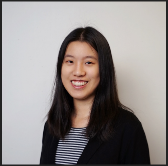
A Probabilistic Separation Logic

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:

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

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 μ are independent if μ is the product of two distributions μ_x and μ_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?

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 $s : \mathcal{X} \to \mathcal{V}$, map from variables to values
- A heap $h : \mathbb{N} \rightharpoonup \mathcal{V}$, partial map from addresses to values

Classical Setting: Heaps

Program states (s, h)

- A store $s : \mathcal{X} \to \mathcal{V}$, map from variables to values
- A heap $h : \mathbb{N} \rightharpoonup \mathcal{V}$, 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

Assertion Logic: Bunched Implications (BI)

Substructural logic (O'Hearn and Pym)

- Start with regular propositional logic $(\top, \bot, \wedge, \vee, \rightarrow)$
- Add a new conjunction ("star"): $P * Q$
- Add a new implication ("magic wand"): $P \mathrel{-\!\!*} Q$

Assertion Logic: Bunched Implications (BI)

Substructural logic (O'Hearn and Pym)

- Start with regular propositional logic $(\top, \bot, \wedge, \vee, \rightarrow)$
- Add a new conjunction ("star"): $P * Q$
- Add a new implication ("magic wand"): $P \mathrel{-\!\!*} Q$

Star is a multiplicative conjunction

- $P \wedge Q$: P and Q hold on the entire state
- $P * Q$: 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 $\circ : S \times S \rightharpoonup S$ (assoc., comm., ...)

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 $\circ : S \times S \rightharpoonup S$ (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 $\circ : S \times S \rightharpoonup S$ (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 $\circ : S \times S \rightharpoonup S$ (assoc., comm., …)

Inductively define states that satisfy formulas

$s \models \top$	always
$s \models \bot$	never
$s \models P \wedge Q$	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 $\circ : S \times S \rightharpoonup S$ (assoc., comm., ...)

Inductively define states that satisfy formulas

$s \models \top$	always
$s \models \bot$	never
$s \models P \wedge Q$	iff $s \models P$ and $s \models Q$
$s \models P * Q$	iff $s_1 \circ s_2 \sqsubseteq s$ with $s_1 \models P$ and $s_2 \models Q$

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

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

Example: Heap Model of BI

Set of states: heaps

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

Monoid operation: combine disjoint heaps

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

Example: Heap Model of BI

Set of states: heaps

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

Monoid operation: combine disjoint heaps

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

Pre-order: extend/project heaps

- $s_1 \sqsubseteq s_2$ iff $\mathsf{dom}(s_1) \subseteq \mathsf{dom}(s_2)$, and s_1, s_2 agree on $\mathsf{dom}(s_1)$

Propositions for Heaps

Atomic propositions: "points-to"

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

Example axioms (not complete)

- Deterministic: $x \mapsto v \wedge y \mapsto w \wedge x = y \rightarrow v = w$
- Disjoint: $x \mapsto v * y \mapsto w \rightarrow x \neq y$

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

Basic Proof Rules

Reading a location

$$\frac{}{\{x \mapsto v\}\ y := *x\ \{x \mapsto v \land y = v\}}\ \textsc{Read}$$

Basic Proof Rules

Reading a location

$$\frac{}{\{x \mapsto v\}\ y := *x\ \{x \mapsto v \land y = v\}}\ \textsc{Read}$$

Writing a location

$$\frac{}{\{x \mapsto v\}\ *x := e\ \{x \mapsto e\}}\ \textsc{Write}$$

The Frame Rule

Properties about unmodified heaps are preserved

$$\frac{\{P\}\ c\ \{Q\} \qquad c \text{ doesn't modify } FV(R)}{\{P * R\}\ c\ \{Q * R\}}\ \textsc{Frame}$$

The Frame Rule

Properties about unmodified heaps are preserved

$$\frac{\{P\}\ c\ \{Q\} \qquad c \text{ doesn't modify } FV(R)}{\{P * R\}\ c\ \{Q * R\}}\ \textsc{Frame}$$

So-called "local reasoning" in SL

- Only need to reason about part of heap used by c
- Note: doesn't hold if $*$ replaced by $\wedge$, due to aliasing!

