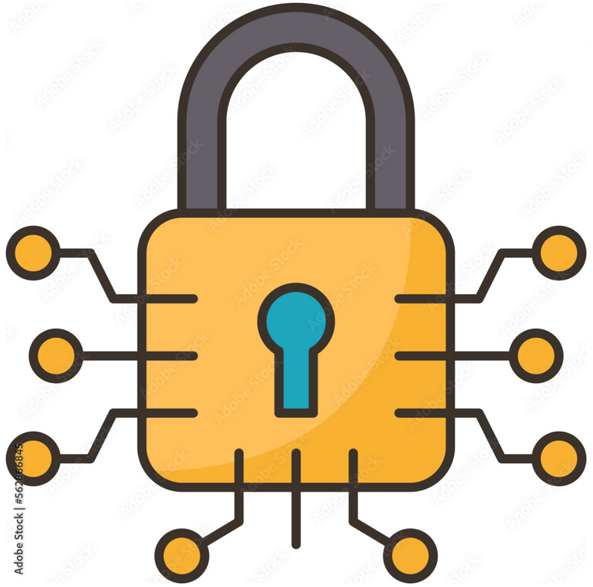
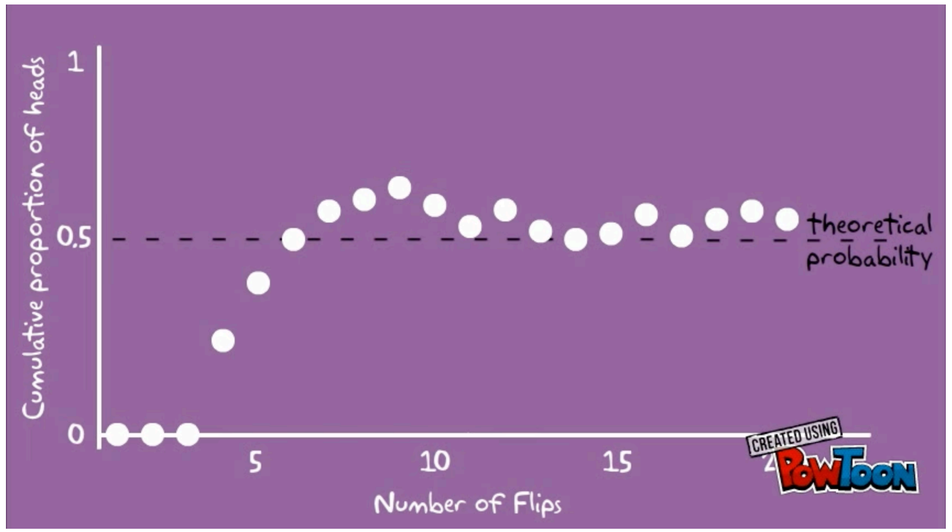
Automating Probabilistic Separation Logic

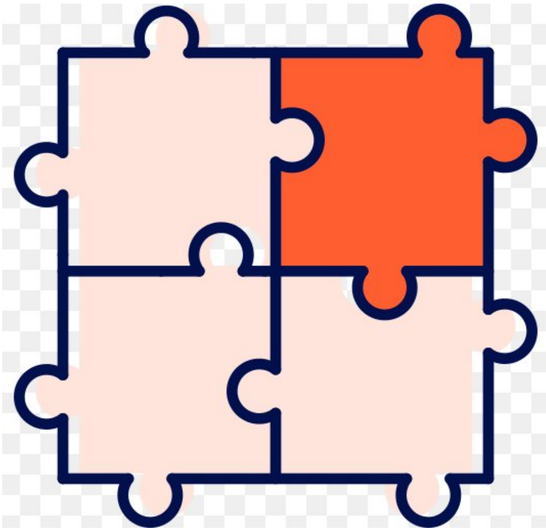
Joint work with Jessica Cho, Steven Holtzen and Justin Hsu

