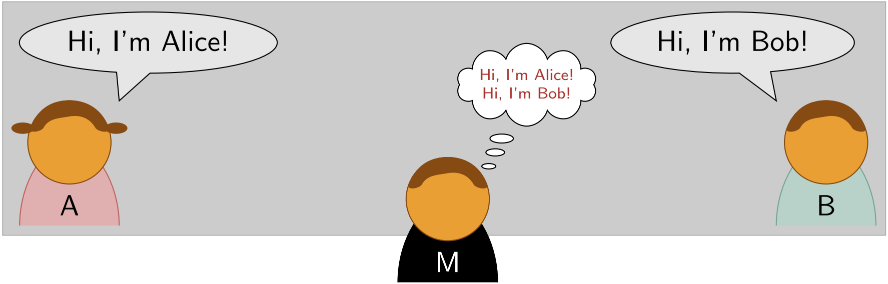
Probabilistic functions and cryptographic oracles in higher-order logic

Andreas Lochbihler

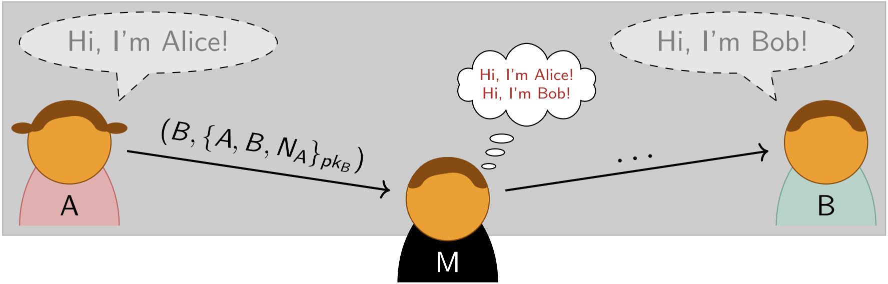
Institute of Information Security
ETH Zurich, Switzerland

