Domain Theory and Task-structured Probabilistic Input/Output Automata

Michael W. Mislove¹

Joint Work With
Aaron Jaggard², Catherine Meadows³ and Roberto Segala⁴

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

FCC Workshop, Pittsburgh, June, 2008
Outline

- General Setting
Outline

- General Setting
- Task-structured Probabilistic Input/Output Automata
Outline

- General Setting
- Task-structured Probabilistic Input/Output Automata
  - Problems with original presentation
Outline

- General Setting
- Task-structured Probabilistic Input/Output Automata
  - Problems with original presentation
- Some domain theory
Outline

- General Setting
- Task-structured Probabilistic Input/Output Automata
  - Problems with original presentation
- Some domain theory
- Key results in hand
Outline

- General Setting
- Task-structured Probabilistic Input/Output Automata
  - Problems with original presentation
- Some domain theory
- Key results in hand
- Summary and Future work
Abstract Setting

- **Implementation**: faithful representation with adversary
- **Idealized simulation**: abstract, simplified representation
Abstract Setting

- **Implementation**: faithful representation with adversary
- **Idealized simulation**: abstract, simplified representation
  - Easier to reason about
  - Requires “obfuscating” idealized process to mimic real-world implementation

*Example*: $\mathcal{P}$ protocol; $\mathcal{F}$ simulation; $\text{Adv}$ adversary; $\mathcal{I}$ ideal adversary; $\mathcal{E}$ environment

$$\mathcal{P} \leq_s \mathcal{F} \iff (\forall \text{Adv})(\exists \mathcal{I})(\forall \mathcal{E}) \quad \mathcal{P}||\text{Adv}||\mathcal{E} \simeq \mathcal{F}||\mathcal{I}||\mathcal{E}$$
Abstract Setting

- **Implementation**: faithful representation with adversary
- **Idealized simulation**: abstract, simplified representation
  - Easier to reason about
  - Requires “obfuscating” idealized process to mimic real-world implementation

*Example*: $\mathcal{P}$ protocol; $\mathcal{F}$ simulation; $Adv$ adversary; $\mathcal{I}$ ideal adversary; $\mathcal{E}$ environment

$$\mathcal{P} \preceq_s \mathcal{F} \iff (\forall Adv)(\exists I)(\forall E) \quad \mathcal{P} \parallel Adv \parallel E \simeq \mathcal{F} \parallel I \parallel E$$

- $\preceq_s$: Trace distributions of observable events from $(\mathcal{P} \parallel Adv \parallel E)$ all appear in the trace distributions of $(\mathcal{F} \parallel I \parallel E)$. 
Separation of Concerns

- Semantic model: Devise model that accommodates concerns arising in protocol analysis
  - Spi-calculus, Probabilistic $\pi$-calculus, PPT-calculus, CSP,...
Separation of Concerns

- **Semantic model**: Devise model that accommodates concerns arising in protocol analysis
  - Spi-calculus, Probabilistic π-calculus, PPT-calculus, CSP, ...
  - Task-structured Probabilistic Input/Output Automata
Separation of Concerns

- **Semantic model**: Devise model that accommodates concerns arising in protocol analysis
  - Spi-calculus, Probabilistic $\pi$-calculus, PPT-calculus, CSP, ...
  - Task-structured Probabilistic Input/Output Automata

- **Mapping to the model**:
  1. Devise encoding of implementation, adversary and environment into model
  2. Devise encoding of idealized functionality, idealized adversary into model
  3. Use model to reason about relationship between (1) and (2)
**Separation of Concerns**

- **Semantic model:** Devise model that accommodates concerns arising in protocol analysis
  - Spi-calculus, Probabilistic $\pi$-calculus, PPT-calculus, CSP, ...
  - Task-structured Probabilistic Input/Output Automata

- **Mapping to the model:**
  1. Devise encoding of implementation, adversary and environment into model
  2. Devise encoding of idealized functionality, idealized adversary into model
  3. Use model to reason about relationship between (1) and (2)

- **Focus of talk:** Task-structured Probabilistic Input/Output Automata
Probabilistic Input/Output Automata

- Mathematical model of concurrent computation
  - *Input/Output*: used to model interactions among component processes
  - *Tasks*: Used to limit power of the adversary
Probabilistic Input/Output Automata

