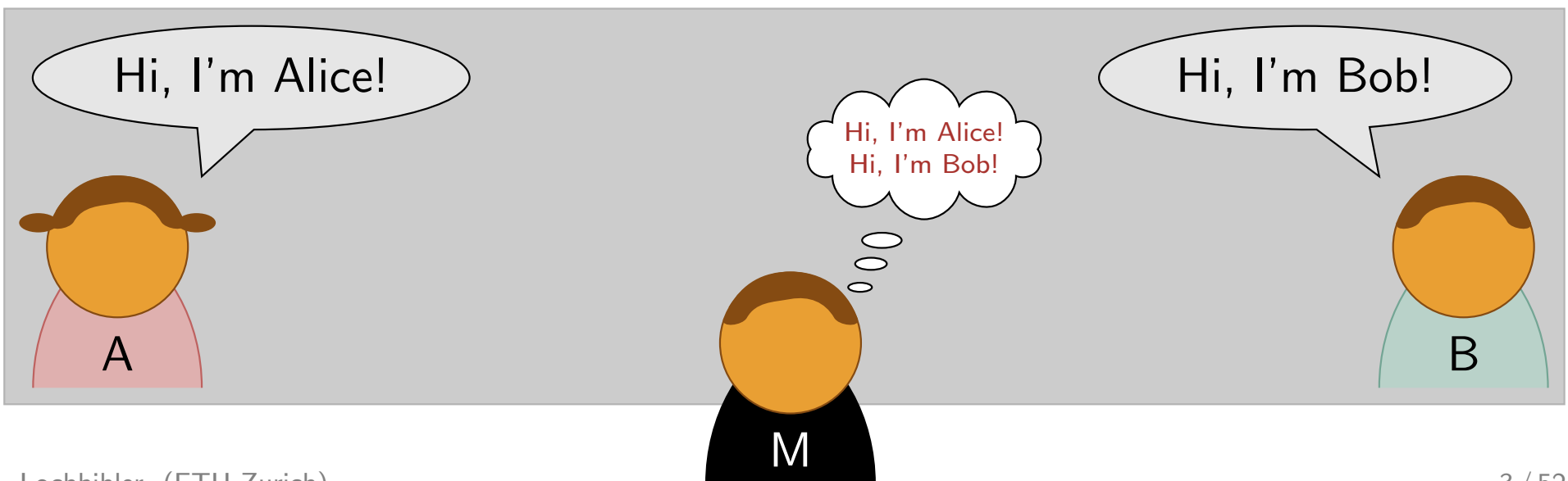
Probabilistic functions and cryptographic oracles in higher-order logic

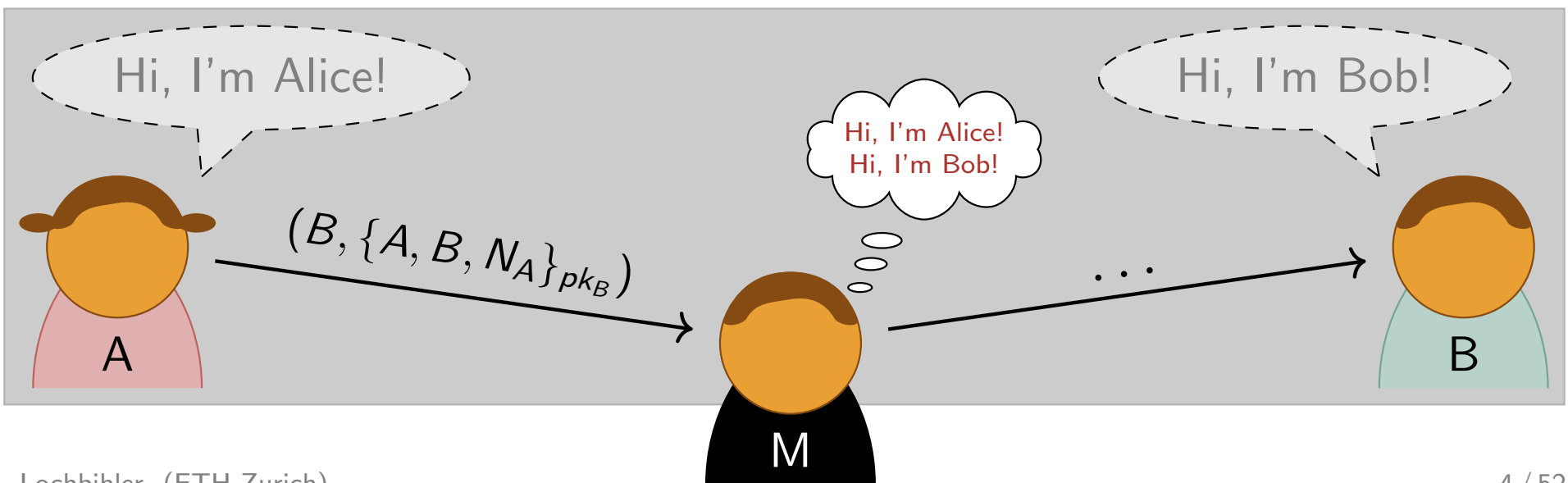
Andreas Lochbihler

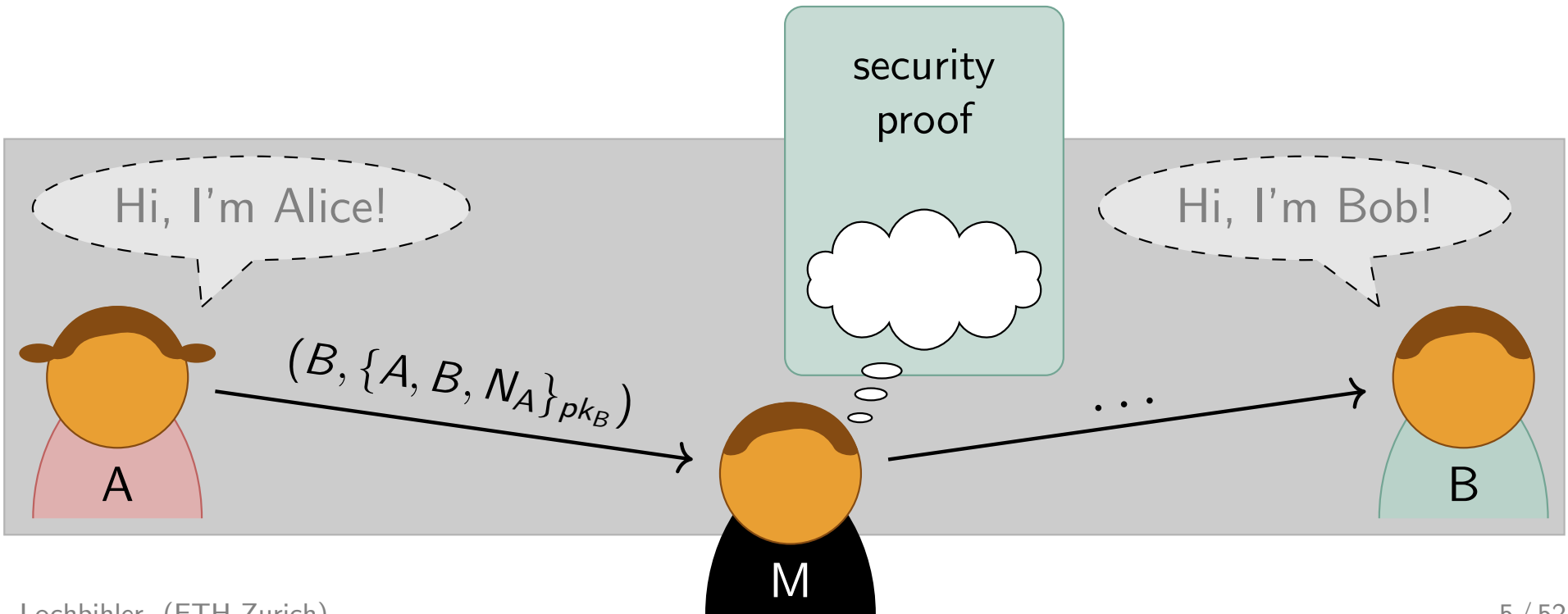
Institute of Information Security

ETH zürich


Hi, I'm Alice!
Hi, I'm Bob!
A
B

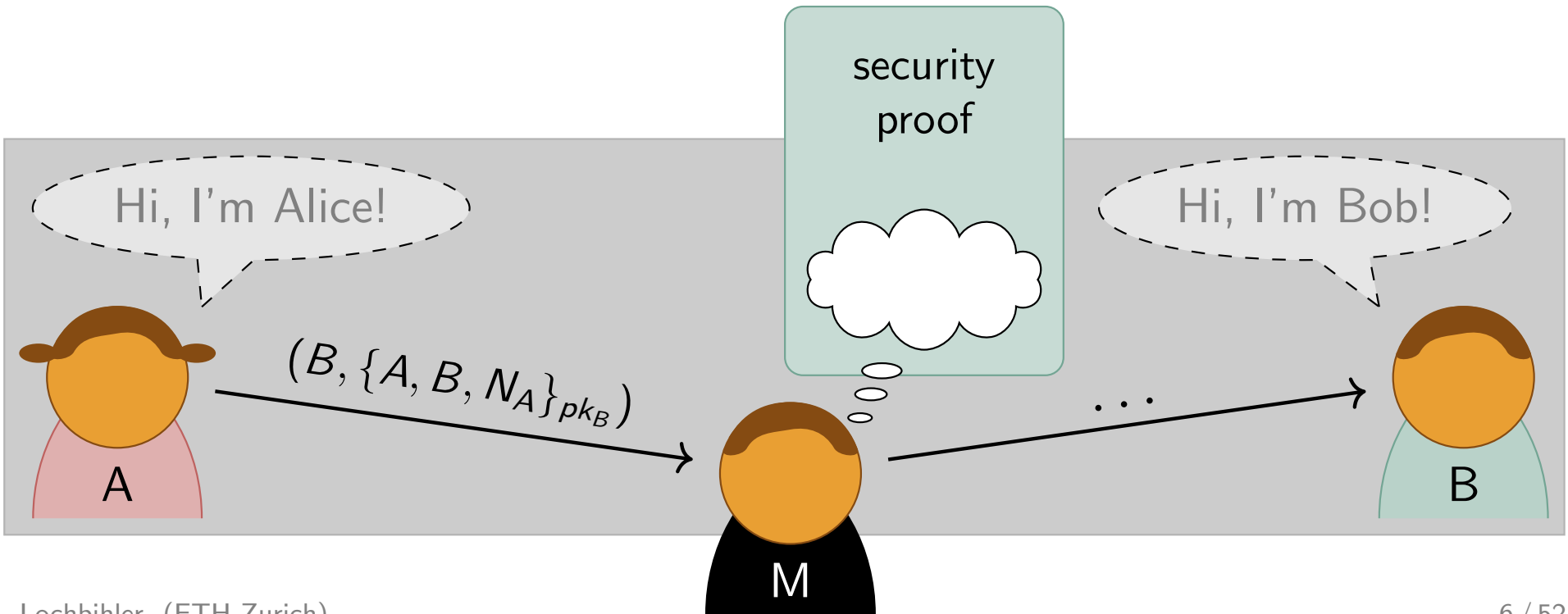

Hi, I'm Alice!
Hi, I'm Alice!
Hi, I'm Bob!
Hi, I'm Bob!
A
M
B


Hi, I'm Alice!
Hi, I'm Bob!
Hi, I'm Alice!
Hi, I'm Bob!
$(B, \{A, B, N_A\}_{pk_B})$
...
A
M
B