A Probabilistic Model of BI

States: Distributions over Memories

States: Distributions over Memories

Memories (not heaps)

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

States: Distributions over Memories

Memories (not heaps)

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

Program states: randomized memories

- States are distributions over memories with same domain
- Formally: $S = \{s \mid s \in \mathsf{Distr}(\mathcal{M}(A)), A \subseteq \mathcal{X}\}$
- When $s \in \mathsf{Distr}(\mathcal{M}(A))$, write $\mathsf{dom}(s)$ 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 $s \in \mathsf{Distr}(\mathcal{M}(A))$ and $s' \in \mathsf{Distr}(\mathcal{M}(B))$. 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'

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'

More formally...

Suppose that $s \in \mathrm{Distr}(\mathcal{M}(A))$ and $s' \in \mathrm{Distr}(\mathcal{M}(B))$. 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 $FV(e, e') \subseteq \mathsf{dom}(s)$, 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

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

Example Axioms (not complete)

Example Axioms (not complete)

Distribution operations

- $x \sim \mathbf{D} \wedge y \sim \mathbf{D} \rightarrow x \wedge y \sim \mathbf{D}$

Example Axioms (not complete)

Distribution operations

- $x \sim \mathbf{D} \wedge y \sim \mathbf{D} \to x \wedge y \sim \mathbf{D}$

Equality and distributions

- $x = y \wedge x \sim \mathbf{Unif} \to y \sim \mathbf{Unif}$

Example Axioms (not complete)

Distribution operations

- $x \sim \mathbf{D} \wedge y \sim \mathbf{D} \to x \wedge y \sim \mathbf{D}$

Equality and distributions

- $x = y \wedge x \sim \mathbf{Unif} \to y \sim \mathbf{Unif}$

Uniformity and products

- $(x \sim \mathbf{Unif} * y \sim \mathbf{Unif}) \to (x, y) \sim \mathbf{Unif}_{\mathbb{B}\times\mathbb{B}}$

Example Axioms (not complete)

Distribution operations

- $x \sim \mathbf{D} \wedge y \sim \mathbf{D} \rightarrow x \wedge y \sim \mathbf{D}$

Equality and distributions

- $x = y \wedge x \sim \mathbf{Unif} \rightarrow y \sim \mathbf{Unif}$

Uniformity and products

- $(x \sim \mathbf{Unif} * y \sim \mathbf{Unif}) \rightarrow (x, y) \sim \mathbf{Unif}_{\mathbb{B} \times \mathbb{B}}$

Uniformity and exclusive-or ($\oplus$)

- $x \sim \mathbf{Unif} * y \sim \mathbf{D} \wedge z = x \oplus y \rightarrow z \sim \mathbf{Unif} * y \sim \mathbf{D}$

Intuitionistic, or Classical?

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

$$\mathsf{Exp} \ni e ::= x \in \mathcal{X} \mid tt \mid f\!f \mid e \wedge e' \mid e \vee e' \mid \cdots$$

$$\mathsf{Com} \ni c ::= \mathsf{skip} \mid x \leftarrow e \mid \; x \xleftarrow{\$} \mathbf{Unif} \; \mid c; c' \mid \mathsf{if}\ e\ \mathsf{then}\ c\ \mathsf{else}\ c'$$

A Toy Probabilistic Language

Program syntax

$$\mathsf{Exp} \ni e ::= x \in \mathcal{X} \mid \mathit{tt} \mid \mathit{ff} \mid e \wedge e' \mid e \vee e' \mid \cdots$$

$$\mathsf{Com} \ni c ::= \mathsf{skip} \mid x \leftarrow e \mid \boxed{x \xleftarrow{\$} \mathbf{Unif}} \mid c;\, c' \mid \mathsf{if}\ e\ \mathsf{then}\ c\ \mathsf{else}\ c'$$

A Toy Probabilistic Language

Program syntax

$$\mathsf{Exp} \ni e ::= x \in \mathcal{X} \mid tt \mid f\!f \mid e \wedge e' \mid e \vee e' \mid \cdots$$

$$\mathsf{Com} \ni c ::= \mathsf{skip} \mid x \leftarrow e \mid \boxed{x \overset{\$}{\leftarrow} \mathbf{Unif}} \mid c; c' \mid \mathsf{if}\ e\ \mathsf{then}\ c\ \mathsf{else}\ c'$$

Semantics: distribution transformers (Kozen)

$$[\![c]\!] : \mathsf{Distr}(\mathcal{M}(\mathcal{X})) \to \mathsf{Distr}(\mathcal{M}(\mathcal{X}))$$

Program Logic Judgments in PSL

P and Q from probabilistic BI, c a probabilistic program

$$\{P\}\ c\ \{Q\}$$

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 $[\![c]\!]s \models Q$.