Computational soundness

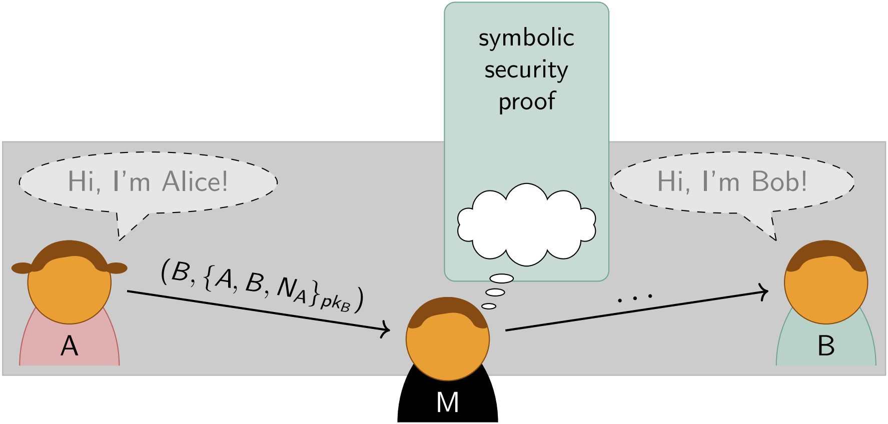
Computational soundness



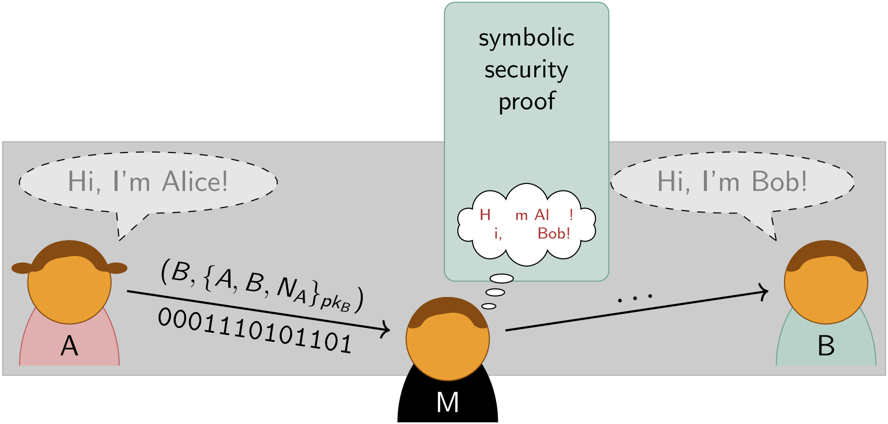
Computational soundness



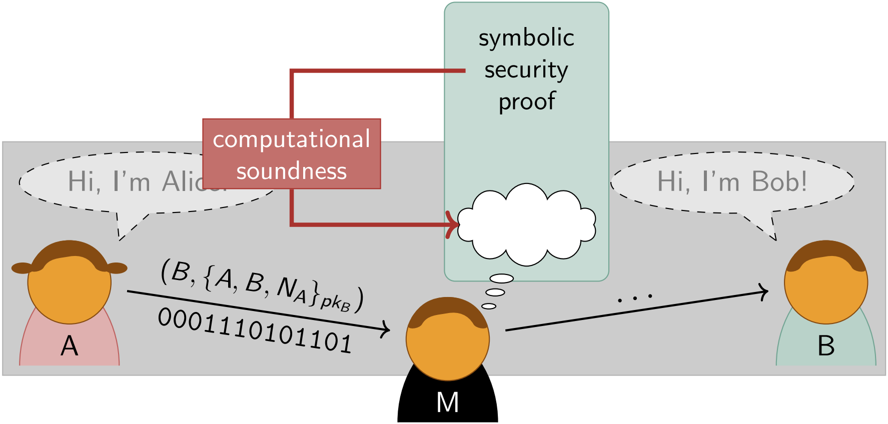
Computational soundness



Computational soundness



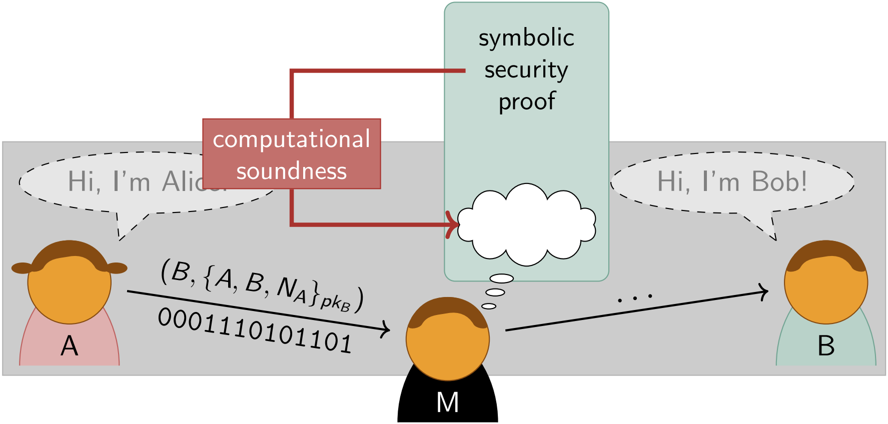
Computational soundness



Computational soundness

Goal: Obtain cryptographic guarantees for symbolic security proofs by formalising a computational soundness proof.

This talk: A framework for formalising computational arguments



Frameworks for formalising computational arguments

	embedding	symbolic messages	proof automation	trusted base
CertiCrypt	deep	⊖	⊖	⊕ Coq
EasyCrypt	axiomatic	⊖	⊕	⊖ EasyCrypt + SMT
Verypto	deep	0	⊖	0 Isabelle + axioms
FCF	semi-shallow	⊕	⊖	⊕ Coq
ours	shallow	⊕	0	⊕ Isabelle

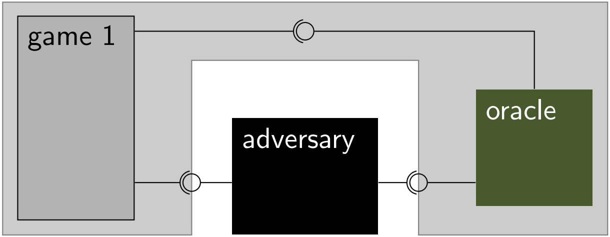
Frameworks for formalising computational arguments

	embedding	symbolic messages	proof automation	trusted base
CertiCrypt	deep	⊖	⊖	⊕ Coq
EasyCrypt	axiomatic	⊖	⊕	⊖ EasyCrypt + SMT
Verypto	deep	0	⊖	0 Isabelle + axioms
FCF	semi-shallow	⊕	⊖	⊕ Coq
ours	shallow	⊕	0	⊕ Isabelle

Contributions

1. Probabilistic language in higher-order logic (HOL)
   - new semantic domain
   - shallow embedding
   - oracle access, monadic sequencing, exceptions, recursion
2. Systematic way to *find* reasoning rules for language primitives
3. Formalised in Isabelle/HOL and applied to small examples

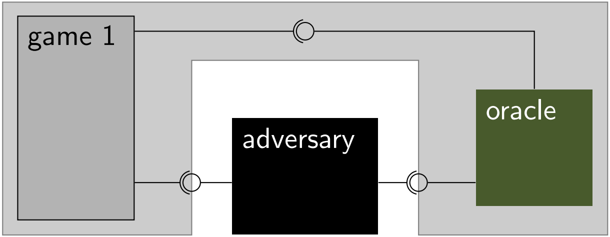
Structure of computational arguments



Security:

Probability of winning game 1 is small for any efficient adversary.