Jialu Bao Software Day Mar. 10th 2026

Probabilistic Independence is Helpful







Security

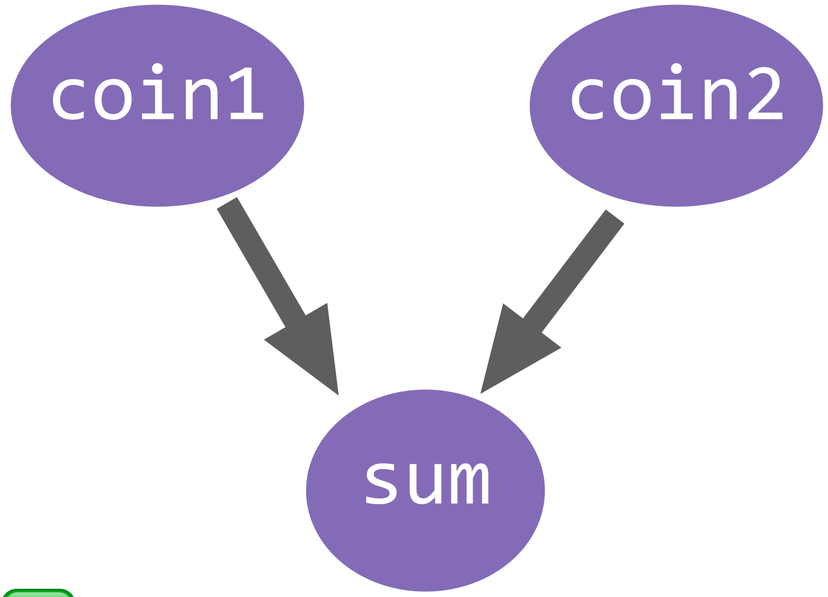
The Law of Large Numbers

Modular Reasoning

How to Identify Independence between Program Variables?

Graphical Approach:

```
coin1 :~ Bern(0.5);
coin2 :~ Bern(0.5);
sum := coin1 + coin2
```

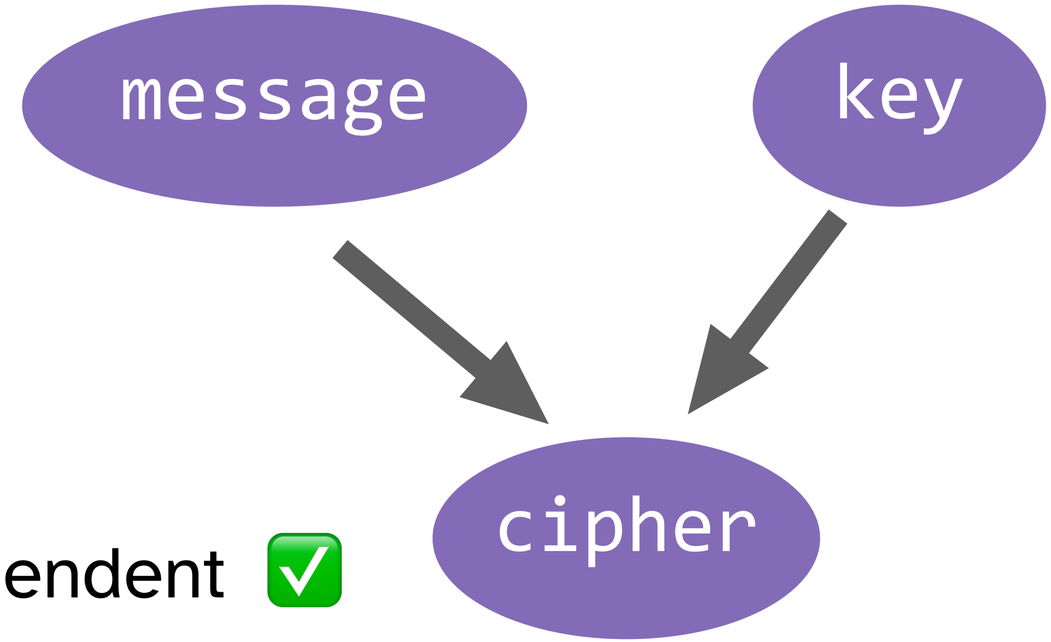


coin1, coin2 independent ✅

How to Identify Independence between Program Variables?