security
proof
Hi, I'm Alice!
Hi, I'm Bob!
$(B, \{A, B, N_A\}_{pk_B})$
...
A
M
B

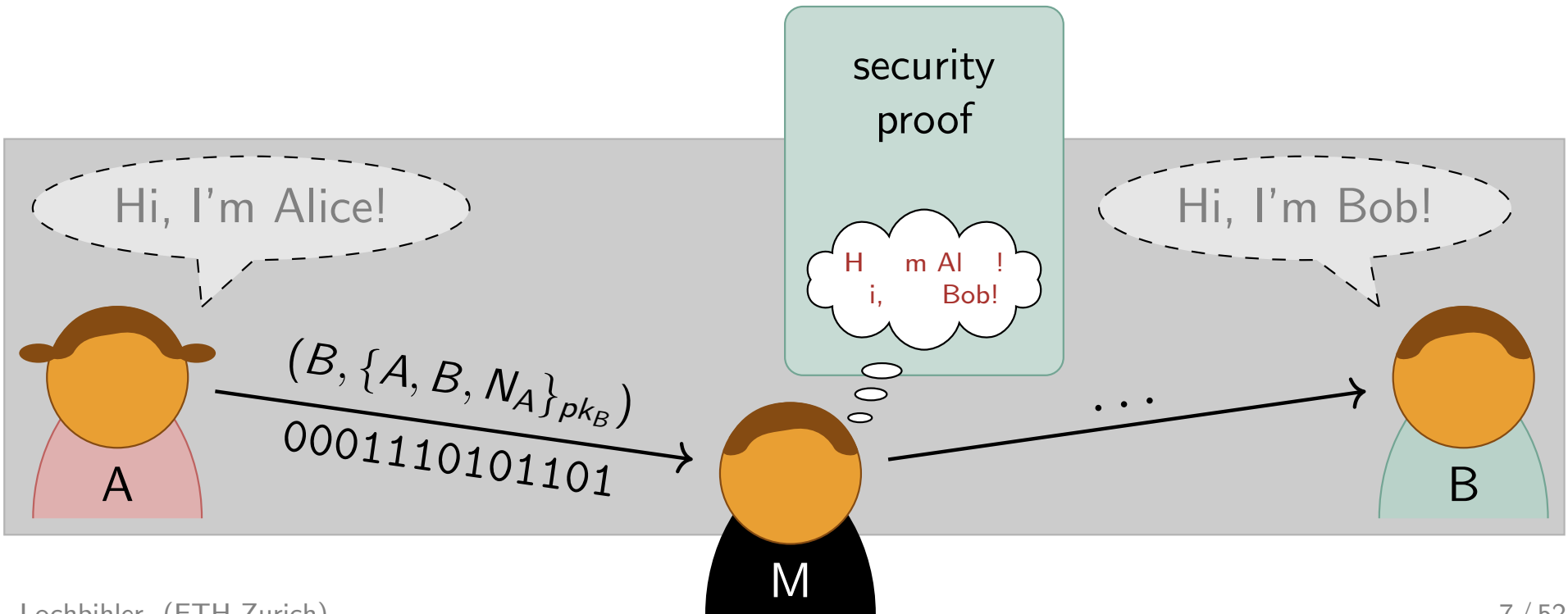
adversary model	Dolev-Yao
cryptography	perfect
reasoning power	unbounded
messages	$\{\langle B, N_A \rangle\}_k$
properties	reachability

ProVerif, OFMC, Scyther, Tamarin



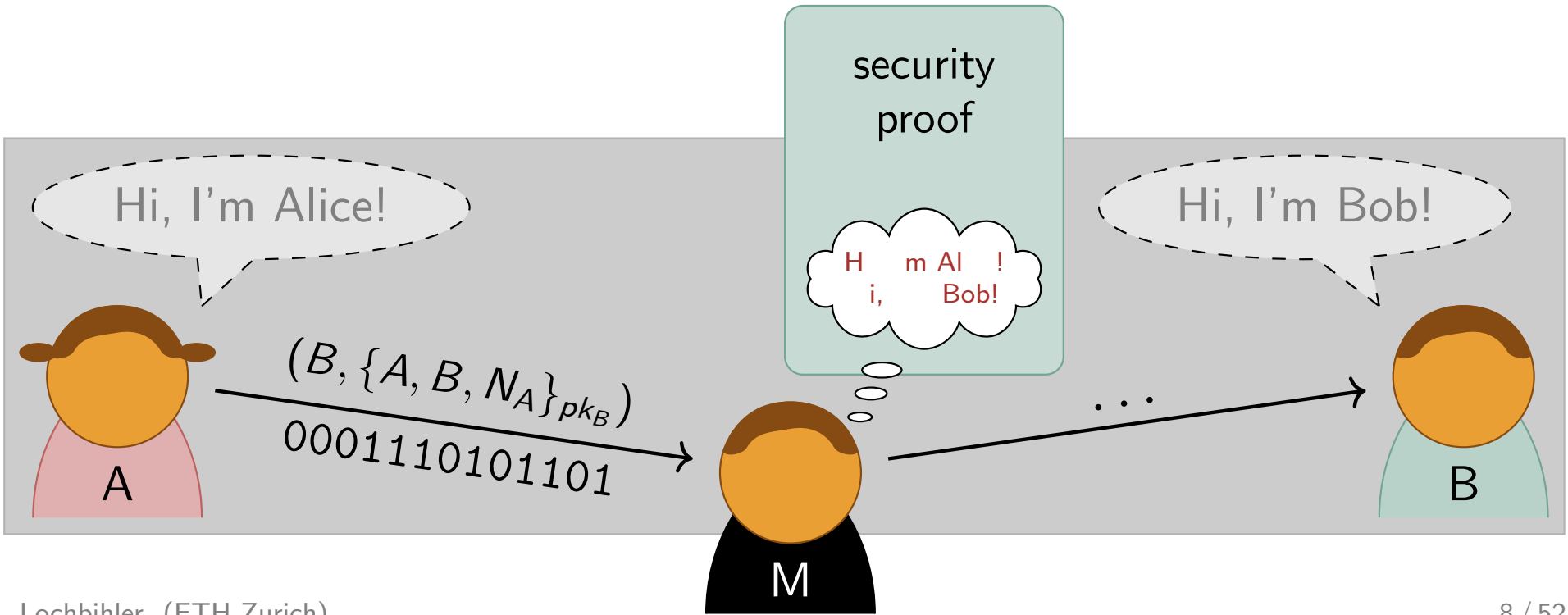
adversary model	Dolev-Yao
cryptography	perfect
reasoning power	unbounded
messages	$\{\langle B, N_A \rangle\}_k$
properties	reachability

ProVerif, OFMC, Scyther, Tamarin

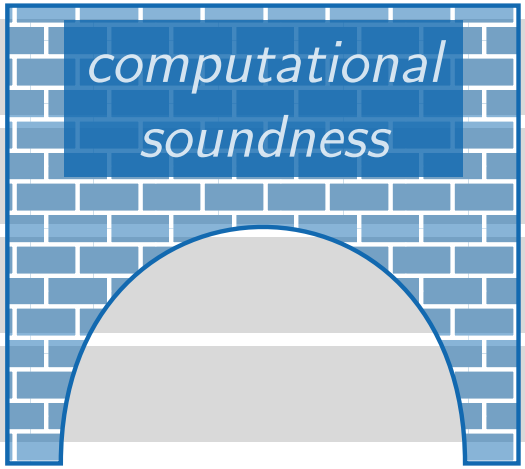


adversary model	Dolev-Yao	computational
cryptography	perfect	
reasoning power	unbounded	efficient
messages	$\{\langle B, N_A \rangle\}_k$	11010111
properties	reachability	games

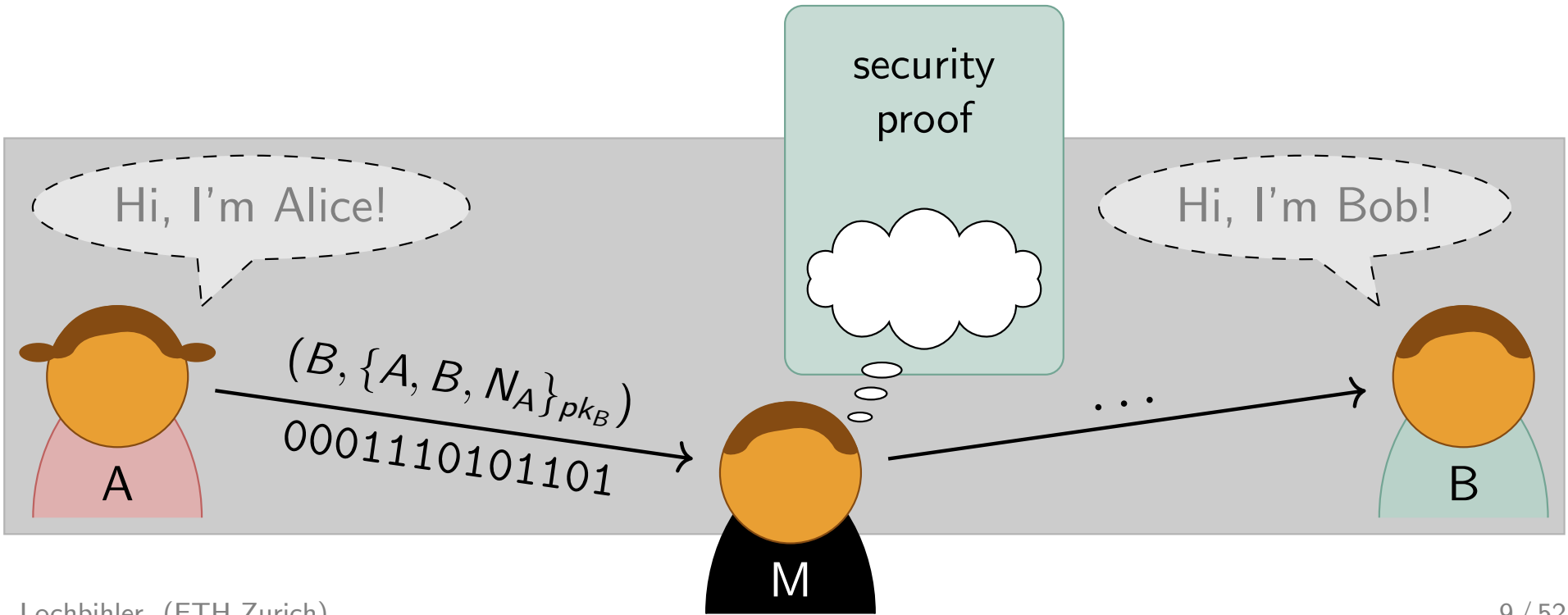
ProVerif, OFMC, Scyther, Tamarin



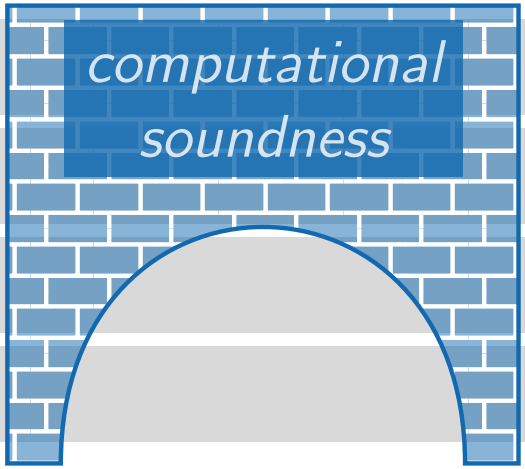
adversary model	Dolev-Yao		computational
cryptography	perfect		
reasoning power	unbounded		efficient
messages	$\{\langle B, N_A \rangle\}_k$		11010111
properties	reachability		games



