Hybrid  Dynamic

“Public” and “Secret”
Facets of Language-based IFC

Associate data with **security levels** to track data flows in programs.
Facets of Language-based IFC

"Public" and "Secret"

Associate data with **security levels** to track data flows in programs

Conservative

Static

Hybrid

Dynamic

Runtime Overhead

Tracking
Facets of Language-based IFC

Associate data with security levels to track data flows in programs.

- **Conservative**
  - Static
  - Hybrid
  - Dynamic

- **Granularity of data flows**
  - Fine-grained
  - Coarse-grained

- **“Public” and “Secret”**

- **Runtime Overhead**
Facets of Language-based IFC

Associate data with **security levels** to track data flows in programs

- **Static**
- **Hybrid**
- **Dynamic**

**Tracking**

- **Conservative**
- **Runtime Overhead**

**Granularity of data flows**

- **Per variable**
- **Fine-grained**
- **Coarse-grained**

“Public” and “Secret”
Facets of Language-based IFC

Associate data with **security levels** to track data flows in programs.

**Tracking**
- **Conservative**
  - *Static*
  - *Hybrid*
  - *Dynamic*

**Runtime Overhead**

**Granularity of data flows**
- **Per variable**
  - *Fine-grained*
- **Per computation**
  - *Coarse-grained*

- "Public" and "Secret"
Plan

Overview of different language-based IFC approaches

- Non Interference
Plan

Overview of different language-based IFC approaches

• Non Interference
Plan

Overview of different language-based IFC approaches

- Non Interference
- 4 IFC Languages
Plan

Overview of different language-based IFC approaches

- Non Interference
- 4 IFC Languages

<table>
<thead>
<tr>
<th></th>
<th>Static</th>
<th>Dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fine-grained</td>
<td>λSFG</td>
<td>λDFG</td>
</tr>
<tr>
<td>Coarse-grained</td>
<td>λSCG</td>
<td>λDCG</td>
</tr>
</tbody>
</table>

Confidentiality & Integrity
Security Policy

*Information flow policies are specified by the security lattice*
Security Policy

Which data flows are allowed

Information flow policies are specified by the security lattice
Security Policy

Information flow policies are specified by the security lattice

Simple lattice for confidentiality:

- Secret
- Public

Which data flows are allowed
Security Policy

Information flow policies are specified by the security lattice.

Simple lattice for *confidentiality*:

- **Public** and **Secret** are security labels.
- Data flows are allowed from **Public** to **Secret**.
Security Policy

Which data flows are allowed

Information flow policies are specified by the security lattice

Simple lattice for confidentiality:

Public and Secret are security labels

“Secret inputs cannot flow to Public outputs”
Security Policy

Which data flows are allowed

Information flow policies are specified by the security lattice

Simple lattice for **confidentiality**:

```
Public and Secret are security labels

Secret

2-point lattice

Public

"Secret inputs cannot flow to Public outputs"
```
Simple lattice for **confidentiality**:

```
Secret

↑

Public

“Secret inputs cannot flow to Public outputs”
```

Formally:

\[
\mathcal{L}^C = ( \{ P, S \}, \sqsubseteq^C, \sqcup^C )
\]
Simple lattice for **confidentiality**: 

```
\begin{align*}
\text{Secret} & \quad \uparrow \\
\text{Public} & \\
\end{align*}
```

“Secret inputs **cannot** flow to Public outputs”

Formally: 

Partial order between labels

\[
L^C = ( \{ P, S \}, \sqsubseteq^C, \sqcup^C )
\]
Simple lattice for **confidentiality**:

Secret

\[ \sqsubseteq^C \]

Public

"**Secret inputs cannot** flow to **Public outputs**"

Formally:

Partial order between labels

\[ L^C = ( \{ P, S \}, \sqsubseteq^C, \sqcup^C ) \]
Simple lattice for **confidentiality**:

```
<table>
<thead>
<tr>
<th>Secret</th>
<th>Public</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

“Secret inputs **cannot** flow to Public outputs”

Formally:

Partial order between labels

\[ \mathcal{L}^C = ( \{ P, S \} , \sqsubseteq^C , \sqcup^C ) \]

where

\[
\begin{align*}
P & \sqsubseteq^C P \\
P & \sqsubseteq^C S \\
S & \sqsubseteq^C S \\
S & \not\sqsubseteq^C P
\end{align*}
\]
Simple lattice for **confidentiality**: 

```
  Secret
   ▲
   ⊑ C
  Public
```

"Secret inputs cannot flow to Public outputs"

Formally:

\[ L^C = ( \{ P, S \}, \sqsubseteq^C, \sqcup^C ) \]
Simple lattice for confidentiality:

“Secret inputs cannot flow to Public outputs”

Formally:

Join Operator (least upper bound)

\[ \mathcal{L}^C = ( \{ P, S \}, \sqsubseteq^C, \sqcup^C ) \]

where

\begin{align*}
P \sqcup^C P &= P \\
S \sqcup^C S &= S \\
P \sqcup^C S &= S \\
S \sqcup^C P &= S
\end{align*}
“Dual” lattice for \textit{integrity}:

Untrusted \hspace{1cm} \sqsubseteq^I \hspace{1cm} \text{Trusted}

“\textbf{Untrusted} inputs \textit{cannot} flow to \textbf{Trusted} outputs”
“Dual” lattice for integrity:

Untrusted

\[ \sqsubseteq^I \]

Trusted

“Untrusted inputs cannot flow to Trusted outputs”

Formally:

\[ L^I = (\{T, U\}, \sqsubseteq^I, \sqcup^I) \]
“Dual” lattice for **integrity**:

```
Untrusted

\[ \subseteq^I \]

Trusted

“**Untrusted** inputs **cannot** flow to **Trusted** outputs”
```

Formally:

