Compositional non-interference
for fine-grained concurrent programs

Dan Frumin  (jww Robbert Krebbers & Lars Birkedal)
29 October, 2019 © Iris Workshop
Radboud University
We want to study security properties of systems formally. In this talk we consider confidentiality through non-interference.
We want to study security properties of systems formally. In this talk we consider *confidentiality* through *non-interference*.

- Confidentiality: secret information is not revealed to an attacker.
We want to study security properties of systems formally. In this talk we consider *confidentiality* through *non-interference*.

- Confidentiality: secret information is not revealed to an attacker.
- Non-interference: varying the secret information does not lead to observably different behavior.
All the variables are divided into two groups.

- *low-sensitivity* variables $l_1, l_2, \ldots$, 
- and *high-sensitivity* variables $h_1, h_2, \ldots$. 

Following this:

- Confidentiality: the data stored in high-sensitivity variables should not leak to low-sensitivity variables.
- Non-interference: changing the values of $h_1, h_2, \ldots$ and running the program does not affect the resulting values of $l_1, l_2, \ldots$.
- Preventing information leaks, e.g., $l_1 \leftarrow ! h_1 + 1$. 

All the variables are divided into two groups.

- *low-sensitivity* variables $l_1, l_2, \ldots$, 
- and *high-sensitivity* variables $h_1, h_2, \ldots$.

Following this:

- Confidentiality: the data stored in high-sensitivity variables should not leak to low-sensitivity variables.
- Non-interference: changing the values of $h_1, h_2, \ldots$ and running the program does not affect the resulting values of $l_1, l_2, \ldots$.
- Preventing information leaks, e.g., $l_1 \leftarrow !h_1 + 1$. 
Type systems for non-interference

Type system where types are annotated with labels from a lattice \( \mathbb{L} \sqsubseteq \mathbb{H} \).

\[
\begin{align*}
\vdash l_i : \text{ref int}^\mathbb{L} & & \quad \vdash h_i : \text{ref int}^\mathbb{H} \\
\vdash e : \text{ref int}^\chi & & \vdash t : \text{int}^\xi & & \xi \sqsubseteq \chi \\
\quad \quad \vdash e \leftarrow t : \text{unit} & & \quad \quad \vdash e + t : \text{int}^{\chi \sqcup \xi}
\end{align*}
\]
Type systems for non-interference

Type system where types are annotated with labels from a lattice $\mathbb{L} \sqsubseteq \mathbb{H}$.

\[ \vdash l_i : \text{ref int}^\mathbb{L} \quad \vdash h_i : \text{ref int}^\mathbb{H} \]

\[ \vdash e : \text{ref int}^\chi \quad \vdash t : \text{int}^\xi \quad \xi \sqsubseteq \chi \quad \vdash e : \text{int}^\chi \quad \vdash t : \text{int}^\xi \]

\[ \vdash e \leftarrow t : \text{unit} \quad \vdash e + t : \text{int}^\chi \sqcup \xi \]

Example:

\[ \vdash l_1 : \text{ref int}^\mathbb{L} \quad \vdash !h_1 : \text{int}^\mathbb{H} \quad \vdash 1 : \text{int}^\mathbb{L} \]

\[ \vdash l_1 : \text{ref int}^\mathbb{L} \quad \vdash !h_1 + 1 : \text{int}^\mathbb{H} \]

\[ \vdash l_1 \leftarrow !h_1 + 1 : \text{unit} \]
Shortcomings of type systems

- Can be extended to cover more PL features (dynamic references, higher-order functions, exceptions), although it is not straightforward.
- Type systems are too weak: in many situations non-interference depends on functional correctness.
Example: value-dependent classifications

\[
\text{let } r = \begin{cases} 
data = \text{ref}(\text{secret}); 
\end{cases} 
is\_\text{classified} = \text{ref}(\text{true}) \\text{in}
\]

\[
\text{while true do}
\]

\[
\text{if } \neg \! r.\text{is\_classified} \quad \text{then out } \leftarrow \! r.\text{data} \\
\text{else } ();
\]

The classification of \(r.\text{data}\) depends on the run-time value \(r.\text{is\_classified}\).
Example: value-dependent classifications

```plaintext
let r = \{ data = ref(secret); is_classified = ref(true) \} in

while true do
  if \neg r.is_classified then
    out <- !r.data
  else
    ();
  r.data <- 0;
  r.is_classified <- false

The classification of r.data depends on the run-time value r.is_classified.
```

- Can we type this program with conventional type systems?
- Is this program secure?
Example: value-dependent classifications

\[
\text{let } r = \begin{cases} 
  \text{data} = \text{ref(secret)}; \\
  \text{is\_classified} = \text{ref(true)} 
\end{cases} \text{ in }
\]

\[\text{while true do}
\]
\[\text{if } \neg ! r.\text{is\_classified} \text{ then out } \leftarrow ! r.\text{data} \text{ else } (); \]
\[r.\text{data} \leftarrow 0; \quad r.\text{is\_classified} \leftarrow \text{false} \]

The classification of \( r.\text{data} \) depends on the run-time value \( r.\text{is\_classified} \).

