Probabilistic Program Analysis
Logic and Analysis

Alessandra Di Pierro
University of Verona, Italy
alessandra.dipierro@univr.it

Herbert Wiklicky
Imperial College London, UK
herbert@doc.ic.ac.uk
Moore-Penrose Pseudo-Inverse

Definition

Let \( C \) and \( D \) be two Hilbert spaces and \( A : C \rightarrow D \) a bounded linear map. A bounded linear map \( A^\dagger = G : D \rightarrow C \) is the Moore-Penrose pseudo-inverse of \( A \) iff

(i) \( A \circ G = P_A \),

(ii) \( G \circ A = P_G \),

where \( P_A \) and \( P_G \) denote orthogonal projections onto the ranges of \( A \) and \( G \).
On finite dimensional vector (Hilbert) spaces we have an inner product \( \langle \cdot, \cdot \rangle \). This allows us to define an adjoint via:

\[
\langle A(x), y \rangle = \langle x, A^*(y) \rangle
\]

- An operator \( A \) is self-adjoint if \( A = A^* \).
- An operator \( A \) is positive, i.e. \( A \succeq 0 \), if there exists an operator \( B \) such that \( A = B^*B \).
- An (orthogonal) projection is a self-adjoint \( E \) with \( EE = E \).

Projections identify (closed) sub-spaces \( \mathcal{Y}_E = \{Ex \mid x \in \mathcal{V} \} \).
On finite dimensional vector (Hilbert) spaces we have an inner product $\langle ., . \rangle$. This allows us to define an adjoint via:

$$\langle A(x), y \rangle = \langle x, A^*(y) \rangle$$

- An operator $A$ is self-adjoint if $A = A^*$.
- An operator $A$ is positive, i.e. $A \succeq 0$, if there exists an operator $B$ such that $A = B^*B$.
- An (orthogonal) projection is a self-adjoint $E$ with $EE = E$.

Projections identify (closed) sub-spaces $Y_E = \{Ex \mid x \in V\}$.
(Orthogonal) Projections – Idempotents

On finite dimensional vector (Hilbert) spaces we have an inner product $\langle . , . \rangle$. This allows us to define an adjoint via:

$$\langle A(x), y \rangle = \langle x, A^*(y) \rangle$$

- An operator $A$ is self-adjoint if $A = A^*$.
- An operator $A$ is positive, i.e. $A \succeq 0$, if there exists an operator $B$ such that $A = B^*B$.
- An (orthogonal) projection is a self-adjoint $E$ with $EE = E$.

Projections identify (closed) sub-spaces $Y_E = \{Ex \mid x \in \mathcal{V}\}$. 
On finite dimensional vector (Hilbert) spaces we have an inner product $\langle ., . \rangle$. This allows us to define an adjoint via:

$$\langle A(x), y \rangle = \langle x, A^*(y) \rangle$$

- An operator $A$ is self-adjoint if $A = A^*$.
- An operator $A$ is positive, i.e. $A \succeq 0$, if there exists an operator $B$ such that $A = B^*B$.
- An (orthogonal) projection is a self-adjoint $E$ with $EE = E$.

Projections identify (closed) sub-spaces $Y_E = \{Ex \mid x \in \mathcal{V} \}$. 
(Orthogonal) Projections – Idempotents

On finite dimensional vector (Hilbert) spaces we have an inner product $\langle ., . \rangle$. This allows us to define an adjoint via:

$$\langle A(x), y \rangle = \langle x, A^*(y) \rangle$$

- An operator $A$ is self-adjoint if $A = A^*$.
- An operator $A$ is positive, i.e. $A \succeq 0$, if there exists an operator $B$ such that $A = B^*B$.
- An (orthogonal) projection is a self-adjoint $E$ with $EE = E$.

Projections identify (closed) sub-spaces $Y_E = \{Ex \mid x \in V\}$. 

Bolzano, 22-26 August 2016 ESSLLI’16
Example: Sign Domain

Enumeration: $\text{Sign} = \{\emptyset, 0, \geq 0, \leq 0, \mathbb{Z}\}$

Free Vector Space: $\mathcal{V}(\text{Sign}) = \left\{ \sum_{s \in \text{Sign}} x_s \cdot s \mid x_i \in \mathbb{R} \right\}$

Example: Sign Domain

Enumeration: \( \text{Sign} = \{\emptyset, 0, \geq 0, \leq 0, \mathbb{Z} \} \)

Free Vector Space: \( \mathcal{V}(\text{Sign}) = \{ \sum_{s \in \text{Sign}} x_s \cdot s \mid x_i \in \mathbb{R} \} \)

Example: Sign Domain

\[ \begin{array}{c}
\bullet \mathbb{Z} \\
\bullet \leq 0 \\
\bullet 0 \\
\bullet \geq 0 \\
\bullet \emptyset \\
\end{array} \]

Enumeration: \( \text{Sign} = \{\emptyset, 0, \geq 0, \leq 0, \mathbb{Z}\} \)

Free Vector Space: \( \mathcal{V}(\text{Sign}) = \left\{ \sum_{s \in \text{Sign}} x_s \cdot s \mid x_i \in \mathbb{R} \right\} \)

Example: Sign Domain

Enumeration: \( \text{Sign} = \{\emptyset, 0, \geq 0, \leq 0, \mathbb{Z}\} \)

Free Vector Space: \( \mathcal{V}(\text{Sign}) = \{ \sum_{s \in \text{Sign}} x_s \cdot s \mid x_i \in \mathbb{R} \} \)

Consider the upward closed sub-domains of \( \{ \emptyset, 0, \geq 0, \leq 0, \mathbb{Z} \} \):

\[
\begin{align*}
\rho_1 &= \{ \mathbb{Z} \} \\
\rho_2 &= \{ \mathbb{Z}, \geq 0 \} \\
\rho_3 &= \{ \mathbb{Z}, 0 \} \\
\rho_4 &= \{ \mathbb{Z}, \emptyset \} \\
\rho_5 &= \{ \mathbb{Z}, \leq 0 \} \\
\rho_6 &= \{ \mathbb{Z}, \geq 0, \emptyset \} \\
\rho_7 &= \{ \mathbb{Z}, \geq 0, 0 \} \\
\rho_8 &= \{ \mathbb{Z}, 0, \emptyset \} \\
\rho_9 &= \{ \mathbb{Z}, \leq 0, 0 \} \\
\rho_{10} &= \{ \mathbb{Z}, \leq 0, \emptyset \} \\
\rho_{11} &= \{ \mathbb{Z}, \geq 0, 0, \emptyset \} \\
\rho_{12} &= \{ \mathbb{Z}, \leq 0, \geq 0, 0, \emptyset \} \\
\rho_{13} &= \{ \mathbb{Z}, \leq 0, 0, \emptyset \} \\
\rho_{14} &= \{ \mathbb{Z}, \leq 0, \geq 0, 0, \emptyset \}
\end{align*}
\]

Identify abstract domains via upward closed operators (uco) \( \rho = \alpha \circ \gamma \) (vs downward closed operators (dco) \( \gamma \circ \alpha \)).
Example: Probabilistic Abstractions $R_n$

$$R_1 = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \quad R_2 = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R_3 = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \quad R_4 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R_5 = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \quad R_6 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$
Example: Probabilistic Abstractions $R_n$

\[
R_7 = \begin{pmatrix}
0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1
\end{pmatrix}, \quad R_8 = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1
\end{pmatrix}
\]

\[
R_9 = \begin{pmatrix}
0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1
\end{pmatrix}, \quad R_{10} = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1
\end{pmatrix}
\]
Example: Probabilistic Abstractions $R_n$
Associate to every PAI \((A, G)\) a projection (similar to uco):

\[ E = AG = AA^\dagger. \]

A general way to construct \(E \cap F\) and (by exploiting de Morgan’s law) also \(E \cup F = (E^\perp \cap F^\perp)^\perp\) is via an infinite approximation sequence and has been suggested by Halmos:

\[ E \cap F = \lim_{n \to \infty} (EFE)^n. \]
Associate to every PAI \((A, G)\) a projection (similar to uco):

\[ E = AG = AA^\dagger. \]

A general way to construct \(E \cap F\) and (by exploiting de Morgan’s law) also \(E \sqcup F = (E^\perp \cap F^\perp)^\perp\) is via an infinite approximation sequence and has been suggested by Halmos:

\[ E \cap F = \lim_{n \to \infty} (EFE)^n. \]
The concrete construction of $E \sqcup F$ and $E \sqcap F$ is in general not trivial. Only for commuting projections we have:

$$E \sqcup F = E + F - EF \quad \text{and} \quad E \sqcap F = EF.$$
The concrete construction of $E \sqcup F$ and $E \sqcap F$ is in general not trivial. Only for commuting projections we have:

$$E \sqcup F = E + F - EF \quad \text{and} \quad E \sqcap F = EF.$$ 

**Example**

Consider a finite set $\Omega$ with a probability structure. For any (measurable) subset $A$ of $\Omega$ define the characteristic function $\chi_A$ with $\chi_A(x) = 1$ if $x \in A$ and 0 otherwise. The characteristic functions are (commutative) projections on random variables using pointwise multiplication, i.e. $X \chi_A \chi_A = X \chi_A$. We have $\chi_{A \cap B} = \chi_A \chi_B$ and $\chi_{A \cup B} = \chi_A + \chi_B - \chi_A \chi_B$. 

Bolzano, 22-26 August 2016  
ESSLLI’16  
Probabilistic Program Analysis  
Slide 10 of 54
Commutative Case

The concrete construction of $E \sqcup F$ and $E \sqcap F$ is in general not trivial. Only for commuting projections we have:

$$E \sqcup F = E + F - EF \quad \text{and} \quad E \sqcap F = EF.$$  

Example

Consider a finite set $\Omega$ with a probability structure. For any (measurable) subset $A$ of $\Omega$ define the characteristic function $\chi_A$ with $\chi_A(x) = 1$ if $x \in A$ and 0 otherwise. The characteristic functions are (commutative) projections on random variables using pointwise multiplication, i.e. $X \chi_A \chi_A = X \chi_A$. We have $\chi_{A \cap B} = \chi_A \chi_B$ and $\chi_{A \cup B} = \chi_A + \chi_B - \chi_A \chi_B$. 
Non-Commutative Case

The Moore-Penrose pseudo-inverse is also useful for computing the $E \sqcap F$ and $E \sqcup F$ of general, non-commuting projections via the **parallel sum**

$$A : B = A(A+B)^\dagger B$$

The **intersection of projections** is given by:

$$E \sqcap F = 2(E : F) = E(E+F)^\dagger F + F(E+F)^\dagger E$$

Israel, Greville: *Generalized Inverses, Theory and Applications*, Springer 03
Define a partial order on self-adjoint operators and projections as follows: $H \sqsubseteq K$ iff $K - H$ is positive, i.e. there exists a $B$ such that $K - H = B^*B$.

Alternatively, order projections by inclusion of their image spaces, i.e. $E \sqsubseteq F$ iff $Y_E \subseteq Y_F$.

