A Probabilistic Separation Logic

Justin Hsu
UW–Madison
Computer Sciences
Brilliant Collaborators

Gilles Barthe

Kevin Liao

Jialu Bao

Simon Docherty

Alexandra Silva
What Is Independence, Intuitively?

Two random variables $x$ and $y$ are independent if they are uncorrelated: the value of $x$ gives no information about the value or distribution of $y$. 
Things that are independent

**Fresh random samples**
- $x$ is the result of a fair coin flip
- $y$ is the result of another, “fresh” coin flip
- More generally: “separate” sources of randomness

**Uncorrelated things**
- $x$ is today’s winning lottery number
- $y$ is the closing price of the stock market
Things that are not independent

Re-used samples

- $x$ is the result of a fair coin flip
- $y$ is the result of the same coin flip

Common cause

- $x$ is today’s ice cream sales
- $y$ is today’s sunglasses sales
What Is Independence, Formally?

**Definition**
Two random variables $x$ and $y$ are independent (in some implicit distribution over $x$ and $y$) if for all values $a$ and $b$:

$$\Pr(x = a \land y = b) = \Pr(x = a) \cdot \Pr(y = b)$$

That is, the distribution over $(x, y)$ is the product of a distribution over $x$ and a distribution over $y$. 
Why Is Independence Useful for Program Reasoning?

Ubiquitous in probabilistic programs
▶ A “fresh” random sample is independent of the state.

Simplifies reasoning about groups of variables
▶ Complicated: general distribution over many variables
▶ Simple: product of distributions over each variable

Preserved under common program operations
▶ Local operations independent of “separate” randomness
▶ Behaves well under conditioning (prob. control flow)
Reasoning about Independence: Challenges

Formal definition isn’t very promising

- Quantification over all values: lots of probabilities!
- Computing exact probabilities: often difficult

How can we leverage the intuition behind probabilistic independence?
Main Observation: Independence is Separation

Two variables $x$ and $y$ in a distribution $\mu$ are independent if $\mu$ is the product of two distributions $\mu_x$ and $\mu_y$ with disjoint domains, containing $x$ and $y$.

Leverage separation logic to reason about independence

- Pioneered by O’Hearn, Reynolds, and Yang
- Highly developed area of program verification research
- Rich logical theory, automated tools, etc.
Our Approach: Two Ingredients

- Develop a probabilistic model of the logic BI
- Design a probabilistic separation logic PSL
Recap: Bunched Implications and Separation Logics
What Goes into a Separation Logic?

- Programs
  - Transform input states to output states
- Assertions
  - Formulas describe pieces of program states
  - Semantics defined by a model of BI (Pym and O'Hearn)
- Program logic
  - Formulas describe programs
  - Assertions specify pre- and post-conditions
What Goes into a Separation Logic?

1. Programs
   - Transform input states to output states
What Goes into a Separation Logic?

1. Programs
   - Transform input states to output states

2. Assertions
   - Formulas describe pieces of program states
   - Semantics defined by a model of BI (Pym and O’Hearn)
What Goes into a Separation Logic?

1. Programs
   - Transform input states to output states

2. Assertions
   - Formulas describe pieces of program states
   - Semantics defined by a model of BI (Pym and O’Hearn)

3. Program logic
   - Formulas describe programs
   - Assertions specify pre- and post-conditions
Classical Setting: Heaps

**Program states** \((s, h)\)

- **A store** \(s : \mathcal{X} \rightarrow \mathcal{V}\), map from variables to values
- **A heap** \(h : \mathbb{N} \rightarrow \mathcal{V}\), partial map from addresses to values
Classical Setting: Heaps

Program states \((s, h)\)
- A store \(s : \mathcal{X} \rightarrow \mathcal{V}\), map from variables to values
- A heap \(h : \mathbb{N} \rightarrow \mathcal{V}\), partial map from addresses to values

Heap-manipulating programs
- Control flow: sequence, if-then-else, loops
- Read/write addresses in heap
- Allocate/free heap cells
Substructural logic (O’Hearn and Pym)

