Formal Methods for Cryptography

Santiago Zanella-Béguelin
santiago@microsoft.com

Microsoft®
Research
Cambridge, UK

2013.03.04–08
18th Estonian Winter School in Computer Science
EWSCS 2013

Online supplemental material

http://easycrypt.gforge.inria.fr/ewscs/

Solutions

1. Roll six traditional fair dice. What is the probability of getting exactly 4 different numbers?

Solutions

1. Roll six traditional fair dice. What is the probability of getting exactly 4 different numbers?

Lazy solution: Haskell

```
let outcomes = sequence (replicate 6 [1..6]) in let fourd
=> 0.5015432098765432
```

That's more than 50%, bet on it!

Solutions

1. Roll six traditional fair dice. What is the probability of getting exactly 4 different numbers?

Lazy solution: Haskell

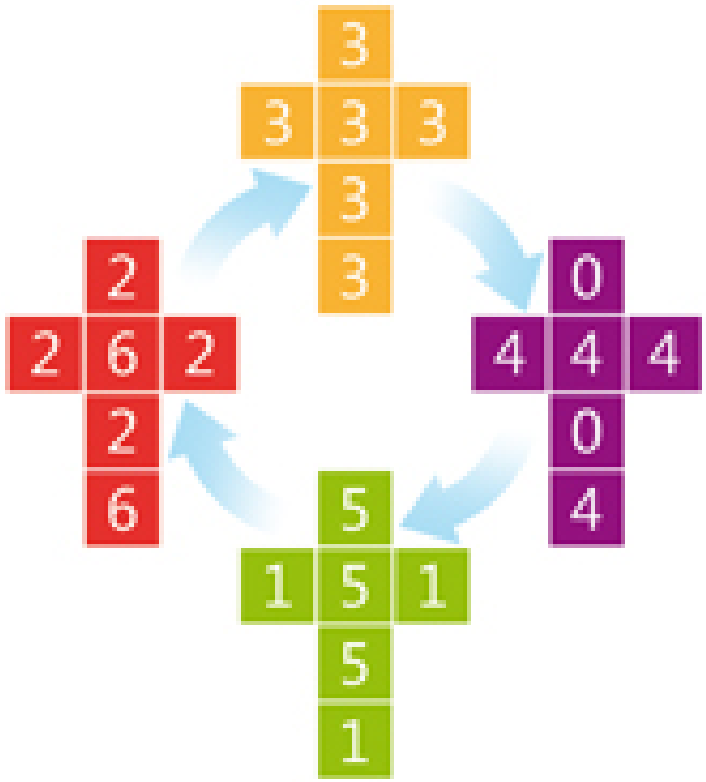
```
let outcomes = sequence (replicate 6 [1..6]) in let fourd
=> 0.5015432098765432
```

That's more than 50%, bet on it!
Reasoned solution:

$$\binom{6}{4}\left(4\times 6\times 5\times 4+\binom{4}{2}\times 6\times 5\times\binom{4}{2}\right)/6^6$$

2. Design a set of four 6-sided dice d_1, d_2, d_3, d_4 such that d_i rolls higher than d_{i+1} with probability more than $1/2$ for $i = 1, 2, 3$. What is the probability p that d_4 rolls higher than d_1? Can you design a set where $p < 1/2$?

2. Design a set of four 6-sided dice d_1, d_2, d_3, d_4 such that d_i rolls higher than d_{i+1} with probability more than $1/2$ for $i = 1, 2, 3$. What is the probability p that d_4 rolls higher than d_1? Can you design a set where $p < 1/2$?



Purple beats Orange with probability $2/3$
Orange beats Red with probability $2/3$
Red beats Green with probability $2/3$
but
Green beats Purple with probability $2/3$!

3. Let x, y be independent uniform random variables over the set of bitstrings of length k. What is the distribution of $x \oplus y$? Now suppose y is constant, what is the distribution of $x \oplus y$?

3. Let x, y be independent uniform random variables over the set of bitstrings of length k. What is the distribution of $x \oplus y$? Now suppose y is constant, what is the distribution of $x \oplus y$?

Answer: x is uniformly distributed no matter the distribution of y

3. Let x, y be independent uniform random variables over the set of bitstrings of length k. What is the distribution of $x \oplus y$? Now suppose y is constant, what is the distribution of $x \oplus y$?