Problem with the Graphical Approach:

```
message :~ Bern(0.7);
key :~ Bern(0.5);
cipher := message xor key
```



message, key independent ✅

message, cipher independent ❓❓

A Logical Approach: Probabilistic Separation Logic

[Barthe, Hsu and Liao 2019]

Assertion logic: specifying program states

The Logic of Bunched Implications [O'Hearn and Pym 1999]

$$P, Q ::= p \in \mathscr{AP} \mid \top \mid \bot \mid P \wedge Q \mid P \vee Q \mid P \Rightarrow Q \mid P * Q \mid P \mathrel{-\!\!*} Q$$

E.g. $\texttt{message} \sim \mathrm{Bern}(0.7) * \texttt{key} \sim \mathrm{Bern}(0.5)$

A Logical Approach: Probabilistic Separation Logic

Program logic: capturing how programs updates states

Judgment

$$\text{program} ::= \text{skip} \mid x := e \mid c_1; c_2 \mid \text{if}_R\ b\ \text{then}\ c_1\ \text{else}\ c_2 \mid \text{while}\ b\ \text{do}\ c \mid x :\sim \text{Unif}_S$$

Distribution Transformer Semantics

$$\textsc{Seqn}\ \frac{\vdash \{\phi\}\ c\ \{\psi\} \qquad \vdash \{\psi\}\ c'\ \{\eta\}}{\vdash \{\phi\}\ c\ ;\ c'\ \{\eta\}}$$

$$\textsc{RSamp}\ \frac{}{\vdash \{\top\}\ x_r :\sim \text{Unif}_S\ \{x_r \sim \text{Unif}_S\}}$$

Program Rules

A Logical Approach: Probabilistic Separation Logic

Program logic: capturing how programs updates states

Structural Rules

$$\textsc{Weak}\ \frac{\vdash \{\phi\}\ c\ \{\psi\} \qquad \phi' \vDash \phi \qquad \psi \vDash \psi'}{\vdash \{\phi'\}\ c\ \{\psi'\}}$$

$$\text{Frame}\ \frac{\vdash \{\phi\}\ c\ \{\psi\} \qquad \text{side conditions}}{\vdash \{\phi\ *\ \eta\}\ c\ \{\psi\ *\ \eta\}}$$

Problem: Need to Construct Proofs Manually

$$\vdash \{\top\}$$

```
message :~ Bern(0.7);
key :~ Bern(0.5);
cipher := message xor key
```

$$\{\text{cipher} * \text{message}\}$$

Can We Automate the Proof Constructions?

Multiple Rules Apply at Each Step

Option 1:

SEQN

$$\vdash \{\top\}\ \texttt{message :\sim Bern(0.7); key :\sim Bern(0.5); cipher := message xor key}\ \{\texttt{cipher} * \texttt{message}\}$$

Multiple Rules Apply at Each Step

Other options:

$$\vdash \{\top\} \quad \texttt{same program} \quad \left\{ \begin{array}{l} \textbf{Many more options here} \\ \texttt{key} \sim \mathrm{Bern}(0.5) * \texttt{message} \wedge \\ \texttt{[cipher = message xor key]} \end{array} \right\} \quad \ldots$$

WEAK

$$\vdash \{\top\} \quad \begin{array}{l} \texttt{message :\sim Bern(0.7);} \\ \texttt{key :\sim Bern(0.5);} \\ \texttt{cipher := message xor key} \end{array} \quad \{\texttt{cipher} * \texttt{message}\}$$