- Start with regular propositional logic ($\top, \bot, \land, \lor, \to$)
- Add a new conjunction ("star"): $P \ast Q$
- Add a new implication ("magic wand"): $P \to\ast Q$
Assertion Logic: Bunched Implications (Bl)

Substructural logic (O’Hearn and Pym)

- Start with regular propositional logic ($\top, \bot, \land, \lor, \rightarrow$)
- Add a new conjunction ("star"): $P \star Q$
- Add a new implication ("magic wand"): $P \rightarrow* Q$

Star is a multiplicative conjunction

- $P \land Q$: $P$ and $Q$ hold on the entire state
- $P \star Q$: $P$ and $Q$ hold on disjoint parts of the entire state
Resource Semantics of BI (O’Hearn and Pym)

Suppose states form a pre-ordered, partial monoid

- Set $S$ of states, pre-order $\sqsubseteq$ on $S$
- Partial operation $\circ : S \times S \rightarrow S$ (assoc., comm., …)
Resource Semantics of BI (O’Hearn and Pym)

Suppose states form a pre-ordered, partial monoid

- Set $\mathcal{S}$ of states, pre-order $\preceq$ on $\mathcal{S}$
- Partial operation $\circ: \mathcal{S} \times \mathcal{S} \rightarrow \mathcal{S}$ (assoc., comm., …)

Inductively define states that satisfy formulas
Resource Semantics of BI (O’Hearn and Pym)

Suppose states form a pre-ordered, partial monoid

- Set $S$ of states, pre-order $\sqsubseteq$ on $S$
- Partial operation $\circ: S \times S \rightarrow S$ (assoc., comm., ...)

Inductively define states that satisfy formulas

$s \models \top$ always
$s \models \bot$ never
Resource Semantics of BI (O’Hearn and Pym)

Suppose states form a pre-ordered, partial monoid

- Set $S$ of states, pre-order $\sqsubseteq$ on $S$
- Partial operation $\circ : S \times S \rightarrow S$ (assoc., comm., …)

Inductively define states that satisfy formulas

- $s \models \top$ always
- $s \models \bot$ never
- $s \models P \land Q$ iff $s \models P$ and $s \models Q$
Resource Semantics of BI (O’Hearn and Pym)

Suppose states form a pre-ordered, partial monoid

- Set $S$ of states, pre-order $\sqsubseteq$ on $S$
- Partial operation $\circ : S \times S \rightarrow S$ (assoc., comm., ...)

Inductively define states that satisfy formulas

\[
\begin{align*}
    s \models \top & \quad \text{always} \\
    s \models \bot & \quad \text{never} \\
    s \models P \land Q & \quad \text{iff } s \models P \text{ and } s \models Q \\
    s \models P \ast Q & \quad \text{iff } s_1 \circ s_2 \sqsubseteq s \text{ with } s_1 \models P \text{ and } s_2 \models Q
\end{align*}
\]

State $s$ can be split into two “disjoint” states, one satisfying $P$ and one satisfying $Q$
Example: Heap Model of BI

Set of states: heaps

$\Gamma S = \mathbb{N} \rightarrow \mathcal{V}$, partial maps from addresses to values
Example: Heap Model of BI

Set of states: heaps
▶ $S' = \mathbb{N} \rightarrow \mathcal{V}$, partial maps from addresses to values

Monoid operation: combine disjoint heaps
▶ $s_1 \circ s_2$ is defined to be union iff $\text{dom}(s_1) \cap \text{dom}(s_2) = \emptyset$
Example: Heap Model of BI

Set of states: heaps

- $S' = \mathbb{N} \rightarrow \mathcal{V}$, partial maps from addresses to values

Monoid operation: combine disjoint heaps

- $s_1 \circ s_2$ is defined to be union iff $\text{dom}(s_1) \cap \text{dom}(s_2) = \emptyset$

Pre-order: extend/project heaps

- $s_1 \sqsubseteq s_2$ iff $\text{dom}(s_1) \subseteq \text{dom}(s_2)$, and $s_1, s_2$ agree on $\text{dom}(s_1)$
Propositions for Heaps

Atomic propositions: “points-to”

- $x \mapsto v$ holds in heap $s$ iff $x \in \text{dom}(s)$ and $s(x) = v$

Example axioms (not complete)

