On Separation Logic, Computational Independence, and Pseudorandomness

Ugo Dal Lago   Davide Davoli   Bruce Kapron

37th IEEE Computer Security Foundations Symposium
July 8-12, 2024 – Enschede, The Netherlands

Separation Logic [Reynolds and O'Hearn, 2002]

Introduced to reason about *heap* manipulating programs.

Separation Logic [Reynolds and O'Hearn, 2002]

Introduced to reason about *heap* manipulating programs.

***Separating* conjunction:**

$h \models \phi * \psi$

Separation Logic [Reynolds and O'Hearn, 2002]

Introduced to reason about *heap* manipulating programs.

***Separating* conjunction:**

$$h \models \phi * \psi \quad :\Leftrightarrow \quad \exists h_1, h_2$$

Separation Logic [Reynolds and O'Hearn, 2002]

Introduced to reason about *heap* manipulating programs.

***Separating* conjunction:**

$$h \models \phi * \psi \quad :\Leftrightarrow \quad \exists h_1, h_2$$

h_1 h_2

x: 3

z: 1

Separation Logic [Reynolds and O'Hearn, 2002]

Introduced to reason about *heap* manipulating programs.

***Separating* conjunction:**

$$h \models \phi * \psi \quad :\Leftrightarrow \quad \exists h_1, h_2 \text{ s.t.} \qquad h_1 \models \phi \text{ and } h_2 \models \psi$$

h_1 h_2

$\boxed{\texttt{x: 3}}$

$\boxed{\texttt{z: 1}}$

Separation Logic [Reynolds and O'Hearn, 2002]

Introduced to reason about *heap* manipulating programs.

***Separating* conjunction:**

$h \models \phi * \psi \quad :\Leftrightarrow \quad \exists h_1, h_2$ s.t. $\quad h_1 \models \phi$ and $h_2 \models \psi$

h_1 $\qquad$ h_2

`x: 3`

`z: 1`

$h_1 \models x = 3 \qquad h_2 \models z = 1$

Separation Logic [Reynolds and O'Hearn, 2002]

Introduced to reason about *heap* manipulating programs.

***Separating* conjunction:**

$$h \models \phi * \psi \quad :\Leftrightarrow \quad \exists h_1, h_2 \text{ s.t.} h_1 \sqcup h_2 \sqsubseteq h, h_1 \models \phi \text{ and } h_2 \models \psi$$

h_1 h_2

`x: 3`

`z: 1`

$h_1 \models x = 3 \qquad h_2 \models z = 1$

Separation Logic [Reynolds and O'Hearn, 2002]

Introduced to reason about *heap* manipulating programs.

***Separating* conjunction:**

$$h \models \phi * \psi \quad :\Leftrightarrow \quad \exists h_1, h_2 \text{ s.t.} h_1 \sqcup h_2 \sqsubseteq h, h_1 \models \phi \text{ and } h_2 \models \psi$$

h_1	h_2	$h_1 \sqcup h_2$		h
`x: 3`		`x: 3`		`x: 3`
			$\sqsubseteq$	`y: 4`
	`z: 1`	`z: 1`		`z: 1`

$h_1 \models x = 3 \qquad h_2 \models z = 1$

Separation Logic [Reynolds and O'Hearn, 2002]

Introduced to reason about *heap* manipulating programs.

Separating **conjunction:**

$$h \models \phi * \psi \quad :\Leftrightarrow \quad \exists h_1, h_2 \text{ s.t.} h_1 \sqcup h_2 \sqsubseteq h, h_1 \models \phi \text{ and } h_2 \models \psi$$

h_1	h_2	$h_1 \sqcup h_2$		h
`x: 3`		`x: 3`		`x: 3`
			$\sqsubseteq$	`y: 4`
	`z: 1`	`z: 1`		`z: 1`

$h_1 \models x = 3$ $\qquad h_2 \models z = 1$ $\qquad h \models x = 3 * z = 1$

Probabilistic Separation Logic (PSL) (1/2)

The interpretation of $\sqcup$ and $\sqsubseteq$ determines the semantics of $*$.

	Heap model (O'Hearn et. al)	**Distribution model (Barthe et al.'s PSL)**
$\sqcup$	Store union	
$\sqsubseteq$	Sub-store	
$*$	Locality	

Probabilistic Separation Logic (PSL) (1/2)

The interpretation of $\sqcup$ and $\sqsubseteq$ determines the semantics of $*$.

