Probabilistic Automata, Probabilistic Models and Crypto-protocols

Michael W. Mislove[1]

Joint Work With
Aaron Jaggard[2], Catherine Meadows[3] and Roberto Segala[4]

[1]Tulane University, New Orleans, LA, Work supported by ONR
[2]DIMACS, Rutgers University, Work supported by NSF
[3]Naval Research Laboratory, Washington, DC
[4]University of Verona, Italy

Protocol eXchange, NPS, January, 2009

Perspective

- Dolev-Yau model is unrealistic
  - Doesn't incorporate cryptographic primitives
  - Adversary can have too much power

Perspective

- Dolev-Yau model is unrealistic
  - Doesn't incorporate cryptographic primitives
  - Adversary can have too much power
- Alternative models try to be more realistic
  - Combine communication mechanisms with cryptographic primitives
  - Limit power of adversary

Perspective

- Dolev-Yau model is unrealistic
  - Doesn't incorporate cryptographic primitives
  - Adversary can have too much power
- Alternative models try to be more realistic
  - Combine communication mechanisms with cryptographic primitives
  - Limit power of adversary
- Attempts to address this utilize some form of probabilistic automata as the computational model
  - Typically state-based
  - Choice of next action from a state can be nondeterministic
  - Result is probability distribution over action–state pairs, or over next state

Perspective

- Dolev-Yau model is unrealistic
  - Doesn't incorporate cryptographic primitives
  - Adversary can have too much power
- Alternative models try to be more realistic
  - Combine communication mechanisms with cryptographic primitives
  - Limit power of adversary
- Attempts to address this utilize some form of probabilistic automata as the computational model
  - Typically state-based
  - Choice of next action from a state can be nondeterministic
  - Result is probability distribution over action–state pairs, or over next state
- Many approaches are arcane, especially in application of mathematical / statistical results to computational models.

System Equations

Models can be of several kinds, e.g.

- *Fully probabilistic:*

$$S \longrightarrow \mathrm{Disc}(\mathrm{Act} \times S)$$

System Equations

Models can be of several kinds, e.g.

- *Fully probabilistic:*

$$S \longrightarrow \mathrm{Disc}(\mathrm{Act} \times S)$$

- *Nondeterministic and Probabilistic:*

$$S \times \mathrm{Act} \longrightarrow \mathrm{Disc}(S) \;\simeq\; S \longrightarrow \mathrm{Disc}(S)^{\mathrm{Act}}$$

System Equations

Models can be of several kinds, e.g.

- *Fully probabilistic:*

$$S \longrightarrow \mathrm{Disc}(\mathrm{Act} \times S)$$

- *Nondeterministic and Probabilistic:*

$$S \times \mathrm{Act} \longrightarrow \mathrm{Disc}(S) \;\simeq\; S \longrightarrow \mathrm{Disc}(S)^{\mathrm{Act}}$$

- *Or Markovian:*

$$S \longrightarrow \mathcal{P}(\mathrm{Act} \times S + \{\perp\})$$

where $\mathcal{P}$ combines nondeterministic and probabilistic choice.

System Equations

Models can be of several kinds, e.g.

- *Fully probabilistic:*

$$S \longrightarrow \mathrm{Disc}(\mathrm{Act} \times S)$$

- *Nondeterministic and Probabilistic:*

$$S \times \mathrm{Act} \longrightarrow \mathrm{Disc}(S) \;\simeq\; S \longrightarrow \mathrm{Disc}(S)^{\mathrm{Act}}$$

- *Or Markovian:*

$$S \longrightarrow \mathcal{P}(\mathrm{Act} \times S + \{\bot\})$$

  where $\mathcal{P}$ combines nondeterministic and probabilistic choice.

Different equations result in different models, but common factor is the need to combine probabilistic and computational reasoning.

Outline

1. Motivating example:
   - Dining Cryptographers (Chaum (1998))
   - Focus is on anonymity

Outline

1. Motivating example:
   - Dining Cryptographers (Chaum (1998))
   - Focus is on anonymity
2. Survey some abstract models
   - Process algebras
   - Probabilistic automata
   - Specialize to PIOAs -
     Task-structured Probabilistic Input/Output Automata

Outline

1. Motivating example:
    - Dining Cryptographers (Chaum (1998))
    - Focus is on anonymity
2. Survey some abstract models
    - Process algebras
    - Probabilistic automata
    - Specialize to PIOAs -
      Task-structured Probabilistic Input/Output Automata
3. Describe abstract automaton to model example.

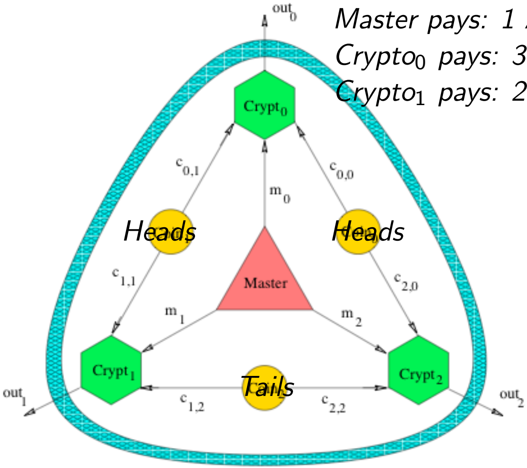
Outline

1. Motivating example:
   - Dining Cryptographers (Chaum (1998))
   - Focus is on anonymity
2. Survey some abstract models
   - Process algebras
   - Probabilistic automata
   - Specialize to PIOAs -
     Task-structured Probabilistic Input/Output Automata
3. Describe abstract automaton to model example.
4. Present mathematical results needed:
   - Give domain-theoretic versions of results from discrete probability theory

Outline

1. Motivating example:
   - Dining Cryptographers (Chaum (1998))
   - Focus is on anonymity
2. Survey some abstract models
   - Process algebras
   - Probabilistic automata
   - Specialize to PIOAs -
     Task-structured Probabilistic Input/Output Automata
3. Describe abstract automaton to model example.
4. Present mathematical results needed:
   - Give domain-theoretic versions of results from discrete probability theory
5. Apply results to model

Dining Cryptographers



Master pays: 1 Agree, 2 Disagree
$Crypto_0$ pays: 3 Disagree
$Crypto_1$ pays: 2 Agree, 1 Disagree

Some Computational Models

- Schneider & Sidiropoulos (1996) *CSP* with nondeterminism replacing probabilistic choice.

Some Computational Models

- Schneider & Sidiropoulos (1996) *CSP* with nondeterminism replacing probabilistic choice.
- Chatzikokolakis, Palamidessi & Panangaden (2008)
    - Nondeterminism supports reasoning only about *fair coins*
    - *Fairness* is crucial for ensuring anonymity

Some Computational Models

- Schneider & Sidiropoulos (1996) *CSP* with nondeterminism replacing probabilistic choice.
- Chatzikokolakis, Palamidessi & Panangaden (2008)
  - Nondeterminism supports reasoning only about *fair coins*
  - *Fairness* is crucial for ensuring anonymity
- Chatzikokolakis & Palamidessi (2007) Probabilistic π-calculus with schedulers

Some Computational Models

- Schneider & Sidiropoulos (1996) *CSP* with nondeterminism replacing probabilistic choice.
- Chatzikokolakis, Palamidessi & Panangaden (2008)
  - Nondeterminism supports reasoning only about *fair coins*
  - *Fairness* is crucial for ensuring anonymity
- Chatzikokolakis & Palamidessi (2007) Probabilistic π-calculus with schedulers
  - Use labels over choices to restrict the power of the adversary

Some Computational Models

- Schneider & Sidiropoulos (1996) *CSP* with nondeterminism replacing probabilistic choice.
- Chatzikokolakis, Palamidessi & Panangaden (2008)
  - Nondeterminism supports reasoning only about *fair coins*
  - *Fairness* is crucial for ensuring anonymity
- Chatzikokolakis & Palamidessi (2007) Probabilistic π-calculus with schedulers
  - Use labels over choices to restrict the power of the adversary
- Garcia, van Rossum & Sokolova (2007) Probabilistic automata with *admissable schedulers*