- **Deterministic**: $x \mapsto v \land y \mapsto w \land x = y \rightarrow v = w$
- **Disjoint**: $x \mapsto v \ast y \mapsto w \rightarrow x \neq y$
The Separation Logic Proper

Programs $c$ from a basic imperative language

- Read from location: $x := *e$
- Write to location: $*e := e'$
The Separation Logic Proper

Programs \( c \) from a basic imperative language
- Read from location: \( x := *e \)
- Write to location: \( *e := e' \)

Program logic judgments

\[
\{ P \} \ c \ \{ Q \}
\]

Reading
Executing \( c \) on any input state satisfying \( P \) leads to an output state satisfying \( Q \), without invalid reads or writes.
Basic Proof Rules

Reading a location
\[
\{ x \mapsto v \} \ y := \star \ x \ \{ x \mapsto v \land y = v \} \]

Writing a location
\[
\{ x \mapsto v \} \star \ x := e \ \{ x \mapsto e \} \]

R/e.sc/a.sc/d.sc
W/r.sc/i.sc/t.sc/e.sc
Basic Proof Rules

Reading a location

\[
\{ x \mapsto v \} \ y := \ast x \ \{ x \mapsto v \land y = v \} \quad \text{READ}
\]
Basic Proof Rules

Reading a location

\[
\{ x \mapsto v \} \ y := \star x \ \{ x \mapsto v \land y = v \} \quad \text{READ}
\]

Writing a location

\[
\{ x \mapsto v \} \ \star x := e \ \{ x \mapsto e \} \quad \text{WRITE}
\]
The Frame Rule

Properties about unmodified heaps are preserved

\[
\begin{align*}
\{ P \} \ c \ \{ Q \} & \quad \text{\textit{c} doesn't modify } FV(R) \quad \boxed{ \text{\textsc{frame} } } \\
\{ P \ast R \} \ c \ \{ Q \ast R \}
\end{align*}
\]
The Frame Rule

Properties about unmodified heaps are preserved

\[
\begin{align*}
\{P\} & \ c & \{Q\} & \quad \text{c doesn’t modify } FV(R) \\
\{P \ast R\} & \ c & \{Q \ast R\}
\end{align*}
\]

So-called “local reasoning” in SL

- Only need to reason about part of heap used by \( c \)
- Note: doesn’t hold if \( \ast \) replaced by \( \wedge \), due to aliasing!
A Probabilistic Model of BI
States: Distributions over Memories

Fix sets $X$ of variables and $V$ of values

Memories indexed by domains $A \subseteq X$

$M(A) = A \rightarrow V$

Program states: randomized memories

States are distributions over memories with same domain

Formally:

$S = \{ s \mid s \in \text{Distr}(M(A)), A \subseteq X \}$

When $s \in \text{Distr}(M(A))$, write $\text{dom}(s)$ for $A$.
States: Distributions over Memories

Memories (not heaps)

- Fix sets $\mathcal{X}$ of variables and $\mathcal{V}$ of values
- Memories indexed by domains $A \subseteq \mathcal{X}$: $\mathcal{M}(A) = A \rightarrow \mathcal{V}$
States: Distributions over Memories

Memories (not heaps)

- Fix sets $\mathcal{X}$ of variables and $\mathcal{V}$ of values
- Memories indexed by domains $A \subseteq \mathcal{X}$: $M(A) = A \rightarrow \mathcal{V}$

Program states: randomized memories

- States are distributions over memories with same domain
- Formally: $S = \{s \mid s \in \text{Distr}(M(A)), A \subseteq \mathcal{X}\}$
- When $s \in \text{Distr}(M(A))$, write $\text{dom}(s)$ for $A$
Monoid: “Disjoint” Product Distribution

Intuition

- Two distributions can be combined iff domains are disjoint
- Combine by taking product distribution, union of domains
Monoid: “Disjoint” Product Distribution

**Intuition**
- Two distributions can be combined iff domains are disjoint
- Combine by taking product distribution, union of domains