	Heap model (O'Hearn et. al)	**Distribution model (Barthe et al.'s PSL)**
$\sqcup$	Store union	Tensor Product
$\sqsubseteq$	Sub-store	
$*$	Locality	

Probabilistic Separation Logic (PSL) (1/2)

The interpretation of $\sqcup$ and $\sqsubseteq$ determines the semantics of $*$.

	Heap model (O'Hearn et. al)	**Distribution model (Barthe et al.'s PSL)**
$\sqcup$	Store union	Tensor Product
$\sqsubseteq$	Sub-store	Marginal Distribution
$*$	Locality	

Probabilistic Separation Logic (PSL) (1/2)

The interpretation of $\sqcup$ and $\sqsubseteq$ determines the semantics of $*$.

	Heap model (O'Hearn et. al)	**Distribution model (Barthe et al.'s PSL)**
$\sqcup$	Store union	Tensor Product
$\sqsubseteq$	Sub-store	Marginal Distribution
$*$	Locality	Statistic Independence

Probabilistic Separation Logic (PSL) (1/2)

The interpretation of $\sqcup$ and $\sqsubseteq$ determines the semantics of $*$.

	Heap model (O'Hearn et. al)	**Distribution model (Barthe et al.'s PSL)**
$\sqcup$	Store union	Tensor Product
$\sqsubseteq$	Sub-store	Marginal Distribution
$*$	Locality	Statistic Independence

Example

$$[\![x \leftarrow \mathsf{unif}(n); y \leftarrow \mathsf{unif}(n)]\!] \models \underbrace{\mathbf{U}(x) * \mathbf{U}(y)}_{x \text{ and } y \text{ are uniform and independent}}$$

Probabilistic Separation Logic (PSL) (1/2)

The interpretation of $\sqcup$ and $\sqsubseteq$ determines the semantics of $*$.

	Heap model (O'Hearn et. al)	**Distribution model (Barthe et al.'s PSL)**
$\sqcup$	Store union	Tensor Product
$\sqsubseteq$	Sub-store	Marginal Distribution
$*$	Locality	Statistic Independence

Example

$$[\![x \leftarrow \mathrm{unif}(n); y \leftarrow \mathrm{unif}(n)]\!] \models \underbrace{\mathbf{U}(x) * \mathbf{U}(y)}_{x \text{ and } y \text{ are uniform and independent}}$$

$$[\![x \leftarrow \mathrm{unif}(n); y \leftarrow x]\!] \not\models \mathbf{U}(x) * \mathbf{U}(y)$$

Probabilistic Separation Logic (PSL) (1/2)

The interpretation of $\sqcup$ and $\sqsubseteq$ determines the semantics of $*$.

	Heap model (O'Hearn et. al)	**Distribution model (Barthe et al.'s PSL)**
$\sqcup$	Store union	Tensor Product
$\sqsubseteq$	Sub-store	Marginal Distribution
$*$	Locality	Statistic Independence

Example

$$[\![x \leftarrow \mathsf{unif}(n); y \leftarrow \mathsf{unif}(n)]\!] \models \underbrace{\mathbf{U}(x) * \mathbf{U}(y)}_{x \text{ and } y \text{ are uniform and independent}}$$

$$[\![x \leftarrow \mathsf{unif}(n); y \leftarrow x]\!] \not\models \mathbf{U}(x) * \mathbf{U}(y)$$

$$\Big([\![x \leftarrow \mathsf{unif}(n); y \leftarrow x]\!] \models \mathbf{U}(x) \wedge \mathbf{U}(y)\Big)$$

Probabilistic Separation Logic (PSL) (2/2)

Probabilistic Separtation Logic can be used to support Hoare style reasoning on cryptographic primitives.

Probabilistic Separation Logic (PSL) (2/2)

Probabilistic Separtation Logic can be used to support Hoare style reasoning on cryptographic primitives.

Example (One Time Pad)

PSL can prove perfect secrecy.

$$\begin{aligned} \texttt{OTP} := \ & \mathit{key} \leftarrow \mathsf{unif}(n); \\ & \mathit{chip} \leftarrow \mathit{msg} \oplus \mathit{key}. \end{aligned}$$

Probabilistic Separation Logic (PSL) (2/2)

Probabilistic Separtation Logic can be used to support Hoare style reasoning on cryptographic primitives.