- Mathematical model of concurrent computation
  - Input/Output: used to model interactions among component processes
  - Tasks: Used to limit power of the adversary

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

- \( S \) - countable set of states, \( s_0 \) - start state
- \( \text{Act} ::= I \cup O \cup H \) - countable set of actions
- \( D \subseteq S \times \text{Act} \times \text{Prob}(S) \) transition relation satisfying:
Probabilistic Input/Output Automata

- Mathematical model of concurrent computation
  - **Input/Output**: used to model interactions among component processes
  - **Tasks**: Used to limit power of the adversary

\[ A = (S, s_0, I, O, H, D) \] where

- \( S \) - countable set of states, \( s_0 \) - start state
- \( \text{Act} ::= I \cup O \cup H \) - countable set of actions
- \( D \subseteq S \times \text{Act} \times \text{Prob}(S) \) transition relation satisfying:
  - **Transition determinism**: \( (s, a, \mu), (s, a, \nu) \in D \implies \mu = \nu \)
  - **Input enabling**: \( (\forall s \in S)(\forall a \in I)(\exists \mu) (s, a, \mu) \in D \)
How to compose PIOAs

\[ A_i = (S_i, s_{A_i}, l_i, O_i, H_i, D_i), \ i = 1, 2 \text{ are compatible if} \]

\[ \text{Act}_i \cap H_{i+1} = \emptyset = O_1 \cap O_2 \]
How to compose PIOAs

\( A_i = (S_i, s_{A_i}, I_i, O_i, H_i, D_i), \ i = 1, 2 \) are compatible if

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

\( A_1 \parallel A_2 = (S_{A_1,A_2}, s_{A_1,A_2}, I_{A_1,A_2}, O_{A_1,A_2}, H_{A_1,A_2}, D_{A_1,A_2}) \) where:

- \( S_{A_1,A_2} = S_1 \times S_2 \)
- \( I_{A_1,A_2} = (I_1 \cup I_2) \setminus (O_1 \cup O_2) \)
- \( O_{A_1,A_2} = O_1 \cup O_2 \)
- \( H_{A_1,A_2} = H_1 \cup H_2 \)
- \( D_{A_1,A_2} = \{ ((s_1, s_2), a, \mu_1 \times \mu_2) \mid (s_1, a, \mu_1) \in D_1 \ \text{or} \ (s_2, a, \mu_2) \in D_2 \}
  \text{and} \ \mu_i = \delta_{s_i} \text{ if } a \not\in \text{Act}_i \} \)
How to compose PIOAs

\( A_i = (S_i, s_{A_i}, I_i, O_i, H_i, D_i), \ i = 1, 2 \) are compatible if

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

\( A_1 \| A_2 = (S_{A_1,A_2}, s_{A_1,A_2}, I_{A_1,A_2}, O_{A_1,A_2}, H_{A_1,A_2}, D_{A_1,A_2}) \) where:

- \( S_{A_1,A_2} = S_1 \times S_2 \)
- \( I_{A_1,A_2} = (I_1 \cup I_2) \setminus (O_1 \cup O_2) \)
- \( O_{A_1,A_2} = O_1 \cup O_2 \)
- \( H_{A_1,A_2} = H_1 \cup H_2 \)
- \( D_{A_1,A_2} = \{((s_1, s_2), a, \mu_1 \times \mu_2) \mid (s_1, a, \mu_1) \in D_1 \text{ or } (s_2, a, \mu_2) \in D_2 \text{ and } \mu_i = \delta_{s_i} \text{ if } a \not\in \text{Act}_i\} \)

\( A \) is closed if \( I_A = \emptyset \)
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 $R \subseteq (O \cup H) \times (O \cup H)$.
  
  **Task:** Any equivalence class of $R$.

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

- **Role:** Reduce power of adversary
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 $s$.

- **Role**: *Reduce power of adversary*

  *Resolve 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 $s$.

- **Role**: *Reduce power of adversary*

  - **Resolve nondeterminism**: In given state, next task specifies which action to execute.

- **Composition**: $\mathcal{R}_{A_1 || A_2} = \mathcal{R}_{A_1} \cup \mathcal{R}_{A_2}$
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 $s$.