Some Computational Models

- Schneider & Sidiropoulos (1996) *CSP* with nondeterminism replacing probabilistic choice.
- Chatzikokolakis, Palamidessi & Panangaden (2008)
  - Nondeterminism supports reasoning only about *fair coins*
  - *Fairness* is crucial for ensuring anonymity
- Chatzikokolakis & Palamidessi (2007) Probabilistic π-calculus with schedulers
  - Use labels over choices to restrict the power of the adversary
- Garcia, van Rossum & Sokolova (2007) Probabilistic automata with *admissable schedulers*
- de Alfaro, Henzinger & Jhala (2001) *Probabilistic modules* consisting of typed variables and a body partitioned by disjoint sets of variables

Some Computational Models

- Schneider & Sidiropoulos (1996) *CSP* with nondeterminism replacing probabilistic choice.
- Chatzikokolakis, Palamidessi & Panangaden (2008)
  - Nondeterminism supports reasoning only about *fair coins*
  - *Fairness* is crucial for ensuring anonymity
- Chatzikokolakis & Palamidessi (2007) Probabilistic π-calculus with schedulers
  - Use labels over choices to restrict the power of the adversary
- Garcia, van Rossum & Sokolova (2007) Probabilistic automata with *admissable schedulers*
- de Alfaro, Henzinger & Jhala (2001) *Probabilistic modules* consisting of typed variables and a body partitioned by disjoint sets of variables
- Canetti, Lynch, et al (2006) Probabilistic input / output automata augmented with *tasks*

  Used to model oblivious transfer under universal composability

Abstract description of Dining Cryptographers

- *Master process:*
  - Chooses payer – herself or one of the cryptographers
  - Informs cryptographers whether they are paying or not

Abstract description of Dining Cryptographers

- *Master process:*
  - Chooses payer – herself or one of the cryptographers
  - Informs cryptographers whether they are paying or not
- *Cryptographers:*
  - Receive information from Master whether they are paying
  - Toss their own coins
  - Inform cryptographer on their right of the outcome
  - Compare their coin and that on the left
  - Announce their result
  - View other announcements

Abstract description of Dining Cryptographers

- *Master process:*
  - Chooses payer – herself or one of the cryptographers
  - Informs cryptographers whether they are paying or not
- *Cryptographers:*
  - Receive information from Master whether they are paying
  - Toss their own coins
  - Inform cryptographer on their right of the outcome
  - Compare their coin and that on the left
  - Announce their result
  - View other announcements

High-level vs. Low-level Nondeterminism

- *Low level:* Doesn't affect outcome
- *High level:* Can affect outcome

Abstract description of Dining Cryptographers

- *Master process:*
  - Chooses payer – herself or one of the cryptographers
  - Informs cryptographers whether they are paying or not
- *Cryptographers:*
  - Receive information from Master whether they are paying
  - Toss their own coins
  - Inform cryptographer on their right of the outcome
  - Compare their coin and that on the left
  - Announce their result
  - View other announcements

Schedulers:

- Set order of Low-level events
  Choose payer, Inform cryptographers, Flip coins, Compare, Announce
- Can't see outcome of High-level (Internal) choices

Probabilistic Input/Output Automata

- Mathematical model of concurrent computation
  - *Input/Output:* used to model interactions among component processes
  - *Hidden:* Internal actions

Probabilistic Input/Output Automata

- Mathematical model of concurrent computation
  - *Input/Output:* used to model interactions among component processes
  - *Hidden:* Internal actions

$\mathcal{A} = (S, s_0, I, O, H, D)$ where

- S - countable set of states, s_0 - start state
- $\text{Act} ::= I \cup O \cup H$ - countable set of actions
- $D \subseteq S \times \text{Act} \times \text{Disc}(S)$ transition relation satisfying:

Probabilistic Input/Output Automata

- Mathematical model of concurrent computation
  - *Input/Output:* used to model interactions among component processes
  - *Hidden:* Internal actions

$\mathcal{A} = (S, s_0, I, O, H, D)$ where

- S - countable set of states, s_0 - start state
- $\text{Act} ::= I \cup O \cup H$ - countable set of actions
- $D \subseteq S \times \text{Act} \times \text{Disc}(S)$ transition relation satisfying:

  Input enabling: $(\forall s \in S)(\forall a \in I)(\exists \mu)\ (s, a, \mu) \in D$

Probabilistic Input/Output Automata

- Mathematical model of concurrent computation
  - *Input/Output:* used to model interactions among component processes
  - *Hidden:* Internal actions

$\mathcal{A} = (S, s_0, I, O, H, D)$ where

- S - countable set of states, s_0 - start state
- $\text{Act} ::= I \cup O \cup H$ - countable set of actions
- $D \subseteq S \times \text{Act} \times \text{Disc}(S)$ transition relation satisfying:

  Input enabling: $(\forall s \in S)(\forall a \in I)(\exists \mu)\ (s, a, \mu) \in D$

  Transition determinism: $(s, a, \mu), (s, a, \nu) \in D \ \Rightarrow\ \mu = \nu$

Probabilistic Input/Output Automata

- Mathematical model of concurrent computation
  - *Input/Output:* used to model interactions among component processes
  - *Hidden:* Internal actions

$\mathcal{A} = (S, s_0, I, O, H, D)$ where

- S - countable set of states, s_0 - start state
- $\text{Act} ::= I \cup O \cup H$ - countable set of actions
- $D \subseteq S \times \text{Act} \times \text{Disc}(S)$ transition relation satisfying:

  Input enabling: $(\forall s \in S)(\forall a \in I)(\exists \mu)\ (s, a, \mu) \in D$

  Transition determinism: $(s, a, \mu), (s, a, \nu) \in D \ \Rightarrow\ \mu = \nu$

$$D \colon S \times \text{Act} \rightharpoonup \text{Disc}(S)$$

How to compose PIOAs

$\mathcal{A}_i = (S_i, s_{\mathcal{A}_i}, I_i, O_i, H_i, D_i),\ \ i = 1, 2$ are *compatible* if

$$\mathrm{Act}_i \cap H_{i+1} = \emptyset = O_1 \cap O_2$$

How to compose PIOAs

$\mathcal{A}_i = (S_i, s_{\mathcal{A}_i}, I_i, O_i, H_i, D_i),\ \ i = 1, 2$ are *compatible* if

$$\text{Act}_i \cap H_{i+1} = \emptyset = O_1 \cap O_2$$

$\mathcal{A}_1 \| \mathcal{A}_2 = (S_{\mathcal{A}_1\|\mathcal{A}_2}, s_{\mathcal{A}_1\|\mathcal{A}_2}, I_{\mathcal{A}_1\|\mathcal{A}_2}, O_{\mathcal{A}_1\|\mathcal{A}_2}, H_{\mathcal{A}_|\|\mathcal{A}_2}, D_{\mathcal{A}_1\|\mathcal{A}_2})$ where:

- $S_{\mathcal{A}_1\|\mathcal{A}_2} = S_1 \times S_2$
- $s_{\mathcal{A}_1\|\mathcal{A}_s} = \langle s_{\mathcal{A}_1}, s_{\mathcal{A}_2} \rangle$
- $I_{\mathcal{A}_1\|\mathcal{A}_2} = (I_1 \cup I_2) \setminus (O_1 \cup O_2)$
- $O_{\mathcal{A}_1\|\mathcal{A}_2} = O_1 \cup O_2$
- $H_{\mathcal{A}_1\|\mathcal{A}_2} = H_1 \cup H_2$
- $D_{\mathcal{A}_1\|\mathcal{A}_2} = \{((s_1, s_2), a, \mu_1 \times \mu_2) \mid (s_1, a, \mu_1) \in D_1 \text{ or } (s_2, a, \mu_2) \in D_2$ and $\mu_i = \delta_{s_i}$ if $a \notin \text{Act}_i\}$

