Robust is the New Black

new criteria for secure compilation

Marco Patrignani
11th December 2017

CISPA
Center for IT-Security, Privacy and Accountability
Special Thanks to:
Contents

Robust Compilation Lattice

Robustly-Safe Compilation
Background

Fully Abstract Compilation to JavaScript

Secure Implementations for Typed Session Abstraction

Authenticated primitives and their compilation

Secure Compilation of Object-Oriented Components to Protected Module Architectures

Beyond Good and Evil

Secure Compilation to Protected Module Architectures

A Secure Compiler for ML Module

An Equivalence-Preserving CPS Translation via Multi-Language Semantics

On Modular and Fully-Abstract Compilation
Background

**Fully abstract compilation (FAC)**
de-facto standard for compiler security
Fully abstract compilation (FAC) de-facto standard for compiler security preservation (and reflection) of contextual equivalence
Background

Fully abstract compilation (FAC) de-facto standard for compiler security preservation (and reflection) of contextual equivalence reduces target attackers to source ones.
Shortcomings for FAC

• inefficient code (memory consumption and runtime checks)
• poor support for multithreaded programs
• complex proofs
Shortcomings for FAC

- inefficient code (memory consumption and runtime checks)
Shortcomings for FAC

- inefficient code (memory consumption and runtime checks)
- poor support for multithreaded programs
Shortcomings for FAC

- inefficient code (memory consumption and runtime checks)
- poor support for multithreaded programs
- complex proofs
What do we Want?

- security-aware criteria
- efficient compiled code
- more manageable proofs
Robust Compilation Lattice
Robust Compilation Lattice (RCL)

• based on hyperproperties (HP)
Robust Compilation Lattice (RCL)

- based on hyperproperties (HP)
  - capture all security properties
  - are organised in subclasses for expressiveness
Robust Compilation Lattice (RCL)

- Based on hyperproperties (HP)
- Capture all security properties
- Organized in subclasses for expressiveness
  - Higher notions are stronger and trickier to achieve
  - Each notion comes in two flavours
    - One with clear HP correspondence
    - One for simpler proofs

$[SP] = \text{KSHP}(1)$

$\text{true}$

$\text{false}$
Robust Compilation Lattice (RCL)

- based on hyperproperties (HP)
  - capture all security properties
  - are organised in subclasses for expressiveness

- higher notions are stronger
Robust Compilation Lattice (RCL)

• based on hyperproperties (HP)
  • capture all security properties
  • are organised in subclasses for expressiveness

• higher notions are stronger
  • and trickier to achieve
Robust Compilation Lattice (RCL)

- based on hyperproperties (HP)
  - capture all security properties
  - are organised in subclasses for expressiveness

- higher notions are stronger
  - and trickier to achieve
- each notion comes in two flavours
Robust Compilation Lattice (RCL)

- based on hyperproperties (HP)
  - capture all security properties
  - are organised in subclasses for expressiveness

- higher notions are stronger
  - and trickier to achieve
- each notion comes in two flavours
  - one with clear HP correspondence
  - one for simpler proofs
Notation

- \( C, C \): components of \( S \) and \( T \)
- \( C \cdot, C \cdot \): contexts
- \( C[C], C[C] \): whole programs
- \( \llbracket \cdot \rrbracket^S_T: C \to C \): compiler from \( S \) to \( T \)
- \( \beta, \beta \): traces (possibly infinite), I/O with an environment
- \( \text{Behav}(P) \): set of traces of \( P \)
- \( \pi, \pi \): prefix (finite)
- \( < \): prefixing
- \( \approx: \text{sth} \times \text{sth} \): cross-language relation
Robust Compilation Lattice

Robust Hyperproperty Preservation

- Robust Subset-Closed Hyperproperty Preservation
  - Robust K-Subset-Closed Hyperproperty Preservation
  - Robust 2-Subset-Closed Hyperproperty Preservation
  - Robust Property Preservation
- Robust Hypersafety Preservation
  - Robust K-Hypersafety Preservation
  - Robust 2-Hypersafety Preservation
- Robust Safety Preservation
Robust Hyperproperty Preservation