**More formally...**
Suppose that \( s \in \text{Distr}(\mathcal{M}(A)) \) and \( s' \in \text{Distr}(\mathcal{M}(B)) \). If \( A, B \) are disjoint, then:

\[
(s \circ s')(m \cup m') = s(m) \cdot s'(m')
\]

for \( m \in \mathcal{M}(A) \) and \( m' \in \mathcal{M}(B) \). Otherwise, \( s \circ s' \) is undefined.
Pre-Order: Extension/Projection

Intuition

- Define $s \sqsubseteq s'$ if $s$ "has less information than" $s'$
- In probabilistic setting: $s$ is a projection of $s'$
Intuition

- Define $s \sqsubseteq s'$ if $s$ “has less information than” $s'$
- In probabilistic setting: $s$ is a projection of $s'$

More formally...

Suppose that $s \in \text{Distr}(\mathcal{M}(A))$ and $s' \in \text{Distr}(\mathcal{M}(B))$. Then $s \sqsubseteq s'$ iff $A \subseteq B$, and for all $m \in \mathcal{M}(A)$, we have:

$$s(m) = \sum_{m' \in \mathcal{M}(B)} s'(m \cup m').$$

That is, $s$ is obtained from $s'$ by marginalizing variables in $B \setminus A$. 
Atomic Formulas

Equalities

- $e = e'$ holds in $s$ iff all variables $FV(e, e') \subseteq \text{dom}(s)$, and $e$ is equal to $e'$ with probability 1 in $s$
Atomic Formulas

Equalities

- $e = e'$ holds in $s$ iff all variables $FV(e, e') \subseteq \text{dom}(s)$, and $e$ is equal to $e'$ with probability 1 in $s$

Distribution laws

- $e \sim \text{Unif}$ holds in $s$ iff $FV(e) \subseteq \text{dom}(s)$, and $e$ is uniformly distributed (e.g., fair coin flip)

- $e \sim \mathbb{D}$ holds in $s$ iff all variables in $FV(e) \subseteq \text{dom}(s)$
Example Axioms (not complete)

Distribution operations

\[ x \sim D \land y \sim D \rightarrow x \land y \sim D \]

Equality and distributions

\[ x = y \land x \sim \text{Unif} \rightarrow y \sim \text{Unif} \]

Uniformity and products

\[ (x \sim \text{Unif}) \land (y \sim \text{Unif}) \rightarrow (x, y) \sim \text{Unif} \times \text{Unif} \]

Uniformity and exclusive-or (\(\oplus\))

\[ x \sim \text{Unif} \land y \sim D \land z = x \oplus y \rightarrow z \sim \text{Unif} \land y \sim D \]
Example Axioms (not complete)

Distribution operations

$\begin{align*}
\text{▶ } & \quad x \sim D \land y \sim D \rightarrow x \land y \sim D \\
\end{align*}$
Example Axioms (not complete)

Distribution operations

▶ \( x \sim D \land y \sim D \rightarrow x \land y \sim D \)

Equality and distributions

▶ \( x = y \land x \sim \text{Unif} \rightarrow y \sim \text{Unif} \)
Example Axioms (not complete)

Distribution operations

\[ x \sim D \land y \sim D \rightarrow x \land y \sim D \]

Equality and distributions

\[ x = y \land x \sim \text{Unif} \rightarrow y \sim \text{Unif} \]

Uniformity and products

\[ (x \sim \text{Unif} \ast y \sim \text{Unif}) \rightarrow (x, y) \sim \text{Unif}_{\mathcal{B} \times \mathcal{B}} \]
Example Axioms (not complete)

**Distribution operations**
- $x \sim D \land y \sim D \rightarrow x \land y \sim D$

**Equality and distributions**
- $x = y \land x \sim \text{Unif} \rightarrow y \sim \text{Unif}$

**Uniformity and products**
- $(x \sim \text{Unif} \ast y \sim \text{Unif}) \rightarrow (x, y) \sim \text{Unif}_{B \times B}$

**Uniformity and exclusive-or (⊕)**
- $x \sim \text{Unif} \ast y \sim D \land z = x \oplus y \rightarrow z \sim \text{Unif} \ast y \sim D$
Intuitionistic, or Classical?