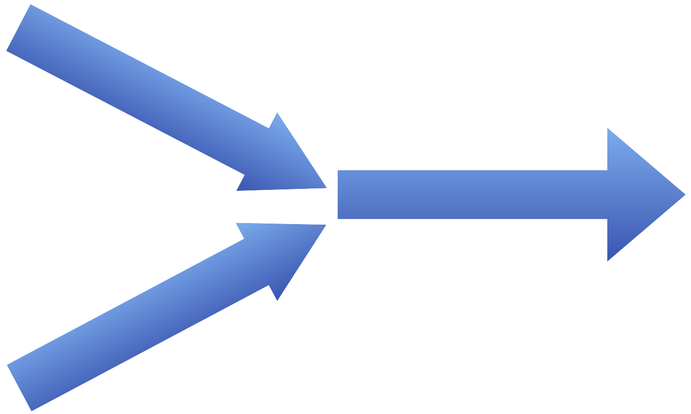
Multiple rules can apply at each step, but only some leads to a successful proof.

It is difficult for an algorithm to pick.

Our Solution to "don't know which rule to pick": A Syntax-Directed Probabilistic Separation Logic

Program Rules

Structural Rules



Syntax-Directed PSL

Exactly one augmented rule for each command.

$$
\begin{array}{l}
\quad \text{skip} \\
\mid \; x := e \\
\mid \; c_1; c_2 \\
\mid \; \text{if}_R \; b \; \text{then} \; c_1 \; \text{else} \; c_2 \\
\mid \; \sout{\text{while } b \text{ do } c} \\
\mid \; x :\sim \text{Unif}_S
\end{array}
$$

Designed to give a strong postcondition.

How to check Proved Postcondition implies the Desired Postcondition?

A Sound and Complete Decision Procedure for Checking Assertion Entailments in Probabilistic Separation Logic

Strategy 1: Reducing to Checking Entailment of Simpler Formulas

For example, $P \vDash Q \wedge R$ iff $P \vDash Q$ and $P \vDash R$

Difficulty:

However, how do we reduce $P \vDash Q * R$ to simpler goals?

Moreover, if given

$\text{message} \sim \text{Bern}(0.7) * \text{key} \sim \text{Bern}(0.5) \wedge [\text{cipher} = \text{key xor message}]$,

can we show it entails $\text{cipher} * \text{key}$?

💡 Algebraic Statistics

Polynomial Characterization of Independence:

Given a distribution μ, ~~Boolean~~ variables X and Y are independent in μ iff

$$a_{00} = \mu(X = 0, Y = 0), a_{01} = \mu(X = 0, Y = 1),$$

$$a_{10} = \mu(X = 1, Y = 0), a_{11} = \mu(X = 1, Y = 1)$$

is a root to the system of polynomial

~~$\{a_{00} \cdot a_{11} - a_{01} \cdot a_{10}\}$~~

A more complicated system of polynomials

💡Our Insight:
Polynomial Characterization of Assertions

A distribution μ satisfies $x \sim \text{bern}(0.5)$ iff

$a_0 = \mu(X = 0), a_1 = \mu(X = 1)$ is a root to the system of polynomial $\{a_1 - 0.5\}$

A distribution μ satisfies $P \wedge Q$ iff

… is a root to the system of polynomial $\text{poly}(P) \cup \text{poly}(Q)$

A distribution μ satisfies $P * Q$ iff

… is a root to the system of polynomial
$\text{poly}(P) \cup \text{poly}(Q) \cup$ extra polynomials to guarantee independence

New Strategy: Reducing to Checking Ideal-membership of polynomials

A distribution μ satisfies P iff it corresponds to a root of poly(P).

Thus, $P \vDash Q$ iff any root of poly(P) is also a root of poly(Q).

This reduces the ideal-membership of polynomials and can be automated through Groebner bases!

Contributions

- A syntax-directed PSL for loop-free probabilistic programs.
- A sound and complete entailment checking procedure.
- An algorithm for checking the validity of PSL Hoare triples.
- An OCaml implementation and a collection of benchmark programs for independence reasoning.

Future Directions

- Support loops
- Automate extensions of PSL

  (Conditioning Modality, $\exists, \forall$, ...)
- Use post-condition to prune proof search
- ...

Automating Probabilistic Separation Logic

Joint work with Jessica Cho, Steven Holtzen and Justin Hsu

Jialu Bao Software Day Mar. 10th 2026