**Definition (RHP)**

\[ \llbracket \cdot \rrbracket_T^S \in \text{RHP} \overset{\text{def}}{=} \forall C, H, H. \]

\[ \text{if } (\forall C. \text{Behav}(C[C]) \in H) \]

\[ \text{and } H \approx_H H \]

\[ \text{then } (\forall C. \text{Behav}(C[\llbracket \cdot \rrbracket_T^S]) \in H) \]
Definition \((\text{RHP})\)

\[
\forall \mathbb{C}, H, H. \\
\text{if } (\forall \mathbb{C}. \text{Behav}(\mathbb{C}[\mathbb{C}]) \in H) \\
\text{and } H \approx_H H \\
\text{then } (\forall \mathbb{C}. \text{Behav}(\mathbb{C}[\mathbb{C}^{\mathbb{S}}]) \in H)
\]
**Definition (RHP)**

\[
\mathbb{[\cdot]}^S_T \in \text{RHP} \overset{\text{def}}{=} \forall C, H, H.
\]

if \( (\forall C. \text{Behav}(C[C]) \in H) \)

and \( H \approx_H H \)

then \( (\forall C. \text{Behav}(C[C]^S_T]) \in H) \)
Hyperproperty Robust Compilation

**Definition (HRC)**

\[
\begin{align*}
\left[ \cdot \right]^S_T \in \text{HRC} & \overset{\text{def}}{=} \forall C, \exists C. \forall \beta, \beta. \beta \approx_{\beta} \beta \\
\beta & \in \text{Behav} \left( C \left[ [C]^S_T \right] \right) \\
\iff \beta & \in \text{Behav} \left( C \left[ C \right] \right)
\end{align*}
\]
Hyperproperty Robust Compilation

**Definition (HRC)**

\[
\begin{align*}
\llbracket \cdot \rrbracket_T^S & \in \text{HRC} \overset{\text{def}}{=} \forall C, \exists C. \forall \beta, \beta. \beta \approx^\beta \beta \\
\beta & \in \text{Behav} \left( C \llbracket C \rrbracket_T^S \right) \\
\iff \beta & \in \text{Behav} \left( C \llbracket C \rrbracket \right)
\end{align*}
\]
Definition (HRC)

\[
\begin{align*}
[\cdot]^S_T \in HRC & \overset{\text{def}}{=} \forall C, \exists C \forall \beta, \beta. \beta \approx^\beta \beta \\
& \iff \beta \in \text{Behav}(C \left[ [C]^S_T \right])
\end{align*}
\]
Robust Compilation Lattice

RHP = HRC (Theorem, Coq’d)

Robust Subset-Closed Hyperproperty Preservation

Robust K-Subset-Closed Hyperproperty Preservation

Robust 2-Subset-Closed Hyperproperty Preservation

Robust Property Preservation

Robust Hypersafety Preservation

Robust K-Hypersafety Preservation

Robust 2-Hypersafety Preservation

Robust Safety Preservation
**RPP: Robust Property Preservation**

**Definition (RPP)**

\[
\left[ \cdot \right]_T^S \in \text{RPP} \overset{\text{def}}{=} \forall C, P, P. \\
\text{if } (\forall C. \text{Behav}(C[C]) \subseteq P) \\
\text{and } P \approx_H P \\
\text{then } (\forall C. \text{Behav}(C[C]_T^S) \subseteq P)
\]
**RC: Robust Compilation**