Structure of computational arguments

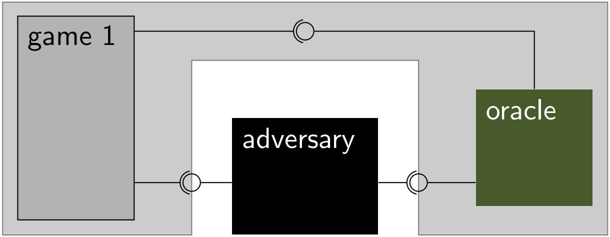


$$
\begin{array}{l}
\text{ind-cpa-rom } \mathcal{A} = \text{try } \ \textbf{do } \{ \\
\quad (pk, sk) \leftarrow \text{key-gen}; \\
\quad b \leftarrow \ \text{uniform } \{0, 1\}; \\
\quad (m_0, m_1, \sigma, s_{\mathcal{O}}) \leftarrow \text{exec}(\mathcal{A}.\text{gen}(pk), \text{rom}, \varnothing); \\
\quad \text{assert } \ (\text{valid-plain}(m_0) \wedge \text{valid-plain}(m_1)); \\
\quad c^* \leftarrow \text{aenc}(pk, \text{if } b \text{ then } m_0 \text{ else } m_1); \\
\quad (b', \_) \leftarrow \text{exec}(\mathcal{A}.\text{guess}(c^*, \sigma), \text{rom}, s_{\mathcal{O}}); \\
\quad \text{return}_{\text{sprob}} \ (b = b') \\
\} \ \text{else } \ \text{uniform } \{0, 1\}
\end{array}
$$

Security:

Probability of winning game 1 is small for any efficient adversary.

Structure of computational arguments



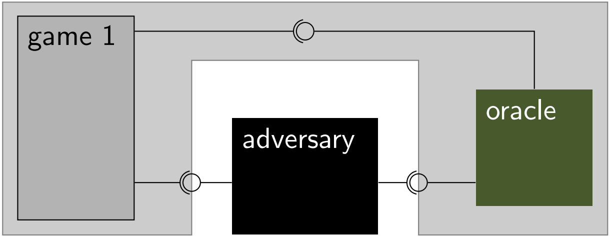
Security:

Probability of winning game 1 is small for any efficient adversary.

$$
\begin{array}{l}
\text{ind-cpa-rom } \mathcal{A} = \text{ try } \textbf{do } \{ \\
\quad (pk, sk) \leftarrow \text{key-gen}; \\
\quad b \leftarrow \text{ uniform } \{0, 1\}; \\
\quad (m_0, m_1, \sigma, s_{\mathcal{O}}) \leftarrow \text{exec}(\mathcal{A}.\text{gen}(pk), \text{rom}, \varnothing); \\
\quad \text{assert } (\text{valid-plain}(m_0) \wedge \text{valid-plain}(m_1)); \\
\quad c^* \leftarrow \text{aenc}(pk, \text{if } b \text{ then } m_0 \text{ else } m_1); \\
\quad (b', \_) \leftarrow \text{exec}(\mathcal{A}.\text{guess}(c^*, \sigma), \text{rom}, s_{\mathcal{O}}); \\
\quad \text{return}_{\text{sprob}} \ (b = b') \\
\} \text{ else uniform } \{0, 1\}
\end{array}
$$

monad

Structure of computational arguments



Security:

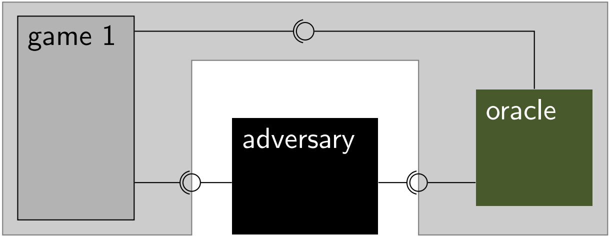
Probability of winning game 1 is small for any efficient adversary.

$$
\begin{array}{l}
\text{ind-cpa-rom } \mathcal{A} = \text{try } \ \mathbf{do} \ \{ \\
\quad (pk, sk) \leftarrow \text{key-gen}; \\
\quad b \leftarrow \text{uniform } \{0, 1\}; \\
\quad (m_0, m_1, \sigma, s_{\mathcal{O}}) \leftarrow \text{exec}(\mathcal{A}.\text{gen}(pk), \text{rom}, \varnothing); \\
\quad \text{assert } \ (\text{valid-plain}(m_0) \wedge \text{valid-plain}(m_1)); \\
\quad c^* \leftarrow \text{aenc}(pk, \text{if } b \text{ then } m_0 \text{ else } m_1); \\
\quad (b', \_) \leftarrow \text{exec}(\mathcal{A}.\text{guess}(c^*, \sigma), \text{rom}, s_{\mathcal{O}}); \\
\quad \text{return}_{\text{sprob}} \ (b = b') \\
\} \ \text{else } \ \text{uniform } \{0, 1\}
\end{array}
$$

monad

sampling

Structure of computational arguments



Security:

Probability of winning game 1 is small for any efficient adversary.