Example (One Time Pad)

PSL can prove perfect secrecy.

$$\begin{aligned}\texttt{OTP} := \; & \mathit{key} \leftarrow \mathsf{unif}(n); \\ & \mathit{chip} \leftarrow \mathit{msg} \oplus \mathit{key}.\end{aligned}$$

In PSL, the following judgment is derivable:

$$\vdash_{\mathsf{PSL}} \{\underbrace{\mathbf{D}(\mathit{msg})}_{\mathit{msg}\text{ is defined}}\} \; \texttt{OTP} \; \{\mathbf{D}(\mathit{msg}) * \mathbf{U}(\mathit{chip})\}$$

Probabilistic Separation Logic (PSL) (2/2)

Probabilistic Separtation Logic can be used to support Hoare style reasoning on cryptographic primitives.

Example (One Time Pad)

PSL can prove perfect secrecy.

$$\begin{aligned} \texttt{OTP} := \ & \mathit{key} \leftarrow \mathsf{unif}(n); \\ & \mathit{chip} \leftarrow \mathit{msg} \oplus \mathit{key}. \end{aligned}$$

In PSL, the following judgment is derivable:

$$\vdash_{\mathsf{PSL}} \{\underbrace{\mathbf{D}(\mathit{msg})}_{\mathit{msg}\text{ is defined}}\}\ \texttt{OTP}\ \{\underbrace{\mathbf{D}(\mathit{msg}) \ast \mathbf{U}(\mathit{chip})}_{\text{perfect secrecy}}\}$$

Towards a Computational Version of PSL

Property		Characterization	Logic
Perfect Secrecy	$\Leftrightarrow$	Statistical Independence	PSL

$\underbrace{\text{Perfect Secrecy} \Leftrightarrow \text{Statistical Independence}}_{\text{[Shannon, 1940]}}$

Towards a Computational Version of PSL

Property		**Characterization**	**Logic**
Perfect Secrecy	$\Leftrightarrow$	Statistical Independence	PSL
[Shannon, 1940]			
Computational Secrecy		?	

Towards a Computational Version of PSL

Property		**Characterization**	**Logic**
Perfect Secrecy	$\Leftrightarrow$	Statistical Independence	PSL
[Shannon, 1940]			
Computational Secrecy		*Computational* independence	

Towards a Computational Version of PSL

Property		Characterization	Logic
Perfect Secrecy	$\Leftrightarrow$	Statistical Independence	PSL

[Shannon, 1940]

Computational Secrecy $\Leftrightarrow$ *Computational* independence

[Fay, 2015] (for families of polysize circuits)

Towards a Computational Version of PSL

Property		**Characterization**	**Logic**
Perfect Secrecy	$\Leftrightarrow$	Statistical Independence	PSL

[Shannon, 1940]

Computational Secrecy $\Leftrightarrow$ *Computational* independence

[Fay, 2015] (for families of polysize circuits)

This work (for polytime programs)

Towards a Computational Version of PSL

Property		**Characterization**	**Logic**
Perfect Secrecy	$\Leftrightarrow$	Statistical Independence	PSL

[Shannon, 1940]

Computational Secrecy	$\Leftrightarrow$	*Computational* independence	?

[Fay, 2015] (for families of polysize circuits)

This work (for polytime programs)

Towards a Computational Version of PSL

Property		**Characterization**	**Logic**
Perfect Secrecy	⇔	Statistical Independence	PSL

[Shannon, 1940]

Computational Secrecy	⇔	*Computational* independence	CSL

[Fay, 2015] (for families of polysize circuits)

This work (for polytime programs)

Cryptographic Separation Logic (CSL)

The interpretation of $\sqcup$ and $\sqsubseteq$ determines the semantics of $*$.

	PSL	CSL

Cryptographic Separation Logic (CSL)

The interpretation of $\sqcup$ and $\sqsubseteq$ determines the semantics of $*$.

	PSL	CSL
$\sqcup$	Tensor Product	Tensor Product

Cryptographic Separation Logic (CSL)

The interpretation of $\sqcup$ and $\sqsubseteq$ determines the semantics of $*$.

	PSL	**CSL**
$\sqcup$	Tensor Product	Tensor Product
$d_1 \sqsubseteq d_2$	d_1 is a marginal of d_2	

Cryptographic Separation Logic (CSL)

The interpretation of $\sqcup$ and $\sqsubseteq$ determines the semantics of $*$.