**Definition (RC)**

\[
\llbracket \cdot \rrbracket^S_T \in RC \overset{\text{def}}{=} \forall C, C, \beta. \exists C, \beta. \beta \approx_{\beta} \beta
\]

if \( \beta \in \text{Behav}(C[\llbracket C \rrbracket_T^S]) \)

then \( \beta \in \text{Behav}(C[C]) \)
Definition (RC)

\[
\begin{align*}
\llbracket \cdot \rrbracket_T^S \in \text{RC} & \overset{\text{def}}{=} \forall C, C, \beta. \exists C, \beta. \beta \approx_\beta \beta \\
\text{if } \beta \in \text{Behav}(C \llbracket C \rrbracket_T^S) & \text{ then } \beta \in \text{Behav}(C \llbracket C \rrbracket)
\end{align*}
\]
Definition (RC)

\[
\begin{align*}
\llbracket \cdot \rrbracket^S_T \in \text{RC} & \overset{\text{def}}{=} \forall \mathcal{C}, \mathcal{C}, \beta. \exists \mathcal{C}, \beta. \beta \approx_{\beta} \beta \\
\text{if } \beta \in \text{Behav}(\mathcal{C} \llbracket \mathcal{C}^S_T \rrbracket) & \text{then } \beta \in \text{Behav}(\mathcal{C} \llbracket \mathcal{C} \rrbracket)
\end{align*}
\]
Robust Compilation Lattice

RHP = HRC (Theorem, Coq’d)

Robust Subset-Closed Hyperproperty Preservation

Robust K-Subset-Closed Hyperproperty Preservation

Robust 2-Subset-Closed Hyperproperty Preservation

Robust Property Preservation

Robust Hypersafety Preservation

Robust K-Hypersafety Preservation

Robust 2-Hypersafety Preservation

Robust Safety Preservation
Robust Compilation Lattice

RHP = HRC (Theorem, Coq’d)

Robust Subset-Closed Hyperproperty Preservation

Robust K-Subset-Closed Hyperproperty Preservation

Robust 2-Subset-Closed Hyperproperty Preservation

RPP = RC (Theorem, Coq’d)

Robust Hypersafety Preservation

Robust K-Hypersafety Preservation

Robust 2-Hypersafety Preservation

Robust Safety Preservation
Robust Safety Property Preservation

Definition (RSPP)

\[\begin{align*}
\llbracket \cdot \rrbracket^S_T \in \text{RSPP} & \overset{\text{def}}{=} \forall C, P \in \text{SP}, P \in \text{SP}. \\
\text{if } (\forall C.\text{Behav}(C[C]) \subseteq P) \\
\text{and } P \approx_H P \\
\text{then } (\forall C.\text{Behav}(C[\llbracket C \rrbracket^S_T]) \subseteq P)
\end{align*}\]
Definition (SRC)

\[[\cdot]_T^S \in RC \overset{\text{def}}{=} \forall C, C', \pi. \exists C, \pi. \pi \approx_{\beta} \pi\]

if \( \pi < \text{Behav}(C[[C]_T^S]) \)

then \( \pi < \text{Behav}(C [C]) \)
Robust Compilation Lattice

RHP = HRC (Theorem, Coq’d)

Robust Subset-Closed Hyperproperty Preservation

Robust K-Subset-Closed Hyperproperty Preservation

Robust 2-Subset-Closed Hyperproperty Preservation

RPP = RC (Theorem, Coq’d)

Robust Hypersafety Preservation

Robust K-Hypersafety Preservation

Robust 2-Hypersafety Preservation

Robust Safety Preservation
Robust Compilation Lattice

RHP = HRC (Theorem, Coq’d)

Robust Subset-Closed Hyperproperty Preservation

Robust K-Subset-Closed Hyperproperty Preservation

Robust 2-Subset-Closed Hyperproperty Preservation

RPP = RC (Theorem, Coq’d)

RSPP = SRC (Theorem, Coq’d)
Proof Techniques
Proof Techniques

Robust Hyperproperty Preservation

Robust Subset-Closed Hyperproperty Preservation

Robust K-Subset-Closed Hyperproperty Preservation

Robust 2-Subset-Closed Hyperproperty Preservation

Robust Property Preservation

Robust safety Preservation

Robust Hypersafety Preservation

Robust K-Hypersafety Preservation

Robust 2-Hypersafety Preservation
Proof Techniques

Robust Hyperproperty Preservation
  Robust Subset-Closed Hyperproperty Preservation
    Robust K-Subset-Closed Hyperproperty Preservation
      Robust 2-Subset-Closed Hyperproperty Preservation
        Robust Property Preservation
    Robust K-Hypersafety Preservation
      Robust 2-Hypersafety Preservation
        Robust Safety Preservation
Proof Techniques