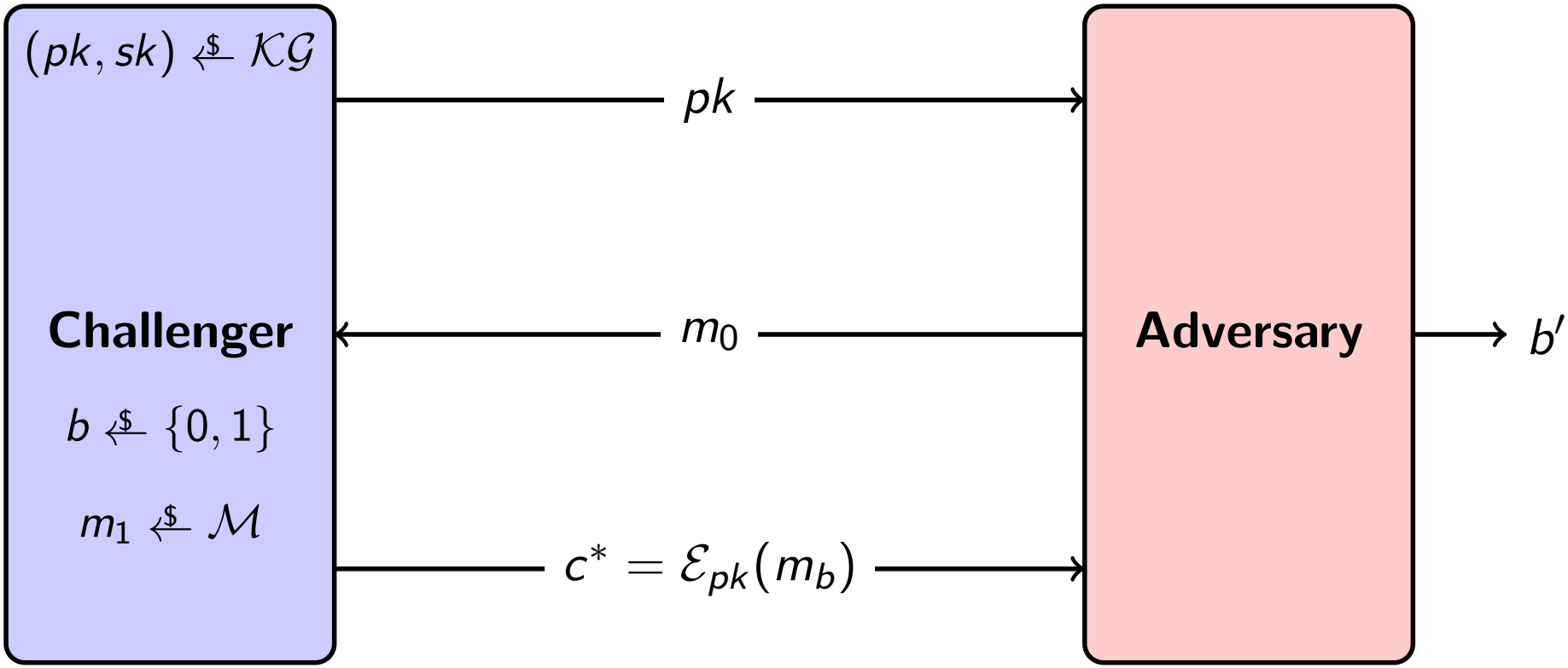
Answer: x is uniformly distributed no matter the distribution of y

4. Show by reduction the equivalence between ROR and CPA security for public-key encryption.

Recall the definitions...

Real-or-Random security

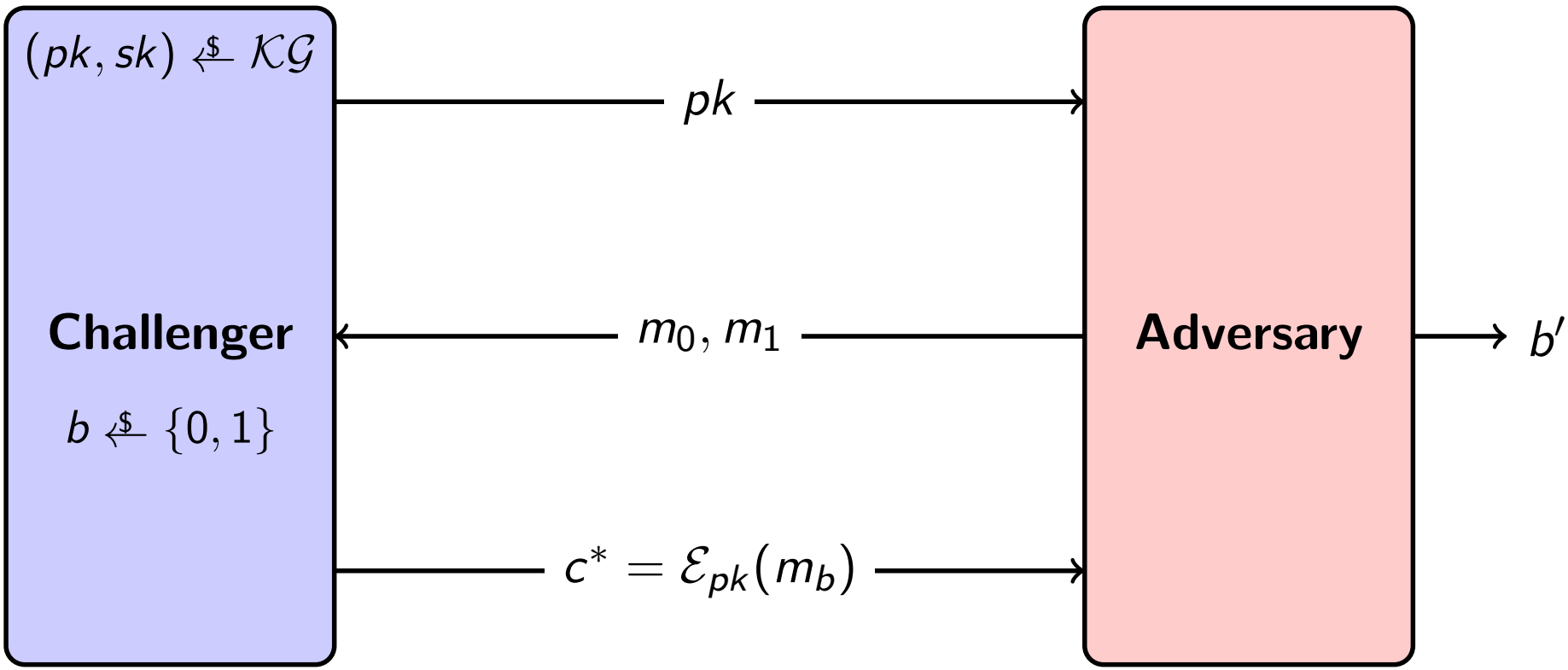
Define the experiment ROR:



Advantage of $\mathcal{A}$: $\mathbf{Adv}^{\mathcal{A}}_{\mathrm{ROR}} \stackrel{\mathrm{def}}{=} 2\ \Pr[\mathrm{ROR} : b = b'] - 1$

Chosen-plaintext security

Define the experiment CPA:



Advantage of $\mathcal{A}$: $\mathbf{Adv}^{\mathcal{A}}_{\mathrm{CPA}} \stackrel{\mathrm{def}}{=} 2\ \Pr[\mathrm{CPA} : b = b'] - 1$

ROR-security is as strong as CPA-security

Given an adversary $\mathcal{B}$ against ROR, construct an adversary $\mathcal{A}$ against CPA with the same advantage:

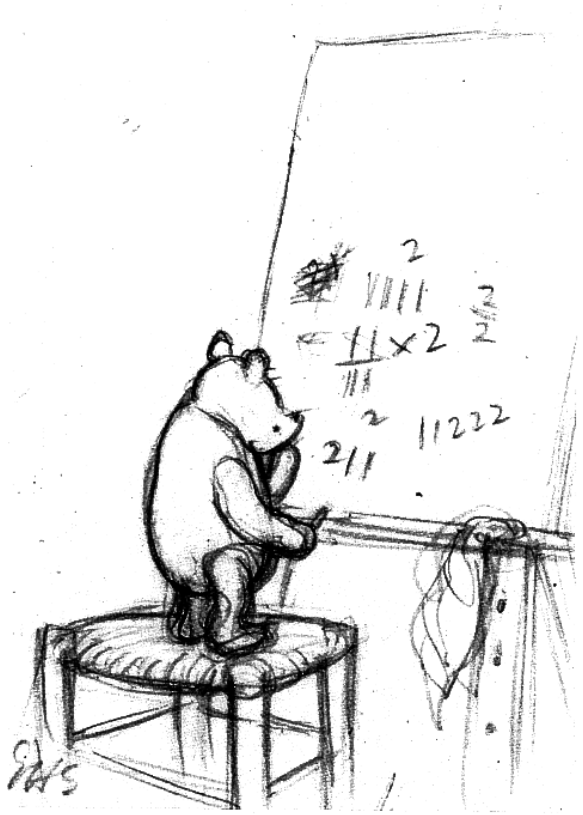
$$\mathcal{A}_1(pk) \stackrel{\text{def}}{=} (m_0, s) \leftarrow \mathcal{B}_1(pk);\ m_1 \xleftarrow{\$} \mathcal{M};\ \text{return } (m_0, m_1, s)$$
$$\mathcal{A}_2(c, s) \stackrel{\text{def}}{=} \text{return } \mathcal{B}_2(c, s)$$

CPA-security is almost as strong as ROR-security

Given an adversary $\mathcal{A}$ against CPA, construct an adversary $\mathcal{B}$ against ROR:

$$\begin{array}{lcl} \mathcal{B}_1(pk) & \stackrel{\text{def}}{=} & (m_0, m_1, s) \leftarrow \mathcal{A}_1(pk);\ d \xleftarrow{\$} \{0,1\};\ \text{return } (m_d, s) \\ \mathcal{B}_2(c, s) & \stackrel{\text{def}}{=} & d' \leftarrow \mathcal{A}_2(c, s);\ \text{return } (d = d') \end{array}$$

Code-Based Cryptographic Proofs



A language-based approach

What if we represent games as programs?

Games	$\Longrightarrow$	probabilistic programs
Game transformations	$\Longrightarrow$	program transformations
Adversaries	$\Longrightarrow$	unspecified procedures
Efficiency	$\Longrightarrow$	Probabilistic Polynomial-Time

A language-based approach

What if we represent games as programs?

Games	$\Longrightarrow$	probabilistic programs
Game transformations	$\Longrightarrow$	program transformations
Adversaries	$\Longrightarrow$	unspecified procedures
Efficiency	$\Longrightarrow$	Probabilistic Polynomial-Time

A language-based approach

What if we represent games as programs?

Games	$\Longrightarrow$	probabilistic programs
Game transformations	$\Longrightarrow$	program transformations
Adversaries	$\Longrightarrow$	unspecified procedures
Efficiency	$\Longrightarrow$	Probabilistic Polynomial-Time

A language-based approach

What if we represent games as programs?

Games	$\Longrightarrow$	probabilistic programs
Game transformations	$\Longrightarrow$	program transformations
Adversaries	$\Longrightarrow$	unspecified procedures
Efficiency	$\Longrightarrow$	Probabilistic Polynomial-Time

A language-based approach

What if we represent games as programs?