- Can we type this program with conventional type systems? \text{No}
- Is this program secure? \text{Yes}
Our solution

Solution: semantic typing & program logic

- We give a relational extension of Iris for reasoning about non-interference

- We model a type system using logical relations

- We prove soundness w.r.t. scheduler-independent notion of non-interference
Solution: semantic typing & program logic

- We give a relational extension of Iris for reasoning about non-interference
  - Enables reasoning about fine-grained concurrency
  - Enables reasoning about functional correctness
- We model a type system using logical relations

- We prove soundness w.r.t. scheduler-independent notion of non-interference
Our solution

Solution: semantic typing & program logic

- We give a relational extension of Iris for reasoning about non-interference
  - Enables reasoning about fine-grained concurrency
  - Enables reasoning about functional correctness
- We model a type system using logical relations
  - Compatibility rules for composing typed programs
  - Can “drop down” to the model to prove semantic typing manually
- We prove soundness w.r.t. scheduler-independent notion of non-interference
The logic
Double weakest precondition

The basic component of SeLoC:

\[ \text{dwp } e_1 \& e_2 \{ \Phi \} \]
Double weakest precondition

The basic component of SeLoC:

\[ \text{dwp:} e_1 \sqcap e_2 \{ \Phi \} \]

- Any two runs of the LHS and the RHS are in a bisimulation and results satisfy the postcondition.
Double weakest precondition

The basic component of SeLoC:

\[
dwp e_1 \& e_2 \{\Phi\}
\]

- Any two runs of the LHS and the RHS are in a bisimulation and results satisfy the postcondition.
Double weakest precondition

The basic component of SeLoC:

\[ \text{dwp } e_1 \& e_2 \{ \Phi \} \]

- Any two runs of the LHS and the RHS are in a bisimulation and results satisfy the postcondition.
- \( e_1 \) and \( e_2 \) have different secret data, but must produce the same public output.
Double weakest precondition

The basic component of SeLoC:

\[
dwp e_1 \& e_2 \{\Phi\}
\]

- Any two runs of the LHS and the RHS are in a bisimulation and results satisfy the postcondition.
- \(e_1\) and \(e_2\) have different secret data, but must produce the same public output.
- Left-hand side and right-hand side resources: \(\ell_1 \mapsto_L v_1\) and \(\ell_2 \mapsto_R v_2\).
Double weakest precondition

The basic component of SeLoC:

\[ \text{dwp } e_1 \& e_2 \{ \Phi \} \]

- Any two runs of the LHS and the RHS are in a bisimulation and results satisfy the postcondition.
- \( e_1 \) and \( e_2 \) have different secret data, but must produce the same public output.
- Left-hand side and right-hand side resources: \( \ell_1 \mapsto_L v_1 \) and \( \ell_2 \mapsto_R v_2 \).
- Soundness statement:

\[
(\forall h_1,\ h_2 \in \mathbb{Z}.\ I_{\text{out}} \vdash \text{dwp } e[h_1/x] \& e[h_2/x] \{ v_1 v_2.\ v_1 = v_2 \}) \implies e \text{ is secure}
\]

\[
I_{\text{out}} \triangleq \exists v \in \mathbb{Z}.\ \text{out} \mapsto_L v * \text{out} \mapsto_R v
\]
Example: value-dependent classifications

Let \( \text{prog secret out} \) be

\[
\begin{align*}
\text{let } r &= \begin{cases} 
\text{data} = \text{ref}(\text{secret}); \\
\text{is\_classified} = \text{ref}(\text{true})
\end{cases} 
\end{align*}
\]

\[
\text{in}
\]

while \text{true} do

if \neg ! \text{r.is\_classified} then

out ← ! \text{r.data}

else

();

r.data ← 0;

r.is\_classified ← \text{false}

Statement we want to prove:

\[
\forall h_1, h_2. \text{l_out} \vdash \text{dwp prog } h_1 \text{ out } \& \text{prog } h_2 \text{ out } \{ v_1 v_2. v_1 = v_2 \}
\]