Robust Hyperproperty Preservation

Robust Subset-Closed Hyperproperty Preservation

Robust K-Subset-Closed Hyperproperty Preservation

Robust 2-Subset-Closed Hyperproperty Preservation

Robust Property Preservation

Robust Hypersafety Preservation

Robust K-Hypersafety Preservation

Robust 2-Hypersafety Preservation

Robust Safety Preservation
Where is Fully Abstract Compilation?
Where is Fully Abstract Compilation?

Robust Hyperproperty Preservation

| Robust Subset-Closed Hyperproperty Preservation

| Robust K-Subset-Closed Hyperproperty Preservation

| Robust 2-Subset-Closed Hyperproperty Preservation

| Robust Property Preservation

| Robust Hypersafety Preservation

| Robust K-Hypersafety Preservation

| Robust 2-Hypersafety Preservation

| Robust Safety Preservation

RSC is very interesting
Where is Fully Abstract Compilation?

- Robust Hyperproperty Preservation
  - Robust Subset-Closed Hyperproperty Preservation
    - Robust K-Subset-Closed Hyperproperty Preservation
      - Robust 2-Subset-Closed Hyperproperty Preservation
        - Robust Property Preservation
      - Robust 2-Hypersafety Preservation
    - Robust K-Hypersafety Preservation
  - Robust /two.osf-Subset-Closed Hyperproperty Preservation
- Robust Hypersafety Preservation
- Robust Safety Preservation

RSC is very interesting
Where is Fully Abstract Compilation?

Robust Hyperproperty Preservation

Robust Subset-Closed Hyperproperty Preservation

Robust K-Subset-Closed Hyperproperty Preservation

Robust 2-Subset-Closed Hyperproperty Preservation

Robust Property Preservation

Robust Safety Preservation

Robust Hypersafety Preservation

Robust K-Hypersafety Preservation

Robust 2-Hypersafety Preservation

Robust /two.osf-Hypersafety Preservation

Robust /two.osf-Subset-Closed Hyperproperty Preservation

RSC is very interesting /two.osf /zero.osf
Where is Fully Abstract Compilation?

Robust Hyperproperty Preservation
- Robust Subset-Closed Hyperproperty Preservation
  - Robust K-Subset-Closed Hyperproperty Preservation
  - Robust 2-Subset-Closed Hyperproperty Preservation
    - Robust Property Preservation
  - Robust 2-Hypersafety Preservation
    - Robust Hypersafety Preservation
      - Robust K-Hypersafety Preservation

Robust Safety Preservation

RSC is very interesting
/two.osf/zero.osf
Where is Fully Abstract Compilation?

Robust Hyperproperty Preservation
- Robust Subset-Closed Hyperproperty Preservation
- Robust 2-Subset-Closed Hyperproperty Preservation
- Robust Property Preservation
- Robust Safety Preservation

Robust K-Hypersafety Preservation
- Robust 2-Hypersafety Preservation

RSC is very interesting
Robustly-Safe Compilation
Why RSC?

- Integrity
- Weak secrecy
- Taint tracking (approx non-interference)
Why RSC?

- integrity
- weak secrecy
- taint tracking
Why RSC?

• integrity
• weak secrecy
• taint tracking (approx non-interference)
Safety how?

- Monitors $M$, $M$ enforce safety
Safety how?

• Monitors $M, \overline{M}$ enforce safety
• Equivalent to classic $\text{assume/} \text{assert}$
Safety how?

• Monitors $M, M'$ enforce safety
• Equivalent to classic `assume/assert`
• Capture `untrusted code` scenario
Safety how?