Program Logic Judgments in PSL

P and Q from probabilistic BI, c a probabilistic program

$$\{P\}\ c\ \{Q\}$$

Validity

For all input states $s \in \mathsf{Distr}(\mathcal{M}(\mathcal{X}))$ 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

Basic Proof Rules in PSL

Assignment

$$\frac{x \notin FV(e)}{\{\top\}\; x \leftarrow e\; \{x = e\}}\ \textsc{Assn}$$

Basic Proof Rules in PSL

Assignment

$$\frac{x \notin FV(e)}{\{\top\}\ x \leftarrow e\ \{x = e\}}\ \textsc{Assn}$$

Sampling

$$\frac{}{\{\top\}\ x \xleftarrow{\$} \mathbf{Unif}\ \{x \sim \mathbf{Unif}\}}\ \textsc{Samp}$$

Conditional Rule in PSL

$$\frac{Q \text{ is ``supported''} \quad \{e = tt * P\}\; c\; \{e = tt * Q\} \quad \{e = f\!f * P\}\; c'\; \{e = f\!f * Q\}}{\{e \sim \mathbf{D} * P\}\ \mathsf{if}\ e\ \mathsf{then}\ c\ \mathsf{else}\ c'\ \{e \sim \mathbf{D} * Q\}}\ \textsc{Cond}$$

Conditional Rule in PSL

$$\frac{Q \text{ is ``supported''} \quad \{e = tt * P\}\ c\ \{e = tt * Q\} \quad \{e = ff * P\}\ c'\ \{e = ff * Q\}}{\{e \sim \mathbf{D} * P\}\ \text{if } e \text{ then } c \text{ else } c'\ \{e \sim \mathbf{D} * Q\}}\ \textsc{Cond}$$

Pre-conditions

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

Conditional Rule in PSL

$$\dfrac{\begin{array}{c} Q \text{ is ``supported''} \\ \{e = tt * P\}\ c\ \{e = tt * Q\} \\ \{e = ff * P\}\ c'\ \{e = ff * Q\} \end{array}}{\{e \sim \mathbf{D} * P\}\ \text{if } e \text{ then } c \text{ else } c'\ \{e \sim \mathbf{D} * Q\}}\ \textsc{Cond}$$

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

$$\frac{\{P\}\, c\, \{Q\} \qquad FV(R) \cap MV(c) = \emptyset \\ \models P \to RV(c) \sim \mathbf{D} \qquad FV(Q) \subseteq RV(c) \cup WV(c)}{\{P * R\}\, c\, \{Q * R\}} \textsc{Frame}$$

Side conditions

The Frame Rule in PSL

$$\frac{\{P\}\ c\ \{Q\} \qquad FV(R) \cap MV(c) = \emptyset \\ \models P \to RV(c) \sim \mathbf{D} \qquad FV(Q) \subseteq RV(c) \cup WV(c)}{\{P * R\}\ c\ \{Q * R\}}\ \textsc{Frame}$$

Side conditions

1. Variables in R are not modified (standard in SL)

The Frame Rule in PSL

$$\frac{\{P\}\ c\ \{Q\} \qquad FV(R) \cap MV(c) = \emptyset \qquad \models P \to RV(c) \sim \mathbf{D} \qquad FV(Q) \subseteq RV(c) \cup WV(c)}{\{P * R\}\ c\ \{Q * R\}}\ \textsc{Frame}$$

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

$$\frac{\{P\}\, c\, \{Q\} \qquad FV(R) \cap MV(c) = \emptyset \\ \models P \to RV(c) \sim \mathbf{D} \qquad FV(Q) \subseteq RV(c) \cup WV(c)}{\{P * R\}\, c\, \{Q * R\}} \ \textsc{Frame}$$

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

$$\frac{\{P\}\, c\, \{Q\} \qquad FV(R) \cap MV(c) = \emptyset \qquad \models P \to RV(c) \sim \mathbf{D} \qquad FV(Q) \subseteq RV(c) \cup WV(c)}{\{P * R\}\, c\, \{Q * R\}}\ \textsc{Frame}$$

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:

$$\frac{\{P\}\, c\, \{Q\} \qquad FV(R) \cap MV(c) = \emptyset \qquad \models P \to RV(c) \sim \mathbf{D} \qquad FV(Q) \subseteq RV(c) \cup WV(c)}{\{P * R\}\, c\, \{Q * R\}} \ \textsc{Frame}$$

$$\frac{}{\{\top\}\, x \xleftarrow{\$} \mathbf{Unif}\ \{x \sim \mathbf{Unif}\}} \ \textsc{Samp}$$