How to compose PIOAs

$\mathcal{A}_i = (S_i, s_{\mathcal{A}_i}, I_i, O_i, H_i, D_i),\ i = 1, 2$ are *compatible* if

$$\mathrm{Act}_i \cap H_{i+1} = \emptyset = O_1 \cap O_2$$

$\mathcal{A}_1 \| \mathcal{A}_2 = (S_{\mathcal{A}_1\|\mathcal{A}_2}, s_{\mathcal{A}_1\|\mathcal{A}_2}, I_{\mathcal{A}_1\|\mathcal{A}_2}, O_{\mathcal{A}_1\|\mathcal{A}_2}, H_{\mathcal{A}_|\|\mathcal{A}_2}, D_{\mathcal{A}_1\|\mathcal{A}_2})$ where:

- $S_{\mathcal{A}_1\|\mathcal{A}_2} = S_1 \times S_2$
- $s_{\mathcal{A}_1\|\mathcal{A}_s} = \langle s_{\mathcal{A}_1}, s_{\mathcal{A}_2} \rangle$
- $I_{\mathcal{A}_1\|\mathcal{A}_2} = (I_1 \cup I_2) \setminus (O_1 \cup O_2)$
- $O_{\mathcal{A}_1\|\mathcal{A}_2} = O_1 \cup O_2$
- $H_{\mathcal{A}_1\|\mathcal{A}_2} = H_1 \cup H_2$
- $D_{\mathcal{A}_1\|\mathcal{A}_2} = \{((s_1, s_2), a, \mu_1 \times \mu_2) \mid (s_1, a, \mu_1) \in D_1 \text{ or } (s_2, a, \mu_2) \in D_2$ and $\mu_i = \delta_{s_i}$ if $a \notin \mathrm{Act}_i\}$

$\mathcal{A}$ is *closed* if $I_{\mathcal{A}} = \emptyset$

Tasks

- **Tasks:** Equivalence relation $\mathcal{R} \subseteq (O \cup H) \times (O \cup H)$.

  Task: Any equivalence class of $\mathcal{R}$.

  Action determinism: For each state s and each task T, at most one action $a \in T$ is enabled in s.

Tasks

- **Tasks:** Equivalence relation $\mathcal{R} \subseteq (O \cup H) \times (O \cup H)$.

  Task: Any equivalence class of $\mathcal{R}$.

  Action determinism: For each state s and each task T, at most one action $a \in T$ is enabled in s.

  Resolves nondeterminism: In given state, next task specifies which action to execute.

Tasks

- **Tasks:** Equivalence relation $\mathcal{R} \subseteq (O \cup H) \times (O \cup H)$.

  Task: Any equivalence class of $\mathcal{R}$.

  Action determinism: For each state s and each task T, at most one action $a \in T$ is enabled in s.

  Resolves nondeterminism: In given state, next task specifies which action to execute.

- **Composition:** $\mathcal{R}_{\mathcal{A}_1 \| \mathcal{A}_2} = \mathcal{R}_{\mathcal{A}_1} \cup \mathcal{R}_{\mathcal{A}_2}$

Tasks

- **Tasks:** Equivalence relation $\mathcal{R} \subseteq (O \cup H) \times (O \cup H)$.

  Task: Any equivalence class of $\mathcal{R}$.

  Action determinism: For each state s and each task T, at most one action $a \in T$ is enabled in s.

  Resolves nondeterminism: In given state, next task specifies which action to execute.
- **Composition:** $\mathcal{R}_{\mathcal{A}_1 \| \mathcal{A}_2} = \mathcal{R}_{\mathcal{A}_1} \cup \mathcal{R}_{\mathcal{A}_2}$
- **Application:** Via *Task Schedules* - $\rho = T_1 T_2 \cdots$

Tasks

- **Tasks:** Equivalence relation $\mathcal{R} \subseteq (O \cup H) \times (O \cup H)$.

  Task: Any equivalence class of $\mathcal{R}$.

  Action determinism: For each state s and each task T, at most one action $a \in T$ is enabled in s.

  Resolves nondeterminism: In given state, next task specifies which action to execute.
- **Composition:** $\mathcal{R}_{\mathcal{A}_1 \| \mathcal{A}_2} = \mathcal{R}_{\mathcal{A}_1} \cup \mathcal{R}_{\mathcal{A}_2}$
- **Application:** Via *Task Schedules* - $\rho = T_1 T_2 \cdots$

Dining Cryptographers:

$$\mathcal{M} \parallel \mathcal{C}rypt_0 \parallel \mathcal{C}rypt_1 \parallel \mathcal{C}rypt_2$$

Tasks

- **Tasks:** Equivalence relation $\mathcal{R} \subseteq (O \cup H) \times (O \cup H)$.

  Task: Any equivalence class of $\mathcal{R}$.

  Action determinism: For each state s and each task T, at most one action $a \in T$ is enabled in s.

  Resolves nondeterminism: In given state, next task specifies which action to execute.
- **Composition:** $\mathcal{R}_{\mathcal{A}_1 \| \mathcal{A}_2} = \mathcal{R}_{\mathcal{A}_1} \cup \mathcal{R}_{\mathcal{A}_2}$
- **Application:** Via *Task Schedules* - $\rho = T_1 T_2 \cdots$

Dining Cryptographers:

$$\mathcal{M} \parallel \mathcal{C}rypt_0 \parallel \mathcal{C}rypt_1 \parallel \mathcal{C}rypt_2$$

$$\mathcal{M} \;::\; Choose\_payer \parallel Inform\_Crypt_0 \parallel Inform\_Crypt_1 \parallel Inform\_Crypt_2$$

Tasks

- **Tasks:** Equivalence relation $\mathcal{R} \subseteq (O \cup H) \times (O \cup H)$.

  Task: Any equivalence class of $\mathcal{R}$.

  Action determinism: For each state s and each task T, at most one action $a \in T$ is enabled in s.

  Resolves nondeterminism: In given state, next task specifies which action to execute.
- **Composition:** $\mathcal{R}_{\mathcal{A}_1 \| \mathcal{A}_2} = \mathcal{R}_{\mathcal{A}_1} \cup \mathcal{R}_{\mathcal{A}_2}$
- **Application:** Via *Task Schedules* - $\rho = T_1 T_2 \cdots$

Dining Cryptographers:

$$\mathcal{M} \parallel \mathcal{C}rypt_0 \parallel \mathcal{C}rypt_1 \parallel \mathcal{C}rypt_2$$

$$\mathcal{M} \;::\; Choose\_payer \parallel Inform\_Crypt_0 \parallel Inform\_Crypt_1 \parallel Inform\_Crypt_2$$

$$\mathcal{C}rypt_i \;::\; Flip\_coin_i \parallel Tell\_Crypt_{i+1} \parallel Input\_coin_{i-1} \parallel Compare_i \parallel Announce_i \parallel Read\_Announce_{i-1} \parallel Read\_Announce_{i+1}$$

PIOA Semantics

$\mathcal{A} = (S, s_0, I, O, H, D)$

Execution fragment:

$\alpha = s_1 a_1 s_2 a_2 \cdots$ with $s_{i+1} \in \mathrm{supp}(\mu_i)$ & $(s_i, a_i, \mu_i) \in D$

$\mathrm{Frags}^*(A) = \bigcup_n \{\alpha \in (S \times \mathrm{Act})^n \times S \mid \alpha \text{ finite execution fragment}\}$

PIOA Semantics

$\mathcal{A} = (S, s_0, I, O, H, D)$

Execution fragment:

$\alpha = s_1 a_1 s_2 a_2 \cdots$ with $s_{i+1} \in \mathrm{supp}(\mu_i)$ & $(s_i, a_i, \mu_i) \in D$

$\mathsf{Frags}^*(A) = \bigcup_n \{\alpha \in (S \times \mathsf{Act})^n \times S \mid \alpha \text{ finite execution fragment}\}$

$\mathsf{fs}(\alpha)$ – first state of α; $\mathsf{ls}(\alpha)$ – last state of α