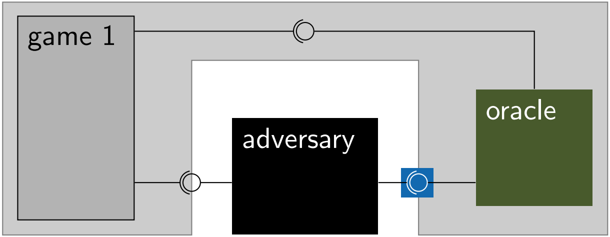
ind-cpa-rom $\mathcal{A}$ = try **do** {
  $(pk, sk) \leftarrow$ key-gen;
  $b \leftarrow$ uniform $\{0, 1\}$;
  $(m_0, m_1, \sigma, s_{\mathcal{O}}) \leftarrow$ exec($\mathcal{A}$.gen(pk), rom, $\varnothing$);
  assert (valid-plain$(m_0) \wedge$ valid-plain(m_1));
  $c^* \leftarrow$ aenc$(pk$, if b then m_0 else $m_1)$;
  $(b', \_) \leftarrow$ exec($\mathcal{A}$.guess(c^*, σ), rom, $s_{\mathcal{O}}$);
  return$_{\text{sprob}}$ $(b = b')$
} else uniform $\{0, 1\}$

monad

sampling

assertions and error handling

Structure of computational arguments

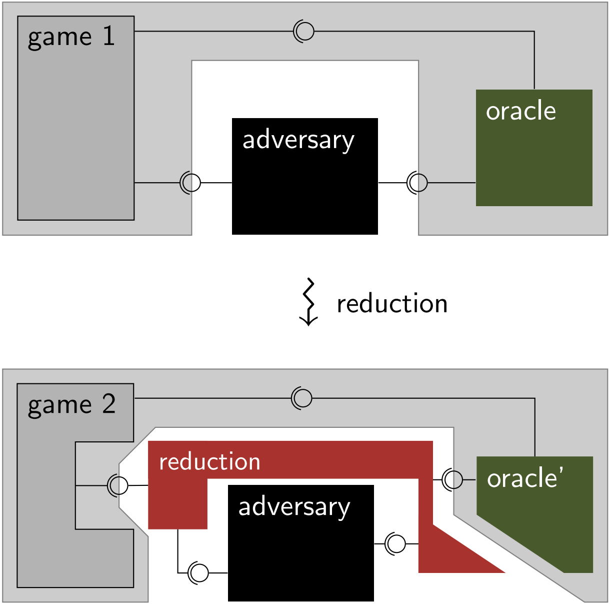


Security:

Probability of winning game 1 is small for any efficient adversary.

$$
\begin{array}{l}
\text{ind-cpa-rom } \mathcal{A} = \text{try } \ \mathbf{do} \ \{ \\
\quad (pk, sk) \leftarrow \text{key-gen}; \\
\quad b \leftarrow \ \text{uniform } \{0, 1\}; \\
\quad (m_0, m_1, \sigma, s_{\mathcal{O}}) \leftarrow \text{exec}(\mathcal{A}.\text{gen}(pk), \text{rom}, \varnothing); \\
\quad \text{assert } \ (\text{valid-plain}(m_0) \wedge \text{valid-plain}(m_1)); \\
\quad c^* \leftarrow \text{aenc}(pk, \text{if } b \text{ then } m_0 \text{ else } m_1); \\
\quad (b', \_) \leftarrow \text{exec}(\mathcal{A}.\text{guess}(c^*, \sigma), \text{rom}, s_{\mathcal{O}}); \\
\quad \text{return}_{\text{sprob}} \ (b = b') \\
\} \ \text{else } \ \text{uniform } \{0, 1\}
\end{array}
$$