Example: Deriving a Better Sampling Rule

Given rules:

$$\frac{\{P\}\, c\, \{Q\} \qquad FV(R) \cap MV(c) = \emptyset \qquad \models P \to RV(c) \sim \mathbf{D} \qquad FV(Q) \subseteq RV(c) \cup WV(c)}{\{P * R\}\, c\, \{Q * R\}}\ \textsc{Frame}$$

$$\frac{}{\{\top\}\ x \xleftarrow{\$} \mathbf{Unif}\ \{x \sim \mathbf{Unif}\}}\ \textsc{Samp}$$

Can derive:

$$\frac{x \notin FV(R)}{\{R\}\ x \xleftarrow{\$} \mathbf{Unif}\ \{x \sim \mathbf{Unif} * R\}}\ \textsc{Samp}^*$$

Example: Deriving a Better Sampling Rule

Given rules:

$$\frac{\{P\}\, c\, \{Q\} \qquad FV(R) \cap MV(c) = \emptyset \qquad \models P \to RV(c) \sim \mathbf{D} \qquad FV(Q) \subseteq RV(c) \cup WV(c)}{\{P * R\}\, c\, \{Q * R\}} \ \textsc{Frame}$$

$$\frac{}{\{\top\}\ x \xleftarrow{\$} \mathbf{Unif}\ \{x \sim \mathbf{Unif}\}} \ \textsc{Samp}$$

Can derive:

$$\frac{x \notin FV(R)}{\{R\}\ x \xleftarrow{\$} \mathbf{Unif}\ \{x \sim \mathbf{Unif} * R\}} \ \textsc{Samp*}$$

Intuitively: fresh random sample is independent of everything

Key Property for Soundness: Restriction

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 $\mathit{dom}(s') = \mathit{dom}(s) \cap FV(P)$.

Intuition

- The only variables that "matter" for P are $FV(P)$
- 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:

$$
\begin{aligned}
&k \xleftarrow{\$} \mathbf{Unif} \,;\\
&c \leftarrow k \oplus m
\end{aligned}
$$

How to Formalize Security?

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

$$k \xleftarrow{\$} \mathbf{Unif};$$

$$c \leftarrow k \oplus m$$

Proving Input-Output Independence for OTP in PSL

$\{m \sim \mathbf{D}\}$ assumption

$k \xleftarrow{\$} \mathbf{Unif};$

$c \leftarrow k \oplus m$

Proving Input-Output Independence for OTP in PSL

$\{m \sim \mathbf{D}\}$ assumption

$k \xleftarrow{\$} \mathbf{Unif};$

$\{m \sim \mathbf{D} * k \sim \mathbf{Unif}\}$ [SAMP*]

$c \leftarrow k \oplus m$

Proving Input-Output Independence for OTP in PSL

$\{m \sim \mathbf{D}\}$ assumption

$k \xleftarrow{\$} \mathbf{Unif};$

$\{m \sim \mathbf{D} * k \sim \mathbf{Unif}\}$ [Samp*]

$c \leftarrow k \oplus m$

$\{m \sim \mathbf{D} * k \sim \mathbf{Unif} \wedge c = k \oplus m\}$ [Assn*]

Proving Input-Output Independence for OTP in PSL

$\{m \sim \mathbf{D}\}$ assumption

$k \xleftarrow{\$} \mathbf{Unif};$

$\{m \sim \mathbf{D} * k \sim \mathbf{Unif}\}$ [SAMP*]

$c \leftarrow k \oplus m$

$\{m \sim \mathbf{D} * k \sim \mathbf{Unif} \wedge c = k \oplus m\}$ [ASSN*]

$\{m \sim \mathbf{D} * c \sim \mathbf{Unif}\}$ 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 $\mathcal{M}(S) \to \text{Distr}(\mathcal{M}(T))$

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

Main Idea: Lift to Markov Kernels

Maps of type $\mathcal{M}(S) \to \mathsf{Distr}(\mathcal{M}(T))$

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

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 $\mu_x : \mathcal{M}(z) \to \mathsf{Distr}(\mathcal{M}(x, z))$, $\mu_y : \mathcal{M}(z) \to \mathsf{Distr}(\mathcal{M}(y, z))$, then x and y are independent conditioned on z.

DIBI: Dependent and Independent BI

DIBI: Dependent and Independent BI

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

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

DIBI: Dependent and Independent BI

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

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

Interaction: reverse exchange law

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

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
- arXiv: `https://arxiv.org/abs/1907.10708`

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

Justin Hsu
UW–Madison
Computer Sciences