Compositional non-interference

for fine-grained concurrent programs

Dan Frumin (jww Robbert Krebbers & Lars Birkedal)

29 October, 2019 @ Iris Workshop

Radboud University

Formal security

We want to study security properties of systems formally. In this talk we consider *confidentiality* through *non-interference*.

Formal security

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.

Formal security

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.

Language-based approach: When an imperative program is secure?

All the variables are divided into two groups.

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

Language-based approach: When an imperative program is secure?

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 $\mathbf{L} \sqsubseteq \mathbf{H}$.

$$\vdash l_i : \texttt{ref int}^{\mathbf{L}} \qquad \vdash h_i : \texttt{ref int}^{\mathbf{H}}$$

$$\frac{\vdash e : \texttt{ref int}^{\chi} \qquad \vdash t : \texttt{int}^{\xi} \qquad \xi \sqsubseteq \chi}{\vdash e \leftarrow t : \texttt{unit}} \qquad \frac{\vdash e : \texttt{int}^{\chi} \qquad \vdash t : \texttt{int}^{\xi}}{\vdash e + t : \texttt{int}^{\chi \sqcup \xi}}$$

Type systems for non-interference

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

$$\vdash l_i : \texttt{ref int}^{\mathbf{L}} \qquad\qquad \vdash h_i : \texttt{ref int}^{\mathbf{H}}$$

$$\frac{\vdash e : \texttt{ref int}^{\chi} \qquad \vdash t : \texttt{int}^{\xi} \qquad \xi \sqsubseteq \chi}{\vdash e \leftarrow t : \texttt{unit}} \qquad\qquad \frac{\vdash e : \texttt{int}^{\chi} \qquad \vdash t : \texttt{int}^{\xi}}{\vdash e + t : \texttt{int}^{\chi \sqcup \xi}}$$

Example:

$$\frac{\vdash l_1 : \texttt{ref int}^{\mathbf{L}} \qquad \dfrac{\vdash !h_1 : \texttt{int}^{\mathbf{H}} \qquad \vdash 1 : \texttt{int}^{\mathbf{L}}}{\vdash !h_1 + 1 : \texttt{int}^{\mathbf{H}}} \qquad \color{red}{\mathbf{H} \sqsubseteq \mathbf{L}}}{\vdash l_1 \leftarrow !h_1 + 1 : \texttt{unit}}$$

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

$$\textbf{let } r = \left\{ \begin{array}{l} data = \textbf{ref}(secret); \\ is\_classified = \textbf{ref}(\textbf{true}) \end{array} \right\} \textbf{ in}$$

$$\begin{array}{ll|l} \textit{while} & \textbf{true } \textit{do} & \\ & \textbf{if } \neg\, !\, r.is\_classified & r.data \leftarrow 0; \\ & \textbf{then } out \leftarrow\, !\, r.data & r.is\_classified \leftarrow \textbf{false} \\ & \textbf{else } (); & \end{array}$$

The classification of $r.data$ depends on the run-time value $r.is\_classified$.

Example: value-dependent classifications

$$\textbf{let } r = \left\{ \begin{array}{l} data = \textbf{ref}(secret); \\ is\_classified = \textbf{ref}(\textbf{true}) \end{array} \right\} \textbf{ in}$$

$$\begin{array}{ll} while & \textbf{true } do \\ & \textbf{if } \neg\, !\, r.is\_classified \\ & \textbf{then } out \leftarrow\, !\, r.data \\ & \textbf{else } (); \end{array} \;\Bigg\|\; \begin{array}{l} r.data \leftarrow 0; \\ r.is\_classified \leftarrow \textbf{false} \end{array}$$

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

$$\textbf{let } r = \left\{ \begin{array}{l} data = \textbf{ref}(secret); \\ is\_classified = \textbf{ref}(\textbf{true}) \end{array} \right\} \textbf{ in}$$

$$\begin{array}{ll|l} \textit{while} & \textbf{true } \textit{do} & \\ & \textbf{if } \neg\, !\, r.is\_classified & r.data \leftarrow 0; \\ & \textbf{then } out \leftarrow\, !\, r.data & r.is\_classified \leftarrow \textbf{false} \\ & \textbf{else } (); & \end{array}$$

The classification of $r.data$ depends on the run-time value $r.is\_classified$.