• Monitors \( M, \overline{M} \) enforce safety
• Equivalent to classic \textit{assume/assert}
• Capture \textit{untrusted code} scenario
• Check heap conditions
Safety how?

- Monitors $M$, men enforce safety
- Equivalent to classic assume/slash.assert
- Capture untrusted code scenario
- Check heap conditions

(Abstract-monitor)

\[
M = (s, \sim, s_0, l, s_c) \quad s_c, H|_l \sim s_f \\
M' = (s, \sim, s_0, l, s_f) \\
M, H \triangleright monitor \rightarrow M', H \triangleright skip
\]
Safety how?

\[ M = (s, \sim, s_0, l, s_c) \quad s_c, H \big|_l \sim s_f \]
\[ M' = (s, \sim, s_0, l, s_f) \]

\[ M, H \rhd \text{monitor} \rightarrow M', H \rhd \text{skip} \]

\[ (\text{Abstract-monitor-fail}) \]
\[ M = (s, \sim, s_0, l, s_c) \quad s_c, H \big|_l \not\sim \_ \]

\[ M, H \rhd \text{monitor} \rightarrow \text{fail} \]
\( \vdash C, M : \text{safe} \stackrel{\text{def}}{=} \text{if } \vdash C : \text{whole} \) 
\[ \text{then } C_0, M \not\xrightarrow{\ast} \text{fail} \]

\( \vdash A : \text{attacker} \stackrel{\text{def}}{=} \text{no monitor inside } A \)

\( \vdash C, M : \text{rs} \stackrel{\text{def}}{=} \text{if } \vdash A : \text{attacker} \) 
\[ \text{then } \vdash A[C], M : \text{safe} \]
Definition (RSC)

\[ \vdash [\cdot]^S_T \circ SRC \overset{\text{def}}{=} \text{if } M \approx M \]

and \[ \vdash C, M : rs \]

then \[ \vdash [C]^S_T, M : rs \]
Languages

• $S$ and $T$ are while languages
Languages

• \( S \) and \( T \) are while languages
• both are untyped
Languages

• $S$ and $T$ are while languages
• both are untyped
• both have $M$, $M$ and monitor instructions
Languages

- S and T are while languages
- both are untyped
- both have M, M and monitor instructions
- S has an abstract heap
Languages

- S and T are while languages
- both are untyped
- both have M, M\text{and} monitor instructions
- S has an abstract heap
  
  \[ H \triangleright e \leftrightarrow v \quad \ell \notin \text{dom}(H) \]  
  
  C, H \triangleright \text{let} x = \text{new} e \text{ in } s
  
  \[ \epsilon \, C, H; \ell \mapsto v \triangleright s[\ell/x] \]  

- T has a concrete heap, abstract capabilities (capabilities / sealing / PMA)
  
  \[ (E-t-new) \]
  
  \[ H = H_1; n \mapsto (v, \eta) \]
  
  \[ H \triangleright e \leftrightarrow v \quad H' = H_1; n+1 \mapsto v : \bot \]
  
  C, H \triangleright \text{let} x = \text{new} e \text{ in } s
  
  \[ \epsilon \, C, H; \ell \mapsto v \triangleright s[\ell/x] \]
Languages

- S and T are while languages
- both are untyped
- both have M, M and monitor instructions
- S has an abstract heap

- T has a concrete heap, abstract capabilities (capabilities / sealing / PMA)
• S and T are while languages
• both are untyped
• both have M, M and monitor instructions
• S has an abstract heap (E-s-alloc)

[H \triangleright e \iff v]

\[\begin{align*}
C, H \triangleright \text{let } x = \text{new } e \text{ in } s \\
\epsilon \rightarrow C, H' \triangleright s[n + 1/x]
\end{align*}\]