$\mathsf{Execs}^*(\mathcal{A}) = \{\alpha \in \mathsf{Frags}^*(\mathcal{A}) \mid \mathsf{fs}(\alpha) = s_0\}$

PIOA Semantics

$\mathcal{A} = (S, s_0, I, O, H, D)$

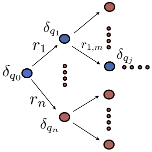
Execution fragment:

$\alpha = s_1 a_1 s_2 a_2 \cdots$ with $s_{i+1} \in \mathrm{supp}(\mu_i)$ & $(s_i, a_i, \mu_i) \in D$

$\mathsf{Frags}^*(A) = \bigcup_n \{\alpha \in (S \times \mathsf{Act})^n \times S \mid \alpha \text{ finite execution fragment}\}$

$\mathsf{fs}(\alpha)$ – first state of α; $\mathsf{ls}(\alpha)$ – last state of α

$\mathsf{Execs}^*(\mathcal{A}) = \{\alpha \in \mathsf{Frags}^*(\mathcal{A}) \mid \mathsf{fs}(\alpha) = s_0\}$



$\mathsf{Prob}(\alpha) = \prod_{i=0}^{n} \mu_{s_i, a_i}(s_{i+1})$

Which α should be used?

PIOA Semantics

$\mathcal{A} = (S, s_0, I, O, H, D)$

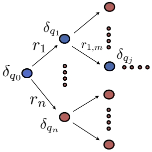
Execution fragment:

$\alpha = s_1 a_1 s_2 a_2 \cdots$ with $s_{i+1} \in \mathrm{supp}(\mu_i)$ & $(s_i, a_i, \mu_i) \in D$

$\mathsf{Frags}^*(A) = \bigcup_n \{\alpha \in (S \times \mathsf{Act})^n \times S \mid \alpha \text{ finite execution fragment}\}$

$\mathsf{fs}(\alpha)$ – first state of α; $\mathsf{ls}(\alpha)$ – last state of α

$\mathsf{Execs}^*(\mathcal{A}) = \{\alpha \in \mathsf{Frags}^*(\mathcal{A}) \mid \mathsf{fs}(\alpha) = s_0\}$



$\mathsf{Prob}(\alpha) = \prod_{i=0}^{n} \mu_{s_i, a_i}(s_{i+1})$

Which α should be used?

Apply task schedule $\rho = T_1 T_2 \cdots$

$$\delta_{s_0} \stackrel{a_{T_1}}{\longrightarrow} \sum_s \mu_{s_0 a_{T_1}}(s) \delta_{s_0 a_{T_1} s}$$

$$\stackrel{a_{T_2}}{\longrightarrow} \sum_s \mu_{s_0 a_{T_1}}(s) \left(\sum_{s'} \mu_{s, a_{T_2}}(s') \delta_{s_0 a_{T_1} s a_{T_2} s'} \right) \cdots$$

Domains

Frags($\mathcal{A}$) is a language from $(S \times \mathrm{Act})^\omega$

Domains

$\text{Frags}(\mathcal{A})$ is a language from $(S \times \text{Act})^\omega$

Chain-complete partial order under the prefix order.

$U \subseteq \text{Frags}(\mathcal{A})$ is *Scott open* if
$U = \uparrow A = \{\beta \mid (\exists \alpha \in A)\ \alpha \sqsubseteq \beta\}$ for some $A \subseteq \text{Frags}^*(\mathcal{A})$.

Domains

$\text{Frags}(\mathcal{A})$ is a language from $(S \times \text{Act})^\omega$

Chain-complete partial order under the prefix order.

$U \subseteq \text{Frags}(\mathcal{A})$ is *Scott open* if
$U = \uparrow A = \{\beta \mid (\exists \alpha \in A)\ \alpha \sqsubseteq \beta\}$ for some $A \subseteq \text{Frags}^*(\mathcal{A})$.

Ω – Borel algebra generated by Scott open sets.

$\text{Prob}(\text{Frags}(\mathcal{A}))$ also chain complete under

$$\mu \sqsubseteq \nu \quad \text{iff} \quad \mu(U) \leq \nu(U)\ (\forall U \text{ Scott open})$$

Domains

$\text{Frags}(\mathcal{A})$ is a language from $(S \times \text{Act})^\omega$

Chain-complete partial order under the prefix order.

$U \subseteq \text{Frags}(\mathcal{A})$ is *Scott open* if
$U = \uparrow A = \{\beta \mid (\exists \alpha \in A)\ \alpha \sqsubseteq \beta\}$ for some $A \subseteq \text{Frags}^*(\mathcal{A})$.

Ω – Borel algebra generated by Scott open sets.

$\text{Prob}(\text{Frags}(\mathcal{A}))$ also chain complete under

$$\mu \sqsubseteq \nu \quad \text{iff} \quad \mu(U) \leq \nu(U)\ (\forall U \text{ Scott open})$$

In particular, $\sum_{i \leq m} r_i \delta_{\alpha_i} \sqsubseteq \sum_{j \leq n} s_j \delta_{\beta_j}$ iff $(\exists t_{i,j} \geq 0)$ with

- $t_{i,j} > 0 \ \Rightarrow \ \alpha_i \sqsubseteq \beta_j$.
- $r_i = \sum_j t_{i,j}\ (\forall i)$.
- $\sum_i t_{i,j} \leq s_j\ (\forall j)$.

The Giry Monad

(X, Ω) measure space $\Rightarrow$ Prob(X, Ω) is a measure space.

So, Prob defines an endofunctor on Meas. This is in fact a monad.

$\eta\colon X \to \text{Prob}(X)$ by $\eta(x) = \delta_x$

$m\colon \text{Prob}(\text{Prob}(X)) \to \text{Prob}(X)$ is just integration.

The Giry Monad

(X, Ω) measure space $\Rightarrow$ Prob(X, Ω) is a measure space.

So, Prob defines an endofunctor on Meas. This is in fact a monad.

$\eta\colon X \to \mathrm{Prob}(X)$ by $\eta(x) = \delta_x$

$m\colon \mathrm{Prob}(\mathrm{Prob}(X)) \to \mathrm{Prob}(X)$ is just integration. For example,

$$m\left(\sum_s \mu_{s_0,a_{T_1}}(s)\left(\sum_{s'} \mu_{s,a_{T_2}}(s')\delta_{s_0 a_{T_1} s a_{T_2} s'}\right)\right) = \sum_{s,s'} \mu_{s_0,a_{T_1}}(s)\mu_{s,a_{T_2}}(s')\delta_{s_0 a_{T_1} s a_{T_2} s'}$$

Defining Apply

$\mathcal{A}$ - Task PIOA; T - task, $A_T = \{\alpha \in \mathsf{Frags}^*(\mathcal{A}) \mid T \text{ enabled in } \mathsf{ls}(\alpha)\}$

Defining Apply

$\mathcal{A}$ - Task PIOA; T - task, $A_T = \{\alpha \in \mathsf{Frags}^*(\mathcal{A}) \mid T \text{ enabled in } \mathsf{ls}(\alpha)\}$

$\mu \in \mathsf{Disc}(\mathsf{Frags}^*(\mathcal{A})) \Rightarrow \mu = \sum_\alpha \mu(\alpha)\delta_\alpha$, so define

$$\begin{aligned}\mathsf{Apply}(\mu, T) &= \sum_{\alpha \notin A_T} \mu(\alpha)\delta_\alpha \\ &\quad + \sum_{\alpha \in A_T} \mu(\alpha) \left(\sum_s \mu_{\mathsf{ls}(\alpha), a_T}(s)\delta_{\alpha a s} \right)\end{aligned}$$

Defining Apply

$\mathcal{A}$ - Task PIOA; T - task, $A_T = \{\alpha \in \mathsf{Frags}^*(\mathcal{A}) \mid T \text{ enabled in } \mathsf{ls}(\alpha)\}$