- Many SLs use classical version of BI (Boolean BI)
  - Pre-order is discrete (trivial)
  - Benefits: can describe heap domain exactly (e.g., empty)
  - Drawbacks: must describe the entire heap

- Our probabilistic model is for intuitionistic BI
  - Pre-order is nontrivial
  - Benefits: can describe a subset of the variables
  - Necessary: other variables might not be independent!
Intuitionistic, or Classical?

Many SLs use classical version of BI (Boolean BI)

- Pre-order is discrete (trivial)
- Benefits: can describe heap domain exactly (e.g., empty)
- Drawbacks: must describe the entire heap
Intuitionistic, or Classical?

Many SLs use classical version of BI (Boolean BI)

▷ Pre-order is discrete (trivial)
▷ Benefits: can describe heap domain exactly (e.g., empty)
▷ Drawbacks: must describe the entire heap

Our probabilistic model is for intuitionistic BI

▷ Pre-order is nontrivial
▷ Benefits: can describe a subset of the variables
▷ Necessary: other variables might not be independent!
A Probabilistic Separation Logic
A Toy Probabilistic Language

Program syntax

\[
\text{Exp} \ni e ::= x \in \mathcal{X} \mid tt \mid ff \mid e \land e' \mid e \lor e' \mid \cdots
\]

\[
\text{Com} \ni c ::= \text{skip} \mid x \leftarrow e \mid x \leftarrow \text{Unif} \mid c; c' \mid \text{if } e \text{ then } c \text{ else } c'
\]
A Toy Probabilistic Language

Program syntax

\[
\text{Exp} \ni e ::= x \in X \mid tt \mid ff \mid e \land e' \mid e \lor e' \mid \ldots \\
\text{Com} \ni c ::= \text{skip} \mid x \leftarrow e \mid x \leftarrow \text{Unif} \mid c; c' \mid \text{if } e \text{ then } c \text{ else } c'
\]
A Toy Probabilistic Language

Program syntax

\[
\begin{align*}
\text{Exp} \ni e & ::= x \in \mathcal{X} \mid \text{tt} \mid \text{ff} \mid e \land e' \mid e \lor e' \mid \cdots \\
\text{Com} \ni c & ::= \text{skip} \mid x \leftarrow e \mid x \leftarrow \text{Unif} \mid c ; c' \mid \text{if } e \text{ then } c \text{ else } c'
\end{align*}
\]

Semantics: distribution transformers (Kozen)

\[
\llbracket c \rrbracket : \text{Distr}(\mathcal{M}(\mathcal{X})) \rightarrow \text{Distr}(\mathcal{M}(\mathcal{X}))
\]
$P$ and $Q$ from probabilistic BI, $c$ a probabilistic program

$\{P\} \xrightarrow{c} \{Q\}$
Program Logic Judgments in PSL

\( P \text{ and } Q \) from probabilistic BI, \( c \) a probabilistic program

\( \{ P \} c \{ Q \} \)

Validity

For all input states \( s \in \text{Distr}(\mathcal{M}(\mathcal{X})) \) satisfying the pre-condition \( s \models P \), the output state \( \llbracket c \rrbracket s \) satisfies the post-condition \( \llbracket c \rrbracket s \models Q \).
Program Logic Judgments in PSL

$P$ and $Q$ from probabilistic BI, $c$ a probabilistic program

\[
\{P\} \ c \ \{Q\}
\]

Validity

For all input states $s \in \text{Distr}(\mathcal{M}(\mathcal{X}))$ satisfying the pre-condition $s \models P$, the output state $[c]s$ satisfies the post-condition $[c]s \models Q$. 
Basic Proof Rules in PSL
Assignment

\[ \frac{x \not\in FV(e)}{\{\top\} \ x \leftarrow e \ \{x = e\} \quad \text{ASSN} } \]
Basic Proof Rules in PSL

Assignment

\[ x \notin FV(e) \quad \Rightarrow \quad \{ \top \} \quad x \leftarrow e \{ x = e \} \]