\[
\text{l_out} \triangleq \exists v \in \mathbb{Z}. \text{out} \leftrightarrow_{L} v \ast \text{out} \leftrightarrow_{R} v
\]
Proof rules

\[
\frac{e_1 \rightarrow_{pure} e'_1 \quad e_2 \rightarrow_{pure} e'_2}{\text{dwp } e'_1 \& e'_2 \{ \Phi \}} \quad \frac{\text{dwp } e_1 \& e_2 \{ \Phi \}}{\text{dwp } e'_1 \& e'_2 \{ \Phi \}}
\]

\[
\frac{\text{dwp } e_1 \& e_2 \{ v_1 \, v_2. \text{dwp } K_1[v_1] \& K_2[v_2] \{ \Phi \} \}}{\text{dwp } K_1[e_1] \& K_2[e_2] \{ \Phi \}}
\]
Proof rules

\[
\frac{e_1 \rightarrow_{\text{pure}} e'_1 \quad e_2 \rightarrow_{\text{pure}} e'_2 \quad \triangledown \text{dwp } e'_1 \& e'_2 \{\Phi\}}{\text{dwp } e_1 \& e_2 \{\Phi\}} \quad \frac{\text{dwp } e_1 \& e_2 \{v_1 v_2. \text{dwp } K_1[v_1] \& K_2[v_2] \{\Phi}\}}{\text{dwp } K_1[e_1] \& K_2[e_2] \{\Phi\}}
\]

\[
\frac{\ell_1 \xrightarrow{q}^{L} v_1 \quad \ell_2 \xrightarrow{q}^{R} v_2 \quad (\ell_1 \xrightarrow{q}^{L} v_1 \ast \ell_2 \xrightarrow{q}^{R} v_2 \ast \Phi(v_1, v_2))}{\text{dwp } !\ell_1 \& !\ell_2 \{\Phi\}}
\]
Proof rules

\[
\begin{align*}
e_1 & \rightarrow_{\text{pure}} e'_1 & e_2 & \rightarrow_{\text{pure}} e'_2 & \triangleright \text{dwp } e'_1 \land e'_2 \{ \Phi \} \\
\text{dwp } e_1 \land e_2 \{ \Phi \} & & \text{dwp } e_1 \land e_2 \{ v_1, v_2. \text{dwp } K_1[v_1] \land K_2[v_2] \{ \Phi \} \} & & \text{dwp } K_1[e_1] \land K_2[e_2] \{ \Phi \}
\end{align*}
\]
Proof rules

\[
\begin{align*}
  e_1 & \rightarrow_{\text{pure}} e'_1 & e_2 & \rightarrow_{\text{pure}} e'_2 & \triangleright \text{dwp } e'_1 & \& & e'_2 & \{ \Phi \} \\
  \text{dwp } e_1 & \& & e_2 & \{ \Phi \} & \quad & \text{dwp } e_1 & \& & e_2 & \{ v_1, v_2. \text{dwp } K_1[v_1] & \& & K_2[v_2] & \{ \Phi \} \} \\
  \text{dwp } K_1[e_1] & \& & K_2[e_2] & \{ \Phi \} \\
\end{align*}
\]

\[
\begin{align*}
  \text{wp}_L e_1 & \{ \Psi_1 \} & \text{wp}_R e_2 & \{ \Psi_2 \} & (\forall v_1, v_2. \Psi_1(v_1) \ast \Psi_2(v_2) \rightarrow \triangleright \Phi(v_1, v_2)) \\
  \text{dwp } e_1 & \& & e_2 & \{ \Phi \} \\
\end{align*}
\]

for atomic \(e_1, e_2\) that do not fork off new threads
Proof rules

\[
\begin{align*}
\text{dwp } e_1 & \rightarrow_{\text{pure}} e_1' & \text{dwp } e_2 & \rightarrow_{\text{pure}} e_2' & \triangleright \text{dwp } e_1' \land e_2' \{ \Phi \} \\
\text{dwp } e_1 & \land e_2 \{ \Phi \} & \text{dwp } e_1 & \land e_2 \{ \Phi \} & \text{dwp } K_1[ e_1 ] \land K_2[ e_2 ] \{ \Phi \}
\end{align*}
\]

\[
\begin{align*}
\text{wp}_L e_1 \{ \Psi_1 \} & \quad \text{wp}_R e_2 \{ \Psi_2 \} & (\forall v_1, v_2. \Psi_1(v_1) \ast \Psi_2(v_2) \rightarrow \triangleright \Phi(v_1, v_2)) \\
\text{dwp } e_1 & \land e_2 \{ \Phi \}
\end{align*}
\]