$\mu \in \mathsf{Disc}(\mathsf{Frags}^*(\mathcal{A})) \Rightarrow \mu = \sum_\alpha \mu(\alpha)\delta_\alpha$, so define

$$\begin{aligned}\mathsf{Apply}(\mu, T) &= \sum_{\alpha \notin A_T} \mu(\alpha)\delta_\alpha \\ &\quad + \sum_{\alpha \in A_T} \mu(\alpha) \left(\sum_s \mu_{\mathsf{ls}(\alpha), a_T}(s)\delta_{\alpha a s} \right)\end{aligned}$$

For $\rho = T_1 \cdots T_n$

$$\mathsf{Apply}(\mu, \rho) = \mathsf{Apply}(\mathsf{Apply}(\mu, T_1), T_2 \cdots T_n))$$

Defining Apply

$\mathcal{A}$ - Task PIOA; T - task, $A_T = \{\alpha \in \mathsf{Frags}^*(\mathcal{A}) \mid T \text{ enabled in } \mathsf{ls}(\alpha)\}$

$\mu \in \mathsf{Disc}(\mathsf{Frags}^*(\mathcal{A})) \Rightarrow \mu = \sum_\alpha \mu(\alpha)\delta_\alpha$, so define

$$\begin{aligned} \mathsf{Apply}(\mu, T) &= \sum_{\alpha \notin A_T} \mu(\alpha)\delta_\alpha \\ &\quad + \sum_{\alpha \in A_T} \mu(\alpha) \left(\sum_s \mu_{\mathsf{ls}(\alpha), a_T}(s)\delta_{\alpha a s} \right) \end{aligned}$$

For $\rho = T_1 \cdots T_n$

$$\mathsf{Apply}(\mu, \rho) = \mathsf{Apply}(\mathsf{Apply}(\mu, T_1), T_2 \cdots T_n))$$

Incomparable Fragments

μ has *incomparable fragments* if $\alpha, \beta \in \operatorname{supp} \mu$ implies α and β are incomparable initial segments in $\mathsf{Frags}^*(\mathcal{A})$.

μ has incomparable fragments $\Rightarrow$ $\mathsf{Apply}(\mu, T)$ has incomparable fragments.

Defining Apply (cont'd)

What about ρ infinite?

Defining Apply (cont'd)

What about ρ infinite?

$\alpha \leq \alpha' \in \mathsf{Frags}^*(\mathcal{A}) \;\Leftrightarrow\; \alpha \text{ prefix of } \alpha'; \quad \uparrow\alpha = \{\alpha' \mid \alpha \leq \alpha'\}$

$\mu \leq \nu \in \mathsf{Disc}(\mathsf{Frags}^*(\mathcal{A})) \;\Leftrightarrow\; \mu(\uparrow\alpha) \leq \nu(\uparrow\alpha)\ (\forall\alpha).$

Defining Apply (cont'd)

What about ρ infinite?

$\alpha \leq \alpha' \in \mathsf{Frags}^*(\mathcal{A}) \;\Leftrightarrow\; \alpha \text{ prefix of } \alpha'; \quad \uparrow\alpha = \{\alpha' \mid \alpha \leq \alpha'\}$

$\mu \leq \nu \in \mathsf{Disc}(\mathsf{Frags}^*(\mathcal{A})) \;\Leftrightarrow\; \mu(\uparrow\alpha) \leq \nu(\uparrow\alpha)\ (\forall\alpha)$.

$\mathsf{Apply}(T)\colon \mathsf{Disc}(\mathsf{Frags}^*(\mathcal{A})) \to \mathsf{Disc}(\mathsf{Frags}^*(\mathcal{A}))$ is *not* monotone,

But $\mu \leq \mathsf{Apply}(\mu, T)\ (\forall\mu)$.

Defining Apply (cont'd)

What about ρ infinite?

$\alpha \leq \alpha' \in \mathsf{Frags}^*(\mathcal{A}) \;\Leftrightarrow\; \alpha$ prefix of α'; $\;\uparrow\alpha = \{\alpha' \mid \alpha \leq \alpha'\}$

$\mu \leq \nu \in \mathsf{Disc}(\mathsf{Frags}^*(\mathcal{A})) \;\Leftrightarrow\; \mu(\uparrow\alpha) \leq \nu(\uparrow\alpha)\ (\forall\alpha)$.

$\mathsf{Apply}(T)\colon \mathsf{Disc}(\mathsf{Frags}^*(\mathcal{A})) \to \mathsf{Disc}(\mathsf{Frags}^*(\mathcal{A}))$ is *not* monotone,

But $\mu \leq \mathsf{Apply}(\mu, T)\ (\forall\mu)$.

So, $\rho = T_1 T_2 \cdots$ infinite implies $\{\mathsf{Apply}(\mu, T_1 \cdots T_n)\}_n$ increasing,

Defining Apply (cont'd)

What about ρ infinite?

$\alpha \leq \alpha' \in \mathsf{Frags}^*(\mathcal{A}) \;\Leftrightarrow\; \alpha$ prefix of α'; $\;\uparrow\alpha = \{\alpha' \mid \alpha \leq \alpha'\}$

$\mu \leq \nu \in \mathsf{Disc}(\mathsf{Frags}^*(\mathcal{A})) \;\Leftrightarrow\; \mu(\uparrow\alpha) \leq \nu(\uparrow\alpha)\ (\forall\alpha)$.

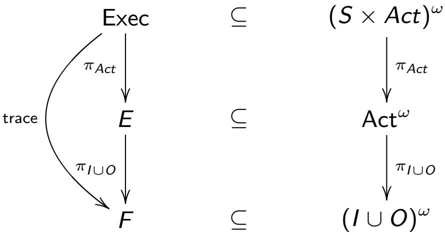
$\mathsf{Apply}(T)\colon \mathsf{Disc}(\mathsf{Frags}^*(\mathcal{A})) \to \mathsf{Disc}(\mathsf{Frags}^*(\mathcal{A}))$ is *not* monotone,

But $\mu \leq \mathsf{Apply}(\mu, T)\ (\forall\mu)$.

So, $\rho = T_1 T_2 \cdots$ infinite implies $\{\mathsf{Apply}(\mu, T_1 \cdots T_n)\}_n$ increasing,

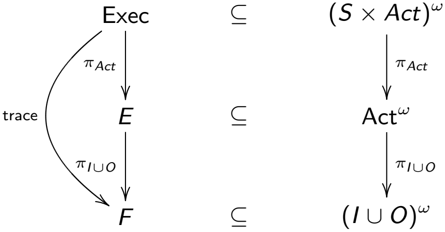
Hence $\mathsf{Apply}(\mu, \rho) = \sup_n \mathsf{Apply}(\mu, T_1 \cdots T_n)$ is well-defined.

Semantics of Observable Events



where $E = \{\alpha|_{\mathrm{Act}^\omega} \mid \alpha \in \mathrm{Exec}\}$ and $F = \{\alpha|_{(I\cup O)^\omega} \mid \alpha \in \mathrm{Exec}\}$

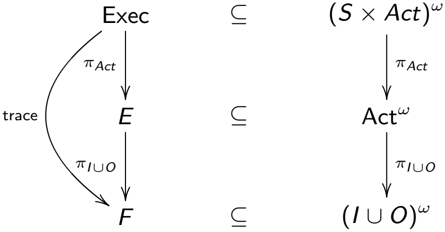
Semantics of Observable Events



where $E = \{\alpha|_{\mathsf{Act}^\omega} \mid \alpha \in \mathsf{Exec}\}$ and $F = \{\alpha|_{(I\cup O)^\omega} \mid \alpha \in \mathsf{Exec}\}$

For Task PIOA $\mathcal{A}$, $\mathsf{tdist}(\mathcal{A}) = \{\mathsf{trace}(\mathsf{Apply}(\delta_{s_0}, \rho)) \mid \rho \text{ task schedule}\}$

Semantics of Observable Events

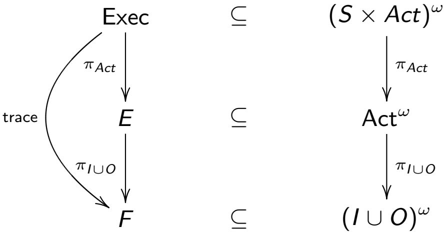


where $E = \{\alpha|_{\mathrm{Act}^\omega} \mid \alpha \in \mathrm{Exec}\}$ and $F = \{\alpha|_{(I \cup O)^\omega} \mid \alpha \in \mathrm{Exec}\}$

For Task PIOA $\mathcal{A}$, $\mathrm{tdist}(\mathcal{A}) = \{\mathrm{trace}(\mathrm{Apply}(\delta_{s_0}, \rho)) \mid \rho \text{ task schedule}\}$

Given $\mathcal{A}$ and $\rho = T_1 T_2 \cdots$, we'd like to have a *task scheduler* – determined in advance – that would represent applying ρ to any $\mu \in \mathrm{Disc}(\mathrm{Frags}^*(A))$.