ProVerif, OFMC,
Scyther, Tamarin



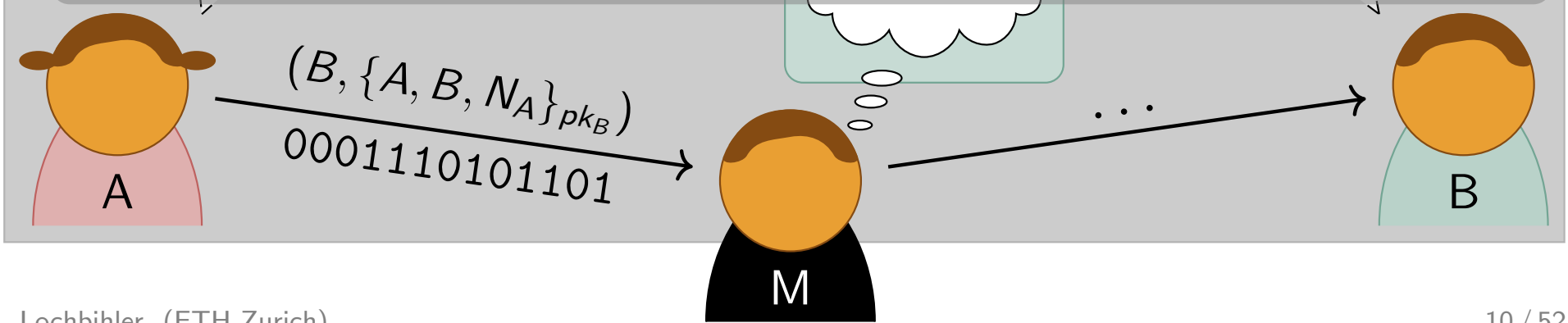
adversary model	Dolev-Yao		computational
cryptography	perfect		
reasoning power	unbounded		efficient
messages	$\{\langle B, N_A \rangle\}_k$		11010111
properties	reachability		games



ProVerif, OFMC,
Scyther, Tamarin

Goal: Obtain computational guarantees for Dolev-Yao 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

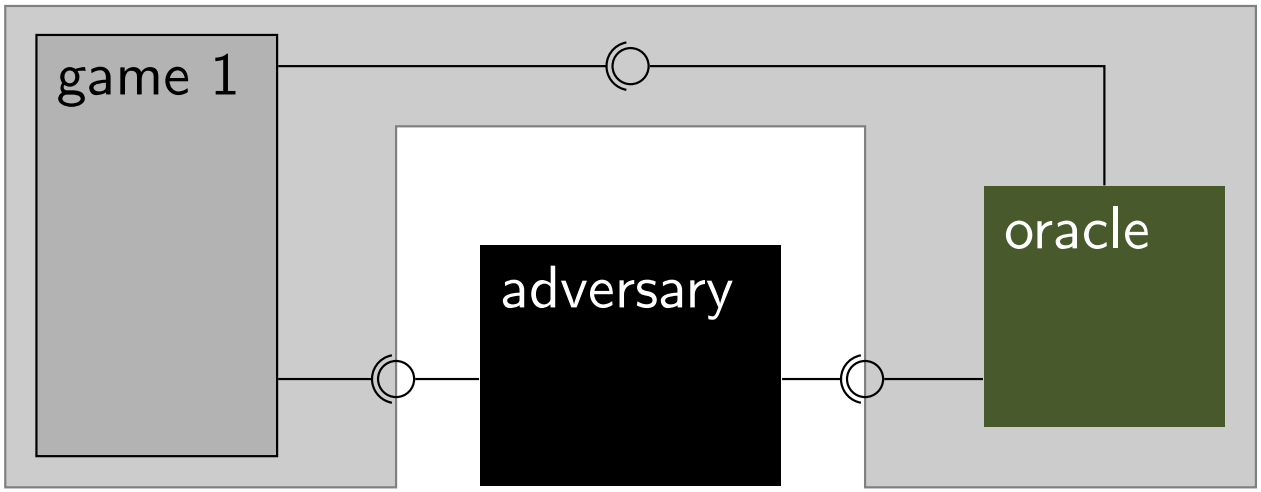
Frameworks for formalising computational arguments

	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

Overview

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. Textbook examples formalised in Isabelle/HOL

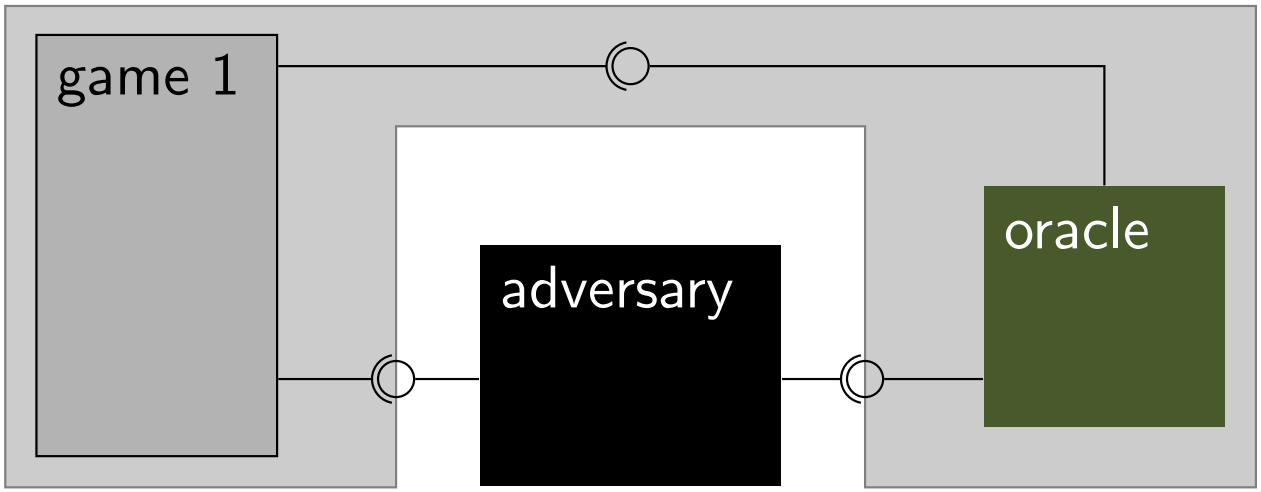
Structure of computational arguments



Security:

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

Structure of computational arguments

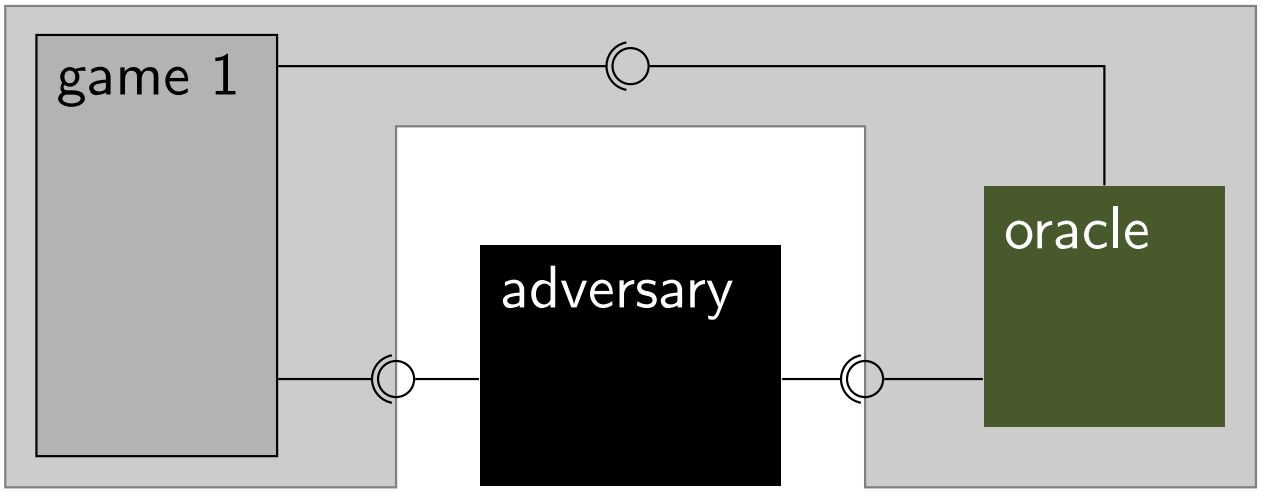


Security:

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

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), RO, $\varnothing$);
  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), RO, $s_{\mathcal{O}}$);
  return$_{\text{sprob}}$ $(b = b')$
} else uniform $\{0, 1\}$

Structure of computational arguments



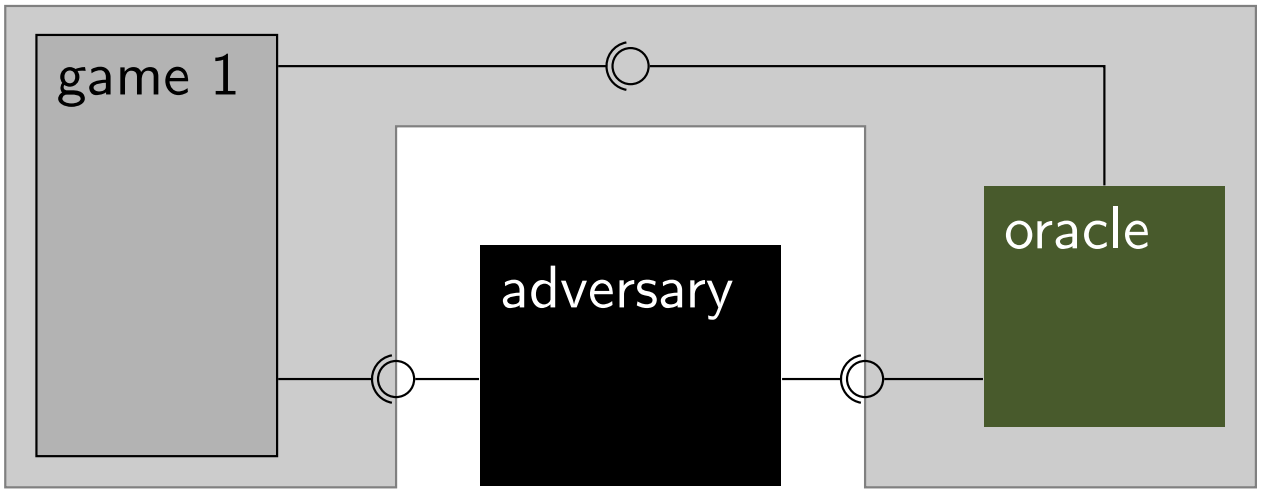
Security:

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

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

monad

Structure of computational arguments



Security:

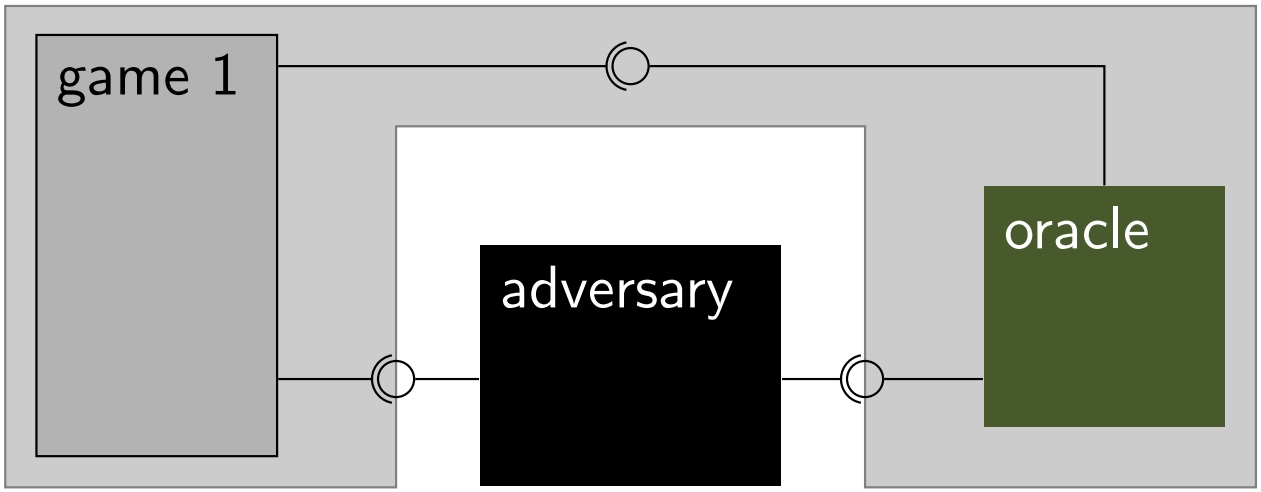
Probability of winning game 1 is small for any efficient adversary.