for atomic \( e_1, e_2 \) that do not fork off new threads

\[
\begin{align*}
\triangleright \text{dwp } e_1 & \land e_2 \{ \text{True} \} & \triangleright \Phi()() \\
\text{dwp } (\text{fork } \{ e_1 \}) \land (\text{fork } \{ e_2 \}) \{ \Phi \}
\end{align*}
\]
Proof rules

\[
\begin{align*}
\frac{e_1 \rightarrow_{\text{pure}} e'_1 \quad e_2 \rightarrow_{\text{pure}} e'_2}{\text{dwp } e_1 \& e_2 \{ \Phi \}}
\quad \frac{\text{dwp } e_1 \& e_2 \{ \Phi \}}{\text{dwp } e'_1 \& e'_2 \{ \Phi \}}
\quad \frac{\text{dwp } e'_1 \& e'_2 \{ \Phi \}}{\text{dwp } e_1 \& e_2 \{ \Phi \}}
\quad \frac{\text{dwp } e_1 \& e_2 \{ \Phi \}}{\text{dwp } K_1[e_1] \& K_2[e_2] \{ \Phi \}}
\end{align*}
\]

\[
\begin{align*}
\frac{\text{wp}_L e_1 \{ \Psi_1 \} \quad \text{wp}_R e_2 \{ \Psi_2 \}}{\text{dwp } e_1 \& e_2 \{ \Phi \}}
\quad (\forall v_1, v_2. \, \Psi_1(v_1) \ast \Psi_2(v_2) \rightarrow \Phi(v_1, v_2))
\end{align*}
\]

for atomic \( e_1, e_2 \) that do not fork off new threads

\[
\begin{align*}
\frac{\text{dwp } e_1 \& e_2 \{ \text{True} \}}{\text{dwp } (\text{fork } \{ e_1 \}) \& (\text{fork } \{ e_2 \}) \{ \Phi \}}
\quad \frac{\text{atomic } e_1, e_2}{\text{dwp } e_1 \& e_2 \{ \Phi \}}
\quad \frac{\text{atomic } e_1, e_2}{\text{dwp } e_1 \& e_2 \{ \Phi \}}
\end{align*}
\]

\[
\begin{align*}
\frac{(\forall v_1, v_2. \, \Psi_1(v_1) \ast \Psi_2(v_2) \rightarrow \Phi(v_1, v_2))}{\text{atomic } e_1, e_2}
\quad \frac{\text{atomic } e_1, e_2}{\text{dwp } e_1 \& e_2 \{ \Phi \}}
\end{align*}
\]
Proof of the example: value-dependent classifications

\[
\text{let } r = \begin{cases} 
\text{data} = \text{ref}(\text{secret}); \\
\text{is\_classified} = \text{ref}(\text{true}) 
\end{cases} \text{ in }
\]

\[
\text{while } \text{true} \text{ do }
\]

\[
\text{if } \neg \! r.\text{is\_classified} \text{ then out } \leftarrow \! r.\text{data} \\
\text{else ()}; \\
\text{r.data } \leftarrow 0; \\
\text{r.is\_classified } \leftarrow \text{false}
\]

Classified \rightarrow \text{Intermediate} \rightarrow \text{Declassified}
Proof of the example: value-dependent classifications

\[
\text{let } r = \begin{cases} 
\text{data} = \text{ref}(\text{secret}); \\
\text{is\_classified} = \text{ref}(\text{true}) 
\end{cases} \text{ in }
\]

\[
\text{while } \text{true} \text{ do }
\]

\[
\text{if } \neg \! r.\text{is\_classified} \text{ then } \text{out} \leftarrow \! r.\text{data} \\
\text{else } ()
\]

\[
r.\text{data} \leftarrow 0; \quad r.\text{is\_classified} \leftarrow \text{false}
\]

Classified \rightarrow \text{Intermediate} \rightarrow \text{Declassified}
Proof of the example: value-dependent classifications

\[
\text{let } r = \begin{cases} 
\text{data} = \text{ref(secret)}; \\
\text{is\_classified} = \text{ref(true)} 
\end{cases} \text{ in} \\
\text{while true do} \\
\text{if } \neg r.\text{is\_classified} \text{ then } \text{out} \leftarrow r.\text{data} \\
\text{then out} \leftarrow !r.\text{data} \text{ else } (); \\
\text{r.\text{data}} \leftarrow 0; \\
\text{r.\text{is\_classified}} \leftarrow \text{false} \\
\text{r.\text{is\_classified}} \leftarrow \text{false}
\]