- Can we type this program with conventional type systems? *No*
- Is this program secure? *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

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
- 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:

$$\mathrm{dwp}\ e_1 \ \&\ e_2\ \{\Phi\}$$

- 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.

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:

$$\text{dwp } e_1 \,\&\, e_2 \,\{\Phi\}$$

- 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: $\ell_1 \mapsto_{\mathsf{L}} v_1$ and $\ell_2 \mapsto_{\mathsf{R}} v_2$.

Double weakest precondition

The basic component of SeLoC:

$$\text{dwp } e_1 \mathbin{\&} e_2 \; \{\Phi\}$$

- 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: $\ell_1 \mapsto_{\mathsf{L}} v_1$ and $\ell_2 \mapsto_{\mathsf{R}} v_2$.
- Soundness statement:

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

$$I_{out} \triangleq \boxed{\exists v \in \mathbb{Z}.\ out \mapsto_{\mathsf{L}} v * out \mapsto_{\mathsf{R}} v}$$

Example: value-dependent classifications

Let *prog secret out* be

$$\textbf{let } r = \begin{Bmatrix} data = \textbf{ref}(secret); \\ is\_classified = \textbf{ref}(\textbf{true}) \end{Bmatrix} \textbf{ in}$$

$$\begin{array}{l|l} \textit{while} \quad \textbf{true} \textit{ do} & \\ \quad \textbf{if} \neg\, !\, r.is\_classified & r.data \leftarrow 0; \\ \quad \textbf{then } out \leftarrow\, !\, r.data & r.is\_classified \leftarrow \textbf{false} \\ \quad \textbf{else } (); & \end{array}$$

Statement we want to prove:

$$\forall h_1, h_2.\ I_{out} \vdash \mathrm{dwp}\ prog\ \ h_1\ \ out\ \&\ prog\ \ h_2\ \ out\ \{v_1 v_2.\ v_1 = v_2\}$$

$$I_{out} \triangleq \boxed{\exists v \in \mathbb{Z}.\ out \mapsto_{\mathrm{L}}\ v * out \mapsto_{\mathrm{R}}\ v}$$

Proof rules

$$\frac{e_1 \rightarrow_{\text{pure}} e_1' \qquad e_2 \rightarrow_{\text{pure}} e_2' \qquad \triangleright \, \text{dwp } e_1' \,\&\, e_2' \, \{\Phi\}}{\text{dwp } e_1 \,\&\, e_2 \, \{\Phi\}}$$

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

Proof rules

$$\frac{e_1 \to_{\mathsf{pure}} e_1' \qquad e_2 \to_{\mathsf{pure}} e_2' \qquad \triangleright \mathsf{dwp}\ e_1' \mathbin{\&} e_2' \,\{\Phi\}}{\mathsf{dwp}\ e_1 \mathbin{\&} e_2 \,\{\Phi\}} \qquad \frac{\mathsf{dwp}\ e_1 \mathbin{\&} e_2 \,\{v_1\, v_2.\, \mathsf{dwp}\ K_1[\, v_1\,] \mathbin{\&} K_2[\, v_2\,] \,\{\Phi\}\}}{\mathsf{dwp}\ K_1[\, e_1\,] \mathbin{\&} K_2[\, e_2\,] \,\{\Phi\}}$$

$$\frac{\ell_1 \overset{q}{\mapsto}_{\mathsf{L}} v_1 \qquad \ell_2 \overset{q}{\mapsto}_{\mathsf{R}} v_2 \qquad (\ell_1 \overset{q}{\mapsto}_{\mathsf{L}} v_1 * \ell_2 \overset{q}{\mapsto}_{\mathsf{R}} v_2 \mathrel{-\!\!*} \Phi(v_1, v_2))}{\mathsf{dwp}\ !\,\ell_1 \mathbin{\&} !\,\ell_2 \,\{\Phi\}}$$