```
\mathcal{L}^I = ( \{ T, U \} , \subseteq^I , \sqcup^I )
```

where

```
T \subseteq^I T  \quad  U \subseteq^I U
T \subseteq^I U  \quad  U \nsubseteq^I T
```
“Dual” lattice for integrity:

Untrusted

\[ \sqsubseteq^I \]

Trusted

“Untrusted inputs cannot flow to Trusted outputs”

Formally:

\[ \mathcal{L}^I = ( \{ T, U \} , \sqsubseteq^I , \sqcup^I ) \]

where

\[ T \sqcup^I T = T \]
\[ U \sqcup^I U = U \]
\[ T \sqcup^I U = U \]
\[ U \sqcup^I P = U \]
Secret
↑ \subseteq^C
Public

Untrusted
↑ \subseteq^I
Trusted
Simple lattice for **confidentiality** and **integrity**:
Simple lattice for **confidentiality** and **integrity**:

- (Secret, Untrusted)
- (Secret, Trusted)
- (Public, Untrusted)
- (Public, Trusted)
Simple lattice for confidentiality and integrity:

\[ (\text{Secret, Untrusted}) \quad \text{Restricted usage} \]

\[ (\text{Secret, Trusted}) \]

\[ (\text{Public, Untrusted}) \]

\[ (\text{Public, Trusted}) \]
Simple lattice for *confidentiality* and *integrity*:

- **(Secret, Untrusted)**: Restricted usage
- **(Secret, Trusted)**
- **(Public, Untrusted)**
- **(Public, Trusted)**: Unrestricted usage
Simple lattice for **confidentiality** and **integrity**: 

- (Secret, Untrusted)
- (Secret, Trusted)
- (Public, Untrusted)
- (Public, Trusted)
Simple lattice for confidentiality and integrity:

\[ \mathcal{L}^{CI} = (\{P,S\} \times \{T,U\}, \sqsubseteq^C \times \sqsubseteq^I, \sqcup^C \times \sqcup^I) \]
Simple lattice for **confidentiality** and **integrity**:

\[
\mathcal{L}^{CI} = ( \{ P, S \} \times \{ T, U \}, \sqsubseteq^C \times \sqsubseteq^I, \sqcup^C \times \sqcup^I )
\]

Notice

\[(S, T) \not\sqsubseteq^{CI} (P, U) \quad (P, U) \not\sqsubseteq^{CI} (S, T)\]
Simple lattice for **confidentiality** and **integrity**:

\[ \mathcal{L}^{CI} = ( \{ P, S \} \times \{ T, U \}, \sqsubseteq^C \times \sqsubseteq^I, \sqcup^C \times \sqcup^I) \]

Notice \( (S, T) \not\sqsubseteq^{CI} (P, U) \) \( (P, U) \not\sqsubseteq^{CI} (S, T) \)
Simple lattice for **confidentiality** and **integrity**:

\[
(\text{Secret, Untrusted}) \\
(\text{Secret, Trusted}) \\
(\text{Public, Untrusted}) \\
(\text{Public, Trusted})
\]

Formally:

\[
\mathcal{L}^{CI} = (\{P, S\} \times \{T, U\}, \sqsubseteq^C \times \sqsubseteq^I, \sqcup^C \times \sqcup^I)
\]

Notice

\[(S, T) \sqcup^CI (P, U)\]
Simple lattice for **confidentiality** and **integrity**:

\[
\mathcal{L}^{CI} = (\{P, S\} \times \{T, U\}, \sqsubseteq^C \times \sqsubseteq^I, \sqcup^C \times \sqcup^I)
\]

Formally:

\[
(S, T) \sqcup^{CI} (P, U) = (S \sqcup^C P, T \sqcup^I U)
\]

Notice
Simple lattice for confidentiality and integrity:

( Secret, Untrusted )

( Secret, Trusted )  ( Public, Untrusted )

( Public, Trusted )

Formally:

\[ \mathcal{L}^{CI} = ( \{P,S\} \times \{T,U\} , \sqcap^C \times \sqcap^I , \sqcup^C \times \sqcup^I ) \]

Notice

\[ (S, T) \sqcup^{CI} (P, U) = (S \sqcup^C P , T \sqcup^I U) = (S , U) \]
General lattice for principals $P$: 
General lattice for principals $P$:

$$P = \{\text{Alice, Bob, Charlie}\}$$
General lattice for principals $P$: $P = \{\text{Alice, Bob, Charlie}\}$
General lattice for principals $P$: $P = \{\text{Alice, Bob, Charlie}\}$
General lattice for principals $P$: $P = \{Alice, Bob, Charlie\}$
General lattice for principals $P$:

$P = \{\text{Alice, Bob, Charlie}\}$
General lattice for principals $P$: 

$P = \{\text{Alice, Bob, Charlie}\}$

Formally:

$L^P = (\mathcal{P}(P), \subseteq, \cup)$
General lattice for principals $P$: $P = \{\text{Alice, Bob, Charlie}\}$

Formally: $\mathcal{L}^P = (\mathcal{P}(P), \subseteq, \cup)$
In general we work with an **abstract lattice** with standard properties

\[ \mathcal{L} = (L, \sqsubseteq, \sqcup) \]

\(\sqsubseteq\) is reflexive, transitive, and antisymmetric.

\(\sqcup\) is idempotent, commutative, and associative.
In general we work with an abstract lattice with standard properties

$$\mathcal{L} = (L, \sqsubseteq, \sqcup)$$

$\sqsubseteq$ is reflexive, transitive, and antisymmetric.

$\sqcup$ is idempotent, commutative, and associative.

$\bot$ element:
In general we work with an **abstract lattice** with standard properties

\[ \mathcal{L} = ( L, \sqsubseteq, \sqcup ) \]

\( \sqsubseteq \) is reflexive, transitive, and antisymmetric.

\( \sqcup \) is idempotent, commutative, and associative.

**Bottom of the lattice**

\( \bot \) element:
In general we work with an abstract lattice with standard properties

\[ L = (L, \sqsubseteq, \sqcup) \]

\( \sqsubseteq \) is reflexive, transitive, and antisymmetric.

\( \sqcup \) is idempotent, commutative, and associative.

\( \bot \) element:
\[ \forall \ell. \ \bot \sqsubseteq \ell \ \land \ \bot \sqcup \ell = \ell \]
In general we work with an abstract lattice with standard properties

\[ \mathcal{L} = ( \mathcal{L}, \sqsubseteq, \sqcup ) \]

\( \sqsubseteq \) is reflexive, transitive, and antisymmetric.

\( \sqcup \) is idempotent, commutative, and associative.

\( \bot \) element:

\[ \forall \ell. \quad \bot \sqsubseteq \ell \land \bot \sqcup \ell = \ell \]

\[ \forall \ell_1 \ell_2 \ell_3. \quad \ell_1 \sqsubseteq \ell_1 \sqcup \ell_2 \land \ell_2 \sqsubseteq \ell_1 \sqcup \ell_2 \]
In general we work with an abstract lattice with standard properties

\[ \mathcal{L} = (L, \sqsubseteq, \sqcup) \]

\( \sqsubseteq \) is reflexive, transitive, and antisymmetric.

\( \sqcup \) is idempotent, commutative, and associative.

\( \perp \) element:

\[ \forall \ell. \perp \sqsubseteq \ell \land \perp \sqcup \ell = \ell \]

Join and partial order “agree”

\[ \forall \ell_1, \ell_2, \ell_3. \ell_1 \sqsubseteq \ell_1 \sqcup \ell_2 \land \ell_2 \sqsubseteq \ell_1 \sqcup \ell_2 \]
Non-Interference

Public outputs must not depend on secret inputs.
Non-Interference

Public outputs must not depend on secret inputs.
Non-Interference

Public outputs must not depend on secret inputs.

Secret Input

Adversarial Program

Secret Output

Public Input

Public Output
Quiz

Do the following programs satisfy non-interference?

\[ h := \text{input}^H() \]
\[ l := \text{input}^L() \]
\[ \text{output}^H(l + h) \]
Quiz

Do the following programs satisfy non-interference?

\[
\begin{align*}
h &:= \text{input}^H() \\
l &:= \text{input}^L() \\
\text{output}^H(l + h) 
\end{align*}
\]
Quiz

Do the following programs satisfy non-interference?

\[
\begin{align*}
    h &:= \text{input}^H() \\
    l &:= \text{input}^L() \\
    \text{output}^H(l + h)
\end{align*}
\]

Public and secret data can flow to secret outputs
Quiz

Do the following programs satisfy non-interference?

h := \text{input}^H() \quad \text{output}^H(l + h)

l := \text{input}^L() \quad \text{output}^L(h + 1)

Public and secret data can flow to secret outputs
Quiz

Do the following programs satisfy non-interference?

\[
\begin{align*}
    &h := \text{input}^H() \\
    &l := \text{input}^L() \\
    &\text{output}^H(l + h) \\
\end{align*}
\]

\[
\begin{align*}
    &h := \text{input}^H() \\
    &\text{output}^L(h + 1) \\
\end{align*}
\]
Quiz

Do the following programs satisfy non-interference?

\[
\begin{align*}
    h & := \text{input}^H() \\
    l & := \text{input}^L() \\
    \text{output}^H(l + h)
\end{align*}
\]

Public and secret data can flow to secret outputs.

\[
\begin{align*}
    h & := \text{input}^H() \\
    \text{output}^L(h + 1)
\end{align*}
\]

Secret data must not flow to public outputs.
Quiz

Do the following programs satisfy non-interference?

**Top Program**

\[
\begin{align*}
\text{h} & := \text{input}^H() \\
\text{l} & := \text{input}^L() \\
\text{output}^H(\text{l} + \text{h})
\end{align*}
\]

*Public and secret data can flow to secret outputs*

This program satisfies non-interference.

**Bottom Program**

\[
\begin{align*}
\text{h} & := \text{input}^H() \\
\text{output}^L(\text{h} + 1)
\end{align*}
\]

*Secret data must not flow to public outputs*

This program violates non-interference. This is an example of an explicit flow.
Quiz

Do the following programs satisfy non-interference?

\[
\begin{align*}
h & := \text{input}^H() \\
\text{if} \ h & \\
\text{output}^L(0)
\end{align*}
\]
Do the following programs satisfy non-interference?

\[
\begin{align*}
    h & := \text{input}^H(()) \\
    \text{if } h \\
    & \quad \text{output}^L(0)
\end{align*}
\]
Quiz

Do the following programs satisfy non-interference?

\[
\begin{align*}
  h & := \text{input}^H() \\
  \text{if } h \\
  \text{output}^L(0)
\end{align*}
\]

The presence of a public output leaks information about the secret.
Quiz

Do the following programs satisfy non-interference?

\[ h := \text{input}^H() \]
\[ \text{if } h \]
\[ \text{output}^L(0) \]

The presence of a public output leaks information about the secret.

This is an example of an \textit{implicit flow}. 
Quiz

Do the following programs satisfy non-interference?

h := \text{input}^H() \\
\text{if } h \\
\text{output}^L(0)

The presence of a public output leaks information about the secret

This is an example of an \textit{implicit flow}

h := \text{input}^H() \\
\text{output}^L(h - h)
Quiz

Do the following programs satisfy non-interference?

\[
\begin{align*}
  h &:= \text{input}^H() \\
  \text{if } h &\\
  \text{output}^L(0)
\end{align*}
\]

The presence of a public output leaks information about the secret.

This is an example of an implicit flow.

\[
\begin{align*}
  h &:= \text{input}^H() \\
  \text{output}^L(h - h)
\end{align*}
\]

This program satisfies non-interference.
Quiz

Do the following programs satisfy non-interference?

\[
\begin{align*}
\text{h := } & \text{input}^H() \\
\text{if h} & \\
\text{output}^L(0) & \\
\end{align*}
\]

The presence of a public output leaks information about the secret

This is an example of an implicit flow

\[
\begin{align*}
\text{h := } & \text{input}^H() \\
\text{output}^L(h - h) & \\
\end{align*}
\]
Do the following programs satisfy non-interference?

```
\begin{align*}
  h & := \text{input}^H() \\
  & \text{if } h \\
  & \quad \text{output}^L(0)
\end{align*}
```

The presence of a public output leaks information about the secret. This is an example of an implicit flow. **Wrong**

```
\begin{align*}
  h & := \text{input}^H() \\
  & \text{output}^L(h - h)
\end{align*}
```

Equivalent to

