Domain Theory and Task-structured Probabilistic Input/Output Automata

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

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; $\mathcal{A}dv$ adversary; $\mathcal{I}$ ideal adversary; $\mathcal{E}$ environment

$\mathcal{P} \leq_S \mathcal{F}$ iff $(\forall \mathcal{A}dv)(\exists \mathcal{I})(\forall \mathcal{E}) \quad \mathcal{P} \| \mathcal{A}dv \| \mathcal{E} \simeq \mathcal{F} \| \mathcal{I} \| \mathcal{E}$

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; $\mathcal{A}dv$ adversary; $\mathcal{I}$ ideal adversary; $\mathcal{E}$ environment

  $\mathcal{P} \leq_S \mathcal{F}$ iff $(\forall \mathcal{A}dv)(\exists \mathcal{I})(\forall \mathcal{E}) \quad \mathcal{P} \| \mathcal{A}dv \| \mathcal{E} \simeq \mathcal{F} \| \mathcal{I} \| \mathcal{E}$

- $\leq_S$: Trace distributions of observable events from $(\mathcal{P} \| \mathcal{A}dv \| \mathcal{E})$ all appear in the trace distributions of $(\mathcal{F} \| \mathcal{I} \| \mathcal{E})$.

Separation of Concerns

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

Separation of Concerns

- *Semantic model:* Devise model that accommodates concerns arising in protocol analysis
  - Spi-calculus, Probabilistic π-calculus, PPT-calculus, CSP,...
  - $\Rightarrow$ **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 π-calculus, PPT-calculus, CSP,...

  $\Rightarrow$ **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
- 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

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

- S - countable set of states, s_0 - start state
- 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: $(s, a, \mu), (s, a, \nu) \in D \;\Rightarrow\; \mu = \nu$

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

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

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

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

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

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

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

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

- $S_{\mathcal{A}_1,\mathcal{A}_2} = S_1 \times S_2$
- $I_{\mathcal{A}_1,\mathcal{A}_2} = (I_1 \cup I_2) \setminus (O_1 \cup O_2)$
- $O_{\mathcal{A}_1,\mathcal{A}_2} = O_1 \cup O_2$
- $H_{\mathcal{A}_1,\mathcal{A}_2} = H_1 \cup H_2$
- $D_{\mathcal{A}_1,\mathcal{A}_2} = \{((s_1, s_2), a, \mu_1 \times \mu_2) \mid (s_1, a, \mu_1) \in D_1 \text{ or } (s_2, a, \mu_2) \in D_2$ 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.
- **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:** $\mathcal{R}_{\mathcal{A}_1 \| \mathcal{A}_2} = \mathcal{R}_{\mathcal{A}_1} \cup \mathcal{R}_{\mathcal{A}_2}$

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

  Resolve 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* - $\rho = T_1 T_2 \cdots$

Example

- *Player:* Flips coin and announces result
- *Opponent:* Announces *Heads* or *Tails*
- If they match, *Player* wins, otherwise *Opponent* wins.
  - *Player:*
    States: ⊥, flipped, announced, initially ⊥
    Actions: Flip: Output: *Heads, Tails*
    P-Announce: *Heads, Tails*
    Tasks: {Flip, P-Announce}
  - *Opponent:*
    States: ⊥, announced, initially ⊥
    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: ⊥, flipped, announced, initially ⊥
    Actions: Flip: Output: *Heads, Tails*
    P-Announce: *Heads, Tails*
    Tasks: {Flip, P-Announce}
  - *Opponent:*
    States: ⊥, announced, initially ⊥
    Actions: O-Announce-Heads, O-Anounce-Tails
    Tasks: {O-Announce-Heads, O-Announce-Tails}
  - Task Schedules:
    Flip,P-Announce,O-Announce-Heads
    Flip,P-Announce,O-Announce-Tails
    Flip,O-Announce-Heads,P-Announce
    Flip,O-Announce-Tails,P-Announce
    O-Announce-Heads,Flip,P-Announce
    O-Announce-Tails,Flip,P-Announce

PIOA Semantics

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

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

Execution fragment:

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

PIOA Semantics

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

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

Execution fragment:

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

$\mathsf{fs}(\alpha)$ – 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

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

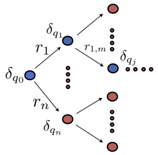
$\mathsf{Frags}^*(A) = \bigcup_n \{\alpha \in (S \times \mathsf{Act})^n \times S \mid \alpha \text{ finite execution fragment}\}$

Execution fragment:

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

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

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



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

Which α should be used?

PIOA Semantics

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

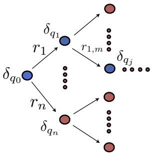
$\text{Frags}^*(A) = \bigcup_n \{\alpha \in (S \times \text{Act})^n \times S \mid \alpha \text{ finite execution fragment}\}$

Execution fragment:

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

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

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



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

Which α 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 $A_T = \{\alpha \in \mathsf{Frags}^*(\mathcal{A}) \mid T \text{ enabled in } \mathsf{ls}(\alpha)\}$

Defining Apply

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

$\mu \in \mathsf{Prob}(\mathsf{Frags}^*(\mathcal{A})) \Rightarrow \mu = \sum_\alpha \mu(\alpha)\delta_\alpha$, 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 $A_T = \{\alpha \in \mathsf{Frags}^*(\mathcal{A}) \mid T \text{ enabled in } \mathsf{ls}(\alpha)\}$

$\mu \in \mathsf{Prob}(\mathsf{Frags}^*(\mathcal{A})) \Rightarrow \mu = \sum_\alpha \mu(\alpha)\delta_\alpha$, 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 (cont'd)

What about ρ infinite?