Classified → Intermediate → Declassified
Proof of the example: value-dependent classifications

\[
\text{let } r = \begin{cases}
  \text{data} &= \text{ref}(\text{secret}); \\
  \text{is\_classified} &= \text{ref}(\text{true})
\end{cases} \in
\]

\[\begin{array}{ll}
\text{while } \text{true} \text{ do} & \\
\text{if } \neg \neg \! r.\text{is\_classified} & r.\text{data} \leftarrow 0; \\
\text{then } \text{out} \leftarrow \! r.\text{data} & r.\text{is\_classified} \leftarrow \text{false} \\
\text{else } () & \\
\end{array}\]

Classified \rightarrow \text{Intermediate} \rightarrow \text{Declassified}
Proof of the example: value-dependent classifications

\[(\text{in\_state(Classified)}) \ast \exists i_1, i_2. \ r_1.is\_classified \mapsto_L \text{true} \ast \\ r_2.is\_classified \mapsto_R \text{true} \ast r_1.data \mapsto_L i_1 \ast r_2.data \mapsto_R i_2)\]

\[\lor (\text{in\_state(Intermediate)}) \ast \exists i. \ r_1.is\_classified \mapsto_L \text{true} \ast \\ r_2.is\_classified \mapsto_R \text{true} \ast r_1.data \mapsto_L i \ast r_2.data \mapsto_R i)\]

\[\lor (\text{in\_state(Declassified)}) \ast \exists i. \ r_1.is\_classified \mapsto_L \text{false} \ast \\ r_2.is\_classified \mapsto_R \text{false} \ast r_1.data \mapsto_L i \ast r_2.data \mapsto_R i)\]
Typing
Semantic typing

We build a type system as an abstraction on top of the logic.

The value-dependent classification example can be “typed” semantically:

\[
\vdash \text{prog \ out \ secret} : \text{unit} \times \text{unit}
\]
We build a type system as an abstraction on top of the logic.
The value-dependent classification example can be “typed” semantically:

\[
\begin{align*}
\vdash prog \ out \ secret : \text{unit} \times \text{unit} & \quad \vdash e : \text{unit} \\
\hline
\vdash prog \ out \ secret ; e : \text{unit}
\end{align*}
\]
Semantic typing

We build a type system as an abstraction on top of the logic.

The value-dependent classification example can be "typed" semantically:

\[
\begin{align*}
\vdash \text{prog} \ \text{out} \ \text{secret} : \text{unit} \times \text{unit} & \quad \vdash e : \text{unit} \\
\vdash \text{prog} \ \text{out} \ \text{secret} ; e : \text{unit}
\end{align*}
\]

Grammar of types:

\[
\tau \in \text{Type} ::= \text{unit} \mid \text{int}^{\chi} \mid \text{bool}^{\chi} \mid \tau \times \tau' \mid \text{ref} \ \tau \mid (\tau \to \tau')^{\chi}
\]

Typing judgements:

\[
\Gamma \vdash e : \tau
\]
Logical relations

We follow the usual structure of logical relations in Iris: from $\Gamma \vdash e : \tau$ to $\Gamma \models e : \tau$.

$$\Gamma \models e : \tau \triangleq \forall \gamma \in \llbracket \Gamma \rrbracket. \ dwp \ \gamma_1(e) \ & \gamma_2(e) \ \{ \llbracket \tau \rrbracket \}$$
Logical relations

We follow the usual structure of logical relations in Iris: from $\Gamma \vdash e : \tau$ to $\Gamma \models e : \tau$.

$$\Gamma \models e : \tau \triangleq \forall \gamma \in [\Gamma]. \ dwp \ \gamma_1(e) \ \& \ \gamma_2(e) \ \{[\tau]\}$$

Types are interpreted as *relations on values* $[[\tau]] : \text{Val} \times \text{Val} \to \text{Prop}$.

$$[[\text{unit}]](\nu_1, \nu_2) \triangleq \nu_1 = \nu_2 = ()$$
Logical relations

We follow the usual structure of logical relations in Iris: from $\Gamma \vdash e : \tau$ to $\Gamma \models e : \tau$.

\[
\Gamma \models e : \tau \triangleq \forall \gamma \in \llbracket \Gamma \rrbracket . \text{dwp } \gamma_1(e) \& \gamma_2(e) \{\llbracket \tau \rrbracket \}
\]

Types are interpreted as relations on values $\llbracket \tau \rrbracket : \text{Val} \times \text{Val} \to \text{Prop}$.

\[
\llbracket \text{unit} \rrbracket (v_1, v_2) \triangleq v_1 = v_2 = ()
\]

\[
\llbracket \text{int}^\chi \rrbracket (v_1, v_2) \triangleq v_1, v_2 \in \mathbb{Z} \ast (\chi = \text{L} \to v_1 = v_2)
\]
Logical relations