(E-t-new)

\[
H = H_1; \ n \mapsto (v, \eta) \quad H \triangleright e \iff v \\
H' = H; \ n + 1 \mapsto v : \bot
\]

\[C, H \triangleright \text{let } x = \text{new } e \text{ in } s \]

\[
\rightarrow C, H' \triangleright s[n + 1/x]
\]
Languages

• S and T are while languages
• Both are untyped
• Both have M, \(M\) and monitor instructions
• T has a concrete heap, abstract capabilities (capabilities / sealing / PMA)

\[(E-t-new)\]
\[
H = H_1; n \leftrightarrow (v, \eta) \quad H \triangleright e \Leftrightarrow v \quad H' = H; n + 1 \leftrightarrow v : \perp
\]
\[
\text{C, } H \triangleright \text{let } x = \text{new } e \text{ in } s \quad \\
\quad \quad \quad \epsilon \rightarrow \text{C, } H' \triangleright s[n + 1/x]
\]

\[(E-t-hide)\]
\[
H \triangleright e \Leftrightarrow n \quad k \notin \text{dom}(H) \quad H = H_1; n \leftrightarrow v : \perp; H_2 \quad H' = H_1; n \leftrightarrow v : k; H_2; k
\]
\[
\text{C, } H \triangleright \text{let } x = \text{hide } e \text{ in } s \quad \\
\quad \quad \quad \epsilon \rightarrow \text{C, } H' \triangleright s[k/x]
\]
Languages

- **S** and **T** are while languages
- both are untyped
- both have **M, M** and monitor instructions
- **S** has an abstract heap

- **T** has a concrete heap, abstract capabilities (capabilities / sealing / PMA)

- !e with e	 x := e with e
• identity except for

\[
\begin{bmatrix}
\text{let } x = \text{new } e
\end{bmatrix}^S
\text{ in } s = \begin{bmatrix}
\text{let } x_{loc} = \text{new } [e]^S_T \text{ in }
\text{let } x_{cap} = \text{hide } x_{loc} \text{ in }
\text{let } x = \langle x_{loc}, x_{cap} \rangle \text{ in } [s]^S_T
\end{bmatrix}^T
\]
Proof Sketch

• if $M, \emptyset \triangleright A [C] \xrightarrow{\bar{\alpha}} M', H \triangleright A [\text{monitor}]$ then $M', H \triangleright A [\text{monitor}] \xrightarrow{\epsilon} M', H \triangleright A [\text{skip}]$

• $M, \emptyset \triangleright A \left[ [C]^S_T \right] \xrightarrow{\bar{\alpha}} M', H \triangleright A [\text{monitor}]$

and we need to prove that

• $M', H \triangleright A [\text{monitor}] \xrightarrow{\epsilon} M', H \triangleright A [\text{skip}]$
Proof Sketch

- if $M, \emptyset \triangleright A[C] \xrightarrow{\alpha} M', H \triangleright A[\text{monitor}]$ then $M', H \triangleright A[\text{monitor}] \xrightarrow{\epsilon} M', H \triangleright A[\text{skip}]$
- $M, \emptyset \triangleright A[\llbracket C \rrbracket_T^S] \xrightarrow{\alpha} M', H \triangleright A[\text{monitor}]$

and we need to prove that

- $M', H \triangleright A[\text{monitor}] \xrightarrow{\epsilon} M', H \triangleright A[\text{skip}]$

$\llbracket \cdot \rrbracket$ takes $\alpha$ and returns $A$
Proof Sketch

- if $M, \alpha /\text{uni2205} \Rightarrow A[C]\ 
\Rightarrow M', H \rightarrow A[monitor]$

- $M', H \rightarrow A[monitor] \epsilon /\text{leftrightline} \rightarrow M', H \rightarrow A[skip]$

- $M, \alpha /\text{uni2205} \Rightarrow A /\text{bracketleft.alt2} J C K S /\text{bracketright.alt2} \alpha /\text{Leftrightline} \Rightarrow M', H \rightarrow A[monitor]$