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), RO, $\varnothing$);
  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), RO, $s_{\mathcal{O}}$);
  return$_{\text{sprob}}$ $(b = b')$
} else uniform $\{0, 1\}$

monad

sampling

Structure of computational arguments



Security:

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

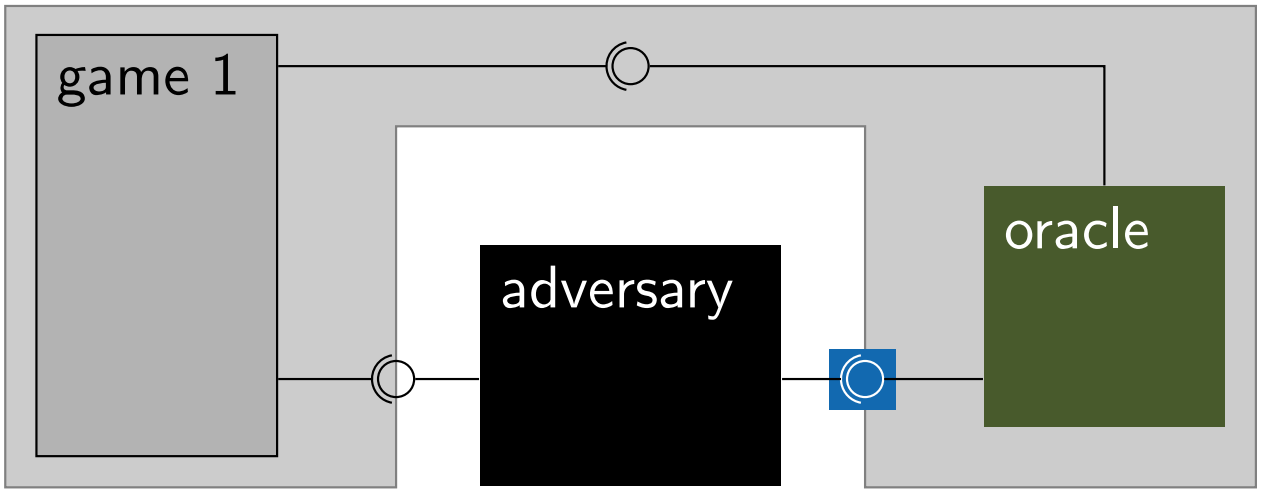
ind-cpa-rom $\mathcal{A}$ = try **do** {
  $(pk, sk) \leftarrow$ key-gen;
  $b \leftarrow$ uniform $\{0, 1\}$;
  $(m_0, m_1, \sigma, s_{\mathcal{O}}) \leftarrow$ exec($\mathcal{A}$.gen(pk), RO, $\varnothing$);
  assert (valid-plain(m_0) $\wedge$ valid-plain(m_1));
  $c^* \leftarrow$ aenc(pk, **if** b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

monad

sampling

assertions and error handling

Structure of computational arguments

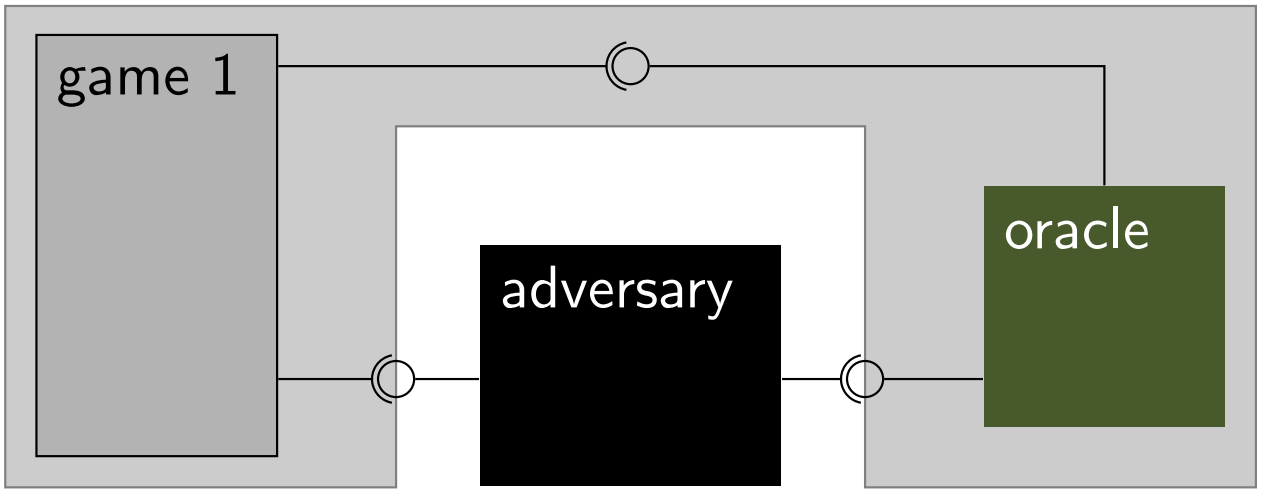


$$
\begin{array}{l}
\text{ind-cpa-rom } \mathcal{A} = \text{ try } \textbf{ do } \{ \\
\quad (pk, sk) \leftarrow \text{key-gen}; \\
\quad b \leftarrow \text{ uniform } \{0,1\}; \\
\quad (m_0, m_1, \sigma, s_{\mathcal{O}}) \leftarrow \text{exec}(\mathcal{A}.\text{gen}(pk), \text{RO}, \varnothing); \\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

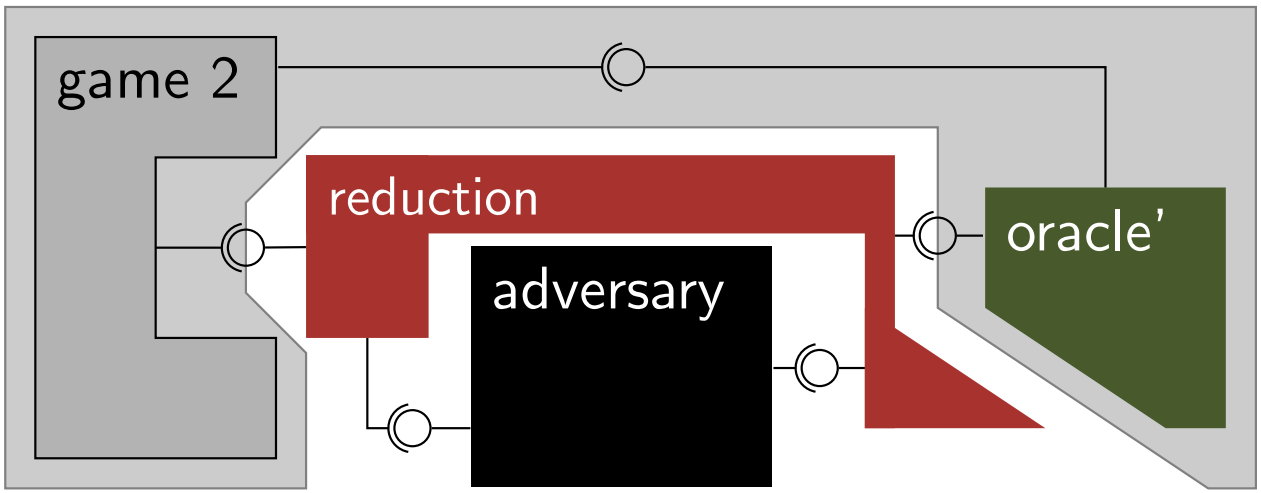
Security:

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

Structure of computational arguments



reduction



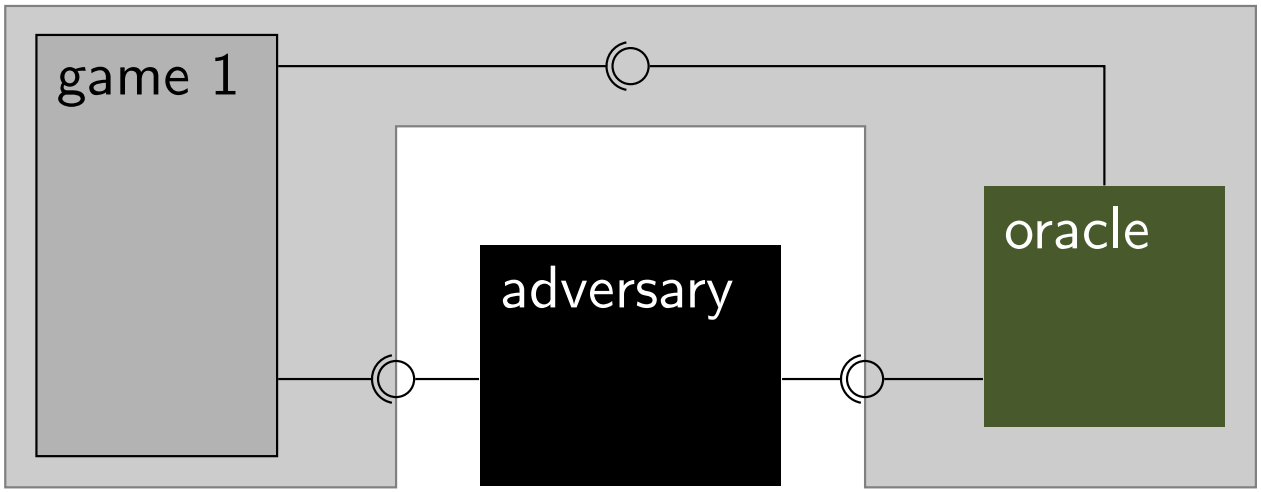
Security:

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

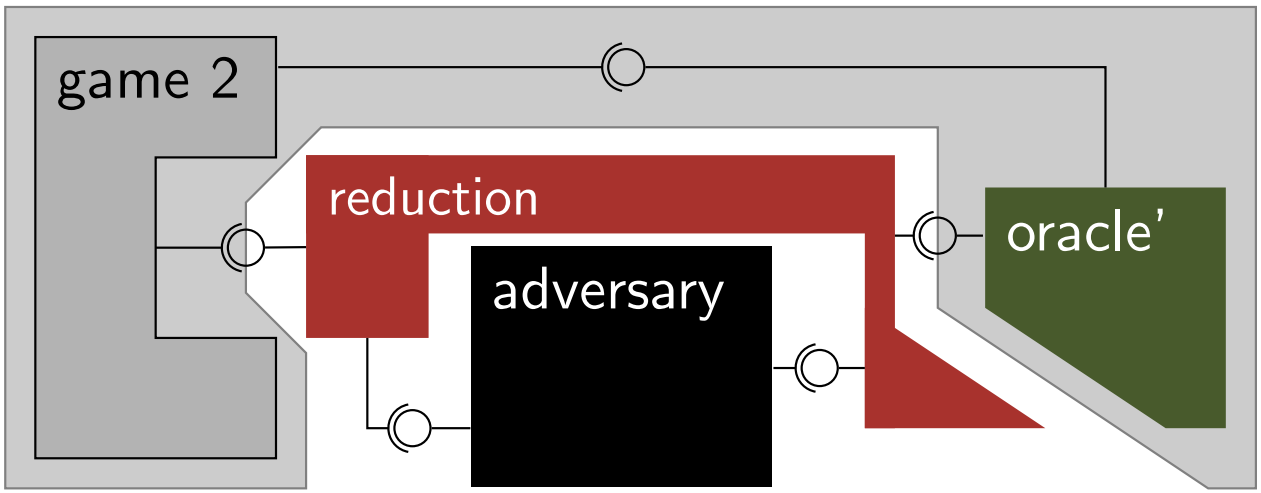
Assumption:

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

Structure of computational arguments



reduction



Security theorem:

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

⇑

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

Discrete subprobabilities as semantic domain

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

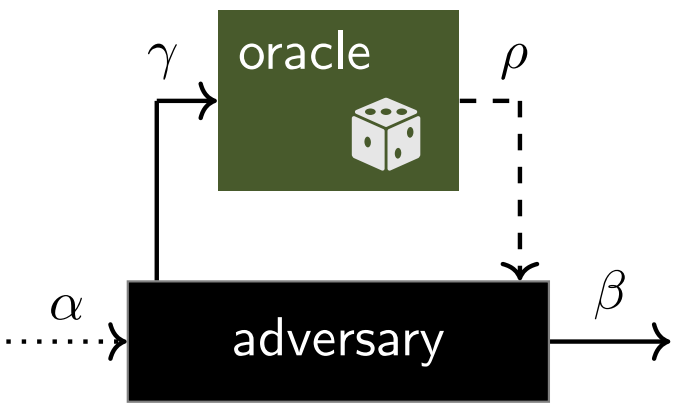
Theorem: α sprob is a chain-complete partial order.