The orthogonal projections form a complete lattice.

The range of the intersection $E \cap F$ is to the closure of the intersection of the image spaces of $E$ and $F$.

The union $E \sqcup F$ corresponds to the union of the images.
Define a partial order on self-adjoint operators and projections as follows: \( H \sqsubseteq K \) iff \( K - H \) is positive, i.e. there exists a \( B \) such that \( K - H = B^*B \).

Alternatively, order projections by inclusion of their image spaces, i.e. \( E \sqsubseteq F \) iff \( \mathcal{Y}_E \subseteq \mathcal{Y}_F \).

The orthogonal projections form a complete lattice.

The range of the intersection \( E \cap F \) is to the closure of the intersection of the image spaces of \( E \) and \( F \).

The union \( E \sqcup F \) corresponds to the union of the images.
Define a **partial order** on self-adjoint operators and projections as follows: \( H \sqsubseteq K \) iff \( K - H \) is **positive**, i.e. there exists a \( B \) such that \( K - H = B^*B \).

Alternatively, order projections by inclusion of their image spaces, i.e. \( E \sqsubseteq F \) iff \( Y_E \subseteq Y_F \).

The orthogonal projections form a complete lattice.

The range of the **intersection** \( E \cap F \) is to the closure of the intersection of the image spaces of \( E \) and \( F \).

The **union** \( E \sqcup F \) corresponds to the union of the images.
Ortholattices I

Non-distributive analogs of Boolean algebras.

**Definition (Ortholattice I)**

An *ortholattice* \((L, \sqsubseteq, \bot, 0, 1)\) is a lattice \((L, \sqsubseteq)\) with universal bounds 0 and 1, i.e.

1. \((L, \sqsubseteq)\) is a partial order (i.e. \(\sqsubseteq\) is reflexive, antisymmetric, and transitive),
2. all pairs of elements \(a, b \in L\) have a least upper bound (sup) denoted by \(a \sqcup b\), and a greatest lower bound (inf) denoted by \(a \sqcap b\),
3. \(0 \sqsubseteq a\) and \(a \sqsubseteq 1\) for all \(a \in L\).

...
Non-distributive analogs of Boolean algebras.

**Definition (Ortholattice I)**

An *ortholattice* \((L, \sqsubseteq, \bot, 0, 1)\) is a lattice \((L, \sqsubseteq)\) with universal bounds 0 and 1, i.e.

1. \((L, \sqsubseteq)\) is a partial order (i.e. \(\sqsubseteq\) is reflexive, antisymmetric, and transitive),

2. all pairs of elements \(a, b \in L\) have a least upper bound (sup) denoted by \(a \sqcup b\), and a greatest lower bound (inf) denoted by \(a \sqcap b\),

3. \(0 \sqsubseteq a\) and \(a \sqsubseteq 1\) for all \(a \in L\).
Non-distributive analogs of Boolean algebras.

Definition (Ortholattice I)

An ortholattice \((L, \sqsubseteq, \bot, 0, 1)\) is a lattice \((L, \sqsubseteq)\) with universal bounds 0 and 1, i.e.

1. \((L, \sqsubseteq)\) is a partial order (i.e. \(\sqsubseteq\) is reflexive, antisymmetric, and transitive),
2. all pairs of elements \(a, b \in L\) have a least upper bound (sup) denoted by \(a \sqcup b\), and a greatest lower bound (inf) denoted by \(a \sqcap b\),
3. \(0 \sqsubseteq a\) and \(a \sqsubseteq 1\) for all \(a \in L\).
Non-distributive analogs of Boolean algebras.

**Definition (Ortholattice I)**

An *ortholattice* \((L, \sqsubseteq, \perp, 0, 1)\) is a lattice \((L, \sqsubseteq)\) with universal bounds 0 and 1, i.e.

1. \((L, \sqsubseteq)\) is a partial order (i.e. \(\sqsubseteq\) is reflexive, antisymmetric, and transitive),
2. all pairs of elements \(a, b \in L\) have a least upper bound (sup) denoted by \(a \sqcup b\), and a greatest lower bound (inf) denoted by \(a \sqcap b\),
3. \(0 \sqsubseteq a\) and \(a \sqsubseteq 1\) for all \(a \in L\).

...
Definition (Ortholattice II)

...and a unary *complementation* operation \( a \mapsto a^\perp \) satisfying:

1. \( a \cap a^\perp = 0 \) and \( a \cup a^\perp = 1 \) for all \( a \in L \),
2. \( (a \cap b)^\perp = a^\perp \cup b^\perp \) and \( (a \cup b)^\perp = a^\perp \cap b^\perp \) for all \( a, b \in L \),
3. \( (a^\perp)^\perp = a \) for all \( a \in L \).

The set \( P(\mathcal{H}) \) of closed-range projections on a Hilbert space \( \mathcal{H} \) is a non-distributive ortholattice

\[
\langle P(\mathcal{H}), \subseteq, \cup, \cap, \cdot^\perp, 1, 0 \rangle
\]
Definition (Ortholattice II)

...and a unary *complementation* operation $a \mapsto a^\perp$ satisfying:

1. $a \sqcap a^\perp = 0$ and $a \sqcup a^\perp = 1$ for all $a \in L$,
2. $(a \sqcap b)^\perp = a^\perp \sqcup b^\perp$ and $(a \sqcup b)^\perp = a^\perp \sqcap b^\perp$ for all $a, b \in L$,
3. $(a^\perp)^\perp = a$ for all $a \in L$.

The set $P(\mathcal{H})$ of closed-range projections on a Hilbert space $\mathcal{H}$ is a non-distributive ortholattice

$$\left\langle P(\mathcal{H}), \sqsubseteq, \sqcup, \sqcap, ^\perp, \mathbf{I}, \mathbf{0} \right\rangle$$
Definition (Ortholattice II)

... and a unary \textit{complementation} operation $a \mapsto a^\perp$ satisfying:

1. $a \sqcap a^\perp = 0$ and $a \sqcup a^\perp = 1$ for all $a \in L$,
2. $(a \sqcap b)^\perp = a^\perp \sqcup b^\perp$ and $(a \sqcup b)^\perp = a^\perp \sqcap b^\perp$ for all $a, b \in L$,
3. $(a^\perp)^\perp = a$ for all $a \in L$.

The set $P(\mathcal{H})$ of closed-range projections on a Hilbert space $\mathcal{H}$ is a non-distributive ortholattice

\[ \langle P(\mathcal{H}), \sqsubseteq, \sqcup, \sqcap, \cdot \downarrow, 1, 0 \rangle \]
Definition (Ortholattice II)

...and a unary complementation operation \( a \mapsto a^\perp \) satisfying:

1. \( a \cap a^\perp = 0 \) and \( a \cup a^\perp = 1 \) for all \( a \in L \),
2. \( (a \cap b)^\perp = a^\perp \cup b^\perp \) and \( (a \cup b)^\perp = a^\perp \cap b^\perp \) for all \( a, b \in L \),
3. \( (a^\perp)^\perp = a \) for all \( a \in L \).

The set \( P(\mathcal{H}) \) of closed-range projections on a Hilbert space \( \mathcal{H} \) is a non-distributive ortholattice

\[ \left< P(\mathcal{H}), \subseteq, \cup, \cap, \cdot, I, 0 \right> \]
Definition (Ortholattice II)

... and a unary *complementation* operation \( a \mapsto a^\perp \) satisfying:

1. \( a \cap a^\perp = 0 \) and \( a \cup a^\perp = 1 \) for all \( a \in L \),
2. \((a \cap b)^\perp = a^\perp \cup b^\perp\) and \((a \cup b)^\perp = a^\perp \cap b^\perp\) for all \( a, b \in L \),
3. \((a^\perp)^\perp = a\) for all \( a \in L \).

The set \( P(H) \) of closed-range projections on a Hilbert space \( H \) is a non-distributive ortholattice

\[
\left\langle P(H), \subseteq, \cup, \cap, \cdot^\perp, 1, 0 \right\rangle
\]
Commutativity and Distributivity

In general, $\sqcap$ and $\sqcup$ in an ortholattice are not distributive, ie.

$$(a \sqcap b) \sqcup (a \sqcap c) \sqsubseteq a \sqcap (b \sqcup c)$$

$$a \sqcup (b \sqcap c) \sqsubseteq (a \sqcup b) \sqcap (a \sqcup c)$$

Two elements $a$ and $b$ in an ortholattice commute, denoted by $[a, b] = 0$, iff

$$a = (a \sqcap b) \sqcup (a \sqcap b)$$

An ortholattice is called an orthomodular lattice if $[a, b] = 0$ implies $[b, a] = 0$. 
Commutativity and Distributivity

In general, $\sqcap$ and $\sqcup$ in an ortholattice are **not distributive**, ie.

\[
(a \sqcap b) \sqcup (a \sqcap c) \subseteq a \sqcap (b \sqcup c)
\]

\[
a \sqcup (b \sqcap c) \subseteq (a \sqcup b) \sqcap (a \sqcup c)
\]

Two elements $a$ and $b$ in an ortholattice **commute**, denoted by $[a, b] = 0$, iff

\[
a = (a \sqcap b) \sqcup (a \sqcap b^\perp)
\]

An ortholattice is called an **orthomodular lattice** if $[a, b] = 0$ implies $[b, a] = 0$. 

Bolzano, 22-26 August 2016 ESSLLI'16 Probabilistic Program Analysis Slide 15 of 54
Commutativity and Distributivity

In general, \( \cap \) and \( \cup \) in an ortholattice are not distributive, ie.

\[
(a \cap b) \cup (a \cap c) \subseteq a \cap (b \cup c)
\]
\[
a \cup (b \cap c) \subseteq (a \cup b) \cap (a \cup c)
\]

Two elements \( a \) and \( b \) in an ortholattice commute, denoted by \([a, b] = 0\), iff

\[
a = (a \cap b) \cup (a \cap b^\perp)
\]

An ortholattice is called an orthomodular lattice if \([a, b] = 0\) implies \([b, a] = 0\).
Example: Projections $\mathbf{P}_n = \mathbf{R}_n \mathbf{R}_n^\dagger$

\[\mathbf{P}_1 = \begin{pmatrix}
\frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} \\
\frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} \\
\frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} \\
\frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} \\
\frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5}
\end{pmatrix}, \quad \mathbf{P}_2 = \begin{pmatrix}
\frac{1}{3} & \frac{1}{3} & 0 & \frac{1}{3} & 0 \\
\frac{1}{3} & \frac{1}{3} & 0 & \frac{1}{3} & 0 \\
0 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 0 & 1
\end{pmatrix}\]

\[\mathbf{P}_3 = \begin{pmatrix}
\frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\
\frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\
0 & 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\
0 & 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\
0 & 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3}
\end{pmatrix}, \quad \mathbf{P}_4 = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 1
\end{pmatrix}\]

\[\mathbf{P}_5 = \begin{pmatrix}
\frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 \\
\frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 \\
\frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 \\
\frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 \\
\frac{1}{3} & \frac{1}{3} & 0 & 0 & 0
\end{pmatrix}, \quad \mathbf{P}_6 = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 \\
0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 \\
0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 \\
0 & \frac{1}{2} & 0 & \frac{1}{2} & 0
\end{pmatrix}\]
Example: Projections $P_n = R_n R_n^\dagger$