Proof rules

$$\frac{e_1 \to_{\text{pure}} e_1' \qquad e_2 \to_{\text{pure}} e_2' \qquad \triangleright \operatorname{dwp} e_1' \mathbin{\&} e_2' \{\Phi\}}{\operatorname{dwp} e_1 \mathbin{\&} e_2 \{\Phi\}} \qquad \frac{\operatorname{dwp} e_1 \mathbin{\&} e_2 \{v_1\, v_2.\, \operatorname{dwp} K_1[\, v_1\,] \mathbin{\&} K_2[\, v_2\,] \{\Phi\}\}}{\operatorname{dwp} K_1[\, e_1\,] \mathbin{\&} K_2[\, e_2\,] \{\Phi\}}$$

$$\frac{\ell_1 \overset{q}{\mapsto}_{\mathsf{L}} v_1 \qquad \ell_2 \overset{q}{\mapsto}_{\mathsf{R}} v_2 \qquad (\ell_1 \overset{q}{\mapsto}_{\mathsf{L}} v_1 * \ell_2 \overset{q}{\mapsto}_{\mathsf{R}} v_2 \mathrel{-\!\!*} \Phi(v_1, v_2))}{\operatorname{dwp} \,!\,\ell_1 \mathbin{\&} \,!\,\ell_2 \{\Phi\}}$$

$$\frac{\ell_1 \overset{q_1}{\mapsto}_{\mathsf{L}} v_1 \qquad \ell_2 \mapsto_{\mathsf{R}} - \qquad (\ell_1 \overset{q_1}{\mapsto}_{\mathsf{L}} v_1 * \ell_2 \mapsto_{\mathsf{R}} v_2 \mathrel{-\!\!*} \Phi(v_1, ()))}{\operatorname{dwp} \,!\,\ell_1 \mathbin{\&} \ell_2 \leftarrow v_2 \{\Phi\}}$$

Proof rules

$$\frac{e_1 \rightarrow_{\mathsf{pure}} e_1' \qquad e_2 \rightarrow_{\mathsf{pure}} e_2' \qquad \triangleright \mathsf{dwp}\ e_1' \mathbin{\&} e_2' \{\Phi\}}{\mathsf{dwp}\ e_1 \mathbin{\&} e_2 \{\Phi\}} \qquad \frac{\mathsf{dwp}\ e_1 \mathbin{\&} e_2 \{v_1\, v_2.\ \mathsf{dwp}\ K_1[\, v_1\,] \mathbin{\&} K_2[\, v_2\,] \{\Phi\}\}}{\mathsf{dwp}\ K_1[\, e_1\,] \mathbin{\&} K_2[\, e_2\,] \{\Phi\}}$$

$$\frac{\mathsf{wp}_{\mathsf{L}}\ e_1 \{\Psi_1\} \qquad \mathsf{wp}_{\mathsf{R}}\ e_2 \{\Psi_2\} \qquad (\forall v_1, v_2.\ \Psi_1(v_1) * \Psi_2(v_2) \mathrel{-\!\!*} \triangleright \Phi(v_1, v_2))}{\mathsf{dwp}\ e_1 \mathbin{\&} e_2 \{\Phi\}}$$

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

Proof rules

$$\frac{e_1 \rightarrow_{\text{pure}} e_1' \qquad e_2 \rightarrow_{\text{pure}} e_2' \qquad \triangleright \text{dwp } e_1' \,\&\, e_2' \,\{\Phi\}}{\text{dwp } e_1 \,\&\, e_2 \,\{\Phi\}} \qquad \frac{\text{dwp } e_1 \,\&\, e_2 \,\{v_1\, v_2.\, \text{dwp } K_1[\, v_1\,] \,\&\, K_2[\, v_2\,] \,\{\Phi\}\}}{\text{dwp } K_1[\, e_1\,] \,\&\, K_2[\, e_2\,] \,\{\Phi\}}$$

$$\frac{\text{wp}_{\text{L}}\ e_1 \,\{\Psi_1\} \qquad \text{wp}_{\text{R}}\ e_2 \,\{\Psi_2\} \qquad (\forall v_1, v_2.\, \Psi_1(v_1) * \Psi_2(v_2) \twoheadrightarrow \triangleright \Phi(v_1, v_2))}{\text{dwp } e_1 \,\&\, e_2 \,\{\Phi\}}$$

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

$$\frac{\triangleright \text{dwp } e_1 \,\&\, e_2 \,\{\text{True}\} \qquad \triangleright \Phi()()}{\text{dwp } (\textbf{fork } \{e_1\}) \,\&\, (\textbf{fork } \{e_2\}) \,\{\Phi\}}$$