Discrete subprobabilities as semantic domain

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

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

First attempt to model oracle access:



$\gamma \to \rho$ sprob

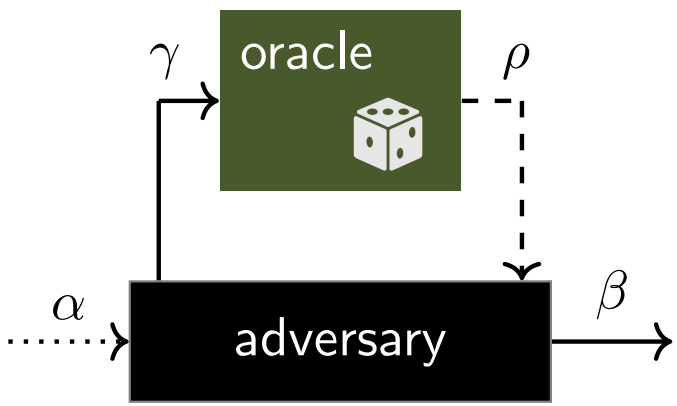
$\alpha \to \beta$ sprob

Discrete subprobabilities as semantic domain

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

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

First attempt to model oracle access:



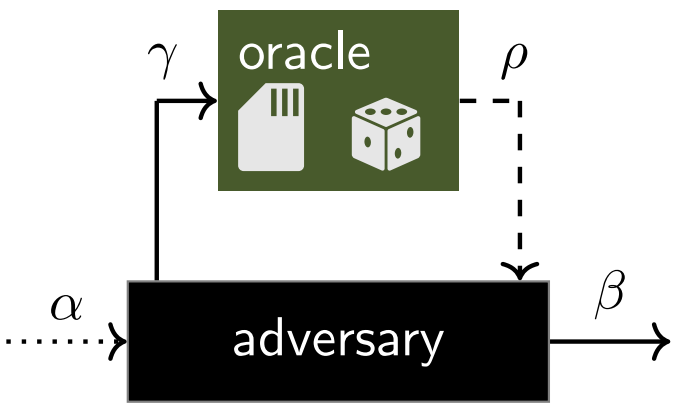
$$(\gamma \to \rho \text{ sprob}) \to \alpha \to \beta \text{ sprob}$$

Discrete subprobabilities as semantic domain

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

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

First attempt to model oracle access:



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

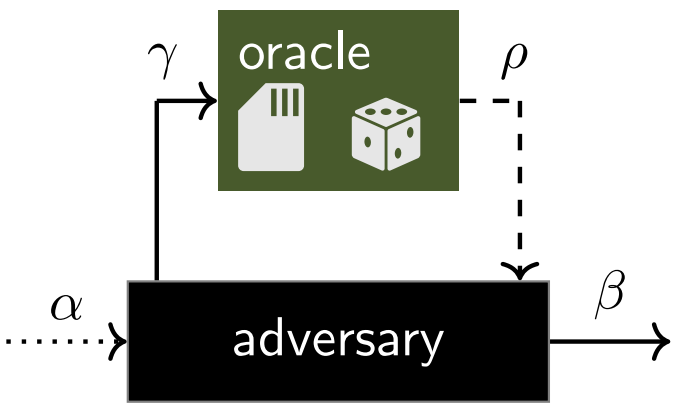
$(\sigma \to \gamma \to (\rho \times \sigma)$ sprob$) \to \sigma \to \alpha \to \beta$ sprob

Discrete subprobabilities as semantic domain

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

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

First attempt to model oracle access:

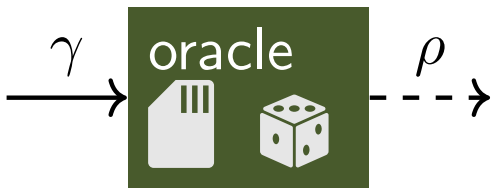


$\sigma \to \gamma \to (\rho \times \sigma)$ sprob ✓

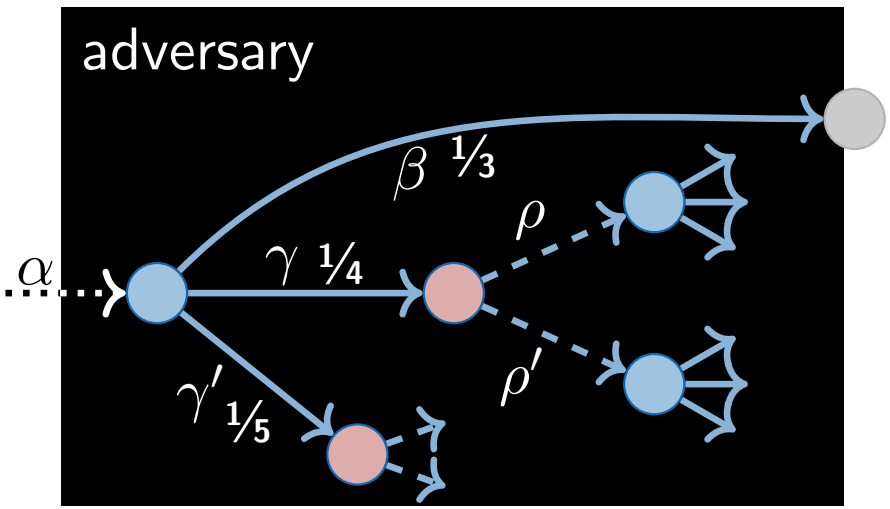
$(\sigma \to \gamma \to (\rho \times \sigma)$ sprob$) \to \sigma \to \alpha \to \beta$ sprob

Generative probabilistic values

Second attempt to model oracle access:

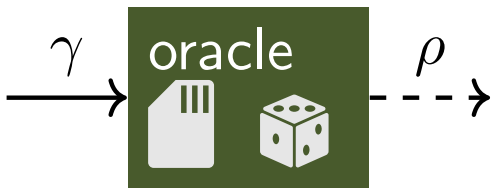


oracle: $\sigma \to \gamma \to (\rho \times \sigma)$ sprob

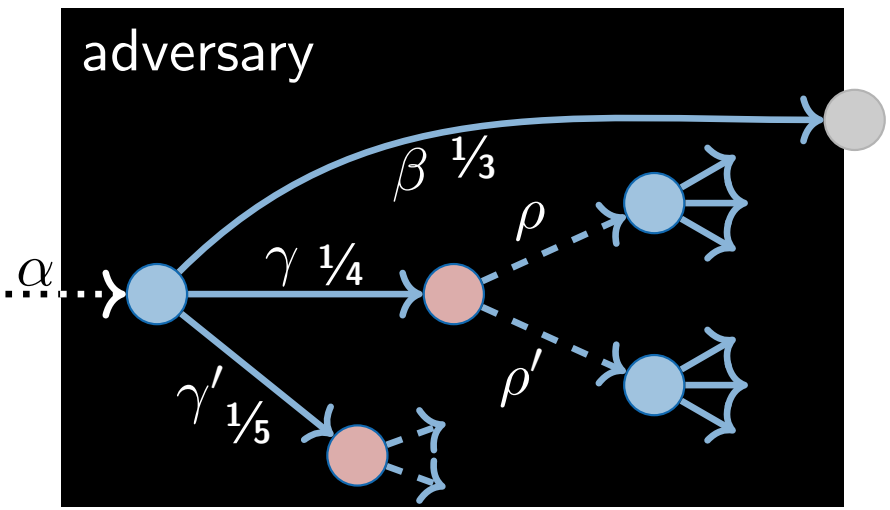


Generative probabilistic values

Second attempt to model oracle access:



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



adversary: $\alpha \to$ gpv

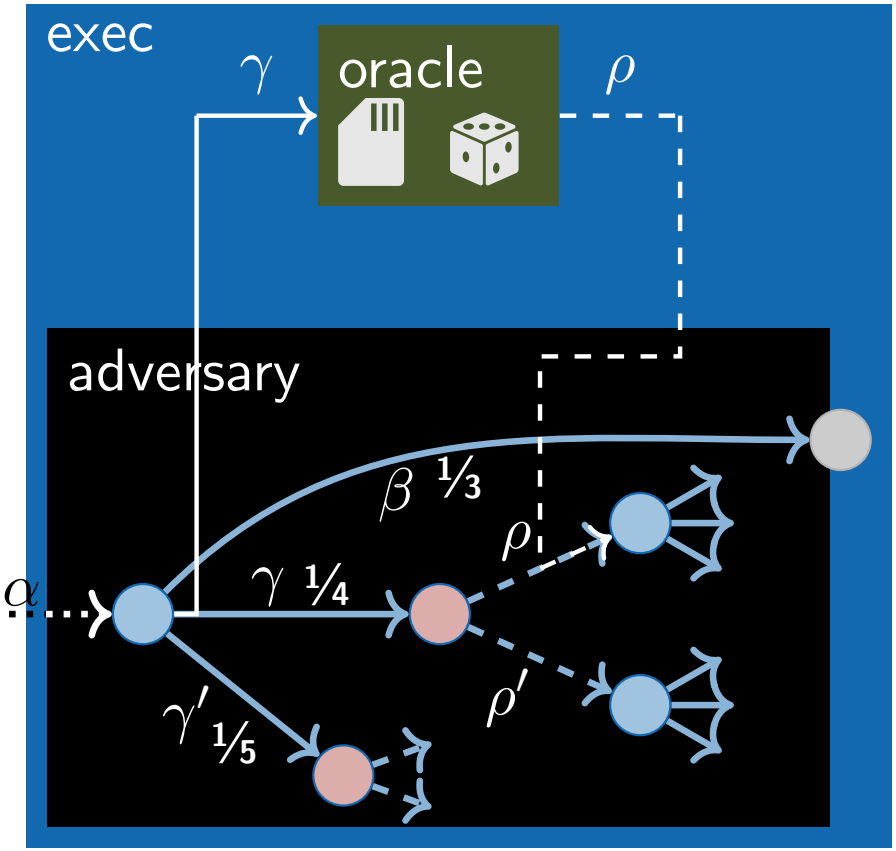
gpv $\cong$ (β $+ \gamma \times$ rpv) sprob
rpv $\cong \rho \to$ gpv

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

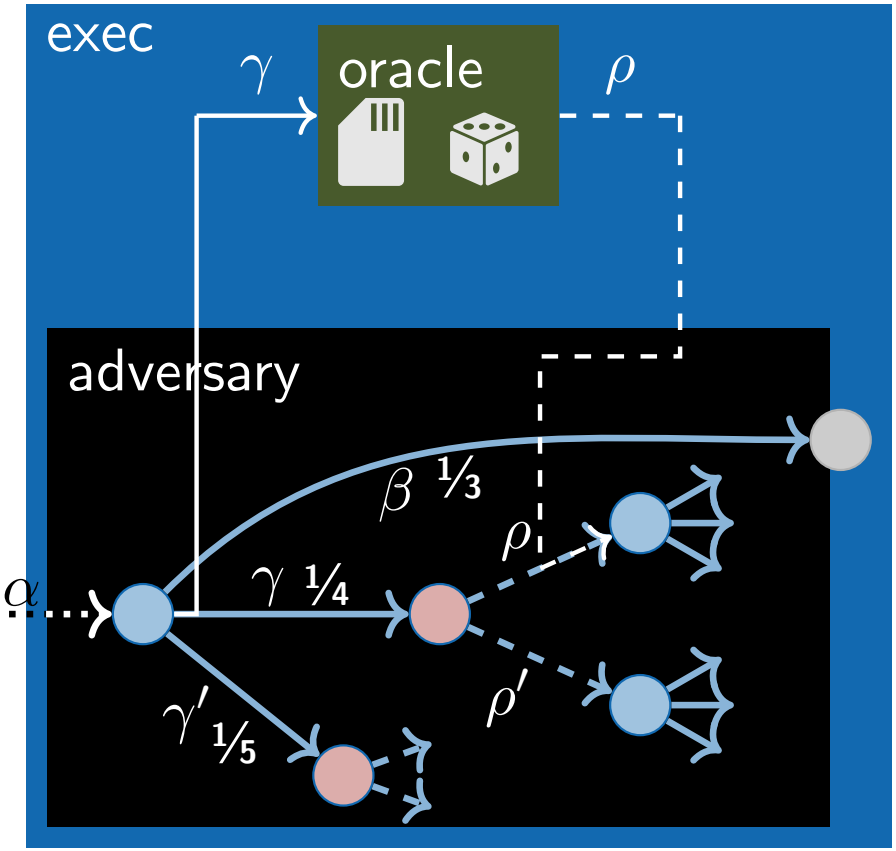
gpv $\cong$ (β + $\gamma \times$ rpv) sprob
rpv $\cong \rho \to$ gpv