We follow the usual structure of logical relations in Iris: from $\Gamma \vdash e : \tau$ to $\Gamma \models e : \tau$.

$$\Gamma \models e : \tau \triangleq \forall \gamma \in \llbracket \Gamma \rrbracket. \text{dwp} \; \gamma_1(e) \land \gamma_2(e) \{\llbracket \tau \rrbracket\}$$

Types are interpreted as *relations on values* $\llbracket \tau \rrbracket : \text{Val} \times \text{Val} \to \text{Prop}$.

$$\llbracket \text{unit} \rrbracket(v_1, v_2) \triangleq v_1 = v_2 = ()$$

$$\llbracket \text{int}^\chi \rrbracket(v_1, v_2) \triangleq v_1, v_2 \in \mathbb{Z}^* (\chi = \text{L} \to v_1 = v_2)$$

$$\llbracket \text{ref } \tau \rrbracket(v_1, v_2) \triangleq v_1, v_2 \in \text{Loc}^* \exists w_1 w_2. v_1 \mapsto_L w_1 \ast v_2 \mapsto_R w_2 \ast \llbracket \tau \rrbracket(w_1, w_2)^{N. (v_1, v_2)}$$
Logical relations

We follow the usual structure of logical relations in Iris: from $\Gamma \vdash e : \tau$ to $\Gamma \models e : \tau$.

$$\Gamma \models e : \tau \triangleq \forall \gamma \in \llbracket \Gamma \rrbracket. \text{dwp} \gamma_1(e) \land \gamma_2(e) \{\llbracket \tau \rrbracket\}$$

Types are interpreted as *relations on values* $\llbracket \tau \rrbracket : \text{Val} \times \text{Val} \to \text{Prop}$.

\[
\begin{align*}
\llbracket \text{unit} \rrbracket(v_1, v_2) & \triangleq v_1 = v_2 = () \\
\llbracket \text{int}^\chi \rrbracket(v_1, v_2) & \triangleq v_1, v_2 \in \mathbb{Z}^\ast (\chi = \text{L} \to v_1 = v_2) \\
\llbracket \text{ref } \tau \rrbracket(v_1, v_2) & \triangleq v_1, v_2 \in \text{Loc}^\ast \exists w_1 w_2. v_1 \mapsto_L w_1 * v_2 \mapsto_R w_2 * \llbracket \tau \rrbracket(w_1, w_2) \overset{\text{N.}}{\to} (v_1, v_2) \\
\llbracket \Gamma \rrbracket(\gamma) & \triangleq \forall x. \llbracket \Gamma(x) \rrbracket(\gamma_1(x), \gamma_2(x))
\end{align*}
\]
Logical relations

Compatibility lemmas:

\[ \vdash e : \text{ref } \tau \quad \text{dwp } e \& e' \{\lbrack \text{ref } \tau \rbrack\} \]

\[ \vdash !e : \tau \quad \text{dwp } !e \& !e' \{\lbrack \tau \rbrack\} \]

Fundamental property:

\[ \Gamma \vdash e : \tau \Rightarrow \Gamma \mid \mid = e : \tau \]

Soundness:

\[ x : \text{int } H \vdash e : \text{int } L \Rightarrow e \text{ is secure} \]

We prove the soundness of the type system using the soundness of dwp.
Logical relations

Compatibility lemmas:

\[ \vdash e : \text{ref } \tau \quad \text{dwp } e \& e' \{[[\text{ref } \tau]]\} \]

\[ \vdash !e : \tau \quad \text{dwp } !e \& !e' \{[[\tau]]\} \]

Fundamental property:

\[ \Gamma \vdash e : \tau \implies \Gamma \models e : \tau \]
Logical relations

Compatibility lemmas:

\[
\begin{align*}
\Gamma \vdash e : \text{ref} \, \tau & \quad \text{dwp} \ e & \& e' \ {\llbracket \text{ref} \, \tau \rrbracket} \\
\Gamma \vdash ! e : \tau & \quad \text{dwp} \ ! e & \& ! e' \ {\llbracket \tau \rrbracket}
\end{align*}
\]

Fundamental property:

\[
\Gamma \vdash e : \tau \implies \Gamma \models e : \tau
\]

Soundness:

\[
x : \text{int}^H \vdash e : \text{int}^L \implies e \text{ is secure}
\]