	PSL	**CSL**
$\sqcup$	Tensor Product	Tensor Product
$d_1 \sqsubseteq d_2$	$d_1 = \lambda x. \sum_y d_2(x, y)$	

Cryptographic Separation Logic (CSL)

The interpretation of $\sqcup$ and $\sqsubseteq$ determines the semantics of $*$.

	PSL	**CSL**
$\sqcup$	Tensor Product	Tensor Product
$d_1 \sqsubseteq d_2$	$d_1 = \lambda x. \sum_y d_2(x, y)$	$d_1 \approx \lambda x. \sum_y d_2(x, y)$

Cryptographic Separation Logic (CSL)

The interpretation of $\sqcup$ and $\sqsubseteq$ determines the semantics of $*$.

	PSL	**CSL**
$\sqcup$	Tensor Product	Tensor Product
$d_1 \sqsubseteq d_2$	$d_1 = \lambda x. \sum_y d_2(x, y)$	$d_1 \approx \lambda x. \sum_y d_2(x, y)$

Every *polytime* distingusher has *negligible* advantage on d_1 and $\lambda x. \sum_y d_2(x, y)$.

Cryptographic Separation Logic (CSL)

The interpretation of $\sqcup$ and $\sqsubseteq$ determines the semantics of $*$.

	PSL	**CSL**
$\sqcup$	Tensor Product	Tensor Product
$d_1 \sqsubseteq d_2$	$d_1 = \lambda x. \sum_y d_2(x, y)$	$d_1 \approx \lambda x. \sum_y d_2(x, y)$

Every *polytime* distingusher has *negligible* advantage on d_1 and $\lambda x. \sum_y d_2(x, y)$.

Theorem (Main Result)

The semantics of the separating conjunction ($*$) in CSL is equivalent to Fay's computational independence.

Polytime Programs

Syntax:

$$\mathtt{P}, \mathtt{R} ::= \ \mathtt{skip} \ \Big| \ r \leftarrow e \ \Big| \ \mathtt{P}; \mathtt{P} \ \Big| \ \mathtt{if}\ r\ \mathtt{then}\ \mathtt{P}\ \mathtt{else}\ \mathtt{P}$$

Polytime Programs

Syntax:

$$\mathtt{P}, \mathtt{R} ::= \ \mathtt{skip} \ \Big| \ r \leftarrow e \ \Big| \ \mathtt{P}; \mathtt{P} \ \Big| \ \mathtt{if} \ r \ \mathtt{then} \ \mathtt{P} \ \mathtt{else} \ \mathtt{P}$$

Type system:

Polytime Programs

Syntax:

$$\mathtt{P}, \mathtt{R} ::= \ \mathtt{skip} \ \Big| \ r \leftarrow e \ \Big| \ \mathtt{P}; \mathtt{P} \ \Big| \ \mathtt{if} \ r \ \mathtt{then} \ \mathtt{P} \ \mathtt{else} \ \mathtt{P}$$

Type system:

- Δ : Variables $\rightarrow$ Size (in terms of the security parameter n).

Polytime Programs

Syntax:

$$\mathtt{P}, \mathtt{R} ::= \ \mathtt{skip} \ \Big| \ r \leftarrow e \ \Big| \ \mathtt{P}; \mathtt{P} \ \Big| \ \mathtt{if}\ r\ \mathtt{then}\ \mathtt{P}\ \mathtt{else}\ \mathtt{P}$$

Type system:

- Δ : Variables $\rightarrow$ Size (in terms of the security parameter n).
- $\Delta \vdash \mathtt{P}$ when $\mathtt{P}$ is polytime in its input.