```
\begin{align*}
  h & := \text{input}^H() \\
  & \quad \text{output}^L(0)
\end{align*}
```

Most IFC languages reject this program. **Correct**
Quiz

Do the following programs satisfy non-interference?

\[
\begin{align*}
    h & := \text{input}^H() \\
    \text{if } h & \\
    \text{output}^L(0)
\end{align*}
\]

This is an example of an \textit{implicit flow}

\[
\begin{align*}
    h & := \text{input}^H() \\
    \text{output}^L(h - h)
\end{align*}
\]

\text{equivalent to}

\[
\begin{align*}
    h & := \text{input}^H() \\
    \text{output}^L(0)
\end{align*}
\]

\text{Most IFC languages reject this program}

\text{False positive}
Outline

Overview of different language-based IFC approaches

• Non Interference

• 4 IFC Languages

<table>
<thead>
<tr>
<th></th>
<th>Static</th>
<th>Dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fine-grained</td>
<td>$\lambda_{SFG}$</td>
<td>$\lambda_{DFG}$</td>
</tr>
<tr>
<td>Coarse-grained</td>
<td>$\lambda_{SCG}$</td>
<td>$\lambda_{DCG}$</td>
</tr>
</tbody>
</table>
Outline

Overview of different language-based IFC approaches

- Non Interference

- 4 IFC Languages

<table>
<thead>
<tr>
<th></th>
<th>Static</th>
<th>Dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fine-grained</td>
<td>$\lambda_{SFG}$</td>
<td>$\lambda_{DFG}$</td>
</tr>
<tr>
<td>Coarse-grained</td>
<td>$\lambda_{SCG}$</td>
<td>$\lambda_{DCG}$</td>
</tr>
</tbody>
</table>
Static Fine-grained IFC

Syntax
Static Fine-grained IFC

Syntax

Labeled Types \( \tau ::= s^\ell \)

Simple Types \( s ::= \text{unit} \mid \tau \to \tau \mid \tau + \tau \mid \tau \times \tau \)
Static Fine-grained IFC

Syntax

Labeled Types \( \tau ::= s^\ell \)

Simple Types \( s ::= \text{unit} \mid \tau \rightarrow \tau \mid \tau + \tau \mid \tau \times \tau \)
Static Fine-grained IFC

Syntax

Label annotation used in IFC type-system

Labeled Types \( \tau ::= s^\ell \)

Simple Types \( s ::= \text{unit} \mid \tau \to \tau \mid \tau + \tau \mid \tau \times \tau \)

Expressions \( e ::= () \mid x \mid \lambda x.e \mid e \ e \)

\mid (e, e) \mid \text{fst}(e) \mid \text{snd}(e)

\mid \text{inl}(e) \mid \text{inr}(e) \mid \text{case}(e, x.e, x.e) \)
Static Fine-grained IFC

Syntax

Labeled Types
\[ \tau ::= s^\ell \]

Simple Types
\[ s ::= \text{unit} \mid \tau \to \tau \mid \tau + \tau \mid \tau \times \tau \]

Expressions
\[ e ::= () \mid x \mid \lambda x. e \mid e \cdot e \]
\[ \mid (e , e) \mid \text{fst}(e) \mid \text{snd}(e) \]
\[ \mid \text{inl}(e) \mid \text{inr}(e) \mid \text{case}(e, x.e, x.e) \]

Values
\[ v ::= () \mid (x.e , \theta) \mid (v, v) \mid \text{inl}(v) \mid \text{inr}(v) \]

Environments
\[ \theta \in \text{Var} \to \text{Value} \]
Static Fine-grained IFC

Syntax

Labeled Types \( \tau ::= s^\ell \)

Simple Types \( s ::= \text{unit} | \tau \rightarrow \tau | \tau + \tau | \tau \times \tau \)

Expressions \( e ::= () | x | \lambda x.e | e\ e | (e,\ e) | \text{fst}(e) | \text{snd}(e) | \text{inl}(e) | \text{inr}(e) | \text{case}(e, x.e, x.e) \)

Values \( v ::= () | (x.e, \theta) | (v, v) | \text{inl}(v) | \text{inr}(v) \)

Environments \( \theta \in \text{Var} \rightarrow \text{Value} \)
Dynamic Semantics  \( e \downarrow^\theta v \)
Dynamic Semantics  \[ e \downarrow^\theta v \]

Standard: no security checks!
Dynamic Semantics \[ e \downarrow^\theta v \]

Static Semantics
\[
\Gamma \vdash e : \tau \quad where \quad \Gamma \in \text{Var} \rightarrow \text{LTypes}
\]
Dynamic Semantics \[ e \downarrow^\theta v \]

Static Semantics \[ \Gamma \vdash e : \tau \quad \text{where} \quad \Gamma \in \text{Var} \rightarrow \text{LTypes} \]

Standard: no security checks!

Well-typed programs are secure
Dynamic Semantics \[ e \downarrow_{\theta} v \]

Static Semantics \[ \Gamma \vdash e : \tau \quad \text{where} \quad \Gamma \in \text{Var} \rightarrow \text{LTypes} \]

**Exercise.** Prove that the following program is *ill-typed*:

\[ \Gamma \not\vdash \text{if } h \text{ then } l_1 \text{ else } l_2 : \text{Bool}^L \]

with typing environment

\[ \Gamma = [ h \mapsto \text{Bool}^H, \ l_1 \mapsto \text{Bool}^L, \ l_2 \mapsto \text{Bool}^L ] \]

**Well-typed program are secure**

**Standard: no security checks!**
Dynamic Semantics \[ e \Downarrow \theta \land \theta \]

Static Semantics

\[ \Gamma \vdash e : \tau \quad \text{where} \quad \Gamma \in \text{Var} \rightarrow \text{LTypes} \]

Exercise. **Prove that the following program is ill-typed:**

\[ \Gamma \not\vdash \text{if } h \text{ then } l_1 \text{ else } l_2 : \text{Bool}^L \]

with typing environment

\[ \Gamma = [ h \mapsto \text{Bool}^H, \ l_1 \mapsto \text{Bool}^L, \ l_2 \mapsto \text{Bool}^L ] \]

where \[ \text{Bool}^\ell \triangleq (\text{unit}^L + \text{unit}^L)^\ell \]

\[ \text{if } e \text{ then } e_1 \text{ else } e_2 \triangleq \text{case}(e, \_ . e_1, \_ . e_2) \]
Dynamic Semantics \[ e \downarrow \theta \nu \]

Static Semantics \[ \Gamma \vdash e : \tau \quad \text{where} \quad \Gamma \in \text{Var} \rightarrow \text{LTypes} \]

**Exercise.** Prove that the following program is **ill-typed**:

\[ \Gamma \not\vdash \text{if } h \text{ then } l_1 \text{ else } l_2 : \text{Bool}^L \]

with typing environment

\[ \Gamma = [ h \mapsto \text{Bool}^H, \ l_1 \mapsto \text{Bool}^L, \ l_2 \mapsto \text{Bool}^L ] \]

where \[ \text{Bool}^L \triangleq (\text{unit}^L + \text{unit}^L)^L \]

\[ \text{if } e \text{ then } e_1 \text{ else } e_2 \triangleq \text{case}(e, \_e_1, \_e_2) \]
Static Semantics

\[ \Gamma \vdash e : \tau \quad where \quad \Gamma \in Var \rightarrow LTypes \]
Observations & Remarks

Elimination rules include security checks
Static Semantics

\[ \Gamma \vdash e : \tau \quad \text{where} \quad \Gamma \in \text{Var} \rightarrow \text{LTypes} \]

Observations & Remarks

Elimination rules include security checks

Avoid implicit leaks through the result
Static Semantics

\[ \Gamma \vdash e : \tau \quad \text{where} \quad \Gamma \in \text{Var} \rightarrow \text{LTypes} \]

Observations & Remarks

Elimination rules include security checks

Introduction rules only generate label \( \bot \)

Avoid implicit leaks through the result
Observations & Remarks

Elimination rules include security checks

Introduction rules only generate label $\perp$

Avoid implicit leaks through the result

Can be increased via subtyping

Static Semantics

$\Gamma \vdash e : \tau$ where $\Gamma \in \text{Var} \rightarrow \text{LTypes}$
Observations & Remarks

Elimination rules include security checks

Introduction rules only generate label ⊥

Avoid implicit leaks through the result

Can be increased via subtyping

To state and prove non-interference we also need:
Observations & Remarks

Elimination rules include security checks

Introduction rules only generate label \( \bot \)

To state and prove non-interference we also need:

\[ \Gamma \vdash v : \tau \]
Observations & Remarks

Elimination rules include security checks

Introduction rules only generate label $\bot$

Can be increased via subtyping

Avoid implicit leaks through the result

To state and prove non-interference we also need:

$\vdash v : \tau$

Similar to the intro rules for expressions
Observations & Remarks

Elimination rules include security checks

Introduction rules only generate label $\bot$

To state and prove non-interference we also need:

- $\Gamma \vdash \nu : \tau$
- $\Gamma \vdash \theta : \Gamma$

Environment and typing contexts “agree”

Avoid implicit leaks through the result

Can be increased via subtyping

Similar to the intro rules for expressions
Subtyping Relation

\[ \tau <: \tau \]

\[ \ell_1 \sqsubseteq \ell_2 \quad S_1 <: S_2 \]

\[ S_1 \ell_1 <: S_2 \ell_2 \]

[Sub-LType]
Subtyping Relation

\[ \tau <: \tau \]

\[ \ell_1 \sqsubseteq \ell_2 \quad s_1 <: s_2 \quad [\text{Sub-LType}] \]

\[ s_1 \ell_1 <: s_2 \ell_2 \]

\[ s <: s \]

\[ \text{unit} <: \text{unit} \quad [\text{Sub-Unit}] \]
Subtyping Relation

\[ \tau <:: \tau \]

\[ \ell_1 \subseteq \ell_2 \quad s_1 <:: s_2 \]

\[ s_1 \ell_1 <:: s_2 \ell_2 \]

[Sub-LType]

\[ s <:: s \]

\[ \text{unit} <:: \text{unit} \]

[Sub-Unit]

\[ \oplus \in \{+,\times\} \quad i \in \{1,2\} \quad \tau_i <:: \tau_i' \]

\[ \tau_1 \oplus \tau_2 <:: \tau_1' \oplus \tau_2' \]
Subtyping Relation

\[ \tau <: \tau \]

\[ \ell_1 \subseteq \ell_2 \quad S_1 <: S_2 \]

\[ S_1 \ell_1 <: S_2 \ell_2 \]

[Sub-LType]

\[ S <: S \]

\[ \text{unit} <: \text{unit} \]

[Sub-Unit]

\[ \oplus \in \{+, \times\} \]

\[ i \in \{1, 2\} \quad \tau_i <: \tau_i' \]

\[ \tau_1 \oplus \tau_2 <: \tau_1' \oplus \tau_2' \]