- and we need to prove that

- $M', H \rightarrow A[monitor] \epsilon /\text{leftrightline} \rightarrow M', H \rightarrow A[skip]$

Plus:

- no need of injective relation
- no need of FA traces

Minus:

- complex because of fine granularity
- still requires backtranslation
Proof Sketch

• if $M,\frac{\alpha}{\text{Leftrightarrow}} A[C] \Rightarrow M', H \frac{\alpha}{\text{Leftrightarrow}} A[\text{monitor}]$ then $M', H \Rightarrow A[\text{monitor}] \frac{\alpha}{\Rightarrow} M', H \Rightarrow A[\text{skip}]$

and we need to prove that

• $M', H \Rightarrow A[\text{monitor}] \frac{\alpha}{\Rightarrow} M', H \Rightarrow A[\text{skip}]$

Plus:

• no need of injective relation
• no need of FA traces

Minus:

• complex because of fine granularity
• still requires backtranslation
• Extend $S$ with a RS type system
Extension

- Extend $S$ with a RS type system
- We can statically know which locations the monitor observes (type $\tau \neq \text{UN}$)
• Extend S with a RS type system
• We can \textit{statically know} which locations the monitor observes (type $\tau \neq \text{UN}$) \textbf{high} locations
Extension

• Extend $S$ with a RS type system
• We can \textit{statically know} which locations the monitor observes (type $\tau \neq \text{UN}$) \textit{high} locations
• $[\cdot]_T^S$ protects only high locations
Extension

• Extend $S$ with a RS type system
• We can statically know which locations the monitor observes (type $\tau \neq \text{UN}$) high locations
• $\llbracket . \rrbracket_S^T$ protects only high locations (efficient!)
Cross-language Bisimulation $\Omega \approx \Omega$

$S$

\[
\begin{array}{ll}
\ell_1 & \leftrightarrow v_1 : \tau_1 \\
\ell_2 & \leftrightarrow v_2 : \tau_2 \\
\ell_3 & \leftrightarrow v_3 : \tau_3 \\
\ell_4 & \leftrightarrow v_4 : \tau_4 \\
\ell_5 & \leftrightarrow v_5 : \tau_5 \\
\ell_6 & \leftrightarrow v_6 : \tau_6 \\
\end{array}
\]

$T$

\[
\begin{array}{ll}
n_1 & \leftrightarrow v_1 : k_1 \\
n_2 & \leftrightarrow v_2 : k_2 \\
n_3 & \leftrightarrow v_3 : k_3 \\
n_4 & \leftrightarrow v_4 : \bot \\
n_5 & \leftrightarrow v_5 : k_5 \\
n_6 & \leftrightarrow v_6 : \bot \\
\end{array}
\]

- No need of injective relation
- No need of any traces
- No need of backtranslation
- Fairly simple
- Scales to multithreaded languages!
Cross-language Bisimulation $\Omega \approx \Omega$

$$\begin{align*}
S & \\
\ell_1 & \mapsto v_1 : \tau_1 \\
\ell_2 & \mapsto v_2 : \tau_2 \\
\ell_3 & \mapsto v_3 : \tau_3 \\
\ell_4 & \mapsto v_4 : \tau_4 \\
\ell_5 & \mapsto v_5 : \tau_5 \\
\ell_6 & \mapsto v_6 : \tau_6
\end{align*}$$

$$\begin{align*}
T & \\
n_1 & \mapsto v_1 : k_1 \\
n_2 & \mapsto v_2 : k_2 \\
n_3 & \mapsto v_3 : k_3 \\
n_4 & \mapsto v_4 : \bot \\
n_5 & \mapsto v_5 : k_5 \\
n_6 & \mapsto v_6 : \bot
\end{align*}$$

- No need of injective relation
- No need of any traces
- No need of backtranslation
- Fairly simple
- Scales to multithreaded languages!
Cross-language Bisimulation $\Omega \approx \Omega$