Structure of computational arguments



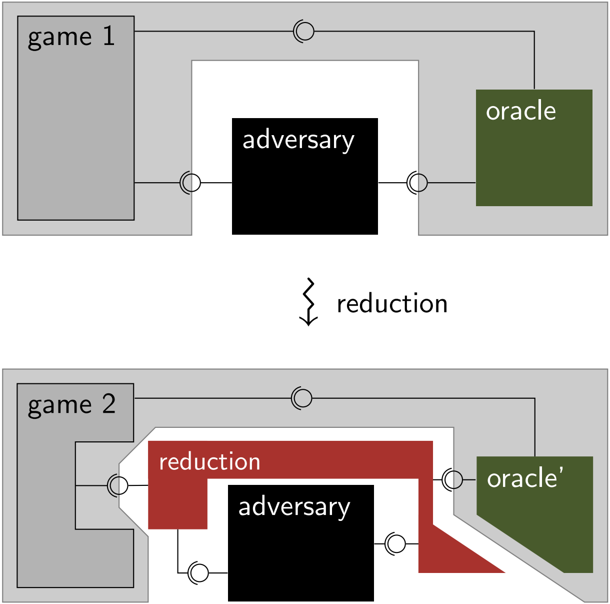
Security:

Probability of winning game 1 is small for any efficient adversary.

Assumption:

Probability of winning game 2 is small for any efficient adversary.

Structure of computational arguments



Security theorem:

Probability of winning game 1 is small for any efficient adversary.

$\Uparrow$

Probability of winning game 2 is small for any efficient adversary.

Discrete subprobabilities as semantic domain

typedef α sprob $= \{ f : \alpha \to \mathbb{R}^+_0 \mid \sum_x f(x) \leq 1 \}$

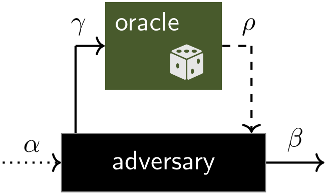
Theorem: α sprob is a chain-complete partial order.

Discrete subprobabilities as semantic domain

typedef α sprob $= \{\, f : \alpha \to \mathbb{R}_0^+ \mid \sum_x f(x) \le 1 \,\}$

Theorem: α sprob is a chain-complete partial order.

First attempt to model oracle access:



$\gamma \to \rho$ sprob

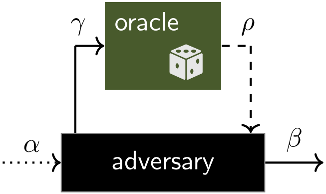
$\alpha \to \beta$ sprob

Discrete subprobabilities as semantic domain

typedef α sprob $= \{\, f : \alpha \to \mathbb{R}_0^+ \mid \sum_x f(x) \le 1 \,\}$

Theorem: α sprob is a chain-complete partial order.

First attempt to model oracle access:



$\gamma \to \rho$ sprob

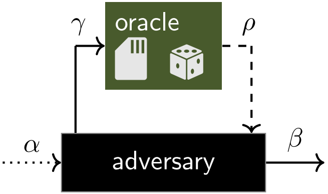
$(\gamma \to \rho$ sprob$) \to \alpha \to \beta$ sprob

Discrete subprobabilities as semantic domain

typedef α sprob $= \{ f : \alpha \to \mathbb{R}_0^+ \mid \sum_x f(x) \leq 1 \}$

Theorem: α sprob is a chain-complete partial order.

First attempt to model oracle access:



$\sigma \to \gamma \to (\rho \times \sigma)$ sprob

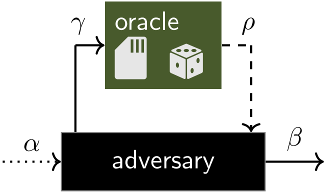
$(\sigma \to \gamma \to (\rho \times \sigma)$ sprob$) \to \boxed{\sigma} \to \alpha \to \beta$ sprob

Discrete subprobabilities as semantic domain

typedef α sprob $= \{\, f : \alpha \to \mathbb{R}_0^+ \mid \sum_x f(x) \le 1 \,\}$

Theorem: α sprob is a chain-complete partial order.

First attempt to model oracle access:

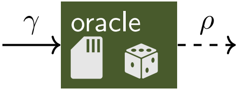


$\sigma \to \gamma \to (\rho \times \sigma)$ sprob

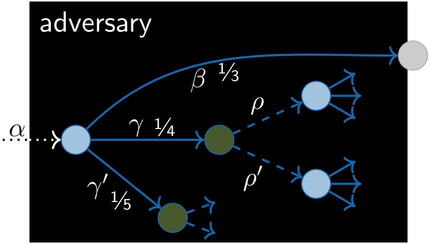
$\cancel{(\sigma \to \gamma \to (\rho \times \sigma)\ \text{sprob}) \to \boxed{\sigma} \to \alpha \to \beta\ \text{sprob}}$

Generative probabilistic values

Second attempt to model oracle access:

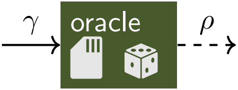


oracle: $\sigma \to \gamma \to (\rho \times \sigma)$ sprob

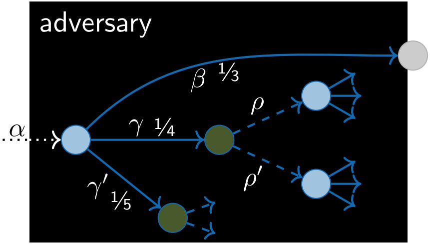


Generative probabilistic values

Second attempt to model oracle access:



oracle: $\sigma \to \gamma \to (\rho \times \sigma)$ sprob



adversary: $\alpha \to$ gpv

gpv $\cong$ ($\beta + \gamma \times$ rpv) sprob

rpv $\cong \rho \to$ gpv

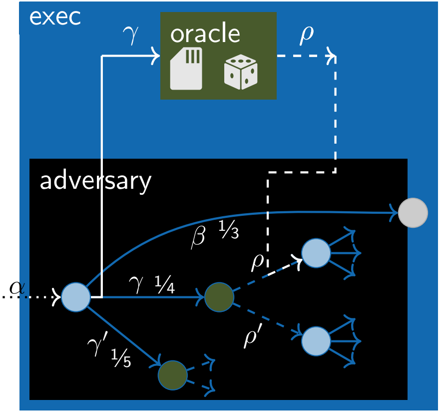
codatatype gpv =

Gpv $((\beta + \gamma \times (\rho \Rightarrow \text{gpv})$ sprob)

Express operators for sequencing, failure, composition , . . .

Generative probabilistic values

Second attempt to model oracle access:



oracle: $\sigma \to \gamma \to (\rho \times \sigma)$ sprob

adversary: $\alpha \to$ gpv

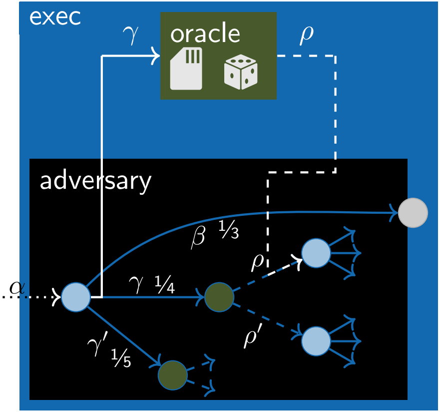
gpv $\cong$ ($\beta + \gamma \times$ rpv) sprob
rpv $\cong \rho \to$ gpv

codatatype gpv =
Gpv (($\beta + \gamma \times (\rho \Rightarrow$ gpv) sprob)

Express operators for sequencing, failure, composition, ...

Generative probabilistic values

Second attempt to model oracle access:



oracle: $\sigma \to \gamma \to (\rho \times \sigma)$ sprob

adversary: $\alpha \to$ gpv

gpv $\cong$ (β $+ \gamma \times$ rpv) sprob
rpv $\cong \rho \to$ gpv

codatatype gpv $=$
Gpv $((\beta + \gamma \times (\rho \Rightarrow \text{gpv})$ sprob$)$

Express operators for sequencing, failure, composition, . . .

gpv also used for reductions and games

Rules for reasoning about game transformations

Equational reasoning

- Shallow embedding supports many equalities
- Example: commutativity for spmf

$$\begin{array}{l} \textbf{do } \{ \\ \quad x \leftarrow ppp; \\ \quad y \leftarrow q; \\ \quad f(x, y) \ \} \end{array} \quad = \quad \begin{array}{l} \textbf{do } \{ \\ \quad y \leftarrow q; \\ \quad x \leftarrow ppp; \\ \quad f(x, y) \ \} \end{array}$$

Rules for reasoning about game transformations

Equational reasoning

- Shallow embedding supports many equalities
- Example: commutativity for spmf

$$\begin{array}{l} \textbf{do}\ \{ \\ \quad x \leftarrow ppp; \\ \quad y \leftarrow q; \\ \quad f(x,y)\ \} \end{array} \quad = \quad \begin{array}{l} \textbf{do}\ \{ \\ \quad y \leftarrow q; \\ \quad x \leftarrow ppp; \\ \quad f(x,y)\ \} \end{array}$$

Relational reasoning

- Lift relation A on outcomes to relation $\uparrow A \uparrow$ on sprobs or gpvs
- Example: sequencing

$$\frac{p \uparrow A \uparrow q \qquad \forall (x,y) \in A.\ f(x)\ \uparrow B \uparrow\ g(y)}{(\textbf{do}\ \{x \leftarrow p; f(x)\ \})\ \uparrow B \uparrow\ (\textbf{do}\ \{y \leftarrow q; g(y)\ \})}$$