- **Role**: Reduce power of adversary

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

- **Composition**: $\mathcal{R}_{A_1 \parallel A_2} = \mathcal{R}_{A_1} \cup \mathcal{R}_{A_2}$

- **Application**: Via *task schedules* - $\rho = T_1 \cdot T_2 \cdots$
Example

- **Player:** Flips coin and announces result
- **Opponent:** Announces *Heads* or *Tails*
- If they match, *Player* wins, otherwise *Opponent* wins.

**Player:**
- States: $\bot$, flipped, announced, initially $\bot$
- Actions: Flip: Output: *Heads*, *Tails*
- P-Announce: *Heads*, *Tails*
- Tasks: \{Flip, P-Announce\}

**Opponent:**
- States: $\bot$, announced, initially $\bot$
- Actions: O-Announce-Heads, O-Anounce-Tails
- Tasks: \{O-Announce-Heads, O-Announce-Tails\}
Example

- **Player:** Flips coin and announces result
- **Opponent:** Announces *Heads* or *Tails*
- If they match, *Player* wins, otherwise *Opponent* wins.

- **Player:**
  - States: \( \bot \), flipped, announced, initially \( \bot \)
  - Actions: Flip: Output: *Heads*, *Tails*
  - P-Announce: *Heads*, *Tails*
  - Tasks: \{Flip, P-Announce\}

- **Opponent:**
  - States: \( \bot \), announced, initially \( \bot \)
  - Actions: O-Announce-Heads, O-Anounce-Tails
  - Tasks: \{O-Announce-Heads, O-Announce-Tails\}

- **Task Schedules:**
  - \( \text{Flip, P-Announce, O-Announce-Heads} \)
  - \( \text{Flip, P-Announce, O-Announce-Tails} \)
  - \( \text{Flip, O-Announce-Heads, P-Announce} \)
  - \( \text{Flip, O-Announce-Tails, P-Announce} \)
  - \( \text{O-Announce-Heads, Flip, P-Announce} \)
  - \( \text{O-Announce-Tails, Flip, P-Announce} \)
### PIOA Semantics

\[ A = (S, s_0, I, O, H, D) \]

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

Execution fragment:

\[ \alpha = s_1 a_1 s_2 a_2 \cdots \text{ with } s_{i+1} \in \text{supp}(\mu_i) \& (s_i, a_i, \mu_i) \in D \]
PIOA Semantics

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

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

Execution fragment:
\[ \alpha = s_1a_1s_2a_2 \cdots \text{ with } s_{i+1} \in \text{supp}(\mu_i) \land (s_i, a_i, \mu_i) \in D \]

\( \text{fs}(\alpha) \) – first state of \( \alpha \); \( \text{ls}(\alpha) \) – last state of \( \alpha \)

\( \text{Execs}^*(\mathcal{A}) = \{ \alpha \in \text{Frags}^*(\mathcal{A}) \mid \text{fs}(\alpha) = s_0 \} \)
**PIOA Semantics**

\[ A = (S, s_0, I, O, H, D) \]

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

Execution fragment:

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

\[ \text{fs}(\alpha) - \text{first state of } \alpha; \quad \text{ls}(\alpha) - \text{last state of } \alpha \]

\[ \text{Execs}^*(A) = \{ \alpha \in \text{Frags}^*(A) \mid \text{fs}(\alpha) = s_0 \} \]

\[ \text{Prob}(\alpha) = \prod_{i=0}^{n} \mu_{q_i,a_i}(q_{i+1}) \]

Which \( \alpha \) should be used?
### PIOA Semantics

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

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

Execution fragment:

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

fs(\(\alpha\)) – first state of \(\alpha\);  \(\text{ls}(\alpha)\) – last state of \(\alpha\)

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

\[ \text{Prob}(\alpha) = \prod_{i=0}^{n} \mu_{q_i,a_i}(q_{i+1}) \]

Which \(\alpha\) should be used?