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

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

Rules for reasoning about game transformations

Equational reasoning

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

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

Relational reasoning

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

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

Finding relational rules

Shape of rule determined by operator type

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

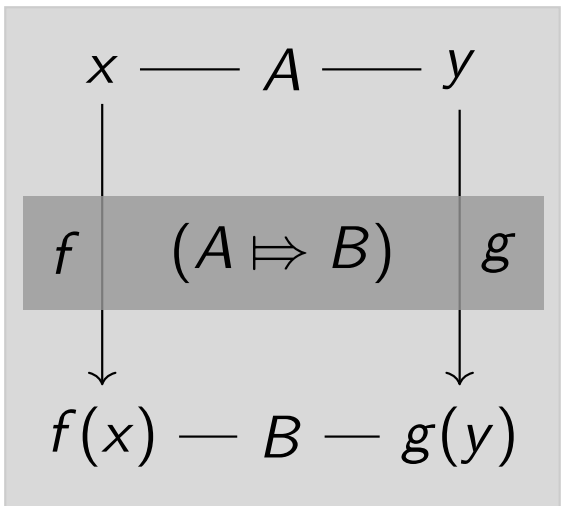
Replace types by relations

Finding relational rules

Shape of rule determined by operator type

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

Replace types by relations

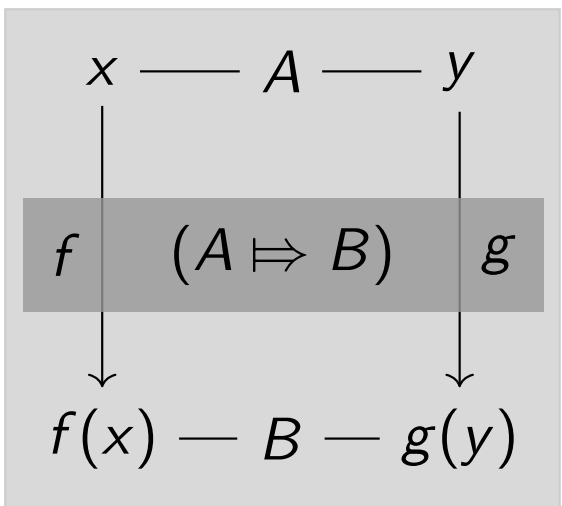


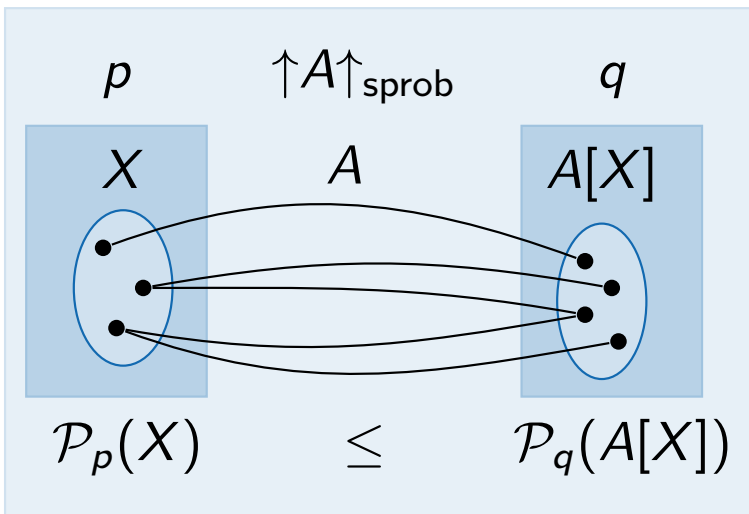
Finding relational rules

Shape of rule determined by operator type

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

Replace types by relations





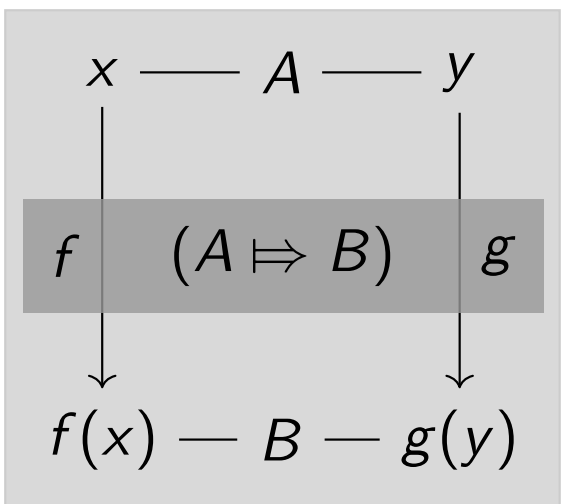
Finding relational rules

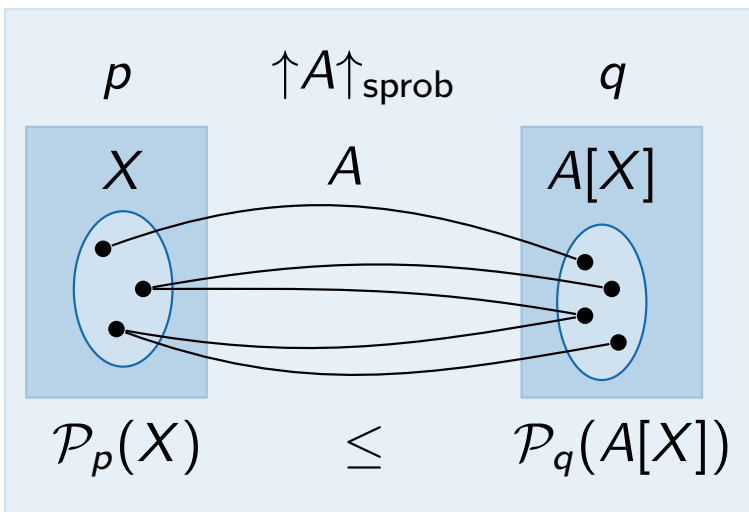
Shape of rule determined by operator type

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

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

Replace types by relations (Reynolds: relational parametricity)





Finding relational rules

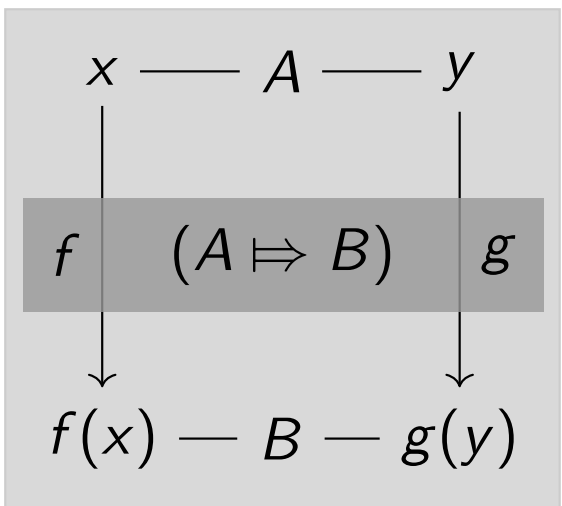
Shape of rule determined by operator type

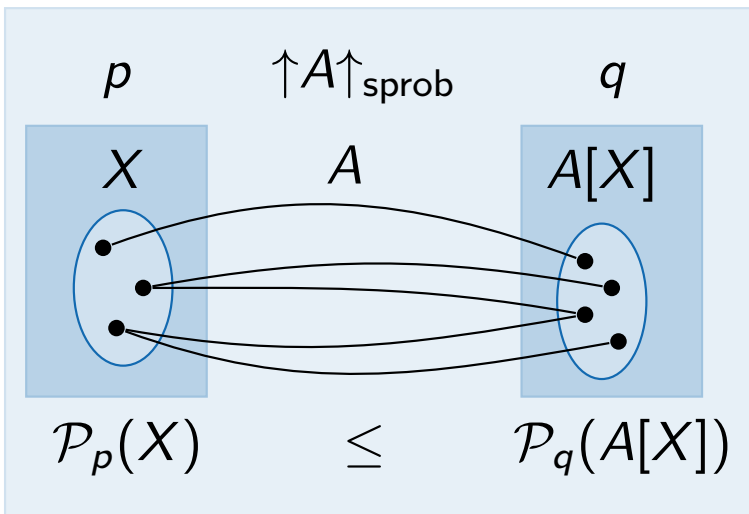
$$\mathsf{bind}_{\mathsf{sprob}} : \alpha\ \mathsf{sprob} \Rightarrow (\alpha \Rightarrow \beta\ \mathsf{sprob}) \Rightarrow \beta\ \mathsf{sprob}$$

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

Replace types by relations (Reynolds: relational parametricity)

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





Finding relational rules

Shape of rule determined by operator type

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

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

Replace types by relations (Reynolds: relational parametricity)

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

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

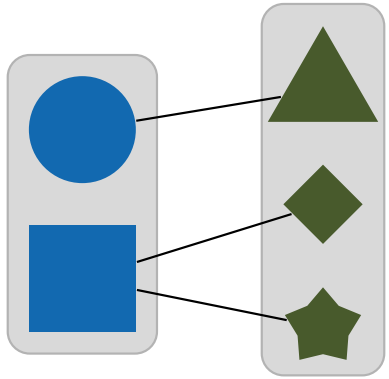
$$\frac{\mathcal{A}_1 \uparrow A\uparrow_{\mathsf{gpv}} \mathcal{A}_2 \qquad \mathcal{O}_1\ (S \Rightarrow (=) \Rightarrow \uparrow(=) \times S\uparrow_{\mathsf{sprob}})\ \mathcal{O}_2 \qquad \sigma_1\ S\ \sigma_2}{\mathsf{exec}(\mathcal{A}_1, \mathcal{O}_1, \sigma_1) \uparrow A \times S\uparrow_{\mathsf{sprob}}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

Sampling is conditionally parametric

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

uniform :

α set	$\Rightarrow$	α sprob
$\uparrow A \uparrow_{\text{set}}$	$\mapsto$	$\uparrow A \uparrow_{\text{sprob}}$

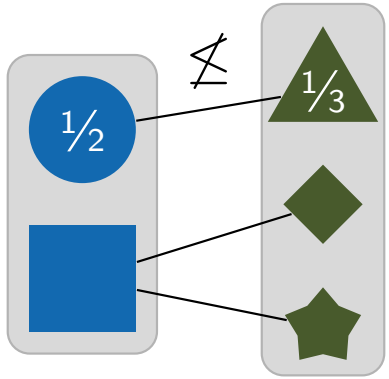


Sampling is conditionally parametric

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

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

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



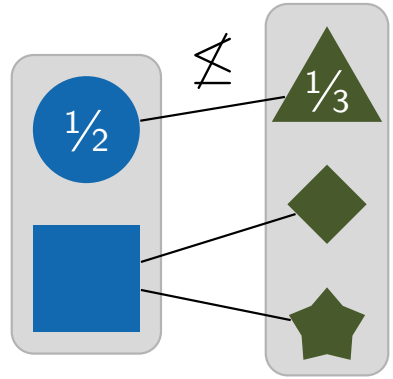
Wadler, Reynolds: Polymorphic equality is not parametric!

Sampling is conditionally parametric

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

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

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



Wadler, Reynolds: Polymorphic equality is not parametric!

A must respect equality!