[Sub-Sum]

[Sub-Pair]

\[ \text{Structural for sums and pairs} \]
Subtyping Relation

\[ \begin{align*}
\tau &<: \tau \\
\ell_1 &\subseteq \ell_2 \quad s_1 <: s_2 \\
s_1 \ell_1 &<: s_2 \ell_2
\end{align*} \] [Sub-LType]

\[ \begin{align*}
s &<: s \\
\oplus &\in \{+, \times\} \\
\text{unit} &<: \text{unit}
\end{align*} \] [Sub-Unit]

\[ \begin{align*}
&\quad i \in \{1, 2\} \quad \tau_i <: \tau_i' \\
\tau_1 \oplus \tau_2 &<: \tau_1' \oplus \tau_2'
\end{align*} \] [Sub-Sum]

\[ \begin{align*}
\tau_1' &<: \tau_1 \quad \tau_2' <: \tau_2' \\
\tau_1 \rightarrow \tau_2 &<: \tau_1' \rightarrow \tau_2'
\end{align*} \] [Sub-Pair]
Subtyping Relation

\[ \tau < : \tau \]

\[
\begin{align*}
\ell_1 & \subseteq \ell_2 \quad S_1 < : S_2 \\
S_1 \ell_1 & \subseteq S_2 \ell_2
\end{align*}
\]  
[Sub-LType]

\[ s < : s \]

\[
\begin{align*}
\text{unit} & < : \text{unit} \\
\oplus & \in \{+, \times\} \quad \begin{align*}
i & \in \{1, 2\} \\
\tau_i & < : \tau_i'
\end{align*} \quad \begin{align*}
\tau_1 \oplus \tau_2 & < : \tau_1' \oplus \tau_2' \\
\tau_1' & < : \tau_1 \quad \tau_2' & < : \tau_2'
\end{align*} \\
\tau_1 \rightarrow \tau_2 & < : \tau_1' \rightarrow \tau_2'
\end{align*}
\]  
[Sub-Unit]  
[Sub-Sum]  
[Sub-Pair]

Covariant in the result
Subtyping Relation

\[ \tau <: \tau \]

\[ \ell_1 \subseteq \ell_2 \quad s_1 <: s_2 \]

\[ s_1 \ell_1 <: s_2 \ell_2 \]

[Sub-LType]

\[ \text{unit} <: \text{unit} \]

[Sub-Unit]

\[ \oplus \in \{+\times\} \]

\[ i \in \{1,2\} \]

\[ \tau_i <: \tau_i' \]

\[ \tau_1 \oplus \tau_2 <: \tau_1' \oplus \tau_2' \]

[Sub-Sum]

[Sub-Pair]

Contravariant in the argument

Covariant in the result
\( \tau <: \tau \)

\[
\begin{align*}
\ell_1 & \sqsubseteq \ell_2 \\
S_1 & <: S_2 \\
S_1 \ell_1 & <: S_2 \ell_2
\end{align*}
\]

[Sub-LType]

\( s <: s \)

\[
\begin{align*}
\text{unit} & <: \text{unit} \\
\oplus & \in \{+, \times\} \\
\tau_i & <: \tau_i' \\
\tau_1 \oplus \tau_2 & <: \tau_1' \oplus \tau_2' \\
\tau_1' & <: \tau_1 \\
\tau_2 & <: \tau_2'
\end{align*}
\]

[Sub-Unit]

[Sub-Sum]

[Sub-Pair]

[Sub-Fun]
Exercise. Prove that $\text{Bool}^H \to \text{Bool}^L <: \text{Bool}^L \to \text{Bool}^H$

$\tau <: \tau$

\[
\ell_1 \subseteq \ell_2 \quad S_1 <: S_2 \quad \frac{}{S_1 \ell_1 <: S_2 \ell_2} \quad \text{[Sub-LType]}
\]

$s <: s$

\[
\text{unit} <: \text{unit} \quad \text{[Sub-Unit]}
\]

\[
\oplus \in \{+ \times\} \quad \frac{i \in \{1,2\} \quad \tau_i <: \tau_i'}{\tau_1 \oplus \tau_2 <: \tau_1' \oplus \tau_2'} \quad \text{[Sub-Sum]}
\]

\[
\frac{\tau_1' <: \tau_1 \quad \tau_2 <: \tau_2'}{\tau_1 \to \tau_2 <: \tau_1' \to \tau_2'} \quad \text{[Sub-Fun]}
\]
Non-Interference for $\lambda^{SFG}$

For all $\lambda^{SFG}$ types, expressions, and values such that:

$$x : \tau \vdash e : \text{Bool}^L$$
Non-Interference for $\lambda^\text{SFG}$

For all $\lambda^\text{SFG}$ types, expressions, and values such that:

\[ x : \tau \vdash e : \text{Bool}^L \]
Non-Interference for $\lambda^{\text{SFG}}$

For all $\lambda^{\text{SFG}}$ types, expressions, and values such that:

$x : \tau \vdash e : \text{Bool}$

- **Secret** input
- **Public** output
Non-Interference for $\lambda^{SFG}$

For all $\lambda^{SFG}$ types, expressions, and values such that:

$\forall \lambda^{SFG} \text{ types, expressions, and values such that:}$

$\text{Secret input} \quad \text{Public output}$

$x : \tau \vdash e : \text{Bool}^L$

where
Non-Interference for $\lambda_{SFG}$

For all $\lambda_{SFG}$ types, expressions, and values such that:

$$x : \tau \vdash e : \text{Bool}^L$$

where

$L$ is the attacker security level
Non-Interference for $\lambda^{SFG}$

For all $\lambda^{SFG}$ types, expressions, and values such that:

$x : \tau \vdash e : \text{Bool}_L$

where

$L$ is the attacker security level

$\tau$ is not observable by the attacker:
Non-Interference for $\lambda^{SFG}$

For all $\lambda^{SFG}$ types, expressions, and values such that:

$\text{Secret input} \quad x : \tau \vdash e : \text{Bool}^L$

$\text{Public output}$

where

$L$ is the attacker security level

$\tau$ is not observable by the attacker:

$\tau = s^\ell \text{ such that } \ell \not\in L$
Non-Interference for $\lambda^{\text{SFG}}$

For all $\lambda^{\text{SFG}}$ types, expressions, and values such that:

$$x : \tau \vdash e : \text{Bool}_L$$
Non-Interference for $\lambda^{SFG}$

For all $\lambda^{SFG}$ types, expressions, and values such that:

$x : \tau \vdash e : \text{Bool}^L$

$v_1 : \tau$

$v_2 : \tau$
Non-Interference for $\lambda^{\text{SFG}}$

For all $\lambda^{\text{SFG}}$ types, expressions, and values such that:

$x : \tau \vdash e : \text{Bool}^L$

Any 2 secret input values

$v_1 : \tau$

$v_2 : \tau$
Non-Interference for $\lambda^{SFG}$

For all $\lambda^{SFG}$ types, expressions, and values such that:

- $\tau \vdash e : \text{Bool}^L$
- $v_1 : \tau$
- $v_2 : \tau$

If $e \Downarrow [x \mapsto v_1] \quad v$

any 2 secret input values

$e \Downarrow [x \mapsto v_2] \quad v'$
Non-Interference for $\lambda^{SFG}$

For all $\lambda^{SFG}$ types, expressions, and values such that:

$$x : \tau \vdash e : \text{Bool}^L$$

Any 2 secret input values

$\nu_1 : \tau$

$\nu_2 : \tau$

If

$$e \downarrow [x \mapsto \nu_1] \nu$$

$$e \downarrow [x \mapsto \nu_2] \nu'$$

Then

$$\nu = \nu'$$
Non-Interference for $\lambda^{SFG}$

For all $\lambda^{SFG}$ types, expressions, and values such that:

\[ x : \tau \vdash e : \text{Bool}^L \]

Any 2 secret input values

\[ v_1 : \tau \]
\[ v_2 : \tau \]

If

\[ e \Downarrow [x \mapsto v_1] \quad v \]
\[ e \Downarrow [x \mapsto v_2] \quad v' \]

then

\[ v = v' \]
Non-Interference for $\lambda^{\text{SFG}}$

For all $\lambda^{\text{SFG}}$ types, expressions, and values such that:

$x : \tau \vdash e : \text{Bool}^L$

Any 2 secret input values

$v_1 : \tau$
$v_2 : \tau$

If

\[ e \Downarrow [x \mapsto v_1] \quad v \]
\[ e \Downarrow [x \mapsto v_2] \quad v' \]

then $v = v'$

“Public outputs do not depend on secret inputs”
Proof Technique

1 Define a \textit{logical relation} for programs giving \textit{equal public outputs}
1. Define a **logical relation** for programs giving **equal public outputs**

\[ E[\tau] = \{ ((e_1, \theta_1), (e_2, \theta_2)) \mid \]
Define a *logical relation* for programs giving *equal public outputs*

\[
E[\tau] = \{ ((e_1, \theta_1), (e_2, \theta_2)) \mid e_1 \downarrow^{\theta_1} v_1 \land e_2 \downarrow^{\theta_2} v_2 \implies (v_1, v_2) \in V[\tau] \}
\]
Proof Technique

1. Define a **logical relation** for programs giving **equal public outputs**

\[
E \left[\tau\right]^L = \left\{ (\left((e_1, \theta_1)\right), (e_2, \theta_2)) \mid \begin{align*}
e_1 \downarrow^{\theta_1} v_1 & \land \ e_2 \downarrow^{\theta_2} v_2 \\
\rightarrow (v_1, v_2) & \in V \left[\tau\right]^L \end{align*} \right\}
\]
Proof Technique

1. Define a **logical relation** for programs giving **equal public outputs**

\[ E[\tau]_L = \{ ((e_1, \theta_1), (e_2, \theta_2)) \mid e_1 \Downarrow^{\theta_1} v_1 \land e_2 \Downarrow^{\theta_2} v_2 \implies (v_1, v_2) \in V[\tau]_L \} \]

2. Prove the **fundamental theorem** of logical relations
**Proof Technique**

1. Define a **logical relation** for programs giving **equal public outputs**

\[
\text{E} \llbracket \tau \rrbracket_L = \{ ((e_1, \theta_1), (e_2, \theta_2)) \mid e_1 \downarrow^{\theta_1} v_1 \land e_2 \downarrow^{\theta_2} v_2 \implies (v_1, v_2) \in \text{V} \llbracket \tau \rrbracket_L \} \]

**Equivalent values at level** \(L\)

2. Prove the **fundamental theorem** of logical relations

If \( \Gamma \vdash e : \tau \) then
Proof Technique

1. Define a **logical relation** for programs giving **equal public outputs**

\[
E[\tau]^L = \{ ((e_1, \theta_1), (e_2, \theta_2)) \mid e_1 \downarrow^{\theta_1} v_1 \land e_2 \downarrow^{\theta_2} v_2 \implies (v_1, v_2) \in V[\tau]^L \}
\]

2. Prove the **fundamental theorem** of logical relations

If \( \Gamma \vdash e : \tau \) then

\[
\forall (\theta_1, \theta_2) \in I[\Gamma]^L \implies ((e, \theta_1), (e, \theta_2)) \in E[\tau]^L
\]
Proof Technique

1 Define a **logical relation** for programs giving **equal public outputs**

\[
\mathcal{E}[\tau]^L = \{ (e_1, \theta_1), (e_2, \theta_2) \mid e_1 \downarrow^{\theta_1} v_1 \land e_2 \downarrow^{\theta_2} v_2 \implies (v_1, v_2) \in \mathcal{V}[\tau]^L \}
\]

Equivalent values at level \(L\)

2 Prove the **fundamental theorem** of logical relations

*If \(\Gamma \vdash e : \tau\) then*

\[
\forall (\theta_1, \theta_2) \in \mathcal{I}[\Gamma]^L \implies ((e, \theta_1), (e, \theta_2)) \in \mathcal{E}[\tau]^L
\]