Proof rules

$$\frac{e_1 \to_{\mathsf{pure}} e_1' \qquad e_2 \to_{\mathsf{pure}} e_2' \qquad \triangleright \mathsf{dwp}\ e_1' \mathbin{\&} e_2' \ \{\Phi\}}{\mathsf{dwp}\ e_1 \mathbin{\&} e_2\ \{\Phi\}} \qquad \frac{\mathsf{dwp}\ e_1 \mathbin{\&} e_2\ \{v_1\, v_2.\ \mathsf{dwp}\ K_1[\,v_1\,] \mathbin{\&} K_2[\,v_2\,]\ \{\Phi\}\}}{\mathsf{dwp}\ K_1[\,e_1\,] \mathbin{\&} K_2[\,e_2\,]\ \{\Phi\}}$$

$$\frac{\mathsf{wp}_{\mathsf{L}}\ e_1\ \{\Psi_1\} \qquad \mathsf{wp}_{\mathsf{R}}\ e_2\ \{\Psi_2\} \qquad (\forall v_1, v_2.\ \Psi_1(v_1) * \Psi_2(v_2) \mathrel{-\!\!*} \triangleright \Phi(v_1, v_2))}{\mathsf{dwp}\ e_1 \mathbin{\&} e_2\ \{\Phi\}}$$

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

$$\frac{\triangleright \mathsf{dwp}\ e_1 \mathbin{\&} e_2\ \{\mathsf{True}\} \qquad \triangleright \Phi()()}{\mathsf{dwp}\ (\textbf{fork}\ \{e_1\}) \mathbin{\&} (\textbf{fork}\ \{e_2\})\ \{\Phi\}} \qquad \frac{\begin{array}{c}\mathsf{atomic}\ e_1, e_2 \\ {}^{\top}\!\Rrightarrow^{\mathcal{E}} \mathsf{dwp}_{\mathcal{E}}\ e_1 \mathbin{\&} e_2 \left\{v_1\, v_2.\ {}^{\mathcal{E}}\!\Rrightarrow^{\top} \Phi(v_1, v_2)\right\}\end{array}}{\mathsf{dwp}\ e_1 \mathbin{\&} e_2\ \{\Phi\}}$$

Proof of the example: value-dependent classifications

$$\textbf{let } r = \begin{Bmatrix} data = \textbf{ref}(secret); \\ is\_classified = \textbf{ref}(\textbf{true}) \end{Bmatrix} \textbf{ in}$$

$$\begin{array}{l} while \quad \textbf{true } do \\ \qquad \textbf{if } \neg\, !\, r.is\_classified \\ \qquad \textbf{then } out \leftarrow\, !\, r.data \\ \qquad \textbf{else } (); \end{array} \;\Bigg\|\; \begin{array}{l} r.data \leftarrow 0; \\ r.is\_classified \leftarrow \textbf{false} \end{array}$$

Proof of the example: value-dependent classifications

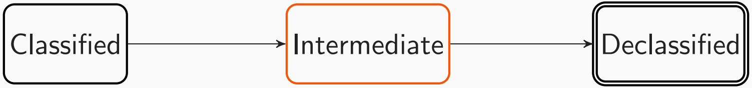
$$\textbf{let } r = \left\{ \begin{array}{l} data = \textbf{ref}(secret); \\ is\_classified = \textbf{ref}(\textbf{true}) \end{array} \right\} \textbf{ in}$$

$$\begin{array}{ll|l} \textit{while} & \textbf{true } \textit{do} & \\ & \textbf{if } \neg\, !\, r.is\_classified & r.data \leftarrow 0; \\ & \textbf{then } out \leftarrow\, !\, r.data & r.is\_classified \leftarrow \textbf{false} \\ & \textbf{else } (); & \end{array}$$

Proof of the example: value-dependent classifications

$$\textbf{let } r = \left\{ \begin{array}{l} data = \textbf{ref}(secret); \\ is\_classified = \textbf{ref}(\textbf{true}) \end{array} \right\} \textbf{ in}$$

$$\begin{array}{ll|l} while & \textbf{true } do & \\ & \textbf{if } \neg\, !\, r.is\_classified & r.data \leftarrow 0; \\ & \textbf{then } out \leftarrow\, !\, r.data & r.is\_classified \leftarrow \textbf{false} \\ & \textbf{else } (); & \end{array}$$