How to find relational rules

Shape of rule determined by operator type

$$\mathsf{bind}_{\mathsf{sprob}} : \begin{array}{ccccccccc} \alpha\ \mathsf{sprob} & \Rightarrow & (& \alpha & \Rightarrow & \beta\ \mathsf{sprob} &) & \Rightarrow & \beta\ \mathsf{sprob} \\ \uparrow A \uparrow_{\mathsf{sprob}} & \Mapsto & (& A & \Mapsto & \uparrow B \uparrow_{\mathsf{sprob}} &) & \Mapsto & \uparrow B \uparrow_{\mathsf{sprob}} \end{array}$$

Replace types by relations

How to find relational rules

Shape of rule determined by operator type

$$\mathrm{bind}_{\mathrm{sprob}} : \alpha\ \mathrm{sprob} \Rightarrow (\alpha \Rightarrow \beta\ \mathrm{sprob}) \Rightarrow \beta\ \mathrm{sprob}$$

$$\forall A\ B.\ \mathrm{bind}_{\mathrm{sprob}}\ (\uparrow A \uparrow_{\mathrm{sprob}} \Mapsto (A \Mapsto \uparrow B \uparrow_{\mathrm{sprob}}) \Mapsto \uparrow B \uparrow_{\mathrm{sprob}})\ \mathrm{bind}_{\mathrm{sprob}}$$

Replace types by relations (Reynolds: relational parametricity)

How to find relational rules

Shape of rule determined by operator type

$$\mathsf{bind}_{\mathsf{sprob}} : \ \alpha\ \mathsf{sprob} \ \Rightarrow \ (\ \alpha \ \Rightarrow \ \beta\ \mathsf{sprob}\ ) \ \Rightarrow \ \beta\ \mathsf{sprob}$$

$$\forall A\ B.\ \mathsf{bind}_{\mathsf{sprob}}\ (\,{\uparrow}A{\uparrow}_{\mathsf{sprob}} \Rrightarrow (\,A \Rrightarrow {\uparrow}B{\uparrow}_{\mathsf{sprob}}\,) \Rrightarrow {\uparrow}B{\uparrow}_{\mathsf{sprob}}\,)\ \mathsf{bind}_{\mathsf{sprob}}$$

Replace types by relations (Reynolds: relational parametricity)

$$\frac{p \uparrow A \uparrow q \qquad \forall (x, y) \in A.\ f(x) \uparrow B \uparrow g(y)}{(\mathbf{do}\ \{x \leftarrow p; f(x)\ \})\uparrow B \uparrow (\mathbf{do}\ \{y \leftarrow q; g(y)\ \})}$$

How to find relational rules

Shape of rule determined by operator type

$$\mathsf{bind}_{\mathsf{sprob}} :\ \alpha\ \mathsf{sprob} \ \Rightarrow\ (\ \alpha\ \Rightarrow\ \beta\ \mathsf{sprob}\ )\ \Rightarrow\ \beta\ \mathsf{sprob}$$

$$\forall A\ B.\ \mathsf{bind}_{\mathsf{sprob}}\ (\ \uparrow A\uparrow_{\mathsf{sprob}}\ \Rrightarrow\ (\ A\ \Rrightarrow\ \uparrow B\uparrow_{\mathsf{sprob}}\ )\ \Rrightarrow\ \uparrow B\uparrow_{\mathsf{sprob}}\ )\ \mathsf{bind}_{\mathsf{sprob}}$$

Replace types by relations (Reynolds: relational parametricity)

$$\frac{p \uparrow A\uparrow q \qquad \forall (x,y)\in A.\ f(x) \uparrow B\uparrow g(y)}{(\mathbf{do}\ \{x \leftarrow p; f(x)\ \})\uparrow B\uparrow (\mathbf{do}\ \{y \leftarrow q; g(y)\ \})}$$

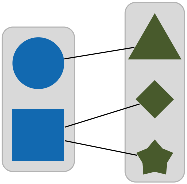
Used this approach to find rules for new operators, e.g.

$$\frac{\mathcal{A}_1 \uparrow A\uparrow_{\mathsf{gpv}} \mathcal{A}_2 \qquad \mathcal{O}_1\ (S \Rrightarrow (=) \Rrightarrow \uparrow(=)\times S\uparrow_{\mathsf{sprob}})\ \mathcal{O}_2 \qquad \sigma_1\ S\ \sigma_2}{\mathsf{exec}(\mathcal{A}_1,\mathcal{O}_1,\sigma_1) \uparrow A\times S\uparrow_{\mathsf{sprob}} \mathsf{exec}(\mathcal{A}_2,\mathcal{O}_2,\sigma_2)}$$