Games	$\Longrightarrow$	probabilistic programs
Game transformations	$\Longrightarrow$	program transformations
Adversaries	$\Longrightarrow$	unspecified procedures
Efficiency	$\Longrightarrow$	Probabilistic Polynomial-Time

A language-based approach

Security definitions, assumptions and games are formalized using a probabilistic programming language

pWhile:

$\mathcal{C}$	::=	skip	nop
	|	$\mathcal{C}$; $\mathcal{C}$	sequence
	|	$\mathcal{V} \leftarrow \mathcal{E}$	assignment
	|	$\mathcal{V} \xleftarrow{\$} \mathcal{DE}$	random sampling
	|	if $\mathcal{E}$ then $\mathcal{C}$ else $\mathcal{C}$	conditional
	|	while $\mathcal{E}$ do $\mathcal{C}$	while loop
	|	$\mathcal{V} \leftarrow \mathcal{P}(\mathcal{E}, \ldots, \mathcal{E})$	procedure call

- $x \xleftarrow{\$} d$: sample the value of x according to distribution d
- The language of expressions ($\mathcal{E}$) and distribution expressions ($\mathcal{DE}$) admits user-defined extensions

Some design choices

- Semantics formalized in the Coq proof assistant
- Deep-embedding
- Strongly-typed language
- Syntax is dependently-typed
  (only well-typed programs are admitted)
- Monadic semantics uses C. Paulin's ALEA Coq library

Representation of programs

Deep-embedding in the language of Coq, using a dependently-typed syntax:

$$
\begin{aligned}
&\textbf{Inductive } \mathcal{I} : \text{Type} := \\
&\mid \text{Assign} : \forall t,\ \mathcal{V}_t \to \mathcal{E}_t \to \mathcal{I} \\
&\mid \text{Rand} \quad : \forall t,\ \mathcal{V}_t \to \mathcal{D}_t \to \mathcal{I} \\
&\mid \text{Cond} \quad : \mathcal{E}_{\mathbb{B}} \to \mathcal{C} \to \mathcal{C} \to \mathcal{I} \\
&\mid \text{While} \ : \mathcal{E}_{\mathbb{B}} \to \mathcal{C} \to \mathcal{I} \\
&\mid \text{Call} \quad : \forall l\ t, \mathcal{P}_{(l,t)} \to\ \mathcal{V}_t \to \mathcal{E}^*_l \to \mathcal{I} \\
&\textbf{where } \mathcal{C} := \mathcal{I}^*
\end{aligned}
$$

Programs are well-typed by construction

Semantics

The Sub-Probability Monad

Distributions represented as monotonic, linear and Scott-continuous functions of type

$$\mathcal{D}(A) \stackrel{\text{def}}{=} (A \to [0,1]) \to [0,1]$$

Intuition: Given $\mu \in \mathcal{D}(A)$ and $f : A \to [0,1]$
$\mu(f)$ represents the expected value of f w.r.t. μ

$$\begin{aligned} \text{unit} &: A \to \mathcal{D}(A) && \stackrel{\text{def}}{=} \lambda x.\ \lambda f.\ f\ x \\ \text{bind} &: \mathcal{D}(A) \to (A \to \mathcal{D}(B)) \to \mathcal{D}(B) && \stackrel{\text{def}}{=} \lambda \mu.\ \lambda F.\ \lambda f.\ \mu(\lambda x.\ F\ x\ f) \end{aligned}$$

Semantics

Programs map an initial memory to a distribution on final memories

$$[\![c \in \mathcal{C}]\!] : \mathcal{M} \to \mathcal{D}(\mathcal{M})$$

The probability of an event is the expected value of its characteristic function:

$$\Pr[c, m : A] \stackrel{\text{def}}{=} [\![c]\!]\ m\ \mathbb{1}_A$$

Programs that do not terminate absolutely generate sub-distributions with total probability mass < 1

Instrumented and parametrized semantics to characterize PPT:

$$[\![c \in \mathcal{C}]\!]_\eta : \mathcal{M} \to \mathcal{D}(\mathcal{M} \times \mathbb{N})$$

Semantics

Programs map an initial memory to a distribution on final memories

$$[\![c \in \mathcal{C}]\!] : \mathcal{M} \to \mathcal{D}(\mathcal{M})$$

The probability of an event is the expected value of its characteristic function:

$$\Pr[c, m : A] \stackrel{\text{def}}{=} [\![c]\!]\ m\ \mathbb{1}_A$$

Programs that do not terminate absolutely generate sub-distributions with total probability mass < 1
Instrumented and parametrized semantics to characterize PPT:

$$[\![c \in \mathcal{C}]\!]_\eta : \mathcal{M} \to \mathcal{D}(\mathcal{M} \times \mathbb{N})$$

Intuition behind semantics

Think of $[\![\mathsf{c}]\!]\ m$ as the expectation operator of the probability distribution induced by the game:

$$[\![\mathsf{c}]\!]\ m\ f = \sum_{m'} f(m')\ \ \Pr[\langle c, m\rangle \Downarrow m']$$

Computing probabilities:

$$\Pr[c, m : A] \quad \stackrel{\text{def}}{=} \quad [\![c]\!]\ m\ \mathbb{1}_A$$

Example.

$$\text{Let c} \stackrel{\text{def}}{=} x \xleftarrow{\$} \{0,1\};\ y \xleftarrow{\$} \{0,1\}$$

$$\Pr[\mathsf{c}, m : x \neq y] = [\![\mathsf{c}]\!]\ m\ \mathbb{1}_{x\neq y} =$$