Proof of the example: value-dependent classifications

$$\textbf{let } r = \left\{ \begin{array}{l} data = \textbf{ref}(secret); \\ is\_classified = \textbf{ref}(\textbf{true}) \end{array} \right\} \textbf{ in}$$

$$\begin{array}{l|l} \textit{while} \quad \textbf{true } do & \\ \quad \textbf{if } \neg\, !\, r.is\_classified & r.data \leftarrow 0; \\ \quad \textbf{then } out \leftarrow\, !\, r.data & r.is\_classified \leftarrow \textbf{false} \\ \quad \textbf{else } (); & \end{array}$$

Proof of the example: value-dependent classifications

$$
\begin{aligned}
&\big(\mathsf{in\_state}(\mathsf{Classified}) * \exists i_1, i_2.\ r_1.\mathit{is\_classified} \mapsto_{\mathsf{L}} \mathbf{true} * \\
&\quad r_2.\mathit{is\_classified} \mapsto_{\mathsf{R}} \mathbf{true} * r_1.\mathit{data} \mapsto_{\mathsf{L}} i_1 * r_2.\mathit{data} \mapsto_{\mathsf{R}} i_2\big) \\
\vee &\big(\mathsf{in\_state}(\mathsf{Intermediate}) * \exists i.\ r_1.\mathit{is\_classified} \mapsto_{\mathsf{L}} \mathbf{true} * \\
&\quad r_2.\mathit{is\_classified} \mapsto_{\mathsf{R}} \mathbf{true} * r_1.\mathit{data} \mapsto_{\mathsf{L}} i * r_2.\mathit{data} \mapsto_{\mathsf{R}} i\big) \\
\vee &\big(\mathsf{in\_state}(\mathsf{Declassified}) * \exists i.\ r_1.\mathit{is\_classified} \mapsto_{\mathsf{L}} \mathbf{false} * \\
&\quad r_2.\mathit{is\_classified} \mapsto_{\mathsf{R}} \mathbf{false} * r_1.\mathit{data} \mapsto_{\mathsf{L}} i * r_2.\mathit{data} \mapsto_{\mathsf{R}} i\big)
\end{aligned}
$$

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*:

$$\models \mathit{prog}\ \mathit{out}\ \mathit{secret} : \mathtt{unit} \times \mathtt{unit}$$

Semantic typing

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

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

$$\frac{\vDash \mathit{prog}\ \ \mathit{out}\ \ \mathit{secret} : \mathtt{unit} \times \mathtt{unit} \qquad \vdash e : \mathtt{unit}}{\vDash \mathit{prog}\ \ \mathit{out}\ \ \mathit{secret}; e : \mathtt{unit}}$$

Semantic typing

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

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

$$\frac{\models \mathit{prog}\ \mathit{out}\ \mathit{secret} : \mathtt{unit} \times \mathtt{unit} \qquad \vdash e : \mathtt{unit}}{\models \mathit{prog}\ \mathit{out}\ \mathit{secret}; e : \mathtt{unit}}$$

Grammar of types:

$$\tau \in \mathsf{Type} ::= \mathtt{unit} \mid \mathtt{int}^{\chi} \mid \mathtt{bool}^{\chi} \mid \tau \times \tau' \mid \mathtt{ref}\ \tau \mid (\tau \to \tau')^{\chi}$$

Typing judgements:

$$\Gamma \vdash e : \tau$$

Logical relations

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

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

Logical relations

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

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

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

$$[\![\mathtt{unit}]\!](v_1, v_2) \triangleq v_1 = v_2 = ()$$

Logical relations

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

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

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

$$[\![\mathtt{unit}]\!](v_1, v_2) \triangleq v_1 = v_2 = ()$$

$$[\![\mathtt{int}^{\chi}]\!](v_1, v_2) \triangleq v_1, v_2 \in \mathbb{Z} * (\chi = \mathbf{L} \to v_1 = v_2)$$