$P_7 = \begin{pmatrix} 1/2 & 1/2 & 0 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 & 0 \\ 0 & 0 & 1/2 & 0 & 1/2 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1/2 & 0 & 1/2 \end{pmatrix}$, $P_8 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1/3 & 1/3 & 1/3 \\ 0 & 0 & 1/3 & 1/3 & 1/3 \\ 0 & 0 & 1/3 & 1/3 & 1/3 \end{pmatrix}$

$P_9 = \begin{pmatrix} 1/2 & 1/2 & 0 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 1/2 \\ 0 & 0 & 0 & 1/2 & 1/2 \end{pmatrix}$, $P_{10} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1/2 & 1/2 & 0 & 0 \\ 0 & 1/2 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 1/2 \\ 0 & 0 & 0 & 1/2 & 1/2 \end{pmatrix}$
Example: Projections $P_n = R_n R_n^\dagger$

\[ P_{11} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} \end{pmatrix}, \quad P_{12} = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \quad P_{13} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \end{pmatrix}, \quad P_{14} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \]
Example: The Lattice $uco(\text{Sign})$
Example: The Lattice $\mathcal{P}(\mathcal{V}(\text{Sign}))$
Example: Combining Projections

\[ P_7 \cap P_8 = \left( \begin{array}{cccc}
\frac{1}{2} & \frac{1}{2} & 0 & 0 \\
\frac{1}{2} & \frac{1}{2} & 0 & 0 \\
0 & 0 & \frac{1}{2} & 0 \\
0 & 0 & 0 & 1 \\
0 & 0 & \frac{1}{2} & 0 \\
\end{array} \right) \cap \left( \begin{array}{cccc}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & \frac{1}{3} & \frac{1}{3} \\
0 & 0 & \frac{1}{3} & \frac{1}{3} \\
0 & 0 & \frac{1}{3} & \frac{1}{3} \\
\end{array} \right) = \left( \begin{array}{cccc}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & \frac{1}{3} & \frac{1}{3} \\
0 & 0 & \frac{1}{3} & \frac{1}{3} \\
0 & 0 & \frac{1}{3} & \frac{1}{3} \\
\end{array} \right) = P_3 \]

In particular, we have \( P_7 \cap P_8 = P_7 P_8 \) as \( P_7 \) and \( P_8 \) commute, i.e. \([P_7, P_8] = P_7 P_8 - P_8 P_7 = O\).
Example: Combining Projections

\[ P_4 \sqcap P_7 = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 1 \\
\end{pmatrix} \sqcap \begin{pmatrix}
1 & 1 & 0 & 0 & 0 \\
2 & 2 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
\end{pmatrix} =
\begin{pmatrix}
\frac{1}{2} & 1 & 1 & 1 & 1 \\
\frac{1}{2} & 1 & 1 & 1 & 1 \\
\frac{1}{2} & 1 & 1 & 1 & 1 \\
\frac{1}{2} & 1 & 1 & 1 & 1 \\
\frac{1}{2} & 1 & 1 & 1 & 1 \\
\end{pmatrix} = P_1
\]

Using the expression \( P_4 \sqcap P_7 = 2P_4(P_4 + P_7)^\dagger P_7 \) as \( P_4 \) and \( P_7 \) do not commute.
Note that the simple multiplication $P_4 P_7$ is different from $P_4 \sqcap P_7$:

\[
P_4 P_7 = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\
0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\
0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\
0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4}
\end{pmatrix}
\begin{pmatrix}
\frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\
\frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\
0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} \\
0 & 0 & 0 & 1 & 0 \\
0 & 0 & \frac{1}{2} & 0 & \frac{1}{2}
\end{pmatrix}
\neq P_4 \sqcap P_7
\]
Definition

Given two vector (Hilbert) spaces $C$ and $D$ and a bounded linear map $F : C \to D$, then we say that a pair of projections $P : C \to C$ and $R : D \to D$ is complete for $F$ iff

$$FP = RFP.$$ 

Given a pair of projections $(P, R)$ for a function $F$, we estimate the precision of the abstraction via the “difference” between $FP$ and its optimal version $RFP$.

$$\text{Prec}_F(P, R) = \|FP - RFP\|.$$
Definition

Given two vector (Hilbert) spaces $C$ and $D$ and a bounded linear map $F : C \to D$, then we say that a pair of projections $P : C \to C$ and $R : D \to D$ is complete for $F$ iff

$$FP = RFP.$$ 

Given a pair of projections $(P, R)$ for a function $F$, we estimate the precision of the abstraction via the “difference” between $FP$ and its optimal version $RFP$.

$$\text{Prec}_F(P, R) = \|FP - RFP\|.$$
Proposition

Let $F : \mathcal{H}_1 \rightarrow \mathcal{H}_2$ be a bounded linear operator between two Hilbert spaces $\mathcal{H}_1$ and $\mathcal{H}_2$, and let $P_1, P_2 \in P(\mathcal{H}_2)$ and $R \in P(\mathcal{H}_1)$. Then we have: if $P_1 \subseteq P_2$ then $\text{Prec}_F(P_1, R) \leq \text{Prec}_F(P_2, R)$. 
### Example: (Relative) Precisions

<table>
<thead>
<tr>
<th></th>
<th>P_1</th>
<th>P_2</th>
<th>P_3</th>
<th>P_4</th>
<th>P_5</th>
<th>P_6</th>
<th>P_7</th>
<th>P_8</th>
<th>P_9</th>
<th>P_10</th>
<th>P_11</th>
<th>P_12</th>
<th>P_13</th>
<th>P_14</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P_2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P_3</td>
<td>1</td>
<td>.75</td>
<td>0</td>
<td>.79</td>
<td>.75</td>
<td>.65</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>.65</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P_4</td>
<td>1</td>
<td>.91</td>
<td>.79</td>
<td>0</td>
<td>.91</td>
<td>0</td>
<td>.79</td>
<td>0</td>
<td>.79</td>
<td>0</td>
<td>.79</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P_5</td>
<td>1</td>
<td>.75</td>
<td>0</td>
<td>.79</td>
<td>.75</td>
<td>.65</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>.65</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P_6</td>
<td>1.10</td>
<td>1</td>
<td>.87</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>.87</td>
<td>0</td>
<td>.87</td>
<td>0</td>
<td>.87</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P_7</td>
<td>1.34</td>
<td>1</td>
<td>0</td>
<td>1.06</td>
<td>1</td>
<td>.87</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>.87</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P_8</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>.82</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>.82</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P_9</td>
<td>1.10</td>
<td>.82</td>
<td>0</td>
<td>.87</td>
<td>.82</td>
<td>.71</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>.71</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P_10</td>
<td>1.07</td>
<td>.91</td>
<td>.87</td>
<td>.87</td>
<td>.91</td>
<td>.71</td>
<td>.87</td>
<td>0</td>
<td>.87</td>
<td>.71</td>
<td>0</td>
<td>.87</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P_11</td>
<td>1.34</td>
<td>1</td>
<td>1</td>
<td>1.22</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P_12</td>
<td>1.34</td>
<td>1</td>
<td>0</td>
<td>1.06</td>
<td>1</td>
<td>.87</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>.87</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P_13</td>
<td>1.10</td>
<td>1</td>
<td>1</td>
<td>1.06</td>
<td>1</td>
<td>.87</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>.87</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P_14</td>
<td>1.34</td>
<td>1</td>
<td>1</td>
<td>1.22</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
The **collecting semantics** of a program $P$ is given by:

$$T(P) = \sum p_{ij} \cdot T(\ell_i, \ell_j)$$

Local effects $T(\ell_i, \ell_j)$: Data Update + Control Step

$$T(\ell_i, \ell_j) = (N_{i1} \otimes N_{i2} \otimes \ldots \otimes N_{iv}) \otimes M_{ij}$$
The **collecting semantics** of a program $P$ is given by:

$$T(P) = \sum p_{ij} \cdot T(\ell_i, \ell_j)$$

Local effects $T(\ell_i, \ell_j)$: Data Update + Control Step

$$T(\ell_i, \ell_j) = (N_{i1} \otimes N_{i2} \otimes \ldots \otimes N_{iv}) \otimes M_{ij}$$
Given a $n \times m$ matrix $A$ and a $k \times l$ matrix $B$:

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{1m} & \cdots & a_{nm} \end{pmatrix}, \quad B = \begin{pmatrix} b_{11} & \cdots & b_{1k} \\ \vdots & \ddots & \vdots \\ b_{1l} & \cdots & b_{kl} \end{pmatrix}$$

The tensor product $A \otimes B$ is then a $nk \times ml$ matrix:

$$A \otimes B = \begin{pmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{1m}B & \cdots & a_{nm}B \end{pmatrix}$$
Kronecker Products

Given a $n \times m$ matrix $A$ and a $k \times l$ matrix $B$:

\[
A = \begin{pmatrix}
a_{11} & \ldots & a_{1n} \\
\vdots & \ddots & \vdots \\
a_{1m} & \ldots & a_{nm}
\end{pmatrix} \quad B = \begin{pmatrix}
b_{11} & \ldots & b_{1k} \\
\vdots & \ddots & \vdots \\
b_{1l} & \ldots & b_{kl}
\end{pmatrix}
\]

The tensor product $A \otimes B$ is then a $nk \times ml$ matrix:

\[
A \otimes B = \begin{pmatrix}
a_{11}B & \ldots & a_{1n}B \\
\vdots & \ddots & \vdots \\
a_{1m}B & \ldots & a_{nm}B
\end{pmatrix}
\]
Abstract Tensor Product

The (algebraic) tensor product of vector spaces $\mathcal{V}_1$, $\mathcal{V}_2$, $\ldots$, $\mathcal{V}_n$ is given by a vector space $\bigotimes_{i=1}^n \mathcal{V}_i$ and a map $p = \bigotimes_{i=1}^n \in \mathcal{L}(\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_n; \bigotimes_{i=1}^n \mathcal{V}_i)$ such that if $\mathcal{W}$ is any vector space and $f \in \mathcal{L}(\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_n; \mathcal{W})$ then there exists a unique map $h : \bigotimes_{i=1}^n \mathcal{V}_i \rightarrow \mathcal{W}$ satisfying $f = h \circ p$.

$$\mathcal{V}(X \times Y) = \mathcal{V}(X) \otimes \mathcal{V}(Y)$$
The (algebraic) tensor product of vector spaces $\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_n$ is given by a vector space $\bigotimes_{i=1}^n \mathcal{V}_i$ and a map $p = \otimes_{i=1}^n \in \mathcal{L}(\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_n; \bigotimes_{i=1}^n \mathcal{V}_i)$ such that if $\mathcal{W}$ is any vector space and $f \in \mathcal{L}(\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_n; \mathcal{W})$ then there exists a unique map $h : \bigotimes_{i=1}^n \mathcal{V}_i \to \mathcal{W}$ satisfying $f = h \circ p$.
The tensor product of $n$ linear operators $A_1, A_2, \ldots, A_n$ is associative (but in general not commutative) and has e.g. the following properties:

1. $$(A_1 \otimes \ldots \otimes A_n) \cdot (B_1 \otimes \ldots \otimes B_n) = A_1 \cdot B_1 \otimes \ldots \otimes A_n \cdot B_n$$

2. $$A_1 \otimes \ldots \otimes (\alpha A_i) \otimes \ldots \otimes A_n = \alpha (A_1 \otimes \ldots \otimes A_i \otimes \ldots \otimes A_n)$$

3. $$A_1 \otimes \ldots \otimes (A_i + B_i) \otimes \ldots \otimes A_n = (A_1 \otimes \ldots \otimes A_i \otimes \ldots \otimes A_n) + (A_1 \otimes \ldots \otimes B_i \otimes \ldots \otimes A_n)$$

4. $$(A_1 \otimes \ldots \otimes A_i \otimes \ldots \otimes A_n)^\dagger = A_1^\dagger \otimes \ldots \otimes A_i^\dagger \otimes \ldots \otimes A_n^\dagger$$
The tensor product of $n$ linear operators $A_1, A_2, \ldots, A_n$ is associative (but in general not commutative) and has e.g. the following properties:

1. $$(A_1 \otimes \ldots \otimes A_n) \cdot (B_1 \otimes \ldots \otimes B_n) = A_1 \cdot B_1 \otimes \ldots \otimes A_n \cdot B_n$$

2. $$A_1 \otimes \ldots \otimes (\alpha A_i) \otimes \ldots \otimes A_n = \alpha (A_1 \otimes \ldots \otimes A_i \otimes \ldots \otimes A_n)$$

3. $$A_1 \otimes \ldots \otimes (A_i + B_i) \otimes \ldots \otimes A_n = (A_1 \otimes \ldots \otimes A_i \otimes \ldots \otimes A_n) + (A_1 \otimes \ldots \otimes B_i \otimes \ldots \otimes A_n)$$

4. $$(A_1 \otimes \ldots \otimes A_i \otimes \ldots \otimes A_n)^\dagger = A_1^\dagger \otimes \ldots \otimes A_i^\dagger \otimes \ldots \otimes A_n^\dagger$$
The tensor product of $n$ linear operators $A_1, A_2, \ldots, A_n$ is associative (but in general not commutative) and has e.g. the following properties:

1. $$(A_1 \otimes \ldots \otimes A_n) \cdot (B_1 \otimes \ldots \otimes B_n) =$$
   $$= A_1 \cdot B_1 \otimes \ldots \otimes A_n \cdot B_n$$

2. $$A_1 \otimes \ldots \otimes (\alpha A_i) \otimes \ldots \otimes A_n =$$
   $$= \alpha(A_1 \otimes \ldots \otimes A_i \otimes \ldots \otimes A_n)$$

3. $$A_1 \otimes \ldots \otimes (A_i + B_i) \otimes \ldots \otimes A_n =$$
   $$= (A_1 \otimes \ldots \otimes A_i \otimes \ldots \otimes A_n) + (A_1 \otimes \ldots \otimes B_i \otimes \ldots \otimes A_n)$$

4. $$(A_1 \otimes \ldots \otimes A_i \otimes \ldots \otimes A_n)^\dagger =$$
   $$= A_1^\dagger \otimes \ldots \otimes A_i^\dagger \otimes \ldots \otimes A_n^\dagger$$
The tensor product of $n$ linear operators $A_1, A_2, \ldots, A_n$ is associative (but in general not commutative) and has e.g. the following properties:

1. \[(A_1 \otimes \ldots \otimes A_n) \cdot (B_1 \otimes \ldots \otimes B_n) = A_1 \cdot B_1 \otimes \ldots \otimes A_n \cdot B_n\]

2. \[A_1 \otimes \ldots \otimes (\alpha A_i) \otimes \ldots \otimes A_n = \alpha(A_1 \otimes \ldots \otimes A_i \otimes \ldots \otimes A_n)\]

3. \[A_1 \otimes \ldots \otimes (A_i + B_i) \otimes \ldots \otimes A_n = (A_1 \otimes \ldots \otimes A_i \otimes \ldots \otimes A_n) + (A_1 \otimes \ldots \otimes B_i \otimes \ldots \otimes A_n)\]

4. \[(A_1 \otimes \ldots \otimes A_i \otimes \ldots \otimes A_n)^\dagger = A_1^\dagger \otimes \ldots \otimes A_i^\dagger \otimes \ldots \otimes A_n^\dagger\]
Tensor Product Properties

The tensor product of $n$ linear operators $A_1, A_2, \ldots, A_n$ is associative (but in general not commutative) and has e.g. the following properties:

1. \[(A_1 \otimes \ldots \otimes A_n) \cdot (B_1 \otimes \ldots \otimes B_n) =
   \]
   \[= A_1 \cdot B_1 \otimes \ldots \otimes A_n \cdot B_n
   \]

2. \[A_1 \otimes \ldots \otimes (\alpha A_i) \otimes \ldots \otimes A_n =
   \]
   \[= \alpha(A_1 \otimes \ldots \otimes A_i \otimes \ldots \otimes A_n)
   \]

3. \[A_1 \otimes \ldots \otimes (A_i + B_i) \otimes \ldots \otimes A_n =
   \]
   \[= (A_1 \otimes \ldots \otimes A_i \otimes \ldots \otimes A_n) + (A_1 \otimes \ldots \otimes B_i \otimes \ldots \otimes A_n)
   \]

4. \[(A_1 \otimes \ldots \otimes A_i \otimes \ldots \otimes A_n)^\dagger =
   \]
   \[= A_1^\dagger \otimes \ldots \otimes A_i^\dagger \otimes \ldots \otimes A_n^\dagger
   \]
Relational Dependency

1: $[m \leftarrow 1]^1$
2: while $[n > 1]^2$ do
3: $[m \leftarrow m \times n]^3$
4: $[n \leftarrow n - 1]^4$
5: end while
6: [stop]$^5$

Input/output behaviour: Parity of $m$ for different values of $n$.

- Probability that $m = \text{even/odd and } n = 1, 2, 3$.
- Probability that $m$ is even/odd, and
  - Probability that $n$ is 1, 2, 3.
- Probability that $m$ is even/odd for $n = 1, 2, 3$. 
Relational Dependency

1: \[ m \leftarrow 1 \]\(^1\);
2: \textbf{while } \[ n > 1 \]\(^2\) \textbf{do}
3: \[ m \leftarrow m \times n \]\(^3\);
4: \[ n \leftarrow n - 1 \]\(^4\)
5: \textbf{end while}
6: \[ \text{stop} \]\(^5\)

Input/output behaviour: Parity of \( m \) for different values of \( n \).

- Probability that \( m = \text{even/odd} \) and \( n = 1, 2, 3 \).
  - Probability that \( m \) is even/odd, and
  - Probability that \( n \) is 1, 2, 3.
- Probability that \( m \) is even/odd for \( n = 1, 2, 3 \).
Relational Dependency

1: \([m \leftarrow 1]^1\);
2: \textbf{while} \([n > 1]^2\ \textbf{do}
3: \quad [m \leftarrow m \times n]^3;
4: \quad [n \leftarrow n - 1]^4
5: \textbf{end while}
6: \textbf{stop}^5

Input/output behaviour: Parity of \(m\) for different values of \(n\).

- Probability that \(m = \text{even/odd}\) and \(n = 1, 2, 3\).
  - Probability that \(m\) is even/odd, and
  - Probability that \(n\) is 1, 2, 3.
- Probability that \(m\) is even/odd for \(n = 1, 2, 3\).
Relational Dependency

1: [\[m \leftarrow 1\]^1;]
2: while \[n > 1\]^2 do
3: \[m \leftarrow m \times n\]^3;
4: \[n \leftarrow n - 1\]^4
5: end while
6: \[\text{stop}\]^5

Input/output behaviour: Parity of \(m\) for different values of \(n\).

- Probability that \(m = \text{even/odd}\) and \(n = 1, 2, 3\).
  - Probability that \(m\) is even/odd, and
  - Probability that \(n\) is 1, 2, 3.
- Probability that \(m\) is even/odd \text{for} \(n = 1, 2, 3\).
Some joint probability distributions can be expressed as tensor product of two (independent) probability distributions \(e\) and \(f\):

\[
\begin{pmatrix}
\frac{2}{9} & \frac{2}{9} & \frac{2}{9} \\
\frac{1}{9} & \frac{1}{9} & \frac{1}{9}
\end{pmatrix}
= \left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right) \otimes \left(\frac{2}{3}, \frac{1}{3}\right)^t
\]

However, in general we can express any joint probability distribution as a linear combination of distributions.

\[
\begin{pmatrix}
0 & \frac{1}{3} & \frac{1}{3} \\
\frac{1}{3} & 0 & 0
\end{pmatrix}
= \frac{1}{3}(e_1 \otimes f_2) + \frac{1}{3}(e_2 \otimes f_1) + \frac{1}{3}(e_3 \otimes f_1)
\]

with \(e_i \in \mathbb{R}^3\) and \(f_j \in \mathbb{R}^2\) (row and column) basis vectors.
Some joint probability distributions can be expressed as tensor product of two (independent) probability distributions $e$ and $f$:

$$
\begin{pmatrix}
\frac{2}{9} & \frac{2}{9} & \frac{2}{9} \\
\frac{1}{9} & \frac{1}{9} & \frac{1}{9}
\end{pmatrix}
= \left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right) \otimes \left(\frac{2}{3}, \frac{1}{3}\right)^t
$$

However, in general we can express any joint probability distribution as a linear combination of distributions.

$$
\begin{pmatrix}
0 & \frac{1}{3} & \frac{1}{3} \\
\frac{1}{3} & 0 & 0
\end{pmatrix}
= \frac{1}{3}(e_1 \otimes f_2) + \frac{1}{3}(e_2 \otimes f_1) + \frac{1}{3}(e_3 \otimes f_1)
$$

with $e_i \in \mathbb{R}^3$ and $f_j \in \mathbb{R}^2$ (row and column) basis vectors.
Some joint probability distributions can be expressed as tensor product of two (independent) probability distributions $e$ and $f$:

\[
\begin{pmatrix}
\frac{2}{9} & \frac{2}{9} & \frac{2}{9} \\
\frac{1}{9} & \frac{1}{9} & \frac{1}{9}
\end{pmatrix}
= \left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right) \otimes \left(\frac{2}{3}, \frac{1}{3}\right)^t
\]