Defining Apply (cont'd)

What about ρ infinite?

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

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

Defining Apply (cont'd)

What about ρ infinite?

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

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

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

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

Defining Apply (cont'd)

What about ρ infinite?

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

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

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

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

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?

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

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

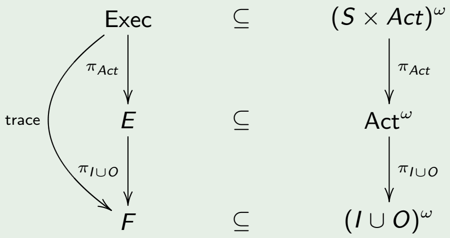
$\mathsf{Apply}(T)\colon \mathsf{Prob}(\mathsf{Frags}^*(\mathcal{A})) \to \mathsf{Prob}(\mathsf{Frags}^*(\mathcal{A}))$ is *not* monotone,

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,

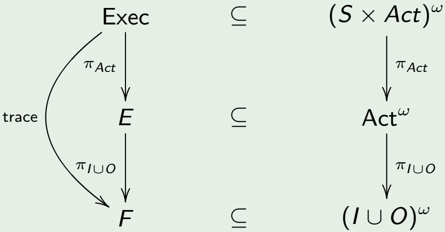
Hence $\mathsf{Apply}(\mu, \rho) = \sup_n \mathsf{Apply}(\mu, T_1 \cdots T_n)$ is well-defined.

Semantics of Observable Events



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

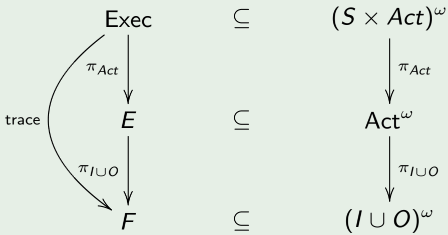
Semantics of Observable Events



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 $E = \{\alpha|_{\mathsf{Act}^\omega} \mid \alpha \in \mathsf{Exec}\}$ 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 $\rho = T_1 \cdots$, we'd like to have a *scheduler* – determined in advance – that would represent applying ρ to any $\mu \in \mathsf{Prob}(\mathsf{Frags}^*(A))$.

Task Schedulers

A *task scheduler* is a map

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

Task 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 - Original Definition

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

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

where $\mu_{\alpha'',a}(s)$ is the probability of landing in state s starting from $\mathsf{ls}(\alpha)$ after executing action a.

Measures from Schedulers - Our Definition

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

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

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

Measures from Schedulers - Our Definition

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

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

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

Neat fact:

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

implies $\epsilon'_{\sigma,\alpha} = \sup_n \epsilon'_{\sigma,\alpha,n}$

Schedulers vs. Task Schedules

Theorem

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

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

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

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

Schedulers vs. Task Schedules

Theorem
Let $\mu \in \mathsf{Prob}(\mathsf{Frags}^*(A))$ 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 $\mu \in \mathsf{Prob}(\mathsf{Frags}^*(A))$ 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
$$\mathsf{Apply}(\delta_{s_0}, \rho) = \epsilon_{\sigma_\rho, \delta_{s_0}}.$$
- $\mathsf{tdist}(\mathcal{A}) \subseteq \{\mathsf{trace}(\epsilon_{\sigma, \delta_{s_0}}) \mid \sigma \text{ deterministic scheduler}\}$.

Simulations

Recall

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

For us,

$$\mathrm{tdist}[\![\mathcal{P} \| \mathcal{A}dv \| \mathcal{E}]\!] \subseteq \mathrm{tdist}[\![\mathcal{F} \| \mathcal{I} \| \mathcal{E}]\!]$$

Simulations

Recall

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

For us,

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

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

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

Simulations

Recall

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

For us,

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

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

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

$$(\mu_1, \mu_2) \in R \;\wedge\; \text{supp}(\mu_1) \subseteq \rho \;\wedge\; \text{supp}(\mu_2) \subseteq \text{full}(f)(\rho)$$
$$\Rightarrow\; (\text{Apply}(\mu_1, T), \text{Apply}(\mu_2, \text{full}(f)(\rho, T)) \in \mathcal{E}(R)$$

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

$(X,\Sigma_X),(Y,\Sigma_Y)$ 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

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

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

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

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

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

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 $\widehat{R} \subseteq \mathbb{V}(\mathbb{V}X) \times \mathbb{V}(\mathbb{V}Y)$

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

$\mathbb{V}$: 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 $\mathbb{V}$
- Application to Dining Cryptograhers
- Application to other protocols, combining the UC of oblivious transfer due to Canetti, et al.