Equivalent input envs at \(L\)
Proof Technique

1. Define a **logical relation** for programs giving equal **public** outputs

   \[ E[\Gamma][\tau]_L = \{ ((e_1, \theta_1), (e_2, \theta_2)) \mid e_1 \Downarrow^{\theta_1} v_1 \land e_2 \Downarrow^{\theta_2} v_2 \implies (v_1, v_2) \in V[\Gamma][\tau]_L \} \]

2. Prove the **fundamental theorem** of logical relations

   *If* \( \Gamma \vdash e : \tau \) *then*

   \[ \forall (\theta_1, \theta_2) \in I[\Gamma]_L \implies ((e, \theta_1), (e, \theta_2)) \in E[\Gamma][\tau]_L \]

3. Derive non-interference as a **corollary**
\[
\lambda^{SFG} \text{ with References}
\]

Syntax with references

Simple Types

\[ S ::= \cdots \mid \text{Ref } \tau \mid \tau \overset{\ell}{\rightarrow} \tau \]
\( \lambda^{SFG} \) with References

Syntax with references

Simple Types

\[
\begin{align*}
s & ::= \cdots \mid \text{Ref } \tau \mid \tau \xrightarrow{\ell} \tau
\end{align*}
\]

Keep tracks of side-effects
\[\lambda^{SFG}\] with References

Syntax with references

Simple Types
\[s ::= \cdots | \text{Ref } \tau | \tau \xrightarrow{\ell} \tau\]

Expressions
\[e ::= \cdots | \text{new } e | !e | e ::= e\]

Keep tracks of side-effects
Syntax with references

Simple Types
\[ s ::= \cdots \mid \text{Ref } \tau \mid \tau \xrightarrow{\ell} \tau \]

Expressions
\[ e ::= \cdots \mid \text{new } e \mid !e \mid e ::= e \]

Values
\[ v ::= \cdots \mid n \]

Keep tracks of side-effects
\( \lambda \text{SFG} \) with References

Syntax with references

Simple Types
\[
s ::= \cdots \mid \text{Ref} \ \tau \mid \tau \xrightarrow{\ell} \tau
\]

Expressions
\[
e ::= \cdots \mid \text{new} \ e \mid !e \mid e ::= e
\]

Values
\[
v ::= \cdots \mid n
\]

Keep tracks of side-effects

Address in store
\[ \lambda^{SFG} \] with References

Syntax with references

**Simple Types**
\[ s ::= \ldots \mid \text{Ref } \tau \mid \tau \xrightarrow{\ell} \tau \]

**Expressions**
\[ e ::= \ldots \mid \text{new } e \mid !e \mid e ::= e \]

**Values**
\[ v ::= \ldots \mid n \]

**Store**
\[ \Sigma \]

Keep tracks of side-effects
\( \lambda^{SFG} \) with References

**Syntax with references**

**Simple Types**

\[ s ::= \cdots \mid \text{Ref} \tau \mid \tau \xrightarrow{\ell} \tau \]

**Expressions**

\[ e ::= \cdots \mid \text{new} e \mid !e \mid e ::= e \]

**Values**

\[ v ::= \cdots \mid n \]

**Store**

\[ \Sigma \]

**Dynamic Semantics**

\[ \langle \Sigma, e \rangle \downarrow^\theta \langle \Sigma', v \rangle \]
\[ \lambda^{\text{SFG}} \] with References

Syntax with references

**Simple Types**
\[ s ::= \cdots | \text{Ref } \tau | \tau \xrightarrow{l} \tau \]

**Expressions**
\[ e ::= \cdots | \text{new } e | \!e | e ::= e \]

**Values**
\[ v ::= \cdots | n \]

**Store**
\[ \Sigma \]

Dynamic Semantics
\[ \langle \Sigma, e \rangle \downarrow^\theta \langle \Sigma', v \rangle \]

Keep tracks of side-effects

Address in store

Standard
Static Semantics

$$\Gamma \vdash_{pc} e : \tau$$
Static Semantics

\[ \Gamma \vdash_{\text{pc}} e : \tau \]

"Program Counter" label
The \( \text{pc} \) label is a \textit{lower bound} on the \textit{write effects} of the program \( e \).
Static Semantics

\[ \Gamma \vdash_{pc} e : \tau \]

"Program Counter" label

\{ Program \ e \ cannot \ create \ and \ write \ references \ labeled \ below \ the \ \mathtt{pc} \}

The \ mathtt{pc} \ label \ is \ a \ lower \ bound \ on \ the \ write \ effects \ of \ the \ program \ \mathtt{e} \
The pc label is a lower bound on the write effects of the program e.
The pc label is a **lower bound** on the **write effects** of the program e.

**Exercise.** Prove that the following program is **ill-typed**:

$$\Gamma \not\vdash_{\text{L}} \text{if } h \text{ then } l := \text{true} \text{ else } () : \text{unit}^H$$
Static Semantics

\[ \Gamma \vdash_{pc} e : \tau \]

The \( pc \) label is a **lower bound** on the **write effects** of the program \( e \)

**Exercise.** Prove that the following program is **ill-typed**:

\[ \Gamma \not\vdash_L \textbf{if } h \textbf{ then } l := \text{true} \textbf{ else } () : \text{unit}^H \]

with typing environment

\[ \Gamma = [ h \mapsto \text{Bool}^H, l \mapsto (\text{Ref Bool}^L)^L ] \]
Subtyping Relation

\[
\begin{array}{c}
\tau_1' <: \tau_1 \quad \tau_2 <: \tau_2' \\
\tau_1 \xrightarrow{\ell} \tau_2 <: \tau_1' \xrightarrow{\ell'} \tau_2' \\
\text{[Sub-Fun]}
\end{array}
\]
Subtyping Relation

\[ \tau_1' <: \tau_1 \quad \tau_2 <: \tau_2' \quad \ell' \sqsubseteq \ell \]

\[ \tau_1 \xrightarrow{\ell} \tau_2 <: \tau_1' \xrightarrow{\ell'} \tau_2' \]

Contravariant
Subtyping Relation

$\text{Contravariant}$

$\tau_1' <: \tau_1$  $\tau_2 <: \tau_2'$  $\ell' \sqsubseteq \ell$

$\tau_1 \xrightarrow{\ell} \tau_2  <:  \tau_1' \xrightarrow{\ell'} \tau_2'$

References?
Subtyping Relation

\[ \begin{align*}
\tau_1' & \lll \tau_1 \\
\tau_2 & \lll \tau_2' \\
\ell' & \subseteq \ell
\end{align*} \]  

[Hyp Fun]

\[ \begin{align*}
\tau_1 & \xrightarrow{\ell} \tau_2 \\
\tau_1' & \xrightarrow{\ell'} \tau_2'
\end{align*} \]

References?

Contravariant

\[ \begin{align*}
\tau' & \lll \tau \\
\text{Ref } \tau & \lll \text{Ref } \tau'
\end{align*} \]  

Covariant
Subtyping Relation

\[ \tau_1' <: \tau_1 \quad \tau_2 <: \tau_2' \quad \ell' \sqsubseteq \ell \]

\[ \tau_1 \xrightarrow{\ell} \tau_2 <: \tau_1' \xrightarrow{\ell'} \tau_2' \]

References ?

Covariant

\[ \tau <: \tau' \]

Ref \( \tau <: \) Ref \( \tau' \)

Contravariant

\[ \tau' <: \tau \]

Ref \( \tau <: \) Ref \( \tau' \)
Subtyping Relation

\[ s <: s \]

\[ \tau_1' <: \tau_1 \quad \tau_2 <: \tau_2' \quad \ell' \sqsubseteq \ell \]

\[ \tau_1 \xrightarrow{\ell} \tau_2 <: \tau_1' \xrightarrow{\ell'} \tau_2' \]

References?

Covariant

\[ \tau <: \tau' \]
\[ \text{Ref } \tau <: \text{Ref } \tau' \]

Invariant

\[ \text{Ref } \tau <: \text{Ref } \tau \]

Contravariant

\[ \tau' <: \tau \]
\[ \text{Ref } \tau <: \text{Ref } \tau' \]
Subtyping Relation

\[ s <: s \]

\[
\frac{\tau_1' <: \tau_1 \quad \tau_2 <: \tau_2'}{\tau_1 \xrightarrow{\ell} \tau_2 <: \tau_1' \xrightarrow{\ell'} \tau_2'}\]

\[ \ell' \subseteq \ell \]

[Sub-Fun]

References?

Covariant

\[
\frac{\tau <: \tau'}{\text{Ref } \tau <: \text{Ref } \tau'}
\]

Invariant

\[
\frac{\tau <: \tau'}{\text{Ref } \tau <: \text{Ref } \tau}
\]

Contravariant

\[
\frac{\tau' <: \tau}{\text{Ref } \tau <: \text{Ref } \tau'}
\]

Contravariant
Exercise

Find a well-typed program that leaks if we consider references **covariant**:

\[
\begin{align*}
\tau &<: \tau' \\
\text{Ref } \tau &<: \text{Ref } \tau'
\end{align*}
\]

Find a well-typed program that leaks if we consider references **contravariant**:

\[
\begin{align*}
\tau' &<: \tau \\
\text{Ref } \tau &<: \text{Ref } \tau'
\end{align*}
\]
Soundness issues!

Covariant

\[ \tau <: \tau' \]

\[
\text{Ref } \tau <: \text{Ref } \tau'
\]

Contravariant

\[ \tau' <: \tau \]

\[
\text{Ref } \tau <: \text{Ref } \tau'
\]
Soundness issues!

Covariant

\[ \tau <: \tau' \]

\[ \text{Ref } \tau <: \text{Ref } \tau' \]

Contravariant

\[ \tau' <: \tau \]

\[ \text{Ref } \tau <: \text{Ref } \tau' \]

Ref Bool\(^L\) can be written as Ref Bool\(^H\)
Soundness issues!

Covariant \( \tau <: \tau' \) 
\[
\frac{\tau <: \tau'}{\text{Ref } \tau <: \text{Ref } \tau'}
\]

Contravariant \( \tau' <: \tau \) 
\[
\frac{\tau' <: \tau}{\text{Ref } \tau <: \text{Ref } \tau'}
\]

Ref \( \text{Bool} \) can be written as \( \text{Ref } \text{Bool} \)

let \( h_{\text{ref}} = l_{\text{ref}} \) in
\[
\begin{align*}
h_{\text{ref}} & := h \\
!l_{\text{ref}} &
\end{align*}
\]
Soundness issues!

Covariant

\[ \tau <: \tau' \]
\[
\text{Ref } \tau <: \text{Ref } \tau'
\]

Ref Bool\textsuperscript{L} can be written as Ref Bool\textsuperscript{H}

Contravariant

\[ \tau' <: \tau \]
\[
\text{Ref } \tau <: \text{Ref } \tau'
\]

Ref Bool\textsuperscript{H} can be read as Ref Bool\textsuperscript{L}

let h\_ref = l\_ref in
h\_ref := h
!l\_ref
Soundness issues!

Covariant

\[ \tau <: \tau' \]

Ref \[ \tau <: \text{Ref} \ \tau' \]

Ref \[ \text{Bool}^L \] can be written as Ref \[ \text{Bool}^H \]

let \[ \text{h\_ref} = \text{l\_ref} \] in
\[ \text{h\_ref} := \text{h} \]

let \[ \text{l\_ref} = \text{h\_ref} \] in
\[ !\text{l\_ref} \]

Contravariant

\[ \tau' <: \tau \]

Ref \[ \tau <: \text{Ref} \ \tau' \]

Ref \[ \text{Bool}^H \] can be read as Ref \[ \text{Bool}^L \]

let \[ \text{l\_ref} = \text{h\_ref} \] in
\[ !\text{l\_ref} \]
Soundness issues!

**Covariant**

\[ \tau <: \tau' \]

\[ \text{Ref } \tau <: \text{Ref } \tau' \]

Ref Bool\(^L\) can be written as Ref Bool\(^H\)