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

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

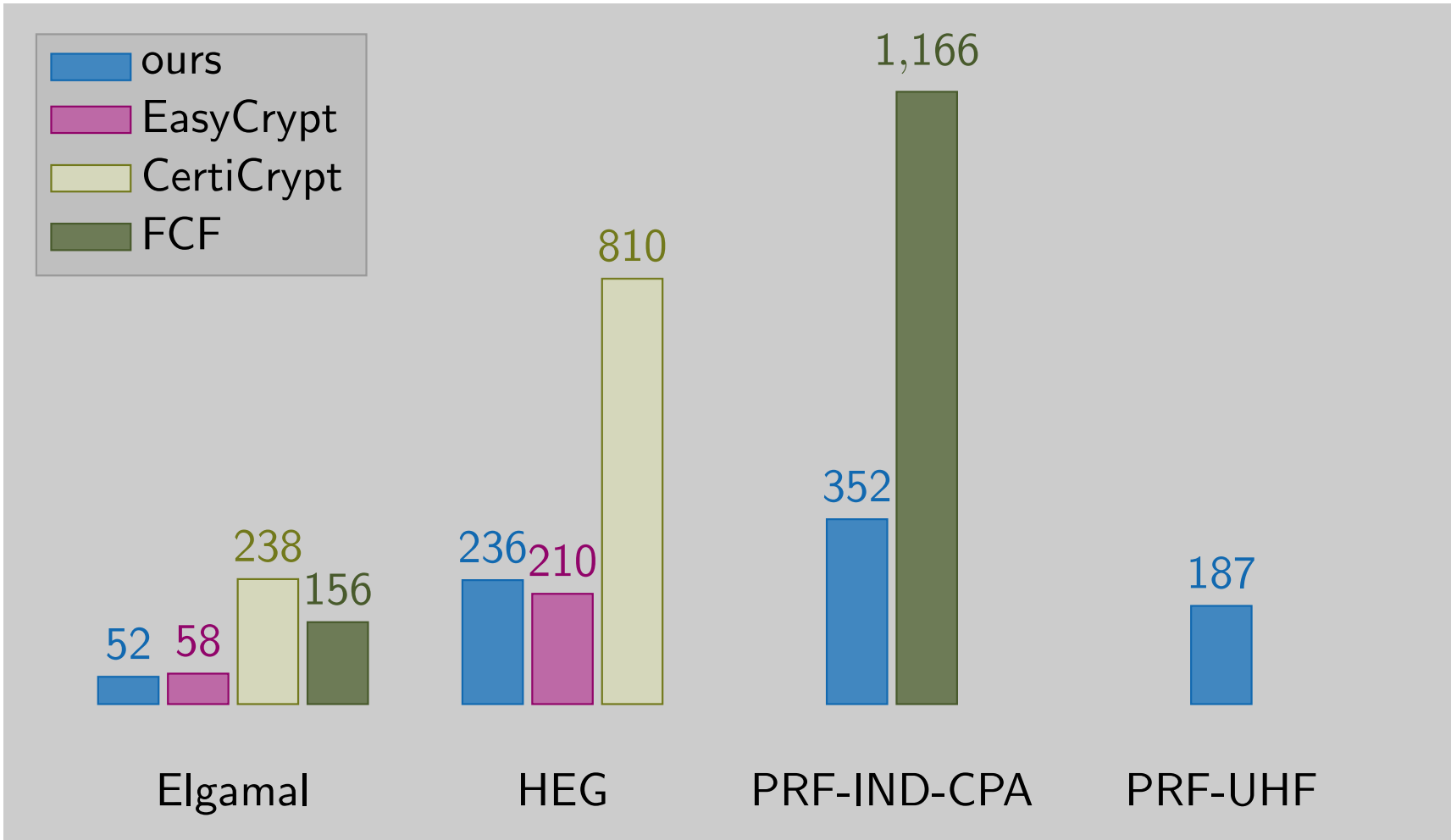
Special case: **one-time pad**

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

Applications

Used framework to verify textbook cryptographic constructions.

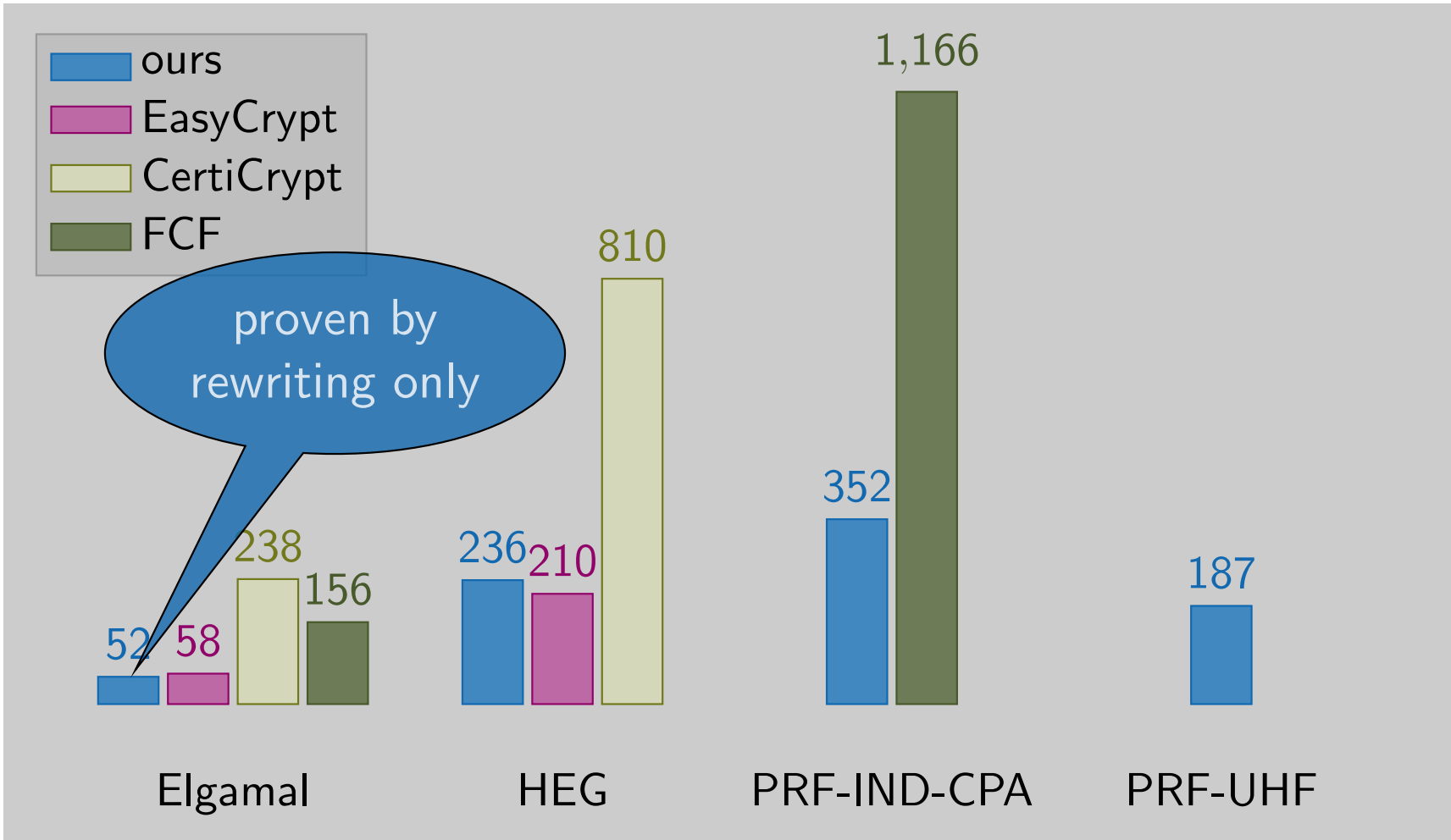
Line counts of proof of concrete security theorem



Applications

Used framework to verify textbook cryptographic constructions.

Line counts of proof of concrete security theorem



```
locale elgamal = elgamal_base + cyclic_group 𝒢 +
  assumes finite_group: "finite (carrier 𝒢)"
begin

lemma advantage_elgamal: "ddh.advantage (elgamal_adversary 𝒜) = ind_cpa.advantage 𝒜"
proof -
  obtain 𝒜1 𝒜2 where [simp]: "𝒜 = (𝒜1, 𝒜2)" by(cases 𝒜)
  note [simp] = order_gt_0_iff_finite finite_group
  have "ddh.ddh_1 (elgamal_adversary 𝒜) = TRY do {
        let q = order 𝒢;
        x ← sample_uniform q;
        y ← sample_uniform q;
        ((msg1, msg2), σ) ← 𝒜1 (g (^) x);
        _ :: unit ← assert_spmf (msg1 ∈ carrier 𝒢 ∧ msg2 ∈ carrier 𝒢);
        b ← coin_spmf;
        z ← map_spmf (λz. g (^) z ⊗ (if b then msg1 else msg2)) (sample_uniform q);
        guess ← 𝒜2 (g (^) y, z) σ;
        return_spmf (guess ⟷ b)
      } ELSE coin_spmf"
    apply(simp add: Let_def try_spmf_bind_out split_def bind_map_spmf o_def ddh.ddh_1_def)
    apply(simp add: bind_commute_spmf[where p=coin_spmf] bind_commute_spmf[where q="𝒜1 _"])
    apply(simp add: bind_commute_spmf[where q="assert_spmf _"])
    done
  also have "... = TRY do {
        let q = order 𝒢;
        x ← sample_uniform q;
        y ← sample_uniform q;
        ((msg1, msg2), σ) ← 𝒜1 (g (^) x);
        _ :: unit ← assert_spmf (msg1 ∈ carrier 𝒢 ∧ msg2 ∈ carrier 𝒢);
        z ← map_spmf (λz. g (^) z) (sample_uniform q);
        guess ← 𝒜2 (g (^) y, z) σ;
        map_spmf (op = guess) coin_spmf
      } ELSE coin_spmf"
    by(simp add: Let_def sample_uniform_one_time_pad cong: bind_spmf_cong_simp)
      (simp add: map_spmf_conv_bind_spmf bind_commute_spmf[where p=coin_spmf])
  also have "... = coin_spmf"
    by(simp add: map_eq_const_coin_spmf Let_def split_def bind_spmf_const try_bind_spmf_loss
```

```
locale elgamal = elgamal_base + cyclic_group 𝒢 +
  assumes finite_group: "finite (carrier 𝒢)"
begin

lemma advantage_elgamal: "ddh.advantage (elgamal_adversary 𝒜) = ind_cpa.advantage 𝒜"
proof -
  obtain 𝒜1 𝒜2 where [simp]: "𝒜 = (𝒜1, 𝒜2)" by(cases 𝒜)
  note [simp] = order_gt_0_iff_finite finite_group
  have "ddh.ddh_1 (elgamal_adversary 𝒜) = TRY do {
        let q = order 𝒢;
        x ← sample_uniform q;
        y ← sample_uniform q;
        ((msg1, msg2), σ) ← 𝒜1 (g (^) x);
        _ :: unit ← assert_spmf (msg1 ∈ carrier 𝒢 ∧ msg2 ∈ carrier 𝒢);
        b ← coin_spmf;
        z ← map_spmf (λz. g (^) z ⊗ (if b then msg1 else msg2)) (sample_uniform q);
        guess ← 𝒜2 (g (^) y, z) σ;
        return_spmf (guess ⟷ b)
      } ELSE coin_spmf"
    apply(simp add: Let_def try_spmf_bind_out split_def bind_map_spmf o_def ddh.ddh_1_def)
    apply(simp add: bind_commute_spmf[where p=coin_spmf] bind_commute_spmf[where q="𝒜1 _"])
    apply(simp add: bind_commute_spmf[where q="assert_spmf _"])
    done
  also have "... = TRY do {
        let q = order 𝒢;
        x ← sample_uniform q;
        y ← sample_uniform q;
        ((msg1, msg2), σ) ← 𝒜1 (g (^) x);
        _ :: unit ← assert_spmf (msg1 ∈ carrier 𝒢 ∧ msg2 ∈ carrier 𝒢);
        z ← map_spmf (λz. g (^) z) (sample_uniform q);
        guess ← 𝒜2 (g (^) y, z) σ;
        map_spmf (op = guess) coin_spmf
      } ELSE coin_spmf"
    by(simp add: Let_def sample_uniform_one_time_pad cong: bind_spmf_cong_simp)
      (simp add: map_spmf_conv_bind_spmf bind_commute_spmf[where p=coin_spmf])
  also have "... = coin_spmf"
    by(simp add: map_eq_const_coin_spmf Let_def split_def bind_spmf_const try_bind_spmf_loss
```