Polytime Programs

Syntax:

$$\mathtt{P}, \mathtt{R} ::= \ \mathtt{skip} \ \Big| \ r \leftarrow e \ \Big| \ \mathtt{P}; \mathtt{P} \ \Big| \ \mathtt{if} \ r \ \mathtt{then} \ \mathtt{P} \ \mathtt{else} \ \mathtt{P}$$

Type system:

- Δ : Variables $\rightarrow$ Size (in terms of the security parameter n).
- $\Delta \vdash \mathtt{P}$ when $\mathtt{P}$ is polytime in its input.
- $[\![\Delta]\!] = \{$distributions of *polysize* stores$\}$

Polytime Programs

Syntax:

$$\mathtt{P}, \mathtt{R} ::= \ \mathtt{skip} \ \big| \ r \leftarrow e \ \big| \ \mathtt{P}; \mathtt{P} \ \big| \ \mathtt{if}\ r\ \mathtt{then}\ \mathtt{P}\ \mathtt{else}\ \mathtt{P}$$

Type system:

- Δ : Variables $\to$ Size (in terms of the security parameter n).
- $\Delta \vdash \mathtt{P}$ when $\mathtt{P}$ is polytime in its input.
- $[\![\Delta]\!] = \{$distributions of *polysize* stores$\}$

Semantics:

$$[\![\Delta \vdash \mathtt{P}]\!] : [\![\Delta]\!] \to [\![\Delta]\!]$$

Polytime Programs

Syntax:

$$\mathtt{P}, \mathtt{R} ::= \mathtt{skip} \;\Big|\; r \leftarrow e \;\Big|\; \mathtt{P}; \mathtt{P} \;\Big|\; \mathtt{if}\ r\ \mathtt{then}\ \mathtt{P}\ \mathtt{else}\ \mathtt{P}$$

Type system:

- Δ : Variables $\rightarrow$ Size (in terms of the security parameter n).
- $\Delta \vdash \mathtt{P}$ when $\mathtt{P}$ is polytime in its input.
- $[\![\Delta]\!] = \{$distributions of *polysize* stores$\}$

Semantics:

$$[\![\Delta \vdash \mathtt{P}]\!] : [\![\Delta]\!] \rightarrow [\![\Delta]\!]$$

Programs are polytime in the security parameter *by construction*.

CSL's Syntax and Semantics

Atomic propositions:

CSL's Syntax and Semantics

Atomic propositions:

$A ::= \mathbf{EQ}(e, g)$ $\qquad$ $[\![e]\!]$ and $[\![g]\!]$ are the same distribution

CSL's Syntax and Semantics

Atomic propositions:

$$\begin{array}{lll} A ::= & \mathbf{EQ}(e, g) & [\![e]\!] \text{ and } [\![g]\!] \text{ are the same distribution} \\ \mid & \mathbf{CI}(e, g) & [\![e]\!] \text{ and } [\![g]\!] \text{ are indistinguishable } ([\![e]\!] \approx [\![g]\!]) \end{array}$$

CSL's Syntax and Semantics

Atomic propositions:

$$\begin{array}{lll} A ::= \mathbf{EQ}(e, g) & & [\![e]\!] \text{ and } [\![g]\!] \text{ are the same distribution} \\ \quad\ \mid \mathbf{CI}(e, g) & & [\![e]\!] \text{ and } [\![g]\!] \text{ are indistinguishable } ([\![e]\!] \approx [\![g]\!]) \end{array}$$

Formulas:

$$\phi ::= (A)^{\Delta} \mid (\phi \wedge \psi)^{\Delta} \mid (\phi * \psi)^{\Delta}$$

CSL's Syntax and Semantics

Atomic propositions:

$$\begin{aligned} A ::= {} & \mathbf{EQ}(e, g) && [\![e]\!] \text{ and } [\![g]\!] \text{ are the same distribution} \\ \mid {} & \mathbf{CI}(e, g) && [\![e]\!] \text{ and } [\![g]\!] \text{ are indistinguishable } ([\![e]\!] \approx [\![g]\!]) \end{aligned}$$

Formulas:

$$\phi ::= (A)^{\Delta} \mid (\phi \wedge \psi)^{\Delta} \mid (\phi * \psi)^{\Delta}$$

- We want to interpret formulas only on those distribution where they have a meaning.

CSL's Syntax and Semantics

Atomic propositions:

$$\begin{array}{lll} A ::= \mathbf{EQ}(e, g) & [\![e]\!] \text{ and } [\![g]\!] \text{ are the same distribution} \\ \quad \mid \mathbf{CI}(e, g) & [\![e]\!] \text{ and } [\![g]\!] \text{ are indistinguishable } ([\![e]\!] \approx [\![g]\!]) \end{array}$$

Formulas:

$$\phi ::= (A)^{\Delta} \mid (\phi \wedge \psi)^{\Delta} \mid (\phi * \psi)^{\Delta}$$

- We want to interpret formulas only on those distribution where they have a meaning.
- $d \models (\phi)^{\Delta}$ means $d \in [\![\Delta]\!]$ and $d \models \phi$.