```plaintext
let h_ref = l_ref in
h_ref := h
!l_ref
```

**Contravariant**

\[ \tau' <: \tau \]

\[ \text{Ref } \tau <: \text{Ref } \tau' \]

Ref Bool\(^H\) can be read as Ref Bool\(^L\)

```plaintext
let l_ref = h_ref in
!l_ref
```

Well-typed but leak!
Covariant \( \times \)

Ref \( \text{Bool}^L \) \textbf{can be written} as Ref \( \text{Bool}^H \)

Contravariant \( \times \)

Ref \( \text{Bool}^H \) \textbf{can be read} as Ref \( \text{Bool}^L \)

References are input (read) and output (write) channels!

Invariant

\[
\text{Ref } \tau \lessdot \text{Ref } \tau
\]
Soundness Proof

Non-Interference for $\lambda^{SFG}$ with higher-order state
Soundness Proof

Non-Interference for $\lambda^{SFG}$ with higher-order state

The store can contain references
Soundness Proof

Non-Interference for $\lambda^{SFG}$ with higher-order state

Step-indexed Kripke logical relation

The store can contain references
Soundness Proof

The store can contain references

Non-Interference for $\lambda^{SFG}$ with higher-order state

Avoid circular reasoning

Step-indexed Kripke logical relation
Soundness Proof

Non-Interference for $\lambda^{SFG}$ with higher-order state

The store can contain references

Avoid circular reasoning

Step-indexed Kripke logical relation

See “On the Expressiveness and Semantics of Information Flow Types” by Rajani and Garg
Outline

Overview of different language-based IFC approaches

• Non Interference

• 4 IFC Languages

<table>
<thead>
<tr>
<th></th>
<th>Static</th>
<th>Dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fine-grained</td>
<td>$\lambda_{SFG}$</td>
<td>$\lambda_{DFG}$</td>
</tr>
<tr>
<td>Coarse-grained</td>
<td>$\lambda_{SCG}$</td>
<td>$\lambda_{DCG}$</td>
</tr>
</tbody>
</table>
Outline

Overview of different language-based IFC approaches

• Non Interference

• 4 IFC Languages

<table>
<thead>
<tr>
<th></th>
<th>Static</th>
<th>Dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fine-grained</td>
<td>$\lambda_{SFG}$</td>
<td>$\lambda_{DFG}$</td>
</tr>
<tr>
<td>Coarse-grained</td>
<td>$\lambda_{SCG}$</td>
<td>$\lambda_{DCG}$</td>
</tr>
</tbody>
</table>
Dynamic Fine-Grained IFC

Enforce dynamic security policies
Dynamic Fine-Grained IFC

Enforce dynamic security policies

Possibly unknown statically
Dynamic Fine-Grained IFC

Enforce dynamic security policies

Possibly unknown statically

Runtime Labels
Dynamic Fine-Grained IFC

Enforce dynamic security policies

Runtime Labels

Possibly unknown statically

Label Introspection

\[
\text{if ( } \text{send} ( \text{, } ) \text{ ) = ( ) }
\]

send( )
Dynamic Fine-Grained IFC

Enforce dynamic security policies

Possibly unknown statically

Runtime Labels

Label Introspection

if ( ♂ = ♂ )
send( ♂ , ♂ )

Useful programming patterns
Dynamic Fine-grained IFC

Syntax

\[ \text{Types} \quad \tau ::= \text{unit} \mid \tau \to \tau \mid \tau + \tau \mid \tau \times \tau \mid \text{Label} \]
Dynamic Fine-grained IFC

Syntax

Types $\tau ::= \text{unit} \mid \tau \rightarrow \tau \mid \tau + \tau \mid \tau \times \tau \mid \text{Label}$
Dynamic Fine-grained IFC

Syntax

Types \( \tau ::= \text{unit} \mid \tau \to \tau \mid \tau + \tau \mid \tau \times \tau \mid \text{Label} \)

Labeled Values \( v ::= r^\ell \)
Dynamic Fine-grained IFC

Syntax

Types
\[ \tau ::= \text{unit} \mid \tau \rightarrow \tau \mid \tau + \tau \mid \tau \times \tau \mid \text{Label} \]

Labeled Values
\[ \nu ::= r^\ell \quad \text{Raw value at security level } \ell \]
Dynamic Fine-grained IFC

**Syntax**

*Types*

\[ \tau ::= \text{unit} \mid \tau \rightarrow \tau \mid \tau + \tau \mid \tau \times \tau \mid \text{Label} \]

*Labeled Values*

\[ v ::= r^\ell \]

*Raw Values*

\[ r ::= () \mid (x.e, \theta) \mid \langle v, v \rangle \]

\[ \mid \text{inl}(v) \mid \text{inr}(v) \mid \ell \]

*Environments*

\[ \theta \in \text{Var} - \text{LValue} \]

*New!*

Raw value at security level \( \ell \)
Dynamic Fine-grained IFC

Syntax

Types \( \tau ::= \text{unit} | \tau \rightarrow \tau | \tau + \tau | \tau \times \tau | \text{Label} \)

Labeled Values \( v ::= r^\ell \) Raw value at security level \( \ell \)

Raw Values \( r ::= () | (x.e, \theta) | \langle v, v \rangle \)

\[ \quad | \text{inl}(v) | \text{inr}(v) | \ell \] Runtime labels

Environments \( \theta \in \text{Var} \rightarrow \text{LValue} \)
Dynamic Fine-grained IFC

\[ \lambda_{DFG} \]

**Syntax**

Types

\[ \tau ::= \text{unit} \mid \tau \to \tau \mid \tau + \tau \mid \tau \times \tau \mid \text{Label} \]

Labeled Values

\[ v ::= r^\ell \]

Raw value at security level \( \ell \)

Raw Values

\[ r ::= () \mid (x.e, \theta) \mid \langle v, v \rangle \]

\[ \mid \text{inl}(v) \mid \text{inr}(v) \mid \ell \]

Runtime labels

Environments

\[ \theta \in \text{Var} \to \text{LValue} \]

Expressions

\[ e ::= \cdots \mid \text{labelOf}(e) \mid \text{getPC} \mid e \triangleleft e \]
Dynamic Fine-grained IFC

Syntax

Types  \( \tau ::= \text{unit} \mid \tau \to \tau \mid \tau + \tau \mid \tau \times \tau \mid \text{Label} \)

Labeled Values  \( v ::= r^\ell \)  

Raw Values  \( r ::= () \mid (x.e, \theta) \mid (v, v) \mid \text{inl}(v) \mid \text{inr}(v) \mid \ell \)

Environments  \( \theta \in \text{Var} \to \text{LValue} \)

Expressions  \( e ::= \cdots \mid \text{labelOf}(e) \mid \text{getPC} \mid e \triangleq? e \)
Semantics

Static $\Gamma \vdash e : \tau$
Semantics

Static \quad \Gamma \vdash e : \tau

Standard: no security checks!
Semantics

Static \[ \Gamma \vdash e : \tau \]

Dynamic \[ e \downarrow^{\theta}_{pc} v \]

Standard: no security checks!
Semantics

Static \[ \Gamma \vdash e : \tau \]

Dynamic \[ e \downarrow_{\theta_{pc}} v \]

Security Monitor

Standard: no security checks!
Semantics

*Static*: $\Gamma \vdash e : \tau$

*Dynamic*: $e \Downarrow_{\mathrm{pc}}^{\theta} v$

*Standard*: no security checks!

*Security Monitor*

*Program Counter*
Semantics

Static: \[ \Gamma \vdash e : \tau \]

Dynamic: \[ e \Downarrow_{\theta_{\text{pc}}} v \]

The monitor propagates labels from inputs to outputs
Label Propagation

The semantics tracks control-flow dependencies with the **program counter** label.

\[ \theta = [ \ x \mapsto \text{true}^H, \ y \mapsto \text{true}^L, \ z \mapsto \text{false}^L ] \]
Label Propagation

The semantics tracks control-flow dependencies with the **program counter** label.

\[
\theta = [ x \mapsto \text{true}^H, \ y \mapsto \text{true}^L, \ z \mapsto \text{false}^L ]
\]
Label Propagation

The semantics tracks control-flow dependencies with the \textit{program counter} label.

\[ \theta = [ x \mapsto \text{true}^H, y \mapsto \text{true}^L, z \mapsto \text{false}^L ] \]
Label Propagation

The semantics tracks control-flow dependencies with the program counter label.

\[ \theta = \{ x \mapsto \text{true}^H, \ y \mapsto \text{true}^L, \ z \mapsto \text{false}^L \} \]
Label Propagation

The semantics tracks control-flow dependencies with the \textit{program counter} label.

\[ \theta = [ x \mapsto \text{true}^H, y \mapsto \text{true}^L, z \mapsto \text{false}^L ] \]
Label Propagation

The semantics tracks control-flow dependencies with the **program counter** label.

\[ \theta = [ x \mapsto \text{true}^H, \ y \mapsto \text{true}^L, \ z \mapsto \text{false}^L ] \]
Dynamic Semantics $e \downarrow_{pc}^\theta v$
Dynamic Semantics

e \Downarrow_{\theta}^{\theta} v

Observations

Introduction rules label the result with the **program counter**

Elimination rules **taint** the result with the intermediate value
Dynamic Semantics \( \lambda SFG \) 

Observations

*Introduction rules label the result with the **program counter***

*Elimination rules **taint** the result with the intermediate value*

Invariant

\[
\text{If } \ e \downarrow_{\text{pc}}^\theta r^\ell \text{ then } \text{pc} \subseteq \ell
\]
Label Introspection

\[ \text{labelOf}(e) \Downarrow_{pc}^{\theta} \]
Label Introspection

\[
e \downarrow_{\theta_{pc}} r \ell
\]

\[
\text{label0f}(e) \downarrow_{\theta_{pc}}
\]
Label Introspection

\[ e \downarrow_{\theta_{pc}} r \ell \]

\[
\text{label0f}(e) \downarrow_{\theta_{pc}} \ell
\]
Label Introspection

\[ e \downarrow_{p_c} r \ell \]

\[ \text{label0f}(e) \downarrow_{p_c} \ell \]

What is the label of the label itself?
Label Introspection

\[ e \downarrow_{\theta_{pc}}^\ell r \ell \]

\[ \text{labelOf}(e) \downarrow_{\theta_{pc}}^\ell \ell \]
The label has the \textbf{same sensitivity} of the result!

\[ e \downarrow_{p_c} r \ell \]

\[ \text{label0f}(e) \downarrow_{p_c} \ell \ell \]
Label Introspection

\[ e \downarrow_{\theta_{pc}} r \ell \]

\[ \text{label0f}(e) \downarrow_{\theta_{pc}} \ell \ell \]

\[ \text{getPC} \downarrow_{\theta_{pc}} \text{pc}^{pc} \]

The label has the same sensitivity of the result!
$\lambda^{DFG}$ with References

Syntax with references