Semantics of Observable Events



where $E = \{\alpha|_{\mathsf{Act}^\omega} \mid \alpha \in \mathsf{Exec}\}$ and $F = \{\alpha|_{(I \cup O)^\omega} \mid \alpha \in \mathsf{Exec}\}$

For Task PIOA $\mathcal{A}$, $\mathsf{tdist}(\mathcal{A}) = \{\mathsf{trace}(\mathsf{Apply}(\delta_{s_0}, \rho)) \mid \rho \text{ task schedule}\}$

Given $\mathcal{A}$ and $\rho = T_1 T_2 \cdots$, we'd like to have a *task scheduler* – determined in advance – that would represent applying ρ to any $\mu \in \mathsf{Disc}(\mathsf{Frags}^*(A))$.

Schedulers

A *task scheduler* is a map

$$\sigma \colon \mathsf{Frags}^*(\mathcal{A}) \to \mathbb{V}(\mathsf{Act}) = \{\mu \mid \mu \text{ subprobability measure}\}$$

Measures from Schedulers

Let $\mathcal{A}$ be a task PIOA and let $\sigma\colon \mathsf{Frags}^*(\mathcal{A}) \to \mathbb{V}(\mathsf{Act})$ be a task scheduler. If $\alpha \in \mathsf{Frags}^*(\mathcal{A})$, we define

$$\epsilon_{\sigma,\alpha} = (1 - ||\sigma(\alpha)||)\delta_\alpha + \sum_{a \in \mathsf{Act}} \sigma(\alpha)(a) \left(\sum_{s} \mu_{\alpha,a}(s) \epsilon_{\sigma,\alpha a s} \right)$$

Measures from Schedulers

Let $\mathcal{A}$ be a task PIOA and let $\sigma\colon \mathsf{Frags}^*(\mathcal{A}) \to \mathbb{V}(\mathsf{Act})$ be a task scheduler. If $\alpha \in \mathsf{Frags}^*(\mathcal{A})$, we define

$$\epsilon_{\sigma,\alpha} = (1 - ||\sigma(\alpha)||)\delta_\alpha + \sum_{a \in \mathsf{Act}} \sigma(\alpha)(a) \left(\sum_s \mu_{\alpha,a}(s)\epsilon_{\sigma,\alpha a s} \right)$$

$$\epsilon_{\sigma,\mu} = \sum_{\alpha \in \mathrm{supp}\,\mu} \mu(\alpha)\epsilon_{\sigma,\alpha}$$

.

Measures from Schedulers

Let $\mathcal{A}$ be a task PIOA and let $\sigma\colon \mathsf{Frags}^*(\mathcal{A}) \to \mathbb{V}(\mathsf{Act})$ be a task scheduler. If $\alpha \in \mathsf{Frags}^*(\mathcal{A})$, we define

$$\epsilon_{\sigma,\alpha} = (1 - ||\sigma(\alpha)||)\delta_\alpha + \sum_{a\in \mathsf{Act}} \sigma(\alpha)(a) \left(\sum_s \mu_{\alpha,a}(s)\epsilon_{\sigma,\alpha a s} \right)$$

$$\epsilon_{\sigma,\mu} = \sum_{\alpha\in \mathrm{supp}\,\mu} \mu(\alpha)\epsilon_{\sigma,\alpha}$$

.

Fact:

$$\begin{aligned} \epsilon_{\sigma,\alpha,0} &= \delta_\alpha \\ \epsilon_{\sigma,\alpha,n+1} &= (1 - ||\sigma(\alpha)||)\delta_\alpha + \sum_{a\in \mathsf{Act}} \sigma(\alpha)(a) \left(\sum_s \mu_{\alpha,a}(s)\epsilon_{\sigma,\alpha a s,n} \right) \end{aligned}$$

implies $\epsilon_{\sigma,\alpha} = \sup_n \epsilon_{\sigma,\alpha,n}$ extends to infinite α.

Schedulers vs. Task Schedules

Theorem

Let $\mu \in \mathsf{Disc}(\mathsf{Frags}^*(A))$ have support consisting of incomparable fragments, and let ρ be a task schedule. Then there is a scheduler $\sigma_\rho \colon \mathsf{Frags}^*(\mathcal{A}) \to \mathbb{V}(\mathsf{Act})$ such that $\mathsf{Apply}(\mu, \rho) = \epsilon_{\sigma_\rho, \mu}$.

Schedulers vs. Task Schedules

Theorem

Let $\mu \in \mathsf{Disc}(\mathsf{Frags}^*(A))$ have support consisting of incomparable fragments, and let ρ be a task schedule. Then there is a scheduler $\sigma_\rho \colon \mathsf{Frags}^*(\mathcal{A}) \to \mathbb{V}(\mathsf{Act})$ such that $\mathsf{Apply}(\mu, \rho) = \epsilon_{\sigma_\rho, \mu}$.

In fact, for $\mu = \sum_\alpha \mu(\alpha)\delta_\alpha$ and $\rho = T\rho'$, the scheduler σ_ρ is deterministic:

$$\sigma_\rho(\alpha) = \begin{cases} \delta_{a_{\mathsf{ls}(\alpha),T}} & \text{if } \alpha \in A_T, \\ \sigma_{\rho'}(\alpha) & \text{if } \sigma_{\rho'}(\alpha) \neq 0 \ \& \ \alpha \notin A_T, \\ 0 & \text{otherwise.} \end{cases}$$

$$\rho = T_1 T_2 \cdots \text{ infinite implies } \sigma_\rho = \cup_n \sigma_{T_1 \cdots T_n}.$$

Schedulers vs. Task Schedules

Theorem

Let $\mu \in \mathsf{Disc}(\mathsf{Frags}^*(A))$ have support consisting of incomparable fragments, and let ρ be a task schedule. Then there is a scheduler $\sigma_\rho \colon \mathsf{Frags}^*(\mathcal{A}) \to \mathbb{V}(\mathsf{Act})$ such that $\mathsf{Apply}(\mu, \rho) = \epsilon_{\sigma_\rho, \mu}$.

Corollary Let $\mathcal{A}$ be a Task PIOA.

- For each task schedule ρ, there is a deterministic task scheduler σ_ρ satisfying
$$\mathsf{Apply}(\delta_{s_0}, \rho) = \epsilon_{\sigma_\rho, \delta_{s_0}}.$$

Schedulers vs. Task Schedules

Theorem

Let $\mu \in \mathsf{Disc}(\mathsf{Frags}^*(A))$ have support consisting of incomparable fragments, and let ρ be a task schedule. Then there is a scheduler $\sigma_\rho \colon \mathsf{Frags}^*(\mathcal{A}) \to \mathbb{V}(\mathsf{Act})$ such that $\mathsf{Apply}(\mu, \rho) = \epsilon_{\sigma_\rho, \mu}$.

Corollary Let $\mathcal{A}$ be a Task PIOA.