Sampling

\[ \{ \top \} \quad x \leftarrow \text{Unif} \{ x \sim \text{Unif} \} \]
Conditional Rule in PSL

\[ Q \text{ is "supported"} \]

\[ \{ e = tt \ast P \} \ c \ \{ e = tt \ast Q \} \]

\[ \{ e = ff \ast P \} \ c' \ \{ e = ff \ast Q \} \]

\[ \{ e \sim D \ast P \} \text{ if } e \text{ then } c \text{ else } c' \ \{ e \sim D \ast Q \} \]
Conditional Rule in PSL

\[ Q \text{ is “supported”} \]
\[ \{ e = tt \ast P \} \ c \ \{ e = tt \ast Q \} \]
\[ \{ e = ff \ast P \} \ c' \ \{ e = ff \ast Q \} \]
\[ \{ e \sim D \ast P \} \ \text{if } e \ \text{then } c \ \text{else } c' \ \{ e \sim D \ast Q \} \]

Pre-conditions

- Inputs to branches derived from \textit{conditioning} on \( e \)
- Independence ensures that \( P \) holds after conditioning
Conditional Rule in PSL

\[
\begin{align*}
Q \text{ is “supported”} \\
\{e = tt \ast P\} \quad c \quad \{e = tt \ast Q\}
\end{align*}
\]

\[
\begin{align*}
\{e = ff \ast P\} \quad c' \quad \{e = ff \ast Q\}
\end{align*}
\]

\[
\begin{align*}
\{e \sim D \ast P\} \text{ if } e \text{ then } c \text{ else } c' \quad \{e \sim D \ast Q\}
\end{align*}
\]

Pre-conditions

- Inputs to branches derived from conditioning on \( e \)
- Independence ensures that \( P \) holds after conditioning

Post-conditions

- Not all post-conditions \( Q \) can be soundly combined
- “Supported”: \( Q \) describes unique distribution (Reynolds)
The Frame Rule in PSL

\[
\{P\} \ c \ \{Q\} \quad \quad FV(R) \cap MV(c) = \emptyset
\]
\[
| \quad = \quad P \rightarrow RV(c) \sim D \quad \quad FV(Q) \subseteq RV(c) \cup WV(c)
\]
\[
\{P \ast R\} \ c \ \{Q \ast R\}
\]

Side conditions
The Frame Rule in PSL

\[
\begin{align*}
\{P\} &\ c \ \{Q\} \quad FV(R) \cap MV(c) = \emptyset \\
| = P \rightarrow RV(c) \sim D &\quad FV(Q) \subseteq RV(c) \cup WV(c) \\
\{P \ast R\} &\ c \ \{Q \ast R\}
\end{align*}
\]

Side conditions

1. Variables in \( R \) are not modified (standard in SL)
The Frame Rule in PSL

\[
\begin{align*}
\{P\} & \ c \ \{Q\} \quad FV(R) \cap MV(c) = \emptyset \\
\models P \rightarrow RV(c) \sim \mathbf{D} \quad FV(Q) \subseteq RV(c) \cup WV(c) \\
\{P \ast R\} & \ c \ \{Q \ast R\}
\end{align*}
\]

Side conditions

1. Variables in \(R\) are not modified (standard in SL)
2. \(P\) describes all variables that might be read
The Frame Rule in PSL

\[
\begin{align*}
\{P\} & \ c \ \{Q\} \quad & FV(R) \cap MV(c) = \emptyset \\
\models P \rightarrow RV(c) \sim D & \quad & FV(Q) \subseteq RV(c) \cup WV(c) \\
\{P \ast R\} & \ c \ \{Q \ast R\} 
\end{align*}
\]

Side conditions

1. Variables in \( R \) are not modified (standard in SL)
2. \( P \) describes all variables that might be read
3. Everything in \( Q \) is freshly written, or in \( P \)
The Frame Rule in PSL

\[
\begin{align*}
\{ P \} & \ c \ \{ Q \} \quad FV(R) \cap MV(c) = \emptyset \\
\models P \rightarrow RV(c) \sim D \quad FV(Q) \subseteq RV(c) \cup WV(c) \\
\{ P * R \} & \ c \ \{ Q * R \}
\end{align*}
\]

Side conditions

1. Variables in \( R \) are not modified (standard in SL)
2. \( P \) describes all variables that might be read
3. Everything in \( Q \) is freshly written, or in \( P \)