```
Simple Types  \( \tau ::= \cdots \mid \text{Ref } \tau \)
```
$\lambda^\text{DFG}$ with References

Syntax with references

Simple Types $\tau ::= \cdots \mid \text{Ref } \tau$

Values $v ::= \cdots \mid n_\ell$
\[ \lambda^\text{DFG} \] with References

Syntax with references

\[ \begin{align*}
\text{Simple Types} & \quad \tau ::= \cdots \mid \text{Ref} \ \tau \\
\text{Values} & \quad v ::= \cdots \mid n_\ell
\end{align*} \]

Reference to data labeled \( \ell \)
\[ \lambda^{\text{DFG}} \text{ with References} \]

**Syntax with references**

**Simple Types**
\[
\tau ::= \cdots \mid \text{Ref } \tau
\]

**Values**
\[
v ::= \cdots \mid n_\ell
\]

**Expressions**
\[
e ::= \cdots \mid \text{new } e \mid !e \mid e := e
\]
\[
\mid \text{labelOfRef}(e)
\]

*Reference to data labeled \( \ell \)
\[ \text{Syntax with references} \]

**Simple Types**

\[ \tau ::= \cdots \mid \text{Ref } \tau \]

**Values**

\[ v ::= \cdots \mid n_\ell \]

**Expressions**

\[ e ::= \cdots \mid \text{new } e \mid \text{!}e \mid e := e \]

\[ \mid \text{labelOfRef}(e) \]

- Reference to data labeled \( \ell \)
- Label introspection on refs
\[\lambda\text{DFG} \text{ with References}\]

Syntax with references

**Simple Types**
\[
\tau ::= \cdots \mid \text{Ref } \tau
\]

**Values**
\[
v ::= \cdots \mid n_\ell
\]

**Expressions**
\[
e ::= \cdots \mid \text{new } e \mid !e \mid e ::= e
\]
\[
\mid \text{labelOfRef}(e)
\]

**Store**
\[
\Sigma \in (\ell : \text{Label}) \to \text{Memory } \ell
\]

**Memory**
\[
\ell \quad M ::= [] \mid r : M
\]
\[
\lambda^{DFG} \text{ with References}
\]

### Syntax with references

**Simple Types**
\[
\tau ::= \cdots | \text{Ref } \tau
\]

**Values**
\[
v ::= \cdots | n_\ell
\]

**Expressions**
\[
e ::= \cdots | \text{new } e | !e | e ::= e
\]

\[
| \text{labelOfRef}(e)
\]

**Store**
\[
\Sigma \in (\ell : \text{Label}) \rightarrow \text{Memory } \ell
\]

**Memory**
\[
\ell \text{ Memory } M ::= [] | r : M
\]

- **Reference to data labeled** \( \ell \)
- **Label introspection on refs**
- **The store is partitioned by label**
Dynamic Semantics

\[ \langle \Sigma, e \rangle \downarrow_{pc}^\theta \langle \Sigma', v \rangle \]

\[ \langle \Sigma, \text{new } e \rangle \downarrow_{pc}^\theta \langle \Sigma'', (n_{\ell})^{pc} \rangle \]

[ New ]
\[ \langle \Sigma, e \rangle \downarrow_{pc}^{\theta} \langle \Sigma', v \rangle \]

\[ \langle \Sigma, e \rangle \downarrow_{pc}^{\theta} \langle \Sigma', r^\ell \rangle \]

\[ \langle \Sigma, \text{new } e \rangle \downarrow_{pc}^{\theta} \langle \Sigma'', (n^\ell)_{pc} \rangle \]

[ New ]
Dynamic Semantics

\[ \langle \Sigma, e \rangle \downarrow_{pc}^{\theta} \langle \Sigma', v \rangle \]

Allocate in memory \( \ell \)

\[ \langle \Sigma, e \rangle \downarrow_{pc}^{\theta} \langle \Sigma', r^\ell \rangle \]

[ New ]

\[ \langle \Sigma, \text{new } e \rangle \downarrow_{pc}^{\theta} \langle \Sigma'', (n\ell)^{pc} \rangle \]
Allocate in memory $\ell$

$$\langle \Sigma, e \rangle \Downarrow_{p_c} \langle \Sigma', r^\ell \rangle$$

$$n = |\Sigma'(\ell)|$$

$$\langle \Sigma, \text{new } e \rangle \Downarrow_{p_c} \langle \Sigma'', (n_\ell)^{p_c} \rangle$$
Allocate in memory $\ell$

$n = |\Sigma'(\ell)|$

\[\langle \Sigma, e \rangle \Downarrow_{pc}^{\theta} \langle \Sigma', r^\ell \rangle\]

\[\langle \Sigma, \text{new } e \rangle \Downarrow_{pc}^{\theta} \langle \Sigma'', (n^\ell)^{pc} \rangle\]
Dynamic Semantics

\[ \langle \Sigma, e \rangle \Downarrow^\theta_{pc} \langle \Sigma', v \rangle \]

Allocate in memory \( \ell \)

Fresh Address

\[ \langle \Sigma, e \rangle \Downarrow_{pc} \langle \Sigma', r^\ell \rangle \]

\[ n = | \Sigma'(\ell) | \quad \Sigma'' = \Sigma'[^{\ell} \mapsto \Sigma'(\ell) [n \mapsto r]] \]

[ New ]

\[ \langle \Sigma, \text{new } e \rangle \Downarrow_{pc} \langle \Sigma'', (n\ell)^{pc} \rangle \]
Dynamic Semantics

\[ \{ \Sigma, e \} \downarrow_{pc} \theta \{ \Sigma', v \} \]

**Allocate in memory \( \ell \)**

\[ \{ \Sigma, e \} \downarrow_{pc} \theta \{ \Sigma', r^\ell \} \]

\[ n = |\Sigma'(\ell)| \quad \Sigma'' = \Sigma'[\ell \mapsto \Sigma'(\ell)[n \mapsto r]] \]

\[ \{ \Sigma, \text{new } e \} \downarrow_{pc} \theta \{ \Sigma'', (n\ell)^{pc} \} \]

**Fresh Address**

**Update the store**

\[ \Sigma'' = \Sigma'[\ell \mapsto \Sigma'(\ell)[n \mapsto r]] \]

[ New ]
Dynamic Semantics

\[ \langle \Sigma, e \rangle \Downarrow^\theta_{pc} \langle \Sigma', v \rangle \]

\[ \langle \Sigma, !e \rangle \Downarrow^\theta_{pc} \]

[ Read ]
Dynamic Semantics

\[ \langle \Sigma, e \rangle \Downarrow_{pc}^\theta \langle \Sigma', v \rangle \]

\[ \langle \Sigma, e \rangle \Downarrow_{pc}^\theta \langle \Sigma', (n_\ell)_{\ell'} \rangle \]

\[ \langle \Sigma, !e \rangle \Downarrow_{pc}^\theta \]
Dynamic Semantics

\[ \langle \Sigma, e \rangle \downarrow_{pc}^{\theta} \langle \Sigma', v \rangle \]

\[ \langle \Sigma, e \rangle \downarrow_{pc} \langle \Sigma', (n_\ell)\ell' \rangle \]

\[ \langle \Sigma, !e \rangle \downarrow_{pc}^{\theta} \]

*Protects the "identity" of the ref*
Dynamic Semantics

\[ \langle \Sigma, e \rangle \downarrow_{pc}^\theta \langle \Sigma', v \rangle \]

Protects the "identity" of the ref

\[ \langle \Sigma, e \rangle \downarrow_{pc}^{\theta} \langle \Sigma', (n_\ell)^{\ell'} \rangle \quad \Sigma'(\ell)[n] = r \]

[ Read ]
Dynamic Semantics

\[ \langle \Sigma, e \rangle \downarrow_{pc}^\theta \langle \Sigma', v \rangle \]

**Protects the “identity” of the ref**

\[ \langle \Sigma, e \rangle \downarrow_{pc}^\theta \langle \Sigma', (n_{\ell})_{\ell'} \rangle \quad \Sigma'(\ell)[n] = r \]

\[ \langle \Sigma, !e \rangle \downarrow_{pc}^\theta \langle \Sigma', r_{\ell} \sqcup \ell' \rangle \]

[Read]
Dynamic Semantics

\[ \langle \Sigma, e \rangle \Downarrow_{pc}^\theta \langle \Sigma', v \rangle \]

Protects the "identity" of the ref

\[ \langle \Sigma, e \rangle \Downarrow_{pc}^\theta \langle \Sigma', (n\ell)\ell' \rangle \quad \Sigma'(\ell)[n] = r \]

[ Read ]

\[ \langle \Sigma, !e \rangle \Downarrow_{pc}^\theta \langle \Sigma', r\ell \sqcup \ell' \rangle \]

Tainted with original label + identity of the ref
Dynamic Semantics

\[ \lambda_{DFG} \]

\[ \langle \Sigma, e \rangle \downarrow_{pc}^{\theta} \langle \Sigma', v \rangle \]

\[ \langle \Sigma, e_1 := e_2 \rangle \downarrow_{pc}^{\theta} \]

[ Write ]
Dynamic Semantics

\[ \langle \Sigma, e \rangle \downarrow_{pc}^\theta \langle \Sigma', v \rangle \]

\[ \langle \Sigma, e_1 \rangle \downarrow_{pc}^\theta \langle \Sigma', (n_\ell)\ell_1 \rangle \]

\[ \langle \Sigma, e_1 := e_2 \rangle \downarrow_{pc}^\theta \]

[Write]
Dynamic Semantics

\[ (\Sigma, e) \Downarrow_{\text{pc}} (\Sigma', v) \]

\[ (\Sigma, e_1) \Downarrow_{\text{pc}} (\Sigma', (n\ell)^{\ell_1}) \]

\[ (\Sigma', e_2) \Downarrow_{\text{pc}} (\Sigma'', r^{\ell_2}) \]

\[ (\Sigma, e_1 := e_2) \Downarrow_{\text{pc}} \]

[ Write ]
Dynamic Semantics

\[ \langle \Sigma, e \rangle \downarrow_{pc}^{\theta} \langle \Sigma', \nu \rangle \]

\[ \langle \Sigma, e_1 \rangle \downarrow_{pc}^{\theta} \langle \Sigma', (n_\ell)^{\ell_1} \rangle \]