But there are no two vectors $e$ and $f$ such that for example

\[
\begin{pmatrix}
0 & \frac{1}{3} & \frac{1}{3} \\
\frac{1}{3} & 0 & 0
\end{pmatrix}
= e \otimes f
\]
Some joint probability distributions can be expressed as tensor product of two (independent) probability distributions $e$ and $f$:

$$
\begin{pmatrix}
\frac{2}{9} & \frac{2}{9} & \frac{2}{9} \\
\frac{1}{9} & \frac{1}{9} & \frac{1}{9}
\end{pmatrix} = \left( \frac{1}{3}, \frac{1}{3}, \frac{1}{3} \right) \otimes \left( \frac{2}{3}, \frac{1}{3} \right)^t
$$

However, in general we can express any joint probability distribution as a linear combination of distributions.

$$
\begin{pmatrix}
0 & \frac{1}{3} & \frac{1}{3} \\
\frac{1}{3} & 0 & 0
\end{pmatrix} = \frac{1}{3}(e_1 \otimes f_2) + \frac{1}{3}(e_2 \otimes f_1) + \frac{1}{3}(e_3 \otimes f_1)
$$

with $e_i \in \mathbb{R}^3$ and $f_j \in \mathbb{R}^2$ (row and column) basis vectors.
Consider compositional (probabilistic) abstractions of the form:

\[
S = \bigoplus_{i=1}^{\nu} S(x_i) \quad \text{with} \quad S(x_i) = (\bigotimes_{k=1}^{i-1} S_{-i}) \otimes S_i \otimes (\bigotimes_{k=i+1}^{\nu} S_{-i})
\]

**Fully Relational:** $S_r$ is $S$ with $S_i = A_i$ and $S_{-i} = A_{-i}$

**Weakly Relational:** $S_w$ is $S$ with $S_i = A_i$ and $S_{-i} = A_{-i}$ or $A_f$

**Non-Relational:** $S_n$ is $S$ with $S_i = A$ and $S_{-i} = A_f$

With $A_f$ forgetful and $A_i$ and $A_{-i}$ nontrivial abstractions. For $S_r$ all factors in $\bigoplus$ are the same; we can take $S_r = S(x_1)$. 
Consider compositional (probabilistic) abstractions of the form:

\[
S = \bigoplus_{i=1}^{v} S(x_i) \quad \text{with} \quad S(x_i) = (\bigotimes_{k=1}^{i-1} S_{-i}) \otimes S_i \otimes (\bigotimes_{k=i+1}^{v} S_{-i})
\]

Fully Relational: \( S_r \) is \( S \) with \( S_i = A_i \) and \( S_{-i} = A_{-i} \)

Weakly Relational: \( S_w \) is \( S \) with \( S_i = A_i \) and \( S_{-i} = A_{-i} \) or \( A_f \)

Non-Relational: \( S_n \) is \( S \) with \( S_i = A \) and \( S_{-i} = A_f \)

With \( A_f \) forgetful and \( A_i \) and \( A_{-i} \) nontrivial abstractions.

For \( S_r \) all factors in \( \bigoplus \) are the same; we can take \( S_r = S(x_1) \).
Consider compositional (probabilistic) abstractions of the form:

\[
S = \bigoplus_{i=1}^{v} S(x_i) \quad \text{with} \quad S(x_i) = (\bigotimes_{k=1}^{i-1} S_{-i}) \otimes S_i \otimes (\bigotimes_{k=i+1}^{v} S_{-i})
\]

**Fully Relational:** \(S_r\) is \(S\) with \(S_i = A_i\) and \(S_{-i} = A_{-i}\)

**Weakly Relational:** \(S_w\) is \(S\) with \(S_i = A_i\) and \(S_{-i} = A_{-i}\) or \(A_f\)

**Non-Relational:** \(S_n\) is \(S\) with \(S_i = A\) and \(S_{-i} = A_f\)

With \(A_f\) forgetful and \(A_i\) and \(A_{-i}\) nontrivial abstractions.
For \(S_r\) all factors in \(\bigoplus\) are the same; we can take \(S_r = S(x_1)\).
Consider compositional (probabilistic) abstractions of the form:

\[ S = \bigoplus_{i=1}^{\nu} S(x_i) \quad \text{with} \quad S(x_i) = \bigotimes_{k=1}^{i-1} S_{-i} \otimes S_i \otimes \bigotimes_{k=i+1}^{\nu} S_{-i} \]

**Fully Relational:** \( S_r \) is \( S \) with \( S_i = A_i \) and \( S_{-i} = A_{-i} \)

**Weakly Relational:** \( S_w \) is \( S \) with \( S_i = A_i \) and \( S_{-i} = A_{-i} \) or \( A_f \)

**Non-Relational:** \( S_n \) is \( S \) with \( S_i = A \) and \( S_{-i} = A_f \)

With \( A_f \) forgetful and \( A_i \) and \( A_{-i} \) nontrivial abstractions.

For \( S_r \) all factors in \( \bigoplus \) are the same; we can take \( S_r = S(x_1) \).
Examples

```
var x:[0..10]; begin x:=k; stop  (k = 1, 4)
```

<table>
<thead>
<tr>
<th>P</th>
<th>R</th>
<th>$S_n$</th>
<th>$S_w$</th>
<th>$S_r$</th>
<th>id</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$S_n$</td>
<td>1.58</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$S_w$</td>
<td>1.58</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$S_r$</td>
<td>1.58</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>id</td>
<td>2.55</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Using cast $d$ abstraction: $A_d$ lifted $\alpha(x) = x \mod d$

$S_n$ is $S$ with $S_i = S_4$, $S_{\neg i} = A_1$

$S_w$ is $S$ with $S_i = S_4$, $S_{\neg i} = A_2$

$S_r$ is $S$ with $S_i = S_{\neg i} = A_4$
Examples

\[\text{var } x: [0..10]; \ y: [0..10]; \begin{align*} \text{begin } x &= y; \text{ stop}
\end{align*}\]

<table>
<thead>
<tr>
<th>(P \setminus R)</th>
<th>(\emptyset)</th>
<th>(S_n)</th>
<th>(S_w)</th>
<th>(S_r)</th>
<th>(id)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\emptyset)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(S_n)</td>
<td>1.73</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(S_w)</td>
<td>2.24</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(S_r)</td>
<td>2.24</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(id)</td>
<td>3.61</td>
<td>3.61</td>
<td>3.61</td>
<td>3.61</td>
<td>0</td>
</tr>
</tbody>
</table>

Using \textit{cast }d\textit{ abstraction : }A_d\textit{ lifted }\alpha(x) = x \mod d

\(S_n\) \text{ is } S \text{ with } S_i = S_4, S_{\neg i} = A_1
\(S_w\) \text{ is } S \text{ with } S_i = S_4, S_{\neg i} = A_2
\(S_r\) \text{ is } S \text{ with } S_i = S_{\neg i} = A_4
Examples

```pascal
var x: [0..10]; y: [0..3]; begin x := 2*y; stop
```

<table>
<thead>
<tr>
<th>P \ R</th>
<th>$\emptyset$</th>
<th>$S_n$</th>
<th>$S_w$</th>
<th>$S_r$</th>
<th>$id$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$S_n$</td>
<td>1.88</td>
<td>0.89</td>
<td>0.89</td>
<td>0.89</td>
<td>0</td>
</tr>
<tr>
<td>$S_w$</td>
<td>2.14</td>
<td>1.52</td>
<td>1.29</td>
<td>1.29</td>
<td>0</td>
</tr>
<tr>
<td>$S_r$</td>
<td>2.24</td>
<td>1.64</td>
<td>1.50</td>
<td>1.41</td>
<td>0</td>
</tr>
<tr>
<td>$id$</td>
<td>3.61</td>
<td>3.60</td>
<td>3.59</td>
<td>3.58</td>
<td>0</td>
</tr>
</tbody>
</table>

Using cast $d$ abstraction: $A_d$ lifted $\alpha(x) = x \mod d$

$S_n$ is $S$ with $S_i = S_4, S_{\neg i} = A_1$

$S_w$ is $S$ with $S_i = S_4, S_{\neg i} = A_2$

$S_r$ is $S$ with $S_i = S_{\neg i} = A_4$
Examples

```
var x:[0..10]; y:[0..3]; begin x:=3*y; stop
```

<table>
<thead>
<tr>
<th></th>
<th>$\emptyset$</th>
<th>$S_n$</th>
<th>$S_w$</th>
<th>$S_r$</th>
<th>$id$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$S_n$</td>
<td>1.77</td>
<td>0.89</td>
<td>0.89</td>
<td>0.89</td>
<td>0</td>
</tr>
<tr>
<td>$S_w$</td>
<td>2.24</td>
<td>1.52</td>
<td>1.29</td>
<td>1.29</td>
<td>0</td>
</tr>
<tr>
<td>$S_r$</td>
<td>2.24</td>
<td>1.64</td>
<td>1.50</td>
<td>1.41</td>
<td>0</td>
</tr>
<tr>
<td>$id$</td>
<td>3.61</td>
<td>3.60</td>
<td>3.59</td>
<td>3.58</td>
<td>0</td>
</tr>
</tbody>
</table>

Using cast $d$ abstraction: $A_d$ lifted $\alpha(x) = x \mod d$

- $S_n$ is $S$ with $S_i = S_4, S_{-i} = A_1$
- $S_w$ is $S$ with $S_i = S_4, S_{-i} = A_2$
- $S_r$ is $S$ with $S_i = S_{-i} = A_4$
Further Work
Conclusions
Conclusions

Some applications of PAI:

- **Approximate Process Equivalences**: The semantics of concurrent processes can be defined via approximate equivalences (e.g. $\epsilon$-bisimulation).

- **Approximate Confinement**: Static analysis of security properties can be sometimes more effective if the security is guaranteed only up to some acceptable percentage threshold.

- **Probabilistic Program Transformation**: Transforming out timing leaks... probabilistically.

- ...
Conclusions

Some applications of PAI:

- **Approximate Process Equivalences**: The semantics of concurrent processes can be defined via approximate equivalences (e.g. $\epsilon$-bisimulation).

- **Approximate Confinement**: Static analysis of security properties can be sometimes more effective if the security is guaranteed only up to some acceptable percentage threshold.

- **Probabilistic Program Transformation**: Transforming out timing leaks... probabilistically.

- ...
Conclusions

Some applications of PAI:

- **Approximate Process Equivalences**: The semantics of concurrent processes can be defined via approximate equivalences (e.g. $\epsilon$-bisimulation).

- **Approximate Confinement**: Static analysis of security properties can be sometimes more effective if the security is guaranteed only up to some acceptable percentage treshold.

- **Probabilistic Program Transformation**: Transforming out timing leaks... probabilistically.

- ...

Bolzano, 22-26 August 2016 ESSLLI’16
Some applications of PAI:

- **Approximate Process Equivalences**: The semantics of concurrent processes can be defined via approximate equivalences (e.g. $\epsilon$-bisimulation).

- **Approximate Confinement**: Static analysis of security properties can be sometimes more effective if the security is guaranteed only up to some acceptable percentage threshold.

- **Probabilistic Program Transformation**: Transforming out timing leaks... probabilistically.

...
Some applications of PAI:

- **Approximate Process Equivalences**: The semantics of concurrent processes can be defined via approximate equivalences (e.g. $\epsilon$-bisimulation).