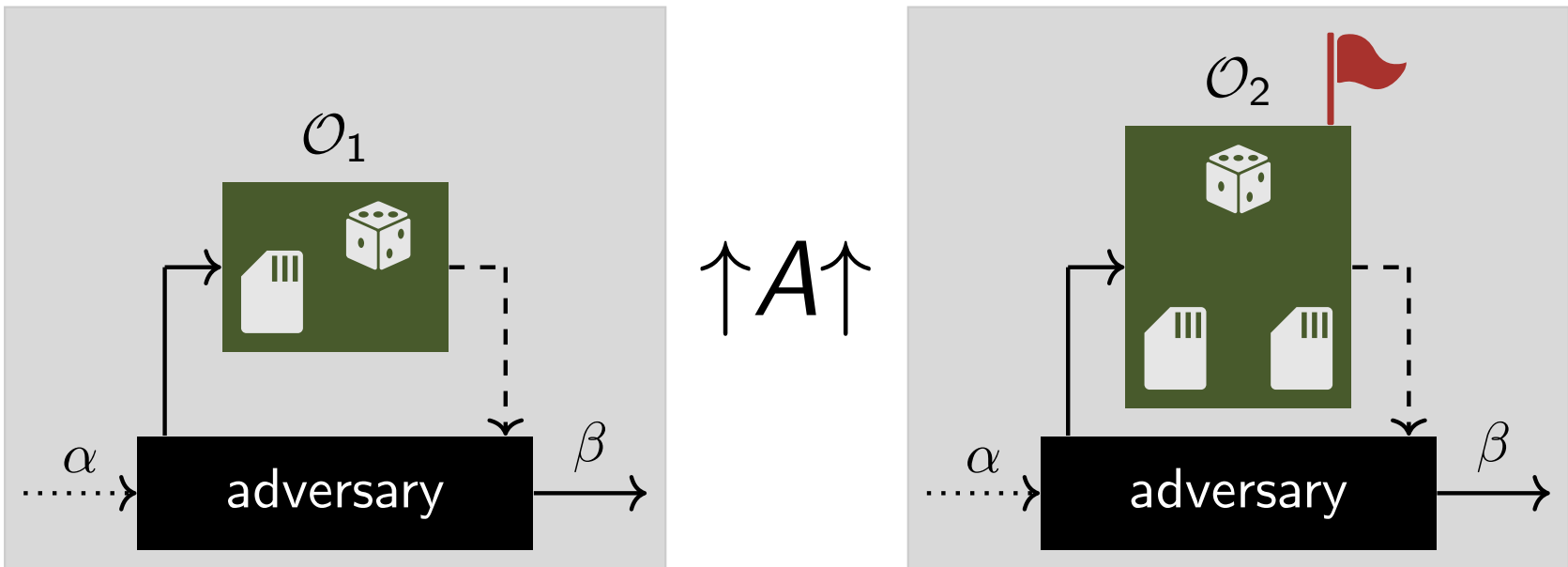
```
locale elgamal = elgamal_base + cyclic_group 𝒢 +
  assumes finite_group: "finite (carrier 𝒢)"
begin

lemma advantage_elgamal: "ddh.advantage (elgamal_adversary 𝒜) = ind_cpa.advantage 𝒜"
proof -
  obtain 𝒜1 𝒜2 where [simp]: "𝒜 = (𝒜1, 𝒜2)" by(cases 𝒜)
  note [simp] = order_gt_0_iff_finite finite_group
  have "ddh.ddh_1 (elgamal_adversary 𝒜) = TRY do {
        let q = order 𝒢;
        x ← sample_uniform q;
        y ← sample_uniform q;
        ((msg1, msg2), σ) ← 𝒜1 (g (^) x);
        _ :: unit ← assert_spmf (msg1 ∈ carrier 𝒢 ∧ msg2 ∈ carrier 𝒢);
        b ← coin_spmf;
        z ← map_spmf (λz. g (^) z ⊗ (if b then msg1 else msg2)) (sample_uniform q);
        guess ← 𝒜2 (g (^) y, z) σ;
        return_spmf (guess ⟷ b)
      } ELSE coin_spmf"
    apply(simp add: Let_def try_spmf_bind_out split_def bind_map_spmf o_def ddh.ddh_1_def)
    apply(simp add: bind_commute_spmf[where p=coin_spmf] bind_commute_spmf[where q="𝒜1 _"])
    apply(simp add: bind_commute_spmf[where q="assert_spmf _"])
    done
  also have "... = TRY do {
        let q = order 𝒢;
        x ← sample_uniform q;
        y ← sample_uniform q;
        ((msg1, msg2), σ) ← 𝒜1 (g (^) x);
        _ :: unit ← assert_spmf (msg1 ∈ carrier 𝒢 ∧ msg2 ∈ carrier 𝒢);
        z ← map_spmf (λz. g (^) z) (sample_uniform q);
        guess ← 𝒜2 (g (^) y, z) σ;
        map_spmf (op = guess) coin_spmf
      } ELSE coin_spmf"
    by(simp add: Let_def sample_uniform_one_time_pad cong: bind_spmf_cong_simp)
      (simp add: map_spmf_conv_bind_spmf bind_commute_spmf[where p=coin_spmf])
  also have "... = coin_spmf"
    by(simp add: map_eq_const_coin_spmf Let_def split_def bind_spmf_const try_bind_spmf_lossless
```

Parametricity and representation independence

- composition preserves parametricity
- exploit theory on relational parametricity

Example: representation independence [Mitchell]



- Prove bisimulation for S: $\mathcal{O}_1 \ (S \Mapsto (=) \Mapsto \uparrow(=) \times S\uparrow) \ \mathcal{O}_2$
- Lift to whole game using parametricity

Random oracle with map $s : \alpha \rightharpoonup \{0,1\}^n$

$$
\begin{array}{l}
\mathsf{RO}(s, x) = \\
\quad \textbf{if } x \in \mathsf{dom}(s) \textbf{ then} \\
\qquad \mathsf{return}_{\mathsf{sprob}}\ (s(x), s) \\
\quad \textbf{else do } \{ \\
\qquad bs \leftarrow \mathsf{uniform}\ \{0,1\}^n; \\
\qquad \mathsf{return}_{\mathsf{sprob}}\ (bs, s(x \mapsto bs))\ \}
\end{array}
$$

Random oracle with map $s : \alpha \rightharpoonup \{0,1\}^n$

Keep track of queries in Q

$\mathsf{RO}(s, x) =$
if $x \in \mathrm{dom}(s)$ **then**
  $\mathrm{return}_{\mathrm{sprob}}\ (s(x), s)$
else do $\{$
  $bs \leftarrow \mathrm{uniform}\ \{0,1\}^n$;
  $\mathrm{return}_{\mathrm{sprob}}\ (bs, s(x \mapsto bs))\ \}$

$\mathsf{RO}'((s, Q), x) =$
if $x \in \mathrm{dom}(s)$ **then**
  $\mathrm{return}_{\mathrm{sprob}}\ (s(x), (s, Q \cup \{x\}))$
else do $\{$
  $bs \leftarrow \mathrm{uniform}\ \{0,1\}^n$;
  $\mathrm{return}_{\mathrm{sprob}}\ (bs, (s(x \mapsto bs), Q \cup \{x\}))\ \}$

Random oracle with map $s : \alpha \rightharpoonup \{0,1\}^n$ — Keep track of queries in Q

$\mathrm{RO}(s, x) =$
if $x \in \mathrm{dom}(s)$ **then**
  $\mathrm{return}_{\mathrm{sprob}}\ (s(x), s)$
else do $\{$
  $bs \leftarrow \mathrm{uniform}\ \{0,1\}^n$;
  $\mathrm{return}_{\mathrm{sprob}}\ (bs, s(x \mapsto bs))\ \}$

$\mathrm{RO}'((s, Q), x) =$
if $x \in \mathrm{dom}(s)$ **then**
  $\mathrm{return}_{\mathrm{sprob}}\ (s(x), (s, Q \cup \{x\}))$
else do $\{$
  $bs \leftarrow \mathrm{uniform}\ \{0,1\}^n$;
  $\mathrm{return}_{\mathrm{sprob}}\ (bs, (s(x \mapsto bs), Q \cup \{x\}))\ \}$

1. Define relation on states: $\overline{s\ \mathrm{S}\ (s, Q)}$
2. Prove bisimulation property:
   - $\varnothing\ \mathrm{S}\ (\varnothing, \{\})$
   - If $s\ \mathrm{S}\ sQ$, then $\mathrm{RO}(s, x) \uparrow(=) \times S \uparrow_{\mathrm{sprob}} \mathrm{RO}'(sQ, x)$.
3. Appeal to representation independence:

$b \leftarrow \mathrm{uniform}\ \{0,1\}$;
$(m_0, m_1, \sigma, s_{\mathcal{O}}) \leftarrow \mathrm{exec}(\mathcal{A}.\mathrm{gen}(pk), \mathrm{RO}, \varnothing)$;
assert $(\text{valid-plain}(m_0) \wedge \text{valid-plain}(m_1))$;
$c^* \leftarrow \mathrm{aenc}(pk, \textbf{if}\ b\ \textbf{then}\ m_0\ \textbf{else}\ m_1)$;
$(b', \_) \leftarrow \mathrm{exec}(\mathcal{A}.\mathrm{guess}(c^*, \sigma), \mathrm{RO}, s_{\mathcal{O}})$;
$\mathrm{return}_{\mathrm{spmf}}\ (b = b')$

Random oracle with map $s : \alpha \rightharpoonup \{0,1\}^n$ — Keep track of queries in Q

$\mathrm{RO}(s, x) =$
if $x \in \mathrm{dom}(s)$ **then**
  $\mathrm{return}_{\mathrm{sprob}}\ (s(x), s)$
else do $\{$
  $bs \leftarrow \mathrm{uniform}\ \{0,1\}^n$;
  $\mathrm{return}_{\mathrm{sprob}}\ (bs, s(x \mapsto bs))\ \}$

$\mathrm{RO}'((s, Q), x) =$
if $x \in \mathrm{dom}(s)$ **then**
  $\mathrm{return}_{\mathrm{sprob}}\ (s(x), (s, Q \cup \{x\}))$
else do $\{$
  $bs \leftarrow \mathrm{uniform}\ \{0,1\}^n$;
  $\mathrm{return}_{\mathrm{sprob}}\ (bs, (s(x \mapsto bs), Q \cup \{x\}))\ \}$

1. Define relation on states: $\overline{s\ \mathrm{S}\ (s, Q)}$
2. Prove bisimulation property:
   - $\varnothing\ \mathrm{S}\ (\varnothing, \{\})$
   - If $s\ \mathrm{S}\ sQ$, then $\mathrm{RO}(s, x) \uparrow(=) \times S \uparrow_{\mathrm{sprob}} \mathrm{RO}'(sQ, x)$.
3. Appeal to representation independence:

$b \leftarrow \mathrm{uniform}\ \{0,1\}$;
$(m_0, m_1, \sigma, s_{\mathcal{O}}) \leftarrow \mathrm{exec}(\mathcal{A}.\mathrm{gen}(pk), \mathrm{RO}', (\varnothing, \{\}))$;
assert $(\text{valid-plain}(m_0) \wedge \text{valid-plain}(m_1))$;
$c^* \leftarrow \mathrm{aenc}(pk, \textbf{if}\ b\ \textbf{then}\ m_0\ \textbf{else}\ m_1)$;
$(b', \_) \leftarrow \mathrm{exec}(\mathcal{A}.\mathrm{guess}(c^*, \sigma), \mathrm{RO}', s_{\mathcal{O}})$;
$\mathrm{return}_{\mathrm{spmf}}\ (b = b')$

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