Logical relations

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

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

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

$$[\![\mathtt{unit}]\!](v_1, v_2) \triangleq v_1 = v_2 = ()$$

$$[\![\mathtt{int}^{\chi}]\!](v_1, v_2) \triangleq v_1, v_2 \in \mathbb{Z} * (\chi = \mathbf{L} \to v_1 = v_2)$$

$$[\![\mathtt{ref}\ \tau]\!](v_1, v_2) \triangleq v_1, v_2 \in \mathit{Loc} * \boxed{\exists w_1\, w_2.\ v_1 \mapsto_{\mathsf{L}} w_1 * v_2 \mapsto_{\mathsf{R}} w_2 * [\![\tau]\!](w_1, w_2)}^{\mathcal{N}.(v_1, v_2)}$$

Logical relations

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

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

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

$$\begin{aligned}
[\![\mathtt{unit}]\!](v_1, v_2) &\triangleq v_1 = v_2 = () \\
[\![\mathtt{int}^{\chi}]\!](v_1, v_2) &\triangleq v_1, v_2 \in \mathbb{Z} * (\chi = \mathbf{L} \to v_1 = v_2) \\
[\![\mathtt{ref}\ \tau]\!](v_1, v_2) &\triangleq v_1, v_2 \in \mathit{Loc} * \boxed{\exists w_1\, w_2.\ v_1 \mapsto_{\mathrm{L}} w_1 * v_2 \mapsto_{\mathrm{R}} w_2 * [\![\tau]\!](w_1, w_2)}^{\mathcal{N}.(v_1, v_2)} \\
[\![\Gamma]\!](\gamma) &\triangleq \forall x.\ [\![\Gamma(x)]\!](\gamma_1(x), \gamma_2(x))
\end{aligned}$$

Logical relations

Compatibility lemmas:

$$\frac{\vdash e : \texttt{ref}\ \tau}{\vdash\ !\,e : \tau} \qquad\qquad \frac{\mathsf{dwp}\ e \,\&\, e'\ \{[\![\texttt{ref}\ \tau]\!]\}}{\mathsf{dwp}\ !\,e \,\&\, !\,e'\ \{[\![\tau]\!]\}}$$

Logical relations

Compatibility lemmas:

$$\frac{\vdash e : \texttt{ref}\ \tau}{\vdash\ !\, e : \tau} \qquad\qquad \frac{\mathsf{dwp}\ e \,\&\, e'\ \{[\![\texttt{ref}\ \tau]\!]\}}{\mathsf{dwp}\ !\, e \,\&\ !\, e'\ \{[\![\tau]\!]\}}$$

Fundamental property:

$$\Gamma \vdash e : \tau \implies \Gamma \models e : \tau$$

Compatibility lemmas:

$$\frac{\vdash e : \texttt{ref}\ \tau}{\vdash\ !\, e : \tau} \qquad\qquad \frac{\text{dwp}\ e\ \&\ e'\ \{[\![\texttt{ref}\ \tau]\!]\}}{\text{dwp}\ !\, e\ \&\ !\, e'\ \{[\![\tau]\!]\}}$$

Fundamental property:

$$\Gamma \vdash e : \tau \implies \Gamma \models e : \tau$$

Soundness:

$$x : \texttt{int}^{\mathbf{H}} \vdash e : \texttt{int}^{\mathbf{L}} \implies e \text{ is secure}$$

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

Soundness

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.

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.

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

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.

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

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

$$e \text{ is secure} \triangleq \forall h_1\ h_2.\ e[h_1/x] \approxeq_L e[h_2/x]$$

$e_1 \approxeq_L e_2 \triangleq$ for any σ s.t. $\sigma(out) \in \mathbb{Z}$, the configurations (e_1, σ) and (e_2, σ) are related by a *strong-low bisimulation* (Sabelfeld & Sands, 2000).