Apply task schedule \(\rho = T_1 T_2 \cdots \)

\[ \delta_{q_0} \xrightarrow{T_1} \sum_q \mu_{q_0,a_{T_1}}(q) \delta_{q_0 a_{T_1}} q \xrightarrow{T_2} \cdots \]
Defining Apply

\( \mathcal{A} \) - Task PIOA; \( T \) - task

\[ A_T = \{ \alpha \in \text{Frags}^*(\mathcal{A}) \mid T \text{ enabled in } !s(\alpha) \} \]
**Defining Apply**

Let $A$ be a Task PIOA; $T$ a task. Then $A_T = \{ \alpha \in \text{Frags}^*(A) \mid T \text{ enabled in } \text{ls} (\alpha) \}$

If $\mu \in \text{Prob} (\text{Frags}^*(A))$, then $\mu = \sum \mu(\alpha) \delta_{\alpha}$, so define:

$$\text{Apply}(\mu, T) = \sum_{\alpha \notin A_T} \mu(\alpha) \delta_{\alpha} + \sum_{\alpha \in A_T} \mu(\alpha) \left( \sum_s \mu_{\text{ls}(\alpha), a_T}(s) \delta_{\alpha a s} \right)$$
Defining Apply

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

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

\[
\text{Apply}(\mu, T) = \sum_{\alpha \notin A_T} \mu (\alpha) \delta_\alpha \\
+ \sum_{\alpha \in A_T} \mu (\alpha) \left( \sum_s \mu |_{\text{ls}(\alpha), a_T (s)} (s) \delta_\alpha as \right)
\]

For \( \rho = T_1 \cdots T_n \)

\[
\text{Apply}(\mu, \rho) = \text{Apply}(\text{Apply}(\mu, T_1), T_2 \cdots T_n))
\]
Defining Apply (cont’d)

What about $\rho$ infinite?
### Defining Apply (cont’d)

**What about \( \rho \) infinite?**

\[
\alpha \leq \alpha' \in \text{Frags}^*(A) \iff \alpha \text{ prefix of } \alpha'; \quad \uparrow \alpha = \{ \alpha' \mid \alpha \leq \alpha' \}
\]

\[
\mu \leq \nu \in \text{Prob}(\text{Frags}^*(A)) \iff \mu(\uparrow \alpha) \leq \nu(\uparrow \alpha) \quad (\forall \alpha).
\]
Defining Apply (cont’d)

What about $\rho$ infinite?

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

$\mu \leq \nu \in \text{Prob}(\text{Frags}^*(A)) \iff \mu(\uparrow \alpha) \leq \nu(\uparrow \alpha) \ (\forall \alpha)$.  

Apply($T$): $\text{Prob}(\text{Frags}^*(A)) \rightarrow \text{Prob}(\text{Frags}^*(A))$ is not monotone,

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

Defining Apply (cont’d)

What about $\rho$ infinite?

$\alpha \leq \alpha' \in \text{Frags}^*(A) \iff \alpha \text{ prefix of } \alpha'; \quad \uparrow \alpha = \{ \alpha' | \alpha \leq \alpha' \}$

$\mu \leq \nu \in \text{Prob}(\text{Frags}^*(A)) \iff \mu(\uparrow \alpha) \leq \nu(\uparrow \alpha) \ (\forall \alpha)$.

Apply$(T): \text{Prob}(\text{Frags}^*(A)) \rightarrow \text{Prob}(\text{Frags}^*(A))$ is not monotone,

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

So, $\rho = T_1 T_2 \cdots$ infinite implies $\{\text{Apply}(\mu, T_1 \cdots T_n)\}_n$ increasing,
Defining Apply (cont’d)

What about $\rho$ infinite?

$\alpha \leq \alpha' \in \text{Ffrags}^*(A) \iff \alpha$ prefix of $\alpha'$; $\uparrow \alpha = \{\alpha' \mid \alpha \leq \alpha'\}$

$\mu \leq \nu \in \text{Prob}^{\text{(Ffrags}^*(A))} \iff \mu(\uparrow \alpha) \leq \nu(\uparrow \alpha)$ (\forall \alpha).

Apply($T$): $\text{Prob}^{\text{(Ffrags}^*(A))} \rightarrow \text{Prob}^{\text{(Ffrags}^*(A))}$ is not monotone,

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

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

Hence $\text{Apply}(\mu, \rho) = \sup_n \text{Apply}(\mu, T_1 \cdots T_n)$ is well-defined.
Semantics of Observable Events

\[
\begin{align*}
\text{Exec} & \subseteq (S \times \text{Act})^\omega \\
E & \subseteq \text{Act}^\omega \\
F & \subseteq (I \cup O)^\omega
\end{align*}
\]

where \( E = \{ \alpha|_{\text{Act}^\omega} \mid \alpha \in \text{Exec} \} \) and \( F = \{ \alpha|_{(I \cup O)}^\omega \mid \alpha \in \text{Exec} \} \)
Semantics of Observable Events

\[
\begin{align*}
\text{Exec} & \subseteq (S \times \text{Act})^\omega \\
E & \subseteq \text{Act}^\omega \\
F & \subseteq (I \cup O)^\omega
\end{align*}
\]