- **Approximate Confinement**: Static analysis of security properties can be sometimes more effective if the security is guaranteed only up to some acceptable percentage threshold.

- **Probabilistic Program Transformation**: Transforming out timing leaks... probabilistically.

- ...
In every choice construct one must make a choice and the probabilities of all choices must sum up to one (certainty). One can’t assume (that the programmer used) normalised probabilities.

We therefore need to normalise probabilities with respect to a context of "competing" probabilities:

\[ \tilde{p} = p[p_1...p_n] = \frac{p}{p_1 + \ldots + p_n}. \]

This can be done at compile-time if all probabilities are constants, but also at runtime in the operational semantics.

Typically one would assume \( p_i \in \mathbb{R} \) or \( p_i \in \mathbb{Q} \). However, we can also use discrete probabilities, i.e. \( p_i \in \mathbb{Z} \).
In every choice construct one must make a choice and the probabilities of all choices must sum up to one (certainty). One can’t assume (that the programmer used) normalised probabilities.

We therefore need to normalise probabilities with respect to a context of "competing" probabilities:

\[ \tilde{p} = p[p_1\ldots p_n] = \frac{p}{p_1 + \ldots + p_n}. \]

This can be done at compile-time if all probabilities are constants, but also at runtime in the operational semantics.

Typically one would assume \( p_i \in \mathbb{R} \) or \( p_i \in \mathbb{Q} \). However, we can also use discrete probabilities, i.e. \( p_i \in \mathbb{Z} \).
In every choice construct one must make a choice and the probabilities of all choices must sum up to one (certainty). One can’t assume (that the programmer used) normalised probabilities.

We therefore need to normalise probabilities with respect to a context of "competing" probabilities:

\[ \tilde{p} = p[p_1...p_n] = \frac{p}{p_1 + \ldots + p_n}. \]

This can be done at compile-time if all probabilities are constants, but also at runtime in the operational semantics.

Typically one would assume \( p_i \in \mathbb{R} \) or \( p_i \in \mathbb{Q} \). However, we can also use discrete probabilities, i.e. \( p_i \in \mathbb{Z} \).
In every choice construct one must make a choice and the probabilities of all choices must sum up to one (certainty). One can’t assume (that the programmer used) normalised probabilities.

We therefore need to normalise probabilities with respect to a context of "competing" probabilities:

\[ \tilde{p} = p[p_1 \ldots p_n] = \frac{p}{p_1 + \ldots + p_n}. \]

This can be done at compile-time if all probabilities are constants, but also at runtime in the operational semantics.

Typically one would assume \( p_i \in \mathbb{R} \) or \( p_i \in \mathbb{Q} \). However, we can also use discrete probabilities, i.e. \( p_i \in \mathbb{Z} \).
In every choice construct one must make a choice and the probabilities of all choices must sum up to one (certainty). One can’t assume (that the programmer used) normalised probabilities.

We therefore need to normalise probabilities with respect to a context of "competing" probabilities:

\[ \tilde{p} = p[p_1...p_n] = \frac{p}{p_1 + \ldots + p_n}. \]

This can be done at compile-time if all probabilities are constants, but also at runtime in the operational semantics.

Typically one would assume \( p_i \in \mathbb{R} \) or \( p_i \in \mathbb{Q} \). However, we can also use discrete probabilities, i.e. \( p_i \in \mathbb{Z} \).
Consider a "duel" between two cowboys:

- Cowboy A – hitting probability $a$
- Cowboy B – hitting probability $b$

1. Choose (non-deterministically) whether A or B starts.
2. Repeat until winner is known:
   - If it is A's turn he will hit/shoot B with probability $a$;
     If B is shot then A is the winner, otherwise it's B's turn.
   - If it is B's turn he will hit/shoot A with probability $b$;
     If A is shot then B is the winner, otherwise it's A's turn.

Question: What is the life expectancy of A or B?
Question: What happens if A is learning to shoot better during the duel? How can we model dynamic probabilities?

Introduced by McIver and Morgan (2005).
Discussed in detail by Gretz, Katoen, McIver (2012)
Duel at High Noon

Consider a "duel" between two cowboys:

- Cowboy $A$ – hitting probability $a$
- Cowboy $B$ – hitting probability $b$

1. Choose (non-deterministically) whether $A$ or $B$ starts.
2. Repeat until winner is known:
   - If it is $A$'s turn he will hit/shoot $B$ with probability $a$;
     If $B$ is shot then $A$ is the winner, otherwise it’s $B$'s turn.
   - If it is $B$'s turn he will hit/shoot $A$ with probability $b$;
     If $A$ is shot then $B$ is the winner, otherwise it’s $A$’s turn.

**Question**: What is the life expectancy of $A$ or $B$?
**Question**: What happens if $A$ is learning to shoot better during the duel? How can we model dynamic probabilities?

Introduced by McIver and Morgan (2005).
Discussed in detail by Gretz, Katoen, McIver (2012)
Consider a "duel" between two cowboys:

- Cowboy $A$ – hitting probability $a$
- Cowboy $B$ – hitting probability $b$

1. Choose (non-deterministically) whether $A$ or $B$ starts.
2. Repeat until winner is known:
   - If it is $A$’s turn he will hit/shoot $B$ with probability $a$;
     If $B$ is shot then $A$ is the winner, otherwise it’s $B$’s turn.
   - If it is $B$’s turn he will hit/shoot $A$ with probability $b$;
     If $A$ is shot then $B$ is the winner, otherwise it’s $A$’s turn.

Question: What is the life expectancy of $A$ or $B$?
Question: What happens if $A$ is learning to shoot better during the duel? How can we model dynamic probabilities?

Introduced by McIver and Morgan (2005).
Discussed in detail by Gretz, Katoen, McIver (2012)
Consider a "duel" between two cowboys:
- Cowboy A – hitting probability $a$
- Cowboy B – hitting probability $b$

1. Choose (non-deterministically) whether A or B starts.
2. Repeat until winner is known:
   - If it is A’s turn he will hit/shoot B with probability $a$;
     If B is shot then A is the winner, otherwise it’s B’s turn.
   - If it is B’s turn he will hit/shoot A with probability $b$;
     If A is shot then B is the winner, otherwise it’s A’s turn.

Question: What is the life expectancy of A or B?
Question: What happens if A is learning to shoot better during the duel? How can we model dynamic probabilities?

Introduced by McIver and Morgan (2005).
Discussed in detail by Gretz, Katoen, McIver (2012)
Duel at High Noon

Consider a "duel" between two cowboys:

- Cowboy $A$ – hitting probability $a$
- Cowboy $B$ – hitting probability $b$

1. Choose (non-deterministically) whether $A$ or $B$ starts.
2. Repeat until winner is known:
   - If it is $A$’s turn he will hit/shoot $B$ with probability $a$;
     If $B$ is shot then $A$ is the winner, otherwise it’s $B$’s turn.
   - If it is $B$’s turn he will hit/shoot $A$ with probability $b$;
     If $A$ is shot then $B$ is the winner, otherwise it’s $A$’s turn.

Question: What is the life expectancy of $A$ or $B$?
Question: What happens if $A$ is learning to shoot better during the duel? How can we model dynamic probabilities?

Introduced by McIver and Morgan (2005).
Discussed in detail by Gretz, Katoen, McIver (2012)
Consider a "duel" between two cowboys:

- Cowboy $A$ – hitting probability $a$
- Cowboy $B$ – hitting probability $b$

1. Choose (non-deterministically) whether $A$ or $B$ starts.
2. Repeat until winner is known:
   - If it is $A$’s turn he will hit/shoot $B$ with probability $a$;
     If $B$ is shot then $A$ is the winner, otherwise it’s $B$’s turn.
   - If it is $B$’s turn he will hit/shoot $A$ with probability $b$;
     If $A$ is shot then $B$ is the winner, otherwise it’s $A$’s turn.

**Question**: What is the life expectancy of $A$ or $B$?

**Question**: What happens if $A$ is learning to shoot better during the duel? How can we model dynamic probabilities?

Introduced by McIver and Morgan (2005).
Discussed in detail by Gretz, Katoen, McIver (2012)
Duel at High Noon

Consider a "duel" between two cowboys:

- Cowboy $A$ – hitting probability $a$
- Cowboy $B$ – hitting probability $b$

1. Choose (non-deterministically) whether $A$ or $B$ starts.
2. Repeat until winner is known:
   - If it is $A$’s turn he will hit/shoot $B$ with probability $a$;
     If $B$ is shot then $A$ is the winner, otherwise it’s $B$’s turn.
   - If it is $B$’s turn he will hit/shoot $A$ with probability $b$;
     If $A$ is shot then $B$ is the winner, otherwise it’s $A$’s turn.

**Question**: What is the life expectancy of $A$ or $B$?

**Question**: What happens if $A$ is learning to shoot better during the duel? How can we model dynamic probabilities?

Introduced by McIver and Morgan (2005).
Discussed in detail by Gretz, Katoen, McIver (2012)
begin
# who's first turn
choose 1:{t:=0} or 1:{t:=1} (t :=
# continue until ...
c := 1;
while c == 1 do
if (t==0) then
    choose ak:{c:=0} or am:{t:=1} (t :=
else
    choose bk:{c:=0} or bm:{t:=0} (t :=
fi;
od;
stop; # terminal loop
end
Example: Duelling Cowboys

The survival chances, i.e. winning probability, for A.
Contexts: Advance Normalisation

For all possible values of the variable probabilities \( p_i \) compute their normalisation, compute the possible contexts.

\[
C[p_1, \ldots, p_n] = \begin{cases} 
\emptyset & \text{if } n = 0 \\
\{[p_1]\} & \text{if } n = 1 \text{ and } p_i \text{ const} \\
\{c \mid c \in \text{Value}(p_1)\} & \text{if } n = 1 \text{ and } p_i \text{ var} \\
\bigcup_{i \in C[p_1]} \{[i] \cdot C[p_2, \ldots, p_n]\} & \text{otherwise, i.e. } n > 1.
\end{cases}
\]

Example

Variable \( x \) with \( \text{Value}(x) = \{0, 1\} \) and a parameter \( p = 0 \) or \( p = 1 \) then contexts are given by:

\[
C[x, 1, p] = \{[0, 1, 0], [1, 1, 0]\} \quad \text{and} \quad C[x, 1, p] = \{[0, 1, 1], [1, 1, 1]\}
\]
Dynamic Probabilities

For all possible values of the variable probabilities test if the current state. With $c_j \in \text{Value}(p_j)$ and $d_i \in \text{Value}(p_i)$ use:

$$P_{p_i[p_1...p_n]}^{c_j[d_1...d_n]} = P(p_i = c_j) \cdot \left( \prod_{k=1,...,n} P(p_k = d_k) \right)$$