We prove the soundness of the type system using the soundness of dwp.
Soundness
DWP is defined as a binary variant of the ordinary WP, but the soundness statement that we want to get is different from the WP adequacy.
DWP is defined as a binary variant of the ordinary WP, but the soundness statement that we want to get is different from the WP adequacy.

\[(\forall h_1, h_2 \in \mathbb{Z}. I_{out} \vdash dwp e[h_1/x] \& e[h_2/x] \{v_1 = v_2\}) \implies e \text{ is secure}\]
DWP is defined as a binary variant of the ordinary WP, but the soundness statement that we want to get is different from the WP adequacy.

\[(\forall h_1, h_2 \in \mathbb{Z}. \ l_{out} \vdash \mathrm{dwp} \ e[h_1/x] \& e[h_2/x] \{v_1 \leftrightarrow v_2. \ v_1 = v_2\}) \implies e \text{ is secure}\]

We take one of the standard definitions of non-interference for concurrent programs:

\[e \text{ is secure} \triangleq (\forall h_1, h_2. \ e[h_1/x] \equiv \mathcal{L} e[h_2/x])\]

\[e_1 \equiv \mathcal{L} e_2 \triangleq \text{for any } \sigma \text{ s.t. } \sigma(out) \in \mathbb{Z}, \text{ the configurations } (e_1, \sigma) \text{ and } (e_2, \sigma) \text{ are related by a strong-low bisimulation} \ (\text{Sabelfeld & Sands, 2000}).\]
Strong-low bisimulation

*Strong-low bisimulation* is a partial equivalence relation $\mathcal{R}$ on configurations such that

- If $(\vec{v}e, \sigma_1) \mathcal{R} (\vec{w}t, \sigma_2)$, then $v = w$;
**Strong-low bisimulation** is a partial equivalence relation \( \mathcal{R} \) on configurations such that

- If \( (v\vec{e}, \sigma_1) \mathcal{R} (w\vec{t}, \sigma_2) \), then \( v = w \);
- If \( (\vec{e}, \sigma_1) \mathcal{R} (\vec{t}, \sigma_2) \), then \( |\vec{e}| = |\vec{t}| \) and \( \sigma_1(\text{out}) = \sigma_2(\text{out}) \);
Strong-low bisimulation

Strong-low bisimulation is a partial equivalence relation $\mathcal{R}$ on configurations such that

- If $(\vec{e}, \sigma_1) \mathcal{R} (\vec{t}, \sigma_2)$, then $\nu = \omega$;
- If $(\vec{e}, \sigma_1) \mathcal{R} (\vec{t}, \sigma_2)$, then $|\vec{e}| = |\vec{t}|$ and $\sigma_1(out) = \sigma_2(out)$;
- The bisimulation condition holds:

$$
\begin{align*}
(e_i, \sigma_1) &\quad (e_0 \ldots e_i \ldots, \sigma_1) \quad \mathcal{R} \quad (t_0 \ldots t_i \ldots, \sigma_2) & (t_i, \sigma_2) \\
\downarrow & \quad \downarrow \\
(e'_i \vec{e}, \sigma'_1) & (e_0 \ldots e'_i \vec{e} \ldots, \sigma'_1) 
\end{align*}
$$
**Strong-low bisimulation** is a partial equivalence relation $\mathcal{R}$ on configurations such that

- If $(v \vec{e}, \sigma_1) \mathcal{R} (w \vec{t}, \sigma_2)$, then $v = w$;
- If $(\vec{e}, \sigma_1) \mathcal{R} (\vec{t}, \sigma_2)$, then $|\vec{e}| = |\vec{t}|$ and $\sigma_1(\text{out}) = \sigma_2(\text{out})$;
- The bisimulation condition holds:

\[
\begin{align*}
(e_i, \sigma_1) & \overset{\mathcal{R}}{\longrightarrow} (e_0 \ldots e_i \ldots, \sigma_1) \\
\downarrow & \quad \quad \quad \quad \quad \downarrow \\
(e_i' \vec{e}, \sigma'_1) & \overset{\mathcal{R}}{\longrightarrow} (e_0 \ldots e_i' \vec{e} \ldots, \sigma'_1) \quad \quad \quad (t_0 \ldots t_i \ldots, \sigma_2) \quad \quad \quad \quad \quad \quad (t_i, \sigma_2) \\
\downarrow & \quad \quad \quad \quad \quad \downarrow \\
(t_i' \vec{t}, \sigma'_2) & \quad \quad \quad \quad \quad \quad (t_0 \ldots t_i' \vec{t} \ldots, \sigma'_2) \quad \quad \quad \quad \quad \quad (t_i' \vec{t}, \sigma'_2)
\end{align*}
\]
Constructing the bisimulation