Variables in the post \( Q \) were independent of \( R \), or are newly independent of \( R \)
Example: Deriving a Better Sampling Rule

Given rules:

\[
\begin{align*}
\{P\} \ c \ \{Q\} & \quad FV(R) \cap MV(c) = \emptyset \\
\models P \rightarrow RV(c) \sim D & \quad FV(Q) \subseteq RV(c) \cup WV(c) \\
\{P \ast R\} \ c \ \{Q \ast R\} & \\
\{\top\} \ x \ \leftarrow^\$ \ \text{Unif} \ \{x \sim \text{Unif}\}
\end{align*}
\]
Example: Deriving a Better Sampling Rule

Given rules:

\[
\begin{align*}
\{P\} c \{Q\} & \quad FV(R) \cap MV(c) = \emptyset \\
\models P \rightarrow RV(c) \sim D & \quad FV(Q) \subseteq RV(c) \cup WV(c) & \text{FRAME} \\
\{P \ast R\} c \{Q \ast R\} & \quad \text{SAMP}
\end{align*}
\]

Can derive:

\[
\begin{align*}
x \notin FV(R) & \quad \text{SAMP*} \\
\{R\} x \leftarrow \text{Unif} \{x \sim \text{Unif} * R\}
\end{align*}
\]
Example: Deriving a Better Sampling Rule

Given rules:

\[
\begin{align*}
\{P\} & \ c \ \{Q\} & \ FV(R) \cap MV(c) = \emptyset \\
\models P \rightarrow RV(c) \sim D & \ FV(Q) \subseteq RV(c) \cup WV(c) \\
\{P \ast R\} & \ c \ \{Q \ast R\} \\
\{\top\} & \ x \ \Leftarrow \ \text{Unif} \ \{x \sim \text{Unif}\}
\end{align*}
\]

Can derive:

\[
\begin{align*}
x \notin FV(R) & \\
\{R\} & \ x \ \Leftarrow \ \text{Unif} \ \{x \sim \text{Unif} \ast R\}
\end{align*}
\]

Intuitively: fresh random sample is independent of everything
Theorem (Restriction)

Let \( P \) be any formula of probabilistic BI, and suppose that \( s \models P \). Then there exists \( s' \sqsubseteq s \) such that \( s' \models P \) and \( \text{dom}(s') = \text{dom}(s) \cap \text{FV}(P) \).

Intuition

- The only variables that “matter” for \( P \) are \( \text{FV}(P) \)
- Tricky for implications; proof “glues” distributions
Verifying an Example
One-Time-Pad (OTP)

Possibly the simplest encryption scheme

- Input: a message $m \in \mathbb{B}$
- Output: a ciphertext $c \in \mathbb{B}$
- Idea: encrypt by taking xor with a uniformly random key $k$
One-Time-Pad (OTP)

Possibly the simplest encryption scheme

- Input: a message $m \in \mathbb{B}$
- Output: a ciphertext $c \in \mathbb{B}$
- Idea: encrypt by taking xor with a uniformly random key $k$

The encoding program:

$$k \leftarrow \text{Unif}_{\mathbb{B}}$$
$$c \leftarrow k \oplus m$$
How to Formalize Security?

Method /one.osf: Uniformity

▶ Show that $c$ is uniformly distributed
▶ Always the same, no matter what the message $m$ is

Method /two.osf: Input-output independence

▶ Assume that $m$ is drawn from some (unknown) distribution
▶ Show that $c$ and $m$ are independent
How to Formalize Security?

Method 1: Uniformity

- Show that $c$ is uniformly distributed
- Always the same, no matter what the message $m$ is
How to Formalize Security?