Intuition behind semantics

Think of $[\![\mathsf{c}]\!]\ m$ as the expectation operator of the probability distribution induced by the game:

$$[\![\mathsf{c}]\!]\ m\ f = \sum_{m'} f(m')\ \ \Pr[\langle c, m\rangle \Downarrow m']$$

Computing probabilities:

$$\Pr[c, m : A] \quad \stackrel{\text{def}}{=} \quad [\![c]\!]\ m\ \mathbb{1}_A$$

Example.

$$\text{Let c} \stackrel{\text{def}}{=} x \xleftarrow{\$} \{0,1\};\ y \xleftarrow{\$} \{0,1\}$$

$$\begin{aligned}\Pr[\mathsf{c}, m : x \neq y] = [\![\mathsf{c}]\!]\ m\ \mathbb{1}_{x\neq y} = \\ \tfrac{1}{4}\ \mathbb{1}_{x\neq y}(m[x \mapsto 0, y \mapsto 0]) \quad + \quad \tfrac{1}{4}\ \mathbb{1}_{x\neq y}(m[x \mapsto 0, y \mapsto 1]) \quad + \\ \tfrac{1}{4}\ \mathbb{1}_{x\neq y}(m[x \mapsto 1, y \mapsto 0]) \quad + \quad \tfrac{1}{4}\ \mathbb{1}_{x\neq y}(m[x \mapsto 1, y \mapsto 1])\end{aligned}$$

Intuition behind semantics

Think of $[\![\mathsf{c}]\!]\ m$ as the expectation operator of the probability distribution induced by the game:

$$[\![\mathsf{c}]\!]\ m\ f = \sum_{m'} f(m')\ \ \Pr[\langle c, m\rangle \Downarrow m']$$

Computing probabilities:

$$\Pr[c, m : A] \quad \stackrel{\text{def}}{=} \quad [\![c]\!]\ m\ \mathbb{1}_A$$

Example.

$$\text{Let c} \stackrel{\text{def}}{=} x \xleftarrow{\$} \{0,1\};\ y \xleftarrow{\$} \{0,1\}$$

$$\Pr[\mathsf{c}, m : x \neq y] = [\![\mathsf{c}]\!]\ m\ \mathbb{1}_{x\neq y} =$$

$$\begin{array}{lllll} 0 & & + \frac{1}{4} & & + \\ \frac{1}{4} & & + 0 & & \end{array}$$

Intuition behind semantics

Think of $[\![\mathsf{c}]\!]\ m$ as the expectation operator of the probability distribution induced by the game:

$$[\![\mathsf{c}]\!]\ m\ f = \sum_{m'} f(m')\ \ \Pr[\langle c, m\rangle \Downarrow m']$$

Computing probabilities:

$$\Pr[c, m : A] \quad \stackrel{\text{def}}{=} \quad [\![c]\!]\ m\ \mathbb{1}_A$$

Example.

$$\text{Let c} \stackrel{\text{def}}{=} x \xleftarrow{\$} \{0,1\};\ y \xleftarrow{\$} \{0,1\}$$

$$\Pr[\mathsf{c}, m : x \neq y] = [\![\mathsf{c}]\!]\ m\ \mathbb{1}_{x\neq y} = \frac{1}{2}$$

Deductive reasoning about programs

- Proving programs correct
- Foundations: program logic (Hoare'69) and weakest precondition calculus (Floyd'67)
- Major advances in:
  - language coverage
    (functions, objects, concurrency, heap)
  - automation
    (decision procedures, SMT solvers, invariant generation)
  - proof engineering
    (intermediate languages)

Hoare logic

- Judgments: $c : P \Rightarrow Q$, often noted $\{P\}c\{Q\}$
- Assertions: P, Q are predicates over program states (typically specified as f.o. formulae over program variables)
- Validity: if $(c, m) \Downarrow m'$ and $m \vDash P$, then $m' \vDash Q$

Some proof rules

$$\frac{}{x \leftarrow e : Q\{x := e\} \Rightarrow Q} \qquad \frac{c_1 : P \Rightarrow Q \quad c_2 : Q \Rightarrow R}{c_1; c_2 : P \Rightarrow R}$$

$$\frac{c_1 : P \wedge e = \mathrm{tt} \Rightarrow Q \quad c_2 : P \wedge e = \mathrm{ff} \Rightarrow Q}{\text{if } e \text{ then } c_1 \text{ else } c_2 : P \Rightarrow Q}$$

$$\frac{c : I \wedge e = \mathrm{tt} \Rightarrow I \quad P \rightarrow I \quad I \wedge e = \mathrm{ff} \rightarrow Q}{\text{while } e \text{ do } c : P \Rightarrow Q}$$

Verification condition generation

- Generate a set of verification conditions from annotated program and post-condition
- If all VCs are valid and $P \rightarrow \mathrm{wp}(c, Q)$ then $C : P \Rightarrow Q$

Selected rules

$$\begin{aligned}
\mathrm{wp}(x \leftarrow e, Q) &= Q\{x := e\} \\
\mathrm{wp}(c_1; c_2, R) &= \mathrm{wp}(c_1, \mathrm{wp}(c_2, R)) \\
\mathrm{wp}(\text{if } e \text{ then } c_1 \text{ else } c_2, Q) &= e{=}\mathrm{tt} \rightarrow \mathrm{wp}(c_1, Q) \wedge e{=}\mathrm{ff} \rightarrow \mathrm{wp}(c_2, Q) \\
\mathrm{wp}(\text{while } e \text{ do } c\ \{I\}, Q) &= I
\end{aligned}$$