- For each task schedule ρ, there is a deterministic task scheduler σ_ρ satisfying
$$\mathsf{Apply}(\delta_{s_0}, \rho) = \epsilon_{\sigma_\rho, \delta_{s_0}}.$$
- $\mathsf{tdist}(\mathcal{A}) \subseteq \{\mathsf{trace}(\epsilon_{\sigma, \delta_{s_0}}) \mid \sigma \text{ deterministic scheduler}\}$.

Expansions and Monads

$(X, \Sigma_X), (Y, \Sigma_Y)$ measure spaces, $R \subseteq X \times Y$. The *lift of* R is $\widehat{R} \subseteq \mathbb{V}X \times \mathbb{V}Y$ defined by

$$(\sum_x r_x \delta_x, \sum_y s_y \delta_y) \in \widehat{R} \;\Leftrightarrow\; \exists t\colon X \times Y \to [0,1] \text{ with}$$

- $r_x = \sum_y t(x,y)\ (\forall x)$
- $\sum_x t(x,y) \leq s_y\ (\forall y)$
- $t(x,y) > 0 \;\Rightarrow\; (x,y) \in R$

Expansions and Monads

$(X, \Sigma_X), (Y, \Sigma_Y)$ measure spaces, $R \subseteq X \times Y$. The *lift of* R is $\widehat{R} \subseteq \mathbb{V}X \times \mathbb{V}Y$ defined by

$$(\sum_x r_x \delta_x, \sum_y s_y \delta_y) \in \widehat{R} \;\Leftrightarrow\; \exists t\colon X \times Y \to [0,1] \text{ with}$$

- $r_x = \sum_y t(x,y)$ $(\forall x)$
- $\sum_x t(x,y) \leq s_y$ $(\forall y)$
- $t(x,y) > 0 \;\Rightarrow\; (x,y) \in R$

But, (X, Σ_X) a measure space implies $\mathbb{V}X$ is a measure space
And $R \subseteq \mathbb{V}X \times \mathbb{V}Y$ implies $\widehat{R} \subseteq \mathbb{V}(\mathbb{V}X) \times \mathbb{V}(\mathbb{V}Y)$

Expansions and Monads

$(X,\Sigma_X),(Y,\Sigma_Y)$ measure spaces, $R \subseteq X \times Y$. The *lift of R* is $\widehat{R} \subseteq \mathbb{V}X \times \mathbb{V}Y$ defined by

$$(\textstyle\sum_x r_x\delta_x, \sum_y s_y\delta_y) \in \widehat{R} \;\Leftrightarrow\; \exists t\colon X \times Y \to [0,1] \text{ with}$$

- $r_x = \sum_y t(x,y)\ (\forall x)$
- $\sum_x t(x,y) \leq s_y\ (\forall y)$
- $t(x,y) > 0 \;\Rightarrow\; (x,y) \in R$

But, (X,Σ_X) a measure space implies $\mathbb{V}X$ is a measure space
And $R \subseteq \mathbb{V}X \times \mathbb{V}Y$ implies $\widehat{R} \subseteq \mathbb{V}(\mathbb{V}X) \times \mathbb{V}(\mathbb{V}Y)$

$(\mu,\nu) \in \mathcal{E}(R) \Leftrightarrow (\exists(\mu',\nu') \in \widehat{R})\ \mu = \int d\mu' \wedge\ \nu = \int d\mu'$

Expansions and Monads

$(X, \Sigma_X), (Y, \Sigma_Y)$ measure spaces, $R \subseteq X \times Y$. The *lift of R* is $\widehat{R} \subseteq \mathbb{V}X \times \mathbb{V}Y$ defined by

$$(\sum_x r_x\delta_x, \sum_y s_y\delta_y) \in \widehat{R} \iff \exists t\colon X \times Y \to [0,1] \text{ with}$$

- $r_x = \sum_y t(x,y)\ (\forall x)$
- $\sum_x t(x,y) \leq s_y\ (\forall y)$
- $t(x,y) > 0 \ \Rightarrow\ (x,y) \in R$

But, (X, Σ_X) a measure space implies $\mathbb{V}X$ is a measure space
And $R \subseteq \mathbb{V}X \times \mathbb{V}Y$ implies $\widehat{R} \subseteq \mathbb{V}(\mathbb{V}X) \times \mathbb{V}(\mathbb{V}Y)$

$$(\mu, \nu) \in \mathcal{E}(R) \Leftrightarrow (\exists(\mu', \nu') \in \widehat{R})\ \mu = \int d\mu' \wedge\ \nu = \int d\mu'$$

$\mathbb{V}\colon$ Meas $\to$ Meas is a *monad*, and $R \mapsto \mathcal{E}(R)$ utilizes the *lifting* and *multiplication* of the monad.

Expansions and Monads

$(X, \Sigma_X), (Y, \Sigma_Y)$ measure spaces, $R \subseteq X \times Y$. The *lift of R* is $\widehat{R} \subseteq \mathbb{V}X \times \mathbb{V}Y$ defined by

$$(\sum_x r_x\delta_x, \sum_y s_y\delta_y) \in \widehat{R} \;\Leftrightarrow\; \exists t\colon X \times Y \to [0,1] \text{ with}$$

- $r_x = \sum_y t(x,y)\ (\forall x)$
- $\sum_x t(x,y) \le s_y\ (\forall y)$
- $t(x,y) > 0 \;\Rightarrow\; (x,y) \in R$

But, (X, Σ_X) a measure space implies $\mathbb{V}X$ is a measure space
And $R \subseteq \mathbb{V}X \times \mathbb{V}Y$ implies $\widehat{R} \subseteq \mathbb{V}(\mathbb{V}X) \times \mathbb{V}(\mathbb{V}Y)$

$(\mu, \nu) \in \mathcal{E}(R) \Leftrightarrow (\exists(\mu', \nu') \in \widehat{R})\ \mu = \int d\mu' \wedge\ \nu = \int d\mu'$

$\mathbb{V}$: Meas $\to$ Meas is a *monad*, and $R \mapsto \mathcal{E}(R)$ utilizes the *lifting* and *multiplication* of the monad. In particular, if $R \subseteq \text{Disc}(\text{Frags}(\mathcal{A})) \times \text{Disc}(\text{Frags}(\mathcal{A}))$, define $\mathcal{E}(R) \subseteq \text{Disc}(\text{Frags}(\mathcal{A})) \times \text{Disc}(\text{Frags}(\mathcal{A}))$ by

$$\begin{aligned}\mathcal{E}(R) &= (m \times m)(\widehat{R})\\ &= \{\langle \mu.\nu\rangle \mid (\exists p_i)(\exists \langle \mu_i, \nu_i\rangle \in R)\ \mu = \sum p_i\mu_i \ \&\ \nu = \sum p_i\nu_i\}\end{aligned}$$

Simulations

Let $(\mathcal{A}_i, \mathcal{R}_i)$ be two task PIOAs and let $f\colon \mathcal{R}_1^* \times \mathcal{R}_1 \to \mathcal{R}_2^*$ be a function. We define $\mathrm{full}(f)\colon \mathcal{R}_1^* \to \mathcal{R}_2^*$ by

$$\begin{aligned} \mathrm{full}(f)(\langle\ \rangle) &= \langle\ \rangle \\ \mathrm{full}(f)(\rho T) &= \mathrm{full}(f)(\rho)\hat{}f(\rho, T) \end{aligned}$$

Simulations

Let $(\mathcal{A}_i, \mathcal{R}_i)$ be two task PIOAs and let $f\colon \mathcal{R}_1^* \times \mathcal{R}_1 \to \mathcal{R}_2^*$ be a function. We define $\mathrm{full}(f)\colon \mathcal{R}_1^* \to \mathcal{R}_2^*$ by

$$\begin{aligned} \mathrm{full}(f)(\langle\,\rangle) &= \langle\,\rangle \\ \mathrm{full}(f)(\rho T) &= \mathrm{full}(f)(\rho)\hat{\ } f(\rho, T) \end{aligned}$$