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

For Task PIOA \( \mathcal{A} \), \( \text{tdist}(\mathcal{A}) = \{ \text{trace}(\text{Apply}(\delta_{s_0}, \rho)) \mid \rho \text{ task schedule} \} \)
Semantics of Observable Events

\[
\text{Exec} \subseteq (S \times \text{Act})^\omega \\
\pi_{\text{Act}}
\]

\[
\text{trace}
\]

\[
E \subseteq \text{Act}^\omega \\
\pi_{\text{I} \cup \text{O}}
\]

\[
F \subseteq (\text{I} \cup \text{O})^\omega \\
\pi_{\text{I} \cup \text{O}}
\]

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

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

Given \( A \) and \( \rho = T_1 \cdots \), we’d like to have a scheduler – determined in advance – that would represent applying \( \rho \) to any \( \mu \in \text{Prob}((\text{Frags}^\star(A))) \).
Task Schedulers

A task scheduler is a map

\[ \sigma : \text{Frags}^*(\mathcal{A}) \rightarrow \mathcal{V}(\text{Act}) = \{ \mu \mid \mu \text{ subprobability measure} \} \]
Task Schedulers

A task scheduler is a map

$$\sigma : \text{Frags}^*(A) \rightarrow \mathbb{V}(\text{Act}) = \{\mu \mid \mu \text{ subprobability measure}\}$$

Measures from Schedulers - Original Definition

If $$\sigma : \text{Frags}^*(A) \rightarrow \mathbb{V}(\text{Act})$$ is a scheduler and $$\alpha \in \text{Frags}^*(A)$$ then define $$\epsilon_\sigma : \text{Prob}((\text{Frags}^*(A)) \rightarrow \text{Prob}((\text{Frags}^*(A))$$ by

$$\epsilon_{\sigma, \alpha} (\uparrow \alpha') = \begin{cases} 0 & \text{if } \alpha' \not\leq \alpha \not\leq \alpha' \\ 1 & \text{if } \alpha' \leq \alpha \\ \epsilon_{\sigma, \alpha} (\uparrow \alpha'') \sigma(\alpha'')(a)\mu_{\alpha'', a}(s) & \text{if } \alpha \leq \alpha' = \alpha'' \text{as,} \end{cases}$$

where $$\mu_{\alpha'', a}(s)$$ is the probability of landing in state $$s$$ starting from $$\text{ls}(\alpha)$$ after executing action $$a$$. 
Measures from Schedulers - Our Definition

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

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

Then, $\epsilon'_{\sigma,\alpha} = \epsilon_{\sigma,\alpha}$ for all schedulers $\sigma$ and $\alpha \in \text{Frags}^*(\mathcal{A})$. 
### Measures from Schedulers - Our Definition

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

$$
\epsilon'_{\sigma, \alpha} = (1 - \|\sigma(\alpha)\|) \delta_{\alpha} + \sum_{a \in \text{Act}} \sigma(\alpha)(a) \left( \sum_{s} \mu_{\alpha, a}(s) \epsilon'_{\sigma, \alpha s} \right)
$$

Then, $\epsilon'_{\sigma, \alpha} = \epsilon_{\sigma, \alpha}$ for all schedulers $\sigma$ and $\alpha \in \text{Frags}^*(\mathcal{A})$.