This gives the LOS Semantics for variable probabilities:

$$\left\{ [[\text{choose}]^n_{i=1} p_i : S_i \ \ldots \ \text{or} \ p_n : S_n \ \text{or} \ \ell] \right\}_{\text{LOS}} = \left\{ S_i \right\}_{\text{LOS}} \cup \bigcup_{i=1}^{n} \left\{ \sum_{c_j \in \text{Value}(p_i)} \sum_{[d_1...d_n] \in C[p_1...p_n]} c_j[d_1...d_n] \cdot P_{p_i[p_1...p_n]}^{c_j[d_1...d_n]} \otimes E(\ell, \text{init}(S_i)) \right\}$$
Dynamic Probabilities

For all possible values of the variable probabilities test if the current state. With $c_j \in \text{Value}(p_j)$ and $d_i \in \text{Value}(p_i)$ use:

$$P_{p_i[p_1 \ldots p_n]}^{c_j[d_1 \ldots d_n]} = P(p_i = c_j) \cdot \left( \prod_{k=1,\ldots,n} P(p_k = d_k) \right)$$

This gives the LOS Semantics for variable probabilities:

$$\{\text{[choose]}^{p_1:S_1} \ldots \text{or } p_n:S_n \text{ or } \ell\} \text{LOS} = \{S_i\} \text{LOS} \cup \bigcup_{i=1}^{n} \left\{ \sum_{c_j \in \text{Value}(p_i)} \sum_{[d_1 \ldots d_n] \in C[p_1 \ldots p_n]} c_j[d_1 \ldots d_n] \cdot P_{p_i[p_1 \ldots p_n]}^{c_j[d_1 \ldots d_n]} \otimes E(\ell, \text{init}(S_i)) \right\}$$
Learning how to shoot straight

begin
# initialise skills of A
akl := ak; aml := am;
# who's first
choose 1:{t:=0} or 1:{t:=1} ro;
# continue until ...
c := 1;
while c == 1 do
    if (t==0) then
        choose akl:{c:=0} or aml:{t:=1} ro
    else
        choose bk:{c:=0} or bm:{t:=0} ro
    fi;
    akl := @inc(akl); aml := @dec(aml);
    od;
stop; # terminal loop
end
Back to the two Cowboys

Learning rate 0.
Back to the two Cowboys

Learning rate 1.
Learning rate 2.
Back to the two Cowboys

Learning rate 4.
Finding the minimum length path vs minimum value of functions

As usual (for now): Take the best non-linear optimisation tool money can’t buy (leave it to "them" to make it work).
Finding the minimum length path vs minimum value of functions

As usual (for now): Take the best non-linear optimisation tool money can’t buy (leave it to "them" to make it work).
Finding the minimum length path vs minimum value of functions

As usual (for now): Take the best non-linear optimisation tool money can’t buy (leave it to "them" to make it work).
A General Approach

- Consider parameterised program $P(p_1, p_2, \ldots, p_n)$ with:

  ... $\lceil \text{choose} \rceil_\ell p_1 : S_1 \text{ or } \ldots \text{ or } p_n : S_n \text{ ro; } \ldots$

- Construct the parametric LOS semantics/operator, i.e.

  $$[P(\lambda_1, \lambda_2, \ldots, \lambda_n)] = T(\lambda_1, \lambda_2, \ldots, \lambda_n)$$

- Establish constraints on functional behaviour, e.g.

  $$\|A^\dagger T(\lambda_1, \lambda_2, \ldots, \lambda_n)A - [S]\| = 0$$

- Additional non-functional (performance) objectives

  $$\min_{\lambda_1, \lambda_2, \ldots, \lambda_n} \Phi(T(\lambda_1, \lambda_2, \ldots, \lambda_n))$$
A General Approach

- Consider parameterised program \( P(\lambda_1, \lambda_2, \ldots, \lambda_n) \) with

\[
\ldots [\text{choose}]^\ell \lambda_1 : S_1 \text{ or } \ldots \text{ or } \lambda_n : S_n \text{; } \ldots
\]

- Construct the parametric LOS semantics/operator, i.e.

\[
[P(\lambda_1, \lambda_2, \ldots, \lambda_n)] = T(\lambda_1, \lambda_2, \ldots, \lambda_n)
\]

- Establish constraints on functional behaviour, e.g.

\[
\|A^\dagger T(\lambda_1, \lambda_2, \ldots, \lambda_n)A - [S]\| = 0
\]

- Additional non-functional (performance) objectives

\[
\min_{\lambda_1, \lambda_2, \ldots, \lambda_n} \Phi(T(\lambda_1, \lambda_2, \ldots, \lambda_n))
\]
A General Approach

- Consider parameterised program \( P(\lambda_1, \lambda_2, \ldots, \lambda_n) \) with

\[
\cdots \left[ \text{opt} \right]^\ell S_1 \text{ or } \cdots \text{ or } S_n \text{ top; } \cdots
\]

- Construct the parametric LOS semantics/operator, i.e.

\[
\lbrack P(\lambda_1, \lambda_2, \ldots, \lambda_n) \rbrack = T(\lambda_1, \lambda_2, \ldots, \lambda_n)
\]

- Establish constraints on functional behaviour, e.g.

\[
\| A^\dagger T(\lambda_1, \lambda_2, \ldots, \lambda_n) A - \lbrack S \rbrack \| = 0
\]

- Additional non-functional (performance) objectives

\[
\min_{\lambda_1, \lambda_2, \ldots, \lambda_n} \Phi(T(\lambda_1, \lambda_2, \ldots, \lambda_n))
\]
A General Approach

- Consider parameterised program \( P(\lambda_1, \lambda_2, \ldots, \lambda_n) \) with

\[
\ldots \text{[choose]}^\ell \lambda_1 : S_1 \text{ or } \ldots \text{ or } \lambda_n : S_n \text{ ro; } \ldots
\]

- Construct the **parametric** LOS semantics/operator, i.e.

\[
\text{[[} P(\lambda_1, \lambda_2, \ldots, \lambda_n) \text{]]} = T(\lambda_1, \lambda_2, \ldots, \lambda_n)
\]

- Establish constraints on functional behaviour, e.g.

\[
\| A^\dagger T(\lambda_1, \lambda_2, \ldots, \lambda_n) A - [S] \| = 0
\]

- Additional non-functional (performance) objectives

\[
\min_{\lambda_1, \lambda_2, \ldots, \lambda_n} \Phi(T(\lambda_1, \lambda_2, \ldots, \lambda_n))
\]
A General Approach

- Consider parameterised program \( P(\lambda_1, \lambda_2, \ldots, \lambda_n) \) with

\[ \ldots \left[ \text{choose} \right]^\ell \lambda_1 : S_1 \text{ or } \ldots \text{ or } \lambda_n : S_n \text{ ro; } \ldots \]

- Construct the **parametric** LOS semantics/operator, i.e.

\[ [P(\lambda_1, \lambda_2, \ldots, \lambda_n)] = T(\lambda_1, \lambda_2, \ldots, \lambda_n) \]

- Establish constraints on functional behaviour, e.g.

\[ \| A^\dagger T(\lambda_1, \lambda_2, \ldots, \lambda_n) A - [S] \| = 0 \]

- Additional non-functional (performance) objectives

\[ \min_{\lambda_1, \lambda_2, \ldots, \lambda_n} \Phi(T(\lambda_1, \lambda_2, \ldots, \lambda_n)) \]
A General Approach

- Consider parameterised program $P(\lambda_1, \lambda_2, \ldots, \lambda_n)$ with

  ...$[\text{choose}]^\ell \lambda_1 : S_1 \text{ or } \ldots \text{ or } \lambda_n : S_n \text{ ro; } \ldots$

- Construct the parametric LOS semantics/operator, i.e.

  $\llbracket P(\lambda_1, \lambda_2, \ldots, \lambda_n) \rrbracket = T(\lambda_1, \lambda_2, \ldots, \lambda_n)$

- Establish constraints on functional behaviour, e.g.

  $A^\dagger T(\lambda_1, \lambda_2, \ldots, \lambda_n) A = \llbracket S \rrbracket$

- Additional non-functional (performance) objectives

  $\min_{\lambda_1, \lambda_2, \ldots, \lambda_n} \Phi(T(\lambda_1, \lambda_2, \ldots, \lambda_n))$
A General Approach

- Consider parameterised program $P(\lambda_1, \lambda_2, \ldots, \lambda_n)$ with

  ...$\{\text{choose}\}^\ell \lambda_1 : S_1 \text{ or } \ldots \text{ or } \lambda_n : S_n \text{ ro; } \ldots$

- Construct the parameteric LOS semantics/operator, i.e.

  $$\llbracket P(\lambda_1, \lambda_2, \ldots, \lambda_n) \rrbracket = T(\lambda_1, \lambda_2, \ldots, \lambda_n)$$

- Establish constraints on functional behaviour, e.g.

  $$\|A^\dagger T(\lambda_1, \lambda_2, \ldots, \lambda_n)A - \llbracket S \rrbracket\| = 0$$

- Additional non-functional (performance) objectives

  $$\min_{\lambda_1, \lambda_2, \ldots, \lambda_n} \Phi(T(\lambda_1, \lambda_2, \ldots, \lambda_n))$$
Swapping: The XOR Trick

Consider the (probabilistic) sketch for swapping \( x \) and \( y \):

\[
\begin{align*}
&[\text{choose}]^1 \lambda_{1,1} : S_1 \text{ or ... or } \lambda_{1,n} : S_n \text{ ro;} \\
&[\text{choose}]^2 \lambda_{2,1} : S_1 \text{ or ... or } \lambda_{2,n} : S_n \text{ ro;} \\
&[\text{choose}]^3 \lambda_{3,1} : S_1 \text{ or ... or } \lambda_{3,n} : S_n \text{ ro;} \\
\end{align*}
\]

with \( S_i \) one of \( i = 1, \ldots, 13 \) different elementary blocks:

\[
\begin{align*}
&[\text{skip}]^1 \\
&[x := y]^2 \quad [x := z]^3 \\
&[y := x]^4 \quad [y := z]^5 \\
&[z := x]^6 \quad [z := y]^7 \\
&[x := (x + y) \mod 2]^8 \quad [x := (x + z) \mod 2]^9 \\
&[y := (y + x) \mod 2]^10 \quad [y := (y + z) \mod 2]^11 \\
&[z := (z + x) \mod 2]^12 \quad [z := (z + y) \mod 2]^13
\end{align*}
\]
Swapping: The XOR Trick

Consider the (probabilistic) sketch for swapping $x$ and $y$:

\[
\begin{align*}
[\text{choose}]^1 & \lambda_{1,1} : S_1 \text{ or } \ldots \text{ or } \lambda_{1,n} : S_n \text{ ro}; \\
[\text{choose}]^2 & \lambda_{2,1} : S_1 \text{ or } \ldots \text{ or } \lambda_{2,n} : S_n \text{ ro}; \\
[\text{choose}]^3 & \lambda_{3,1} : S_1 \text{ or } \ldots \text{ or } \lambda_{3,n} : S_n \text{ ro};
\end{align*}
\]

with $S_i$ one of $i = 1, \ldots, 13$ different elementary blocks:

\[
\begin{align*}
[\text{skip}]^1 \\
[x := y]^2 & [x := z]^3 \\
[y := x]^4 & [y := z]^5 \\
[z := x]^6 & [z := y]^7 \\
[x := (x + y) \text{ mod } 2]^8 & [x := (x + z) \text{ mod } 2]^9 \\
[y := (y + x) \text{ mod } 2]^10 & [y := (y + z) \text{ mod } 2]^11 \\
[z := (z + x) \text{ mod } 2]^12 & [z := (z + y) \text{ mod } 2]^13
\end{align*}
\]
Swapping: Parameterised LOS and Objective