Sampling is almost parametric

$$\mathcal{P}_{\mathsf{uniform}\ \Omega}(X) = \frac{|X|}{|\Omega|}$$

$$\mathsf{uniform} : \begin{array}{ccc} \alpha\ \mathsf{set} & \Rightarrow & \alpha\ \mathsf{sprob} \\ \uparrow A \uparrow_{\mathsf{set}} & \Mapsto & \uparrow A \uparrow_{\mathsf{sprob}} \end{array}$$

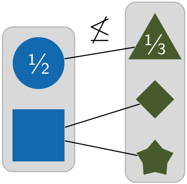


Sampling is almost parametric

$$\mathcal{P}_{\mathsf{uniform}\ \Omega}(X) = \frac{|X|}{|\Omega|}$$

$$\mathsf{uniform} : \alpha\ \mathsf{set} \Rightarrow \alpha\ \mathsf{sprob}$$

$$\neg\,\forall A.\ \mathsf{uniform}\ (\uparrow A \uparrow_{\mathsf{set}} \Mapsto \uparrow A \uparrow_{\mathsf{sprob}})\ \mathsf{uniform}$$



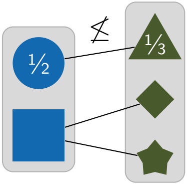
Wadler, Reynolds: Polymorphic equality is not parametric!

Sampling is almost parametric

$$\mathcal{P}_{\mathsf{uniform}\ \Omega}\,(X) = \frac{|X|}{|\Omega|}$$

$$\mathsf{uniform} : \alpha\ \mathsf{set} \Rightarrow \alpha\ \mathsf{sprob}$$

$$\neg\,\forall A.\ \mathsf{uniform}\ (\uparrow\! A\!\uparrow_{\mathsf{set}} \Mapsto\ \uparrow\! A\!\uparrow_{\mathsf{sprob}})\ \mathsf{uniform}$$



Wadler, Reynolds: Polymorphic equality is not parametric!

A must respect equality!

$$\frac{(=)\ (A \Mapsto A \Mapsto \mathsf{rel}_{\mathsf{bool}})\ (=)}{\mathsf{uniform}\ (\uparrow\! A\!\uparrow_{\mathsf{set}} \Mapsto\ \uparrow\! A\!\uparrow_{\mathsf{sprob}})\ \mathsf{uniform}}$$

$$\frac{A \text{ is bijection between } \Omega_1 \text{ and } \Omega_2}{\mathsf{uniform}\ \Omega_1\ \ \uparrow\! A\!\uparrow_{\mathsf{sprob}}\ \ \mathsf{uniform}\ \Omega_2}$$

Special case: one-time pad

$$\mathsf{map}_{\mathsf{sprob}}\ (\mathsf{xor}\ m)\ (\mathsf{uniform}\ \{\,0,1\,\}^n) = \mathsf{uniform}\ \{\,0,1\,\}^n$$

Application examples

Used framework to verify 3 cryptographic constructions.

Line counts of proof of concrete security theorem

Cryptographic construction	Framework ours	CertiCrypt	EasyCrypt	FCF
Elgamal	52	238	58	156
Hashed Elgamal (ROM)	236	810	210	
PRF-IND-CPA	352			1166

Summary

- Probabilistic language with oracles in HOL

$$\mathsf{gpv} \cong (\beta + \gamma \times \mathsf{rpv})\ \mathsf{sprob}$$
$$\mathsf{rpv} \cong \rho \to \mathsf{gpv}$$

- Relational parametricity yields reasoning principles

$$\mathsf{uniform} : \alpha\ \mathsf{set} \Rightarrow \alpha\ \mathsf{sprob}$$
$$\mathsf{uniform}\ (\uparrow A \uparrow_{\mathsf{set}} \Mapsto \uparrow A \uparrow_{\mathsf{sprob}})\ \mathsf{uniform} \quad \textit{if} \quad (=)\ (A \Mapsto A \Mapsto \mathsf{rel}_{\mathsf{bool}})\ (=)$$

- Shallow embedding yields short proofs

Summary

- Probabilistic language with oracles in HOL

$$\text{gpv} \cong (\beta + \gamma \times \text{rpv})\ \text{sprob}$$
$$\text{rpv} \cong \rho \to \text{gpv}$$

- Relational parametricity yields reasoning principles

$$\text{uniform} : \alpha\ \text{set} \Rightarrow \alpha\ \text{sprob}$$
$$\text{uniform}\ (\uparrow A \uparrow_{\text{set}} \Mapsto \uparrow A \uparrow_{\text{sprob}})\ \text{uniform} \quad \textit{if} \quad (=)\ (A \Mapsto A \Mapsto \text{rel}_{\text{bool}})\ (=)$$

- Shallow embedding yields short proofs

Next steps

- Formalise a computational soundness proof in the framework
- Explore the connection between parametricity and other relational program logics