\[ \ell_1 \subseteq \ell \]

\[ \langle \Sigma', e_2 \rangle \downarrow_{pc}^{\theta} \langle \Sigma'', r^{\ell_2} \rangle \]

\[ \ell_2 \subseteq \ell \]

\[ \langle \Sigma, e_1 := e_2 \rangle \downarrow_{pc}^{\theta} \]

Security Checks

[ Write ]
Dynamic Semantics

\[ \langle \Sigma, e \rangle \Downarrow_{pc} \langle \Sigma', v \rangle \]

\[ \langle \Sigma, e_1 \rangle \Downarrow_{pc} \langle \Sigma', (n\ell)\ell_1 \rangle \]
\[ \ell_1 \sqsubseteq \ell \]

\[ \langle \Sigma', e_2 \rangle \Downarrow_{pc} \langle \Sigma'', r\ell_2 \rangle \]
\[ \ell_2 \sqsubseteq \ell \]

\[ \langle \Sigma, e_1 := e_2 \rangle \Downarrow_{pc} \]

\[ \ell_1 \sqsubseteq \ell \]

The decision of writing this reference must not depend on data above the label of the reference.
Dynamic Semantics

\[ \langle \Sigma, e \rangle \downarrow_{\theta}^{\bowtie} \langle \Sigma', v \rangle \]

\[ \langle \Sigma, e_1 \rangle \downarrow_{\theta}^{\bowtie} \langle \Sigma', (n_{\ell})\ell_1 \rangle \]

\[ \ell_1 \subseteq \ell \]

\[ \langle \Sigma', e_2 \rangle \downarrow_{\theta}^{\bowtie} \langle \Sigma'', \ell_2 \rangle \]

\[ \ell_2 \subseteq \ell \]

\[ \langle \Sigma, e_1 := e_2 \rangle \downarrow_{\theta}^{\bowtie} \]

\[ \ell_1 \subseteq \ell \quad \text{The decision of writing this reference must not depend on data above the label of the reference} \]

\[ \ell_2 \subseteq \ell \quad \text{Must not write data above the label of the reference} \]
\[\langle \Sigma, e \rangle \downarrow_{pc}^{\theta} \langle \Sigma', v \rangle\]

\[\langle \Sigma, e_1 \rangle \downarrow_{pc}^{\theta} \langle \Sigma', (n_{\ell})^{\ell_1} \rangle\]

\[\ell_1 \subseteq \ell\]

\[\langle \Sigma', e_2 \rangle \downarrow_{pc}^{\theta} \langle \Sigma'', r^{\ell_2} \rangle\]

\[\ell_2 \subseteq \ell\]

\[\Sigma''' = \Sigma''[\ell \mapsto \Sigma''(\ell)[n \mapsto r]]\]

\[\langle \Sigma, e_1 := e_2 \rangle \downarrow_{pc}^{\theta} \langle \Sigma''', (\_)^{pc} \rangle\]

\[\ell_1 \subseteq \ell\]

The decision of writing \textbf{this} reference must not depend on data above the label of the reference.

\[\ell_2 \subseteq \ell\]

Must not write data above the label of the reference.
1. Define the *low-equivalence* relation $V_1 \approx^T_L V_2$.
Proof Technique

1. Define the low-equivalence relation $v_1 \approx^T_L v_2$.

$V_1$ and $V_2$ are indistinguishable at security level $L$. 
Proof Technique

1. Define the low-equivalence relation $v_1 \approx_L v_2$

2. Prove that the semantics preserves the relation:

   $\theta_1 \approx \theta_2$

   $c_1 \approx c_2$

$v_1$ and $v_2$ are indistinguishable at security level $L$
Proof Technique

1. Define the **low-equivalence** relation $V_1 \approx^L V_2$

2. Prove that the semantics **preserves** the relation:

   $\theta_1 \approx \theta_2$

   $C_1 \approx C_2$

   if

   $C_1 \downarrow^{\theta_1}_{pc} C_1'$

   $C_2 \downarrow^{\theta_2}_{pc} C_2'$

$V_1$ and $V_2$ are indistinguishable at security level $L$
Proof Technique

1. Define the low-equivalence relation $V_1 \approx^T_L V_2$

2. Prove that the semantics preserves the relation:

\[
\begin{align*}
\theta_1 \approx \theta_2 \\
C_1 \approx C_2
\end{align*}
\]

if

\[
\begin{align*}
C_1 \downarrow^{\theta_1}_{pc} C_1' \\
C_2 \downarrow^{\theta_2}_{pc} C_2'
\end{align*}
\]

then $C_1' \approx C_2'$
Proof Technique

1. Define the **low-equivalence** relation

\[ V_1 \approx^T_L V_2 \]

2. Prove that the semantics **preserves** the relation:

\[ \theta_1 \approx \theta_2 \]
\[ C_1 \approx C_2 \]

\[ \text{if } C_1 \downarrow^p_c C_1' \]
\[ C_2 \downarrow^p_c C_2' \]

then \[ C_1' \approx C_2' \]

3. Derive non-interference as a **corollary**
Outline

Overview of different language-based IFC approaches

- Non Interference
- 4 IFC Languages

<table>
<thead>
<tr>
<th></th>
<th>Static</th>
<th>Dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fine-grained</td>
<td>$\lambda_{SFG}$</td>
<td>$\lambda_{DFG}$</td>
</tr>
<tr>
<td>Coarse-grained</td>
<td>$\lambda_{SCG}$</td>
<td>$\lambda_{DCG}$</td>
</tr>
</tbody>
</table>
Outline

Overview of different language-based IFC approaches

- Non Interference
- 4 IFC Languages

<table>
<thead>
<tr>
<th></th>
<th>Static</th>
<th>Dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fine-grained</td>
<td>$\lambda_{SFG}$</td>
<td>$\lambda_{DFG}$</td>
</tr>
<tr>
<td>Coarse-grained</td>
<td>$\lambda_{SCG}$</td>
<td>$\lambda_{DCG}$</td>
</tr>
</tbody>
</table>
### Outline

**Overview of different language-based IFC approaches**

- **Non Interference**
  - ✅

- **4 IFC Languages**

<table>
<thead>
<tr>
<th></th>
<th>Static</th>
<th>Dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fine-grained</td>
<td>$\lambda_{SFG}$</td>
<td>$\lambda_{DFG}$</td>
</tr>
<tr>
<td>Coarse-grained</td>
<td>$\lambda_{SCG}$</td>
<td>$\lambda_{DCG}$</td>
</tr>
</tbody>
</table>

✅
References

**Introduction and Surveys**

- Language-based information-flow security
  Andrei Sabelfeld and Andrew C. Myers

- A Perspective on Information-Flow Control
  Daniel Hedin and Andrei Sabelfeld

**Different Variants of Non-Interference**

- From dynamic to static and back: Riding the roller coaster of information-flow control research
  Andrei Sabelfeld and Alejandro Russo
Fine-Grained IFC

- **Static**
  - On the Expressiveness and Semantics of Information Flow Types
    - Vineet Rajani and Deepak Garg

- **Dynamic**
  - Efficient purely dynamic information flow analysis
    - Thomas H. Austin and Cormac Flanagan

- **Hybrid**
  - Type-Driven Gradual Security with References
    - Matías Toro, Ronald García, Éric Tanter
Coarse-Grained IFC

MAC, A Verified Static Information-Flow Control Library
Marco Vassena, Alejandro Russo, Pablo Buiras, Lucas Waye

Flexible Dynamic Information Flow Control in Presence of Exceptions
Deian Stefan, Alejandro Russo, John Mitchell, and David Mazières

HLIO: Mixing Static and Dynamic Typing for Information-Flow Control in Haskell
Pablo Buiras, Dimitrios Vytiniotis, and Alejandro Russo
Covert Channels

Addressing Covert Termination and Timing Channels in Concurrent Information Flow Systems
Deian Stefan, Alejandro Russo, Pablo Buiras, Amit Levy, John C. Mitchell, and David Mazières

Securing Concurrent Lazy Programs Against Information Leakage
Marco Vassena, Joachim Breitner and Alejandro Russo

Foundations for Parallel Information Flow Control Runtime Systems
Marco Vassena, Gary Soeller, Peter Amidon, Matthew Chan, and Deian Stefan

From trash to treasure: timing-sensitive garbage collection
Mathias V. Pedersen and Aslan Askarov

A Library For Removing Cache-based Attacks in Concurrent Information Flow Systems
Pablo Buiras, Deian Stefan, Amit Levy, Alejandro Russo, and David Mazières
Declassification and Endorsement

Declassification: Dimensions and principles
Andrei Sabelfeld and David Sands

A Semantic Framework for Declassification and Endorsement
Aslan Askarov and Andrew C. Myers

Nonmalleable Information Flow Control
Ethan Cecchetti, Andrew C. Myers, Owen Arden