Method 1: Uniformity

- Show that $c$ is uniformly distributed
- Always the same, no matter what the message $m$ is

Method 2: Input-output independence

- Assume that $m$ is drawn from some (unknown) distribution
- Show that $c$ and $m$ are independent
Proving Input-Output Independence for OTP in PSL

\[ k \leftarrow \text{Unif}_\varphi \]

\[ c \leftarrow k \oplus m \]
\{m \sim D\} 

\k \leftarrow \text{Unif}^\$ 

c \leftarrow k \oplus m 

\text{assumption}
Proving Input-Output Independence for OTP in PSL

\{m \sim D\}

\[k \leftarrow \text{Unif}\]

\{m \sim D \ast k \sim \text{Unif}\}

\[c \leftarrow k \oplus m\]
{m \sim D}

\begin{align*}
k & \leftarrow \text{Unif}^\circ \\
\{m \sim D \ast k \sim \text{Unif}\} & \quad [\text{SAMP}^*] \\
c & \leftarrow k \oplus m \\
\{m \sim D \ast k \sim \text{Unif} \land c = k \oplus m\} & \quad [\text{ASSN}^*]
\end{align*}
Proving Input-Output Independence for OTP in PSL

\{ m \sim D \} \hspace{2cm} \text{assumption}

k \leftarrow \text{Unif}_9

\{ m \sim D * k \sim \text{Unif} \} \hspace{2cm} [\text{SAMP}^*]

c \leftarrow k \oplus m

\{ m \sim D * k \sim \text{Unif} \land c = k \oplus m \} \hspace{2cm} [\text{ASSN}^*]

\{ m \sim D * c \sim \text{Unif} \} \hspace{2cm} \text{XOR axiom}
Recent Directions:
Conditional Independence
What is Conditional Independence (CI)?

Two random variables $x$ and $y$ are independent conditioned on $z$ if they are only correlated through $z$: fixing any value of $z$, the value of $x$ gives no information about the value of $y$. 
Main Idea: Lift to Markov Kernels

Maps of type $\mathcal{M}(S) \rightarrow \text{Distr}(\mathcal{M}(T))$

- $S \subseteq T$: maps must “preserve input to output”
- Plain distributions encoded as $\mathcal{M}(\emptyset) \rightarrow \text{Distr}(\mathcal{M}(T))$
Main Idea: Lift to Markov Kernels

Maps of type $\mathcal{M}(S) \rightarrow \text{Distr}(\mathcal{M}(T))$

- $S \subseteq T$: maps must “preserve input to output”
- Plain distributions encoded as $\mathcal{M}(\emptyset) \rightarrow \text{Distr}(\mathcal{M}(T))$

CI expressible in terms of kernels
Let $\odot$ be Kleisli composition and $\otimes$ be “parallel” composition. If we can decompose:

$$\mu = \mu_z \odot (\mu_x \otimes \mu_y)$$

with $\mu_x : \mathcal{M}(z) \rightarrow \text{Distr}(\mathcal{M}(x, z))$, $\mu_y : \mathcal{M}(z) \rightarrow \text{Distr}(\mathcal{M}(y, z))$, then $x$ and $y$ are independent conditioned on $z$. 
DIBI: Dependent and Independent BI

Main idea: add a non-commutative conjunction $P \# Q$

States are now kernels

$P \ast Q$: parallel composition of kernels

$P \# Q$: Kleisli composition of kernels

Interaction: reverse exchange law

$(P \# Q) \ast (R \# S) \vdash (P \ast R) \# (Q \ast S)$

Reverse of the usual direction (cf. Concurrent Kleene Algebra)
Main idea: add a non-commutative conjunction $P ; Q$

- States are now kernels
- $P \ast Q$: parallel composition of kernels
- $P ; Q$: Kleisli composition of kernels
DIBI: Dependent and Independent BI

Main idea: add a non-commutative conjunction $P ; Q$

- States are now kernels
- $P * Q$: parallel composition of kernels
- $P ; Q$: Kleisli composition of kernels

Interaction: reverse exchange law

$$(P ; Q) * (R ; S) \vdash (P * R) ; (Q * S)$$

Reverse of the usual direction (cf. Concurrent Kleene Algebra)
See the Papers for More Details

A Probabilistic Separation Logic (POPL 2020)
- Extensions to PSL: deterministic variables, loops, etc.
- Many examples from cryptography, security of ORAM

A Logic to Reason about Dependence and Independence
- Details about DIBI, sound and complete Hilbert system
- Models capturing join dependency in relational algebra
- A separation logic (CPSL) based on DIBI
- arXiv: available soon, or send an email
A Probabilistic Separation Logic