$R \subseteq \mathrm{Disc}(\mathrm{Exec}(\mathcal{A}_1)) \times \mathrm{Disc}(\mathrm{Exec}(\mathcal{A}_2))$ is a *simulation* if

- $(\mu_1, \mu_2) \in R \;\Rightarrow\; \mathrm{tdist}(\mu_1) \subseteq \mathrm{tdist}(\mu_2)$
- $(\delta_{s_{0,1}}, \delta_{s_{0,2}}) \in R$
- $(\exists f : \mathcal{R}_1^* \times \mathcal{R}_1 \to \mathcal{R}_2^*)(\forall \rho \in \mathcal{R}_1^*)(\forall T \in \mathcal{R}_1)$

$$\begin{aligned} &(\mu_1, \mu_2) \in R \;\wedge\; \mathrm{supp}(\mu_1) \subseteq \rho \;\wedge\; \mathrm{supp}(\mu_2) \subseteq \mathrm{full}(f)(\rho) \\ &\quad\Rightarrow\; \langle \mathrm{Apply}(\mu_1, T), \mathrm{Apply}(\mu_2, \mathrm{full}(f)(\rho, T)) \rangle \in \mathcal{E}(R) \end{aligned}$$

Simulations

Let $(\mathcal{A}_i, \mathcal{R}_i)$ be two task PIOAs and let $f\colon \mathcal{R}_1^* \times \mathcal{R}_1 \to \mathcal{R}_2^*$ be a function. We define $\text{full}(f)\colon \mathcal{R}_1^* \to \mathcal{R}_2^*$ by

$$\begin{aligned}\text{full}(f)(\langle\,\rangle) &= \langle\,\rangle \\ \text{full}(f)(\rho T) &= \text{full}(f)(\rho)\hat{\ } f(\rho, T)\end{aligned}$$

Theorem (Canetti, et al)
Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be comparable task-PIOAs that are closed and action-deterministic. If there exists a simulation relation from $\mathcal{A}_1$ to $\mathcal{A}_2$, then $\text{tdist}(\mathcal{A}_1) \subseteq \text{tdist}(\mathcal{A}_2)$.

Remember

$$\mathcal{M} \parallel \mathcal{C}rypt_0 \parallel \mathcal{C}rypt_1 \parallel \mathcal{C}rypt_2$$

$$\mathcal{M} \;::\; Choose\_payer \parallel Inform\_Crypt_0 \parallel Inform\_Crypt_1 \parallel Inform\_Crypt_2$$

$$\mathcal{C}rypt_i \;::\; Flip\_coin_i \parallel Tell\_Crypt_{i+1} \parallel Input\_coin_{i-1} \parallel Compare_i \parallel Announce_i \parallel Read\_Announce_{i-1} \parallel Read\_Announce_{i+1}$$

Remember

$$\mathcal{M} \parallel \mathcal{C}rypt_0 \parallel \mathcal{C}rypt_1 \parallel \mathcal{C}rypt_2$$

$\mathcal{M}$ :: *Choose_payer* $\parallel$ *Inform_Crypt*$_0$ $\parallel$ *Inform_Crypt*$_1$ $\parallel$ *Inform_Crypt*$_2$

$\mathcal{C}rypt_i$ :: *Flip_coin*$_i$ $\parallel$ *Tell_Crypt*$_{i+1}$ $\parallel$ *Input_coin*$_{i-1}$ $\parallel$ *Compare*$_i$
$\parallel$ *Announce*$_i$ $\parallel$ *Read_Announce*$_{i-1}$ $\parallel$ *Read_Announce*$_{i+1}$

Task Schedule:

Choose_payer; *Inform_Crypt*$_0$; *Inform_Crypt*$_1$; *Inform_Crypt*$_2$;
Flip_coin$_0$; *Flip_coin*$_1$; *Flip_coin*$_2$; *Tell_crypt*$_1$; *Tell_crypt*$_2$; *Tell_crypt*$_0$;
Compare$_0$; *Compare*$_1$; *Compare*$_2$; *Announce*$_0$; *Announce*$_1$; *Announce*$_2$

Remember

$$\mathcal{M} \parallel \mathcal{C}rypt_0 \parallel \mathcal{C}rypt_1 \parallel \mathcal{C}rypt_2$$

$\mathcal{M}$ :: $Choose\_payer \parallel Inform\_Crypt_0 \parallel Inform\_Crypt_1 \parallel Inform\_Crypt_2$

$\mathcal{C}rypt_i$ :: $Flip\_coin_i \parallel Tell\_Crypt_{i+1} \parallel Input\_coin_{i-1} \parallel Compare_i$
$\parallel Announce_i \parallel Read\_Announce_{i-1} \parallel Read\_Announce_{i+1}$

Task Schedule:

$Choose\_payer; Inform\_Crypt_0; Inform\_Crypt_1; Inform\_Crypt_2;$
$Flip\_coin_0; Flip\_coin_1; Flip\_coin_2; Tell\_crypt_1; Tell\_crypt_2; Tell\_crypt_0;$
$Compare_0; Compare_1; Compare_2; Announce_0; Announce_1; Announce_2$

$\mathcal{M}_i$ ::= Master who chooses $Crypt_i$

$$\mathcal{M} := \sum_i r_i \delta_{\mathcal{M}_i}, \qquad \sum_i r_i = 1$$

Dining Cryptographers

One approach:

$\mathcal{M}_i$ ::= Master who chooses Crypt_i

Show $(\forall i)(\forall \sigma)(\forall O)$

$$\mu_\sigma(\mathcal{M}_{i+1} \parallel \mathit{Crypto}_0 \parallel \mathit{Crypt}_1 \parallel \mathit{Crypt}_2))(O)|_{\mathit{Crypt}_i} =$$
$$\mu_\sigma(\mathcal{M}_{i+2} \parallel \mathit{Crypto}_0 \parallel \mathit{Crypt}_1 \parallel \mathit{Crypt}_2))(O)|_{\mathit{Crypt}_i}$$

Dining Cryptographers

One approach:

$\mathcal{M}_i$ ::= Master who chooses $Crypt_i$

Show $(\forall i)(\forall \sigma)(\forall O)$

$$\mu_\sigma(\mathcal{M}_{i+1} \parallel Crypto_0 \parallel Crypt_1 \parallel Crypt_2))(O)|_{Crypt_i} = \mu_\sigma(\mathcal{M}_{i+2} \parallel Crypto_0 \parallel Crypt_1 \parallel Crypt_2))(O)|_{Crypt_i}$$

Definition: (Barghava & Palamidessi)

Protocol satisfies *strong probabilistic anonymity* if

$$(\forall \sigma)(\forall O)\ \mu_\sigma(O \mid \mathcal{M}_i) = \mu_\sigma(O \mid \mathcal{M}_{i+1}) = \mu_\sigma(O \mid \mathcal{M}_{i+2})$$

Dining Cryptographers

One approach:

$\mathcal{M}_i$ ::= Master who chooses $Crypt_i$

Show $(\forall i)(\forall \sigma)(\forall O)$

$$\mu_\sigma(\mathcal{M}_{i+1} \parallel Crypto_0 \parallel Crypt_1 \parallel Crypt_2))(O)|_{Crypt_i} = \mu_\sigma(\mathcal{M}_{i+2} \parallel Crypto_0 \parallel Crypt_1 \parallel Crypt_2))(O)|_{Crypt_i}$$

Definition: (Barghava & Palamidessi)
Protocol satisfies *strong probabilistic anonymity* if

$$(\forall \sigma)(\forall O)\ \mu_\sigma(O \mid \mathcal{M}_i) = \mu_\sigma(O \mid \mathcal{M}_{i+1}) = \mu_\sigma(O \mid \mathcal{M}_{i+2})$$

But

$$\mu_\sigma(O \mid \mathcal{M}_i) = \frac{\mu_\sigma(O \wedge \mathcal{M}_i)}{\mu(\mathcal{M}_i)} = \mu_\sigma(O)$$

since the observation O is independent of which cryptographer $\mathcal{M}$ chooses.