CSL's Syntax and Semantics

Atomic propositions:

$$
\begin{array}{lll}
A ::= \mathbf{EQ}(e, g) & [\![e]\!] \text{ and } [\![g]\!] \text{ are the same distribution} \\
\quad \mid \mathbf{CI}(e, g) & [\![e]\!] \text{ and } [\![g]\!] \text{ are indistinguishable } ([\![e]\!] \approx [\![g]\!])
\end{array}
$$

Formulas:

$$\phi ::= (A)^{\Delta} \mid (\phi \wedge \psi)^{\Delta} \mid (\phi * \psi)^{\Delta}$$

- We want to interpret formulas only on those distribution where they have a meaning.
- $d \models (\phi)^{\Delta}$ means $d \in [\![\Delta]\!]$ and $d \models \phi$.
- The environments of ϕ and ψ are always smaller than Δ

CSL's Syntax and Semantics

Atomic propositions:

$$
\begin{array}{lll}
A ::= \mathbf{EQ}(e, g) & [\![e]\!] \text{ and } [\![g]\!] \text{ are the same distribution} \\
\quad \mid \mathbf{CI}(e, g) & [\![e]\!] \text{ and } [\![g]\!] \text{ are indistinguishable } ([\![e]\!] \approx [\![g]\!])
\end{array}
$$

Formulas:

$$\phi ::= (A)^{\Delta} \mid (\phi \wedge \psi)^{\Delta} \mid (\phi * \psi)^{\Delta}$$

- We want to interpret formulas only on those distribution where they have a meaning.
- $d \models (\phi)^{\Delta}$ means $d \in [\![\Delta]\!]$ and $d \models \phi$.
- The environments of ϕ and ψ are always smaller than Δ
- They must also be disjoint for $*$.

Typed Separating Conjunctions

The standard interpretation of $*$ is ambiguous:

Typed Separating Conjunctions

The standard interpretation of $*$ is ambiguous:

$$h \models \phi * \psi \qquad :\Leftrightarrow \qquad \exists \underbrace{h_1, h_2}_{\text{what is their domain?}} \text{s.t.} \ldots$$

Typed Separating Conjunctions

The standard interpretation of $*$ is ambiguous:

$$d \models \phi * \psi \qquad :\Leftrightarrow \qquad \exists \underbrace{d_1, d_2}_{\text{what is their domain?}} \text{ s.t. } \ldots$$

- In PSL, we do not know which variables are independent.

Typed Separating Conjunctions

The standard interpretation of $*$ is ambiguous:

$$d \models \phi * \psi \qquad :\Leftrightarrow \qquad \exists \underbrace{d_1, d_2}_{\text{what is their domain?}} \text{s.t.} \ldots$$

- In PSL, we do not know which variables are independent.

$((\phi)^\Gamma * (\psi)^\Theta)^\Delta$ is a formula $\Rightarrow$ Γ and Θ have disjoint domains.

Typed Separating Conjunctions

The standard interpretation of $*$ is ambiguous:

$$d \models \phi * \psi \qquad :\Leftrightarrow \qquad \exists \underbrace{d_1, d_2}_{\text{what is their domain?}} \text{ s.t.} \ldots$$

- In PSL, we do not know which variables are independent.
- CSL *formulas* tell us which variables are independent.

$((\phi)^{\Gamma} * (\psi)^{\Theta})^{\Delta}$ is a formula $\Rightarrow$ Γ and Θ have disjoint domains.

CSL's Rules

Judgments

$$\{(\phi)^{\Delta}\}\ \Delta \vdash \mathtt{P}\ \{(\psi)^{\Delta}\}$$

CSL's Rules

Judgments

$$\{(\phi)^{\Delta}\}\ \Delta \vdash \mathtt{P}\ \{(\psi)^{\Delta}\}$$

For every $d \in [\![\Delta]\!]$, if $d \models \phi$, then $[\![\Delta \vdash \mathtt{P}]\!](d) \models \psi$.

CSL's Rules

Judgments

$$\{(\phi)^{\Delta}\}\ \Delta \vdash \mathtt{P}\ \{(\psi)^{\Delta}\}$$

For every $d \in [\![\Delta]\!]$, if $d \models \phi$, then $[\![\Delta \vdash \mathtt{P}]\!](d) \models \psi$.