Strong-low bisimulation

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 $v = w$;

Strong-low bisimulation

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 $\sigma_1(out) = \sigma_2(out)$;

Strong-low bisimulation

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 $\sigma_1(out) = \sigma_2(out)$;
- The bisimulation condition holds:

$$
\begin{array}{ccccc}
(e_i, \sigma_1) & & (e_0 \dots e_i \dots, \sigma_1) \ \text{—}\mathcal{R}\text{—}\ (t_0 \dots t_i \dots, \sigma_2) & & (t_i, \sigma_2) \\
\downarrow & & \downarrow \qquad\qquad\qquad\qquad\qquad & & \\
(e_i' \vec{e}, \sigma_1') & & (e_0 \dots e_i' \vec{e} \dots, \sigma_1') \qquad\qquad\qquad & &
\end{array}
$$

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 $\sigma_1(out) = \sigma_2(out)$;
- The bisimulation condition holds:

$$
\begin{array}{ccccccc}
(e_i, \sigma_1) & \quad & (e_0 \ldots e_i \ldots, \sigma_1) & \text{---}\ \mathcal{R}\ \text{---} & (t_0 \ldots t_i \ldots, \sigma_2) & \quad & (t_i, \sigma_2) \\
\downarrow & & \downarrow & & \downarrow & & \downarrow \\
(e_i' \vec{e}, \sigma_1') & & (e_0 \ldots e_i' \vec{e} \ldots, \sigma_1') & \text{- -}\ \mathcal{R}\ \text{- -} & (t_0 \ldots t_i' \vec{t} \ldots, \sigma_2') & & (t_i' \vec{t}, \sigma_2')
\end{array}
$$

Constructing the bisimulation

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

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

$$\text{True} \vdash \left({}^{\top}\!\Rrightarrow^{\emptyset} \triangleright {}^{\emptyset}\!\Rrightarrow^{\top}\right)^{n} \Rrightarrow_{\top} SR(\sigma_1, \sigma_2) * I_{out} *$$

$$\text{dwp } e_0 \;\&\; t_0 \; \{v_1 \, v_2.\, v_1 = v_2\} \; *$$

$$\mathop{\Huge *}_{1 \le i \le m}.\ \text{dwp } e_i \;\&\; t_i \; \{\text{True}\}$$

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

Constructing the bisimulation

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

$$
\begin{aligned}
(e_0 e_1 \ldots e_m, \sigma_1)\ \mathcal{R}\ (t_0 t_1 \ldots t_m, \sigma_2) \triangleq \ & \text{there exists } n \text{ such that} \\
\text{True} \vdash \left({}^{\top}\!\Rrightarrow^{\emptyset} \triangleright {}^{\emptyset}\!\Rrightarrow^{\top}\right)^{n} \Rrightarrow_{\top} & SR(\sigma_1, \sigma_2) * I_{out} * \\
& \text{dwp } e_0 \,\&\, t_0 \,\{v_1\, v_2.\, v_1 = v_2\} * \\
& \textstyle\mathop{\Huge *}_{1 \le i \le m}.\ \text{dwp } e_i \,\&\, t_i \,\{\text{True}\}
\end{aligned}
$$

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

We get

$$
(I_{out} \vdash \text{dwp } e_1 \,\&\, e_2 \,\{v_1 v_2.\, v_1 = v_2\}) \implies (e_1, \sigma)\ \mathcal{R}^*\ (e_2, \sigma)
$$

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 situation we want to permit *some* information leakage in a controlled way.

We can encode a specific form of deliberate information release in SeLoC.

Example: Alice's calendar

Consider the following example (Constanzo & Shao, 2014):

$$\textbf{let}\ \mathit{cal}\ A = \mathit{iter}\ (\lambda\, i\ v.\, \textbf{if}\, (v == 0)\, \textbf{then}\ \mathit{print}(i))\ A$$

- 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

Specification of the example: Alice's calendar

$$\textbf{let}\ cal\ A = iter\ (\lambda\, i\ v.\, \textbf{if}\,(v == 0)\ \textbf{then}\ print(i))\ A$$

In what sense the program is secure?

Specification of the example: Alice's calendar

$$\textbf{let}\ cal\ A = iter\ (\lambda i\ v.\ \textbf{if}\ (v == 0)\ \textbf{then}\ print(i))\ A$$

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

$$\textbf{let}\ cal\ A = iter\ (\lambda\, i\ \ v.\, \textbf{if}\, (v == 0)\, \textbf{then}\ print(i))\ A$$

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.

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

where $A_1 \sim A_2 \triangleq |A_1| = |A_2| \wedge \forall i.\, A_1[i] = 0 \iff A_2[i] = 0$.

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).

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.

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.

More information: `https://arxiv.org/abs/1910.00905`.