The while rule generates two proof obligations

$$I \wedge e = \mathrm{tt} \rightarrow \mathrm{wp}(c, I) \qquad I \wedge e = \mathrm{ff} \rightarrow Q$$

Relational Hoare logic

- Judgments: $\models c_1 \sim c_2 : P \Rightarrow Q$
- Assertions: P, Q are binary relations over program states (specified as f.o. formulae over tagged program variables)
- Validity: if $P \models (m_1, m_2)$, $(c_1, m_1) \Downarrow m_1'$ and $(c_2, m_2) \Downarrow m_2'$, then $Q \models (m_1', m_2')$
- May require co-termination.

Verification methods

- Embedding into Hoare logic:
  - Self-composition (Barthe et al.'04)
  - Cross-products (Zaks and Pnueli'08, Barthe et al.'11)
- Relational Hoare Logic (Benton'04)

Relational Hoare Logic

Selected rules

$$\frac{\vDash c_1 \sim c_2 : \Psi \Rightarrow \Phi \quad \Psi' \to \Psi \quad \Phi \to \Phi'}{\vDash c_1 \sim c_2 : \Psi' \Rightarrow \Phi'}\,[\text{Sub}]$$

$$\frac{\vDash c_1 \sim c_2 : \Psi \Rightarrow \Phi \quad \vDash c_2 \sim c_3 : \Psi' \Rightarrow \Phi'}{\vDash c_1 \sim c_3 : \Psi \circ \Psi' \Rightarrow \Phi \circ \Phi'}\,[\text{Comp}]$$

$$\frac{\vDash c_1 \sim c_1' : \Psi \Rightarrow \Phi' \quad \vDash c_2 \sim c_2' : \Phi' \Rightarrow \Phi}{\vDash c_1; c_2 \sim c_1'; c_2' : \Psi \Rightarrow \Phi}\,[\text{Seq}]$$

$$\frac{}{\vDash x \leftarrow e \sim x \leftarrow e' : \Phi\{x\langle 1\rangle := e\langle 1\rangle, x\langle 2\rangle := e'\langle 2\rangle\} \Rightarrow \Phi}\,[\text{Asn}]$$

$$\frac{\begin{array}{l}\Psi \to e\langle 1\rangle \;\leftrightarrow\; e'\langle 2\rangle \\ \vDash c_1 \sim c_1' : \Psi \wedge e\langle 1\rangle \Rightarrow \Phi \quad \vDash c_2 \sim c_2' : \Psi \wedge \neg e\langle 1\rangle \Rightarrow \Phi\end{array}}{\vDash \text{if } e \text{ then } c_1 \text{ else } c_2 \sim \text{if } e' \text{ then } c_1' \text{ else } c_2' : \Psi \Rightarrow \Phi}\,[\text{Cond}]$$

Probabilistic Relational Hoare Logic

Probabilistic extension of Benton's Relational Hoare Logic

- Judgments: $\models c_1 \sim c_2 : P \Rightarrow Q$
- Assertions: P, Q are binary relations over program states (specified as f.o. formulae over tagged program variables)
- Validity: if $P \models (m_1, m_2)$, $[\![c_1]\!]m_1 = \mu_1$ and $[\![c_2]\!]m_2 = \mu_2$, then $Q^\sharp \models (\mu_1, \mu_2)$

$Q^\sharp$ lifts relation Q from states to distributions over states

Lifting Relations to Distributions

- Let $\mu_1 \in \mathcal{D}(A), \mu_2 \in \mathcal{D}(B)$, $\Psi \subseteq A \times B$.
- $\Psi^\sharp \subseteq \mathcal{D}(A) \times \mathcal{D}(B)$ is the smallest relation that satisfies:
  - If $(s, t) \models \Psi$ then $(\mathrm{unit}(s), \mathrm{unit}(t)) \models \Psi^\sharp$
    ($\mathrm{unit}(s)$ is the point distribution on s)
  - If $(\mu_i, \nu_i) \models \Psi^\sharp$ and $\sum_i p_i = 1$, then