Rules

$$\frac{r \notin \mathsf{FV}(e)}{\vdash \{(\top)^{\Delta}\}\ \Delta \vdash r \leftarrow e\ \{(\mathbf{EQ}(r, e))^{\Delta}\}}\ \mathsf{Asgn}$$

CSL's Rules

Judgments

$$\{(\phi)^{\Delta}\}\ \Delta \vdash \mathtt{P}\ \{(\psi)^{\Delta}\}$$

For every $d \in [\![\Delta]\!]$, if $d \models \phi$, then $[\![\Delta \vdash \mathtt{P}]\!](d) \models \psi$.

Rules

$$\frac{r \notin \mathsf{FV}(e)}{\vdash \{(\top)^{\Delta}\}\ \Delta \vdash r \leftarrow e\ \{(\mathbf{EQ}(r, e))^{\Delta}\}}\ \mathsf{Asgn}$$

in PSL:

$$\frac{r \notin \mathsf{FV}(e)}{\vdash \{\top\}\ r \leftarrow e\ \{\mathbf{EQ}(r, e)\}}\ \mathsf{Asgn}$$

CSL's Rules

Judgments

$$\{(\phi)^{\Delta}\}\ \Delta \vdash \mathtt{P}\ \{(\psi)^{\Delta}\}$$

For every $d \in [\![\Delta]\!]$, if $d \models \phi$, then $[\![\Delta \vdash \mathtt{P}]\!](d) \models \psi$.

Rules

$$\frac{r \notin \mathsf{FV}(e)}{\vdash \{(\top)^{\Delta}\}\ \Delta \vdash r \leftarrow e\ \{(\mathbf{EQ}(r, e))^{\Delta}\}}\ \text{Asgn}$$

$$\frac{\vdash \{(\phi)^{\Gamma}\}\ \Gamma \vdash \mathtt{P}\ \{(\psi)^{\Gamma}\}}{\vdash \{((\phi)^{\Gamma} * (\xi)^{\Theta})^{\Delta}\}\ \Delta \vdash \mathtt{P}\ \{((\psi)^{\Gamma} * (\xi)^{\Theta})^{\Delta}\}}\ \text{Frame}$$

in PSL:

$$\frac{r \notin \mathsf{FV}(e)}{\vdash \{\top\}\ r \leftarrow e\ \{\mathbf{EQ}(r, e)\}}\ \text{Asgn}$$

Difficulty with loops

The standard Hoare rule for for-loops is *unsound* in CSL:

$$\frac{\forall i.\ \vdash \{\phi(i)\}\ \mathtt{P}\ \{\phi(i+1)\}}{\vdash \{\phi(0)\}\ \mathtt{for}\ i = 0\ \mathtt{to}\ n\ \mathtt{do}\ \mathtt{P}\ \{\phi(n)\}}$$

Difficulty with loops

The standard Hoare rule for for-loops is *unsound* in CSL:

$$\frac{\forall i. \vdash \{\phi(i)\}\ \mathtt{P}\ \{\phi(i+1)\}}{\vdash \{\phi(0)\}\ \mathtt{for}\ i = 0\ \mathtt{to}\ n\ \mathtt{do}\ \mathtt{P}\ \{\phi(n)\}}$$

Example

$\phi(i) := \mathbf{CU}(e_i)$ e_i is computationally uniform.

Difficulty with loops

The standard Hoare rule for for-loops is *unsound* in CSL:

$$\frac{\forall i.\vdash \{\phi(i)\}\ \mathtt{P}\ \{\phi(i+1)\}}{\vdash \{\phi(0)\}\ \mathtt{for}\ i=0\ \mathtt{to}\ n\ \mathtt{do}\ \mathtt{P}\ \{\phi(n)\}}$$

Example

$\phi(i) := \mathbf{CU}(e_i)$ e_i is computationally uniform.