<table>
<thead>
<tr>
<th>$S$</th>
<th>$\tau \neq \text{UN}_{\text{high}}$</th>
<th>low</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\ell_1 \mapsto v_1 : \tau_1$</td>
<td>$\ell_4 \mapsto v_4 : \tau_4$</td>
<td></td>
</tr>
<tr>
<td>$\ell_2 \mapsto v_2 : \tau_2$</td>
<td>$\ell_5 \mapsto v_5 : \tau_5$</td>
<td></td>
</tr>
<tr>
<td>$\ell_3 \mapsto v_3 : \tau_3$</td>
<td>$\ell_6 \mapsto v_6 : \tau_6$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$T$</th>
<th>high</th>
<th>low</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_1 \mapsto v_1 : k_1$</td>
<td>$n_4 \mapsto v_4 : \bot$</td>
<td></td>
</tr>
<tr>
<td>$n_2 \mapsto v_2 : k_2$</td>
<td>$n_5 \mapsto v_5 : k_5$</td>
<td></td>
</tr>
<tr>
<td>$n_3 \mapsto v_3 : k_3$</td>
<td>$n_6 \mapsto v_6 : \bot$</td>
<td></td>
</tr>
</tbody>
</table>

- Plus:
  - no need of injective relation
  - no need of any traces
  - no need of backtranslation
  - fairly simple
  - scales to multithreaded languages!
Cross-language Bisimulation $\Omega \approx \Omega$

$S$  
\[ \ell_1 \mapsto v_1 : \tau_1 \]  
\[ \ell_2 \mapsto v_2 : \tau_2 \]  
$\approx$  
\[ \ell_3 \mapsto v_3 : \tau_3 \]

$\tau \neq UN_{\text{high}}$  

$\text{low}$  
\[ \ell_4 \mapsto v_4 : \tau_4 \]  
\[ \ell_5 \mapsto v_5 : \tau_5 \]  
\[ \ell_6 \mapsto v_6 : \tau_6 \]

$T$  
\[ n_1 \mapsto v_1 : k_1 \]  
\[ n_2 \mapsto v_2 : k_2 \]  
\[ n_3 \mapsto v_3 : k_3 \]

$\approx$  

$\text{high}$  
\[ k_1 \]  
\[ k_2 \]  
\[ k_3 \]

$\text{low}$  
\[ n_4 \mapsto v_4 : \bot \]  
\[ n_5 \mapsto v_5 : k_5 \]  
\[ n_6 \mapsto v_6 : \bot \]

Plus:  
- no need of injective relation  
- no need of any traces  
- no need of backtranslation  
- fairly simple  
- scales to multithreaded languages!
Cross-language Bisimulation $\Omega \approx \Omega$

$S$

$\tau \neq \text{UN}_{\text{high}}$

$l_1 \mapsto v_1 : \tau_1$

$l_2 \mapsto v_2 : \tau_2$

$l_3 \mapsto v_3 : \tau_3$

$\text{low}$

$l_4 \mapsto v_4 : \tau_4$

$l_5 \mapsto v_5 : \tau_5$

$l_6 \mapsto v_6 : \tau_6$

$T$

$\approx$

$n_1 \mapsto v_1 : k_1$

$n_2 \mapsto v_2 : k_2$

$n_3 \mapsto v_3 : k_3$

$\text{high}$

$\circ$

$\approx$

$k_1$

$k_2$

$k_3$

$\text{low}$

$n_4 \mapsto v_4 : \perp$

$n_5 \mapsto v_5 : k_5$

$n_6 \mapsto v_6 : \perp$

Plus:

• no need of injective relation
• no need of any traces
• no need of backtranslation
• fairly simple
• scales to multithreaded languages!
Cross-language Bisimulation $\Omega \simeq \Omega$

Plus:

- no need of injective relation
- no need of any traces
- no need of backtranslation
- fairly simple
- scales to multithreaded languages!
• motivated the Robust Compilation Lattice
• inspected elements of RCL
• zoomed in an instance of Robustly Safe Compilation
• discussed proof techniques for RSC
Conclusion