$$\left(\sum_i p_i \, \mu_i, \sum_i p_i \, \nu_i \right) \models \Psi^\sharp$$

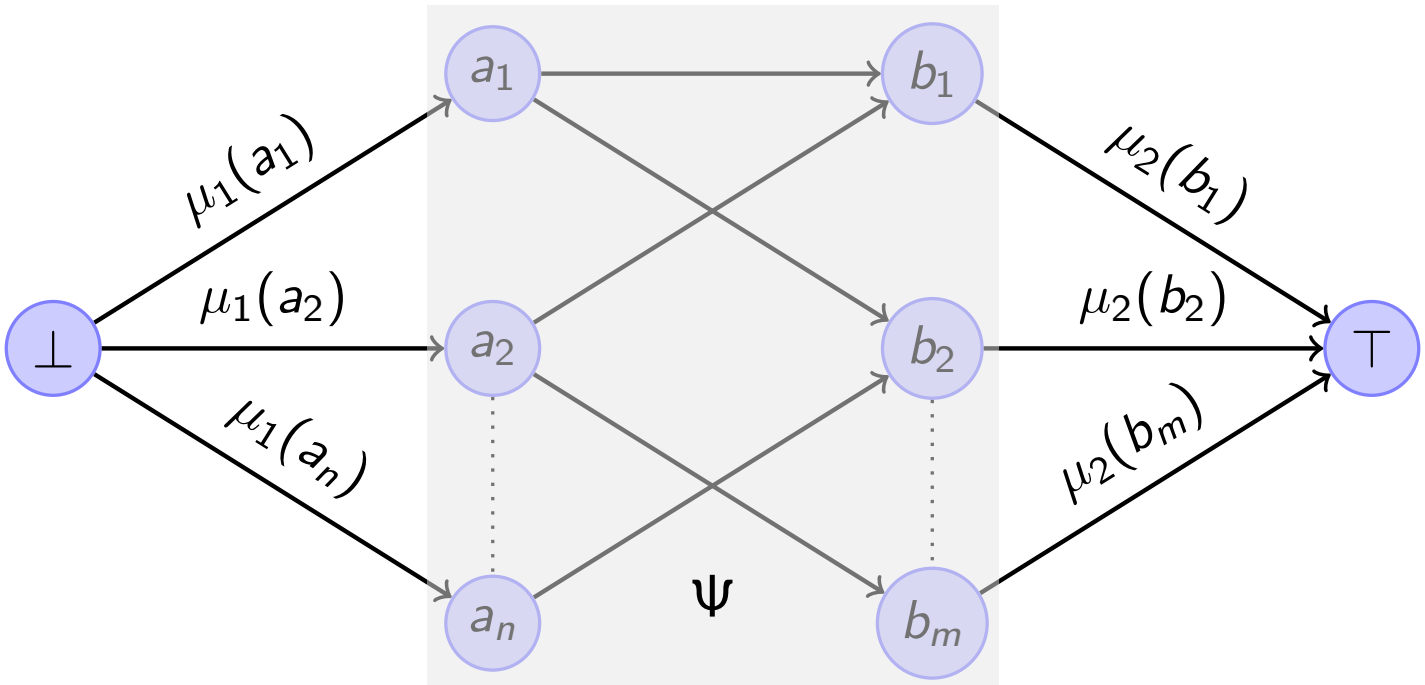
Lifting Relations to Distributions

Alternatively, $(\mu_1, \mu_2) \models \Psi^\sharp$ iff there exists $\mu \in \mathcal{D}(\mathcal{A} \times B)$ s.t.

- The 1^{st} projection of μ coincides with μ_1
- The 2^{nd} projection of μ coincides with μ_2
- The support of μ is a subset of Ψ

Lifting Relations to Distributions

Alternatively, $(\mu_1, \mu_2) \models \Psi^\sharp$ iff the maximum flow in the following network is 1:

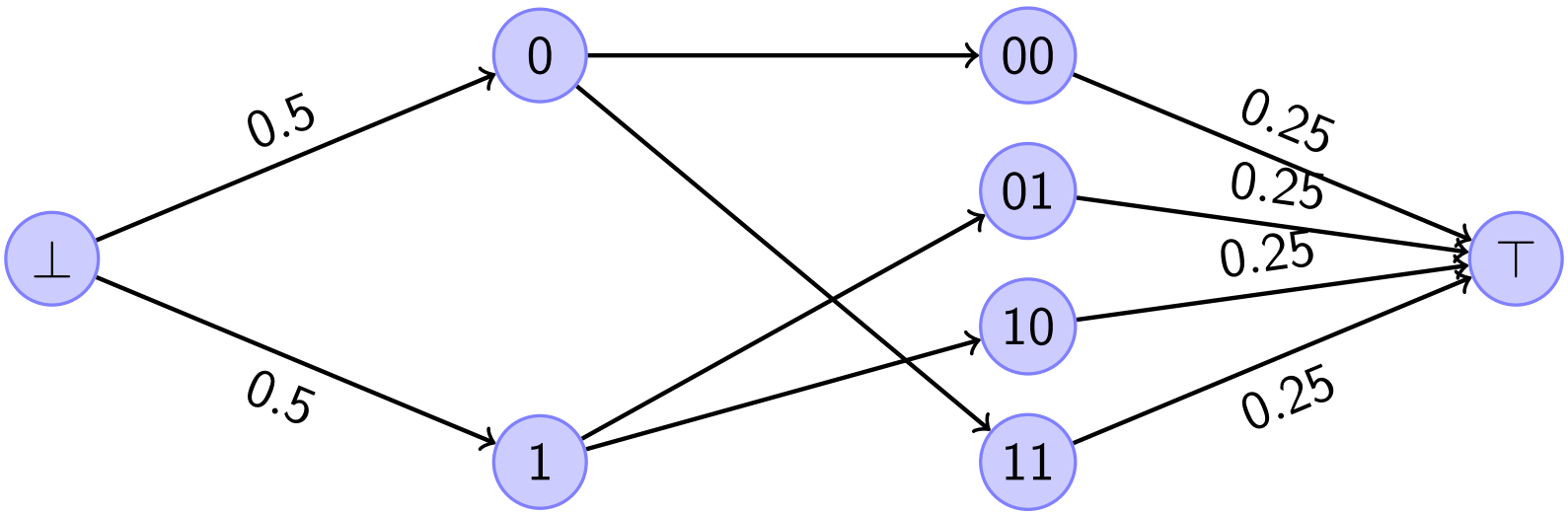


Example

$$c_1 \stackrel{\text{def}}{=} x \xleftarrow{\$} \{0,1\} \qquad c_2 \stackrel{\text{def}}{=} x \xleftarrow{\$} \{0,1\};\ y \xleftarrow{\$} \{0,1\}$$