Assume:

$$\models \{\mathbf{CU}(e_i)\}\ \mathtt{P}\ \{\underbrace{\mathbf{CU}(e_{i+1})}_{\text{with negligible advantage } n^i/2^n}\}.$$

Difficulty with loops

The standard Hoare rule for for-loops is *unsound* in CSL:

$$\frac{\forall i. \vdash \{\phi(i)\}\ \mathtt{P}\ \{\phi(i+1)\}}{\vdash \{\phi(0)\}\ \mathtt{for}\ i = 0\ \mathtt{to}\ n\ \mathtt{do}\ \mathtt{P}\ \{\phi(n)\}}$$

Example

$$\phi(i) := \mathbf{CU}(e_i) \qquad e_i \text{ is computationally uniform.}$$

Assume:

$$\models \{\mathbf{CU}(e_i)\}\ \mathtt{P}\ \{\underbrace{\mathbf{CU}(e_{i+1})}_{\text{with negligible advantage } n^i/2^n}\}.$$

After n iterations, the bound on the advantage is: $n^n/2^n$.

Pseudo One Time Pad

The Pseudo One Time Pad is the *computationally secret* variant of the One Time Pad.

Pseudo One Time Pad

The Pseudo One Time Pad is the *computationally secret* variant of the One Time Pad.

$$\begin{aligned}\mathtt{OTP} := \ & \mathit{key} \leftarrow \mathsf{unif}(n);\\ & \mathit{chip} \leftarrow \mathit{msg} \oplus \mathit{key}.\end{aligned}$$

Pseudo One Time Pad

The Pseudo One Time Pad is the *computationally secret* variant of the One Time Pad.

$$\texttt{POTP} := \begin{aligned}[t] & \mathit{key} \leftarrow \mathsf{unif}(n); \\ & r \leftarrow g(\mathit{key}); \\ & \mathit{chip} \leftarrow \mathit{msg} \oplus r. \end{aligned}$$

g is a *pseudorandom* generator.

Pseudo One Time Pad

The Pseudo One Time Pad is the *computationally secret* variant of the One Time Pad.

$$\begin{aligned}\texttt{POTP} := \; & \mathit{key} \leftarrow \mathsf{unif}(n); \\ & r \leftarrow g(\mathit{key}); \\ & \mathit{chip} \leftarrow \mathit{msg} \oplus r.\end{aligned}$$

g is a *pseudorandom* generator.

$$\vdash_{\mathsf{CSL}} \{(\top)^{\Delta}\}\; \Delta \vdash \texttt{POTP}\; \{\underbrace{((\top)^{\{\mathit{msg}:\ldots\}} * (\mathbf{CU}(\mathit{chip}))^{\{\mathit{chip}:\ldots\}})^{\Delta}}_{\text{computational secrecy}}\}$$

Pseudo One Time Pad

The Pseudo One Time Pad is the *computationally secret* variant of the One Time Pad.

$$\begin{aligned}\texttt{POTP} := \ & \mathit{key} \leftarrow \mathsf{unif}(n);\\ & r \leftarrow g(\mathit{key});\\ & \mathit{chip} \leftarrow \mathit{msg} \oplus r.\end{aligned}$$

g is a *pseudorandom* generator.

$$\vdash_{\mathrm{CSL}} \{(\top)^{\Delta}\}\ \Delta \vdash \texttt{POTP}\ \{\underbrace{((\top)^{\{\mathit{msg}:\ldots\}} * (\mathbf{CU}(\mathit{chip}))^{\{\mathit{chip}:\ldots\}})^{\Delta}}_{\text{computational secrecy}}\}$$

$$\vdash_{\mathrm{PSL}} \{\underbrace{\mathbf{D}(\mathit{msg})}_{\mathit{msg}\text{ is defined}}\}\ \texttt{OTP}\ \{\underbrace{\mathbf{D}(\mathit{msg}) * \mathbf{U}(\mathit{chip})}_{\text{perfect secrecy}}\}$$

Conclusion and Future Work

Conclusion

- CSL is a separation logic for *computational independence*.

Conclusion and Future Work

Conclusion

- CSL is a separation logic for *computational independence*.
- The inference rules of CSL are similar to those of PSL.

Conclusion and Future Work

Conclusion

- CSL is a separation logic for *computational independence*.
- The inference rules of CSL are similar to those of PSL.
- CSL can be used to prove *computational secrecy*, thanks to our Fay-style characterization.

Conclusion and Future Work

Conclusion

- CSL is a separation logic for *computational independence*.
- The inference rules of CSL are similar to those of PSL.
- CSL can be used to prove *computational secrecy*, thanks to our Fay-style characterization.

Future work

- Extend the language supported by CSL with for-loops.