**Neat fact:**

$$
\begin{align*}
\epsilon'_{\sigma, \alpha, 0} &= \delta_{\alpha} \\
\epsilon'_{\sigma, \alpha, n+1} &= (1 - \|\sigma(\alpha)\|) \delta_{\alpha} + \sum_{a \in \text{Act}} \sigma(\alpha)(a) \left( \sum_{s} \mu_{\alpha, a}(s) \epsilon'_{\sigma, \alpha as, n} \right)
\end{align*}
$$

implies $\epsilon'_{\sigma, \alpha} = \sup_n \epsilon'_{\sigma, \alpha, n}$
Schedulers vs. Task Schedules

Theorem

Let $\mu \in \text{Prob}(\text{Frags}^*(\mathcal{A}))$ have support consisting of incomparable fragments, and let $\rho$ be a task schedule. Then there is a scheduler $\sigma_\rho : \text{Frags}^*(\mathcal{A}) \rightarrow \mathbb{V}(\text{Act})$ such that $\text{Apply}(\mu, \rho) = \epsilon_{\sigma_\rho, \mu}$. 
Schedulers vs. Task Schedules

**Theorem**

Let \( \mu \in \text{Prob}(\text{Frags}^*(A)) \) have support consisting of incomparable fragments, and let \( \rho \) be a task schedule. Then there is a scheduler \( \sigma_\rho : \text{Frags}^*(A) \rightarrow \text{V}(\text{Act}) \) such that \( \text{Apply}(\mu, \rho) = \epsilon_{\sigma_\rho, \mu} \).

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

\[
\sigma_\rho(\alpha) = \begin{cases} 
\delta_{\text{is}(\alpha), T} & \text{if } \alpha \in A_T \cap \text{supp} \mu, \\
\sigma_{\rho'}(\alpha) & \text{if } \sigma_{\rho'}(\alpha) \neq 0, \\
0 & \text{otherwise.}
\end{cases}
\]

\( \rho = T_1 \cdots \) infinite implies \( \sigma_\rho = \bigcup_n \sigma_{T_1 \cdots T_n} \).
Schedulers vs. Task Schedules

**Theorem**
Let $\mu \in \text{Prob}(\text{Frags}^*(A))$ have support consisting of incomparable fragments, and let $\rho$ be a task schedule. Then there is a scheduler $\sigma_\rho : \text{Frags}^*(A) \rightarrow \mathbb{V}(\text{Act})$ such that $\text{Apply}(\mu, \rho) = \epsilon_{\sigma_\rho, \mu}$.

**Corollary** Let $A$ be a Task PIOA.

- For each task schedule $\rho$, there is a deterministic task scheduler $\sigma_\rho$ satisfying

\[
\text{Apply}(\delta_{s_0}, \rho) = \epsilon_{\sigma_\rho, \delta_{s_0}}.
\]
**Schedulers vs. Task Schedules**

**Theorem**
Let $\mu \in \text{Prob}(\text{Frags}^*(A))$ have support consisting of incomparable fragments, and let $\rho$ be a task schedule. Then there is a scheduler $\sigma_\rho : \text{Frags}^*(A) \rightarrow \bigvee(\text{Act})$ such that $\text{Apply}(\mu, \rho) = \epsilon_{\sigma_\rho, \mu}$.

**Corollary** Let $A$ be a Task PIOA.

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

Recall

\[(P \parallel Adv \parallel E) \simeq (F \parallel I \parallel E)\]

For us,

\[\text{tdist}[P \parallel Adv \parallel E] \subseteq \text{tdist}[F \parallel I \parallel E]\]
Simulations

Recall

\[(P || Adv || E) \simeq (F || I || E)\]

For us,

\[\text{tdist}[P || Adv || E] \subseteq \text{tdist}[F || I || E]\]

Let \((A_i, R_i)\) be two task PIOAs and let \(f : R_1^* \times R_1 \rightarrow R_2^*\) be a function. We define \(\text{full}(f) : R_1^* \rightarrow R_2^*\) by

\[
\begin{align*}
\text{full}(f)(\langle \rangle) &= \langle \rangle \\
\text{full}(f)(\rho T) &= \text{full}(f)(\rho)^* f(\rho, T)
\end{align*}
\]
Simulations

Recall

\[(\mathcal{P} \parallel A \parallel E) \simeq (\mathcal{F} \parallel I \parallel E)\]

For us,

\[\text{tdist}[\mathcal{P} \parallel A \parallel E] \subseteq \text{tdist}[\mathcal{F} \parallel I \parallel E]\]

\(R \subseteq \text{Prob}(\text{Exec}(\mathcal{A}_1)) \times \text{Prob}(\text{Exec}(\mathcal{A}_2))\) is a simulation if