The specific bisimulation that we construct is $\mathcal{R}^*$ where

$$(e_0 e_1 \ldots e_m, \sigma_1) \mathcal{R} (t_0 t_1 \ldots t_m, \sigma_2) \triangleq \text{there exists } n \text{ such that}$$

$$\text{True} \vdash \left(\top \implies ^\top \implies ^\emptyset \implies ^\emptyset \implies ^\top \right)^n \implies ^\top \text{SR}(\sigma_1, \sigma_2) \ast l_{out} \ast$$

$$\text{dwp } e_0 \& t_0 \{v_1 v_2. v_1 = v_2\} \ast$$

$$\ast_{1 \leq i \leq m.} \text{dwp } e_i \& t_i \{\text{True}\}$$

Proof that $\mathcal{R}^*$ is a bisimulation relies on the soundness of the update modality in Iris.
The specific bisimulation that we construct is $\mathcal{R}^*$ where

\[
(e_0 e_1 \ldots e_m, \sigma_1) \mathcal{R} (t_0 t_1 \ldots t_m, \sigma_2) \triangleq \text{there exists } n \text{ such that }
\]

\[
\text{True} \vdash \left( \top \not\Rightarrow \emptyset \not\Rightarrow \top \right)^n \not\Rightarrow \top \text{SR}(\sigma_1, \sigma_2) * l_{out} *
\]

\[
dwp e_0 & t_0 \{ v_1 v_2. v_1 = v_2 \} *
\]

\[
\star_{1 \leq i \leq m.} \text{dwp } e_i & t_i \{ \text{True} \}
\]

Proof that $\mathcal{R}^*$ is a bisimulation relies on the soundness of the update modality in Iris.

We get

\[
(l_{out} \vdash \text{dwp } e_1 & e_2 \{ v_1 v_2. v_1 = v_2 \}) \implies (e_1, \sigma) \mathcal{R}^* (e_2, \sigma)
\]
Deliberate information release
Deliberate information release

Problem:

- “Vanilla” non-interference does not allow information flow high-sensitivity input to low-sensitivity output at all.
- In many real-world situations we want to permit some information leakage in a controlled way.

We can encode a specific form of deliberate information release in SeLoC.
Consider the following example (Constanzo & Shao, 2014):

```
let cal A = iter (λ i v. if (v == 0) then print(i)) A
```

- the input list $A$ represents a calendar of Alice
- $A[i]$ contains information about the $i$-th day:
  - $A[i] = 0 \implies$ Alice is free on that day;
  - $A[i] = t \implies$ Alice has a meeting scheduled at time $t$.
- Alice wants to share the days when she is free, but she does not want to share the exact times she is busy
let cal A = iter (λ i v. if (v == 0) then print(i)) A

In what sense the program is secure?
**Specification of the example: Alice’s calendar**

\[
\text{let } \text{cal } A = \text{iter } (\lambda i \ v. \ \text{if } (v == 0) \ \text{then } \text{print}(i)) \ A
\]

In what sense the program is secure?

If the attacker already knows on which days Alice is free, then they do not learn anything more about her calendar.
Specification of the example: Alice’s calendar

\[
\text{let } cal\ A = \text{iter } (\lambda\ i\ v.\ \text{if } (v == 0) \text{ then } \text{print}(i))\ A
\]

In what sense the program is secure?
If the attacker already knows on which days Alice is free, then they do not learn anything more about her calendar.

\[
\forall A_1, A_2.\ A_1 \sim A_2 \vdash \text{dwp cal } A_1 \& \text{cal } A_2 \{v_1 \& v_2.\ v_1 = v_2\}
\]

where \( A_1 \sim A_2 \triangleq \left| A_1 \right| = \left| A_2 \right| \& \forall i.\ A_1[i] = 0 \iff A_2[i] = 0.\)
Further work on deliberate information release

The relation $\sim$ on the calendars can be generalized to arbitrary PERs on values (Sabelfeld & Meyers 2003) and we can readily do that in SeLoC.

Further work:

- Formalizing the soundness statement.
- Integrating into the type system.
- Controlling *where* and *when* the information is released (e.g., the information is leaked only after the password is entered).
SeLoC: logic for proving non-interference of fine-grained concurrent programs.

- Formalized in Iris in Coq.
- A model of an IFC type system with semantic typing.
- Modular HOCAST-style dwp specifications.
Conclusion

SeLoC: logic for proving non-interference of fine-grained concurrent programs.

- Formalized in Iris in Coq.
- A model of an IFC type system with semantic typing.
- Modular HOCAP-style dwp specifications.

Thank you for your attention.