- c_1 generates a distribution μ_1 over $\{0,1\}$
- c_2 generates a distribution μ_2 over $\{0,1\}^2$
- Consider $\Psi \stackrel{\text{def}}{=} x\langle 1\rangle = x\langle 2\rangle \oplus y\langle 2\rangle$

Q: Does $(\mu_1, \mu_2) \models \Psi^\sharp$ hold?



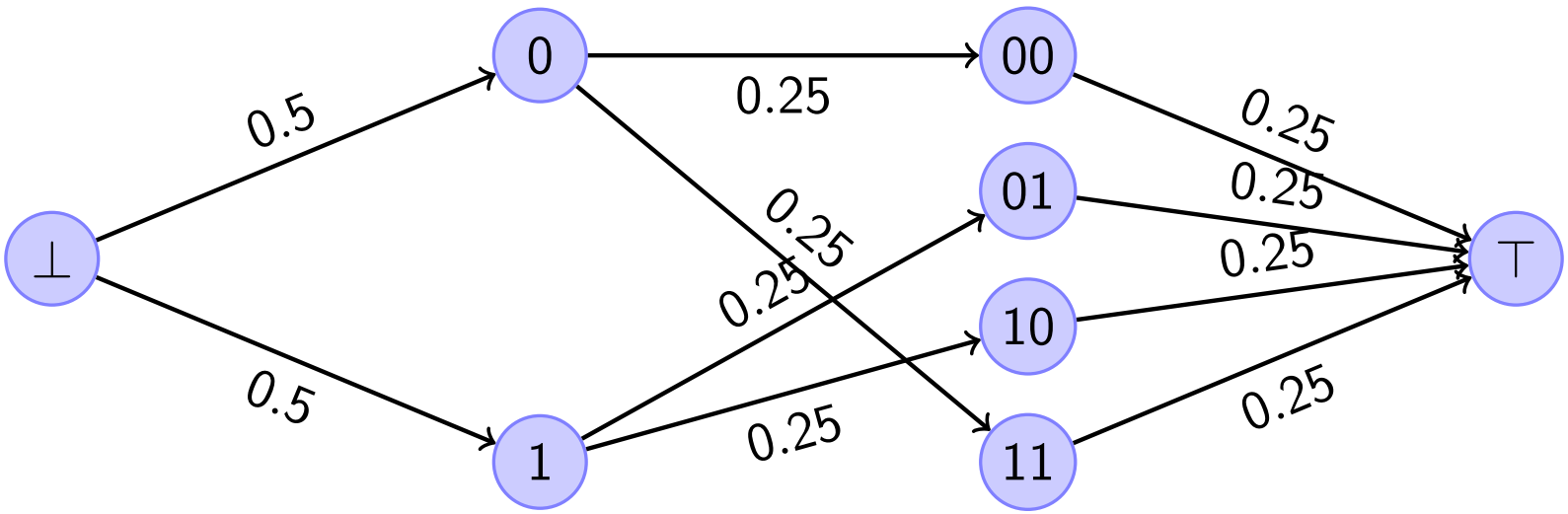
Example

$$c_1 \stackrel{\text{def}}{=} x \xleftarrow{\$} \{0,1\} \qquad c_2 \stackrel{\text{def}}{=} x \xleftarrow{\$} \{0,1\};\ y \xleftarrow{\$} \{0,1\}$$

- c_1 generates a distribution μ_1 over $\{0,1\}$
- c_2 generates a distribution μ_2 over $\{0,1\}^2$
- Consider $\Psi \stackrel{\text{def}}{=} x\langle 1\rangle = x\langle 2\rangle \oplus y\langle 2\rangle$

Q: Does $(\mu_1, \mu_2) \models \Psi^\sharp$ hold?

A: Yes, because we can construct a flow of value 1 in the corresponding network



Excercises

1. Define as sugar syntax a statement assert(b) in pWHILE, that cuts off execution *traces* that do not satisfy b. Hint: define first an abort statement s.t. forall $m : \mathcal{M}$, $f : \mathcal{M} \to [0,1]$, $[\![\text{abort}]\!]\ m\ f = 0$.

2. Define a uniform denotational semantics for pWHILE: $[\![c]\!]^* : \mathcal{D}(\mathcal{M}) \to \mathcal{D}(\mathcal{M})$

3. We saw during this lecture that the following judgment is valid:

$$\vDash x \xleftarrow{\$} \{0,1\}^k; y \leftarrow x \oplus z \simeq^{\{z\}}_{\{x,y\}} y \xleftarrow{\$} \{0,1\}^k; x \leftarrow y \oplus z$$

   Is this stronger judgment valid?

$$\vDash x \xleftarrow{\$} \{0,1\}^k; y \leftarrow x \oplus z \simeq^{\{z\}}_{\{x,y,z\}} y \xleftarrow{\$} \{0,1\}^k; x \leftarrow y \oplus z$$