- \((\mu_1, \mu_2) \in R \Rightarrow \text{tdist}(\mu_1) \subseteq \text{tdist}(\mu_2)\)
- \((\delta_{s_0,1}, \delta_{s_0,2}) \in R\)
- \((\exists f : \mathcal{R}_1^* \times \mathcal{R}_1 \rightarrow \mathcal{R}_2^*)(\forall \rho \in \mathcal{R}_1^*)(\forall T \in \mathcal{R}_1)\)

\[\text{Apply}(\mu_1, T), \text{Apply}(\mu_2, \text{full}(f)(\rho, T)) \in \mathcal{E}(R)\]
Simulations

Recall

\[(\mathcal{P}\|\mathcal{A}dv\|\mathcal{E}) \simeq (\mathcal{F}\|\mathcal{I}\|\mathcal{E})\]

For us,

\[\text{tdist}[\mathcal{P}\|\mathcal{A}dv\|\mathcal{E}] \subseteq \text{tdist}[\mathcal{F}\|\mathcal{I}\|\mathcal{E}]\]

**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 

\[\text{tdist}(\mathcal{A}_1) \subseteq \text{tdist}(\mathcal{A}_2).\]
Expansions and Monads

Let $(X, \Sigma_X), (Y, \Sigma_Y)$ be measure spaces, $R \subseteq X \times Y$. The lift of $R$ is defined as

\[ \hat{R} \subseteq \forall X \times \forall Y \] defined by

\[
(\sum_x r_x \delta_x, \sum_y s_y \delta_y) \in \hat{R} \iff \exists t: X \times Y \rightarrow [0, 1] \text{ with }
\]

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

Let $(X, \Sigma_X), (Y, \Sigma_Y)$ be measure spaces, $R \subseteq X \times Y$. The lift of $R$ is $\hat{R} \subseteq \forall X \times \forall Y$ defined by

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

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

But, $(X, \Sigma_X)$ a measure space implies $\forall X$ is a measure space
And $R \subseteq \forall X \times \forall Y$ implies $\hat{R} \subseteq \forall(\forall X) \times \forall(\forall Y)$
Expansions and Monads

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

\[
\left(\sum_x r_x \delta_x, \sum_y s_y \delta_y\right) \in \hat{R} \iff \exists t: X \times Y \to [0, 1] \text{ with }
\]

- \(r_x = \sum_y t(x, y) \hspace{1cm} (\forall x)\)
- \(\sum_x t(x, y) \leq s_y \hspace{1cm} (\forall y)\)
- \(t(x, y) > 0 \Rightarrow (x, y) \in R\)

But, \((X, \Sigma_X)\) a measure space implies \(\mathbb{V}X\) is a measure space

And \(R \subseteq \mathbb{V}X \times \mathbb{V}Y\) implies \(\hat{R} \subseteq \mathbb{V}(\mathbb{V}X) \times \mathbb{V}(\mathbb{V}Y)\)

\((\mu, \nu) \in \mathcal{E}(R) \iff (\exists (\mu', \nu') \in \hat{R}) \mu = \int d\mu' \wedge \nu = \int d\mu'\)
Expansions and Monads

$(X, \Sigma_X), (Y, \Sigma_Y)$ measure spaces, $R \subseteq X \times Y$. The lift of $R$ is $\hat{R} \subseteq \forall X \times \forall Y$ defined by

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

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

But, $(X, \Sigma_X)$ a measure space implies $\forall X$ is a measure space.

And $R \subseteq \forall X \times \forall Y$ implies $\hat{R} \subseteq \forall(\forall X) \times \forall(\forall Y)$

$$(\mu, \nu) \in \mathcal{E}(R) \iff (\exists (\mu', \nu') \in \hat{R}) \mu = \int d\mu' \land \nu = \int d\mu'$$

$\forall$: Meas $\to$ Meas is a monad, and $R \mapsto \mathcal{E}(R)$ utilizes the lifting and multiplication of the monad.
Summary

- Gave outline of how domain theory can clarify structure of Task PIOAs.
- In particular, task schedulers, the measures they induce and their relation to task schedules emerge more clearly.
Summary

- Gave outline of how domain theory can clarify structure of Task PIOAs.
- In particular, task schedulers, the measures they induce and their relation to task schedules emerge more clearly.

Future Work

- More about the use of the monad $\nabla$
- Application to Dining Cryptographers
- Application to other protocols, combining the UC of oblivious transfer due to Canetti, et al.