Using 13 transfer functions $F_1 \ldots F_{13}$ to define

$$T(\lambda_{ij}) = \prod_{i=1}^{3} T_i(\lambda_{ij}) \quad \text{with} \quad T_i(\lambda_{ij}) = \sum_{j=1}^{13} \lambda_{ij} F_j$$

For one-bit variables $x$, $y$ the intended behaviour (on $\mathbb{R}^2 \otimes \mathbb{R}^2$):

$$S = \begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1
\end{pmatrix} \quad x \mapsto 0 \quad y \mapsto 0 \quad x \mapsto 0 \quad y \mapsto 1 \quad x \mapsto 1 \quad y \mapsto 0 \quad x \mapsto 1 \quad y \mapsto 1$$

Objective: $\min \Phi_{00}(\lambda_{ij}) = \|A^\dagger T(\lambda_{ij})A - S\|_2$ or $\min \Phi_{\rho\omega}(\lambda_{ij})$ which also penalises for reading or writing to $z$; using the abstraction $A = I_{(4)} \otimes A_{f(2)} = \text{diag}(1, 1, 1, 1) \otimes (1, 1)^t$.  

Bolzano, 22-26 August 2016 ESSLLI’16 Probabilistic Program Analysis Slide 51 of 54
Swapping: Parameterised LOS and Objective

Using 13 transfer functions $F_1 \ldots F_{13}$ to define

$$T(\lambda_{ij}) = \prod_{i=1}^{3} T_i(\lambda_{ij}) \quad \text{with} \quad T_i(\lambda_{ij}) = \sum_{j=1}^{13} \lambda_{ij} F_j$$

For one-bit variables $x, y$ the intended behaviour (on $\mathbb{R}^2 \otimes \mathbb{R}^2$):

$$S = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$x \mapsto 0 \quad y \mapsto 0$$

$$x \mapsto 0 \quad y \mapsto 1$$

$$x \mapsto 1 \quad y \mapsto 0$$

$$x \mapsto 1 \quad y \mapsto 1$$

Objective: $\min \phi_{00}(\lambda_{ij}) = \|A^\dagger T(\lambda_{ij})A - S\|_2$ or $\min \phi_{\rho\omega}(\lambda_{ij})$ which also penalises for reading or writing to $z$; using the abstraction $A = I(4) \otimes A_f(2) = \text{diag}(1, 1, 1, 1) \otimes (1, 1)^t$. 
Swapping: Parameterised LOS and Objective

Using 13 transfer functions $F_1 \ldots F_{13}$ to define

$$T(\lambda_{ij}) = \prod_{i=1}^{3} T_i(\lambda_{ij}) \quad \text{with} \quad T_i(\lambda_{ij}) = \sum_{j=1}^{13} \lambda_{ij} F_j$$

For one-bit variables $x, y$ the intended behaviour (on $\mathbb{R}^2 \otimes \mathbb{R}^2$):

$$S = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$x \mapsto 0$  $y \mapsto 0$

$x \mapsto 0$  $y \mapsto 1$

$x \mapsto 1$  $y \mapsto 0$

$x \mapsto 1$  $y \mapsto 1$

Objective: $\min \Phi_{00}(\lambda_{ij}) = \|A^\dagger T(\lambda_{ij})A - S\|_2$ or $\min \Phi_{\rho\omega}(\lambda_{ij})$ which also penalises for reading or writing to $z$; using the abstraction $A = I_{(4)} \otimes A_{f(2)} = \text{diag}(1, 1, 1, 1) \otimes (1, 1)^t$. 

Bolzano, 22-26 August 2016  ESSLLI’16  Probabilistic Program Analysis  Slide 51 of 54
Swapping: Test Runs

Using \texttt{octave}: if we start with a swap which uses \( z \), like

\[
[z := x]^6; [x := y]^2; [y := z]^5
\]

represented by \( \lambda_{ij} \) given as:

\[
\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
\]

For \( \min \Phi_{00} \) we get no change; but with \( \min \Phi_{11} \) (after 12 iterations) we get with \texttt{octave} the optimal \( \lambda_{ij} \)'s:

\[
\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0
\end{pmatrix}
\]

This corresponds to the program:

\[
[y := (y+x) \mod 2]^10; [x := (x+y) \mod 2]^8; [y := (y+x) \mod 2]^10
\]
Swapping: Test Runs

Using \texttt{octave}: if we start with a swap which uses $z$, like

\[
[z := x]^6; [x := y]^2; [y := z]^5
\]

represented by $\lambda_{ij}$ given as:

\[
\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
\]

For min $\Phi_{00}$ we get no change; but with min $\Phi_{11}$ (after 12 iterations) we get with \texttt{octave} the optimal $\lambda_{ij}$'s:

\[
\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
\]

This corresponds to the program:

\[
[y := (y+x) \mod 2]^10; [x := (x+y) \mod 2]^8; [y := (y+x) \mod 2]^10
\]
Swapping: Test Runs

Using \textit{octave}: if we start with a swap which uses \( z \), like

\[
[z := x]^6; \quad [x := y]^2; \quad [y := z]^5
\]

represented by \( \lambda_{ij} \) given as:

\[
\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
\]

For \( \min \Phi_{00} \) we get no change; but with \( \min \Phi_{11} \) (after 12 iterations) we get with \textit{octave} the optimal \( \lambda_{ij} \)'s:

\[
\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
\]

This corresponds to the program:

\[
[y := (y+x) \mod 2]^10; \quad [x := (x+y) \mod 2]^8; \quad [y := (y+x) \mod 2]^10
\]
Swapping: Test Runs

Using **octave**: if we start with a swap which uses \( z \), like

\[
[z := x]^{6}; \ [x := y]^{2}; \ [y := z]^{5}
\]

represented by \( \lambda_{ij} \) given as:

\[
\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
\]

For \( \min \Phi_{00} \) we get no change; but with \( \min \Phi_{11} \) (after 12 iterations) we get with **octave** the optimal \( \lambda_{ij} \)'s:

\[
\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0
\end{pmatrix}
\]

This corresponds to the program:

\[
[y := (y+x) \mod 2]^{10}; \ [x := (x+y) \mod 2]^{8}; \ [y := (y+x) \mod 2]^{10}
\]
Swapping: Test Runs

Using \texttt{octave}: if we start with a swap which uses \( z \), like

\[
[z := x]^6; [x := y]^2; [y := z]^5
\]

represented by \( \lambda_{ij} \) given as:

\[
\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
\]

For \( \min \Phi_{00} \) we get no change; but with \( \min \Phi_{11} \) (after 12 iterations) we get with \texttt{octave} the optimal \( \lambda_{ij} \)’s:

\[
\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0
\end{pmatrix}
\]

This corresponds to the program:

\[
[y := (y+x) \mod 2]^10; [x := (x+y) \mod 2]^8; [y := (y+x) \mod 2]^10
\]
Swapping: Test Runs

For randomly chosen initial values for $\lambda_{ij}$:

\[
\begin{pmatrix}
.70 & .30 & .72 & .84 & .51 & .70 & .76 & .47 & .63 & .63 & .93 & .55 & .68 \\
.74 & .22 & .37 & .70 & .67 & .13 & .93 & .69 & .30 & .88 & .03 & .52 & .80 \\
.59 & .49 & .01 & .69 & .22 & .23 & .10 & .01 & .10 & .22 & .03 & .55 & .11
\end{pmatrix}
\]

For $\min \Phi_{11}$ (after 9 iterations) we get the optimal $\lambda_{ij}$’s:

\[
\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0
\end{pmatrix}
\]

This corresponds to the program:

\[
[y := (y+x) \mod 2]^{10}; \ [x := (x+y) \mod 2]^{8}; \ [y := (y+x) \mod 2]^{10}
\]

For $\Phi_{00}$ we may also get: $[z := x]^6; \ [x := y]^2; \ [y := z]^5$. 
Swapping: Test Runs

For randomly chosen initial values for $\lambda_{ij}$:

\[
\begin{pmatrix}
0.70 & 0.30 & 0.72 & 0.84 & 0.51 & 0.70 & 0.76 & 0.47 & 0.63 & 0.63 & 0.93 & 0.55 & 0.68 \\
0.74 & 0.22 & 0.37 & 0.70 & 0.67 & 0.13 & 0.93 & 0.69 & 0.30 & 0.88 & 0.03 & 0.52 & 0.80 \\
0.59 & 0.49 & 0.01 & 0.69 & 0.22 & 0.23 & 0.10 & 0.01 & 0.10 & 0.22 & 0.03 & 0.55 & 0.11
\end{pmatrix}
\]

For $\min \Phi_{11}$ (after 9 iterations) we get the optimal $\lambda_{ij}$’s:

\[
\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0
\end{pmatrix}
\]

This corresponds to the program:

\[
[y := (y+x) \mod 2]^\text{10}; \ [x := (x+y) \mod 2]^\text{8}; \ [y := (y+x) \mod 2]^\text{10}
\]

For $\Phi_{00}$ we may also get: $[z := x]^6; \ [x := y]^2; \ [y := z]^5$. 

Swapping: Test Runs

For randomly chosen initial values for $\lambda_{ij}$:

$\begin{pmatrix}
.70 & .30 & .72 & .84 & .51 & .70 & .76 & .47 & .63 & .63 & .93 & .55 & .68 \\
.74 & .22 & .37 & .70 & .67 & .13 & .93 & .69 & .30 & .88 & .03 & .52 & .80 \\
.59 & .49 & .01 & .69 & .22 & .23 & .10 & .01 & .10 & .22 & .03 & .55 & .11
\end{pmatrix}$

For $\min \Phi_{11}$ (after 9 iterations) we get the optimal $\lambda_{ij}$’s:

$\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0
\end{pmatrix}$

This corresponds to the program:

$[y := (y+x) \mod 2]^{10}; \ [x := (x+y) \mod 2]^{8}; \ [y := (y+x) \mod 2]^{10}$

For $\Phi_{00}$ we may also get: $[z := x]^6; \ [x := y]^2; \ [y := z]^5$. 
Swapping: Test Runs

For randomly chosen initial values for $\lambda_{ij}$:

\[
\begin{pmatrix}
0.70 & 0.30 & 0.72 & 0.84 & 0.51 & 0.70 & 0.76 & 0.47 & 0.63 & 0.63 & 0.93 & 0.55 & 0.68 \\
0.74 & 0.22 & 0.37 & 0.70 & 0.67 & 0.13 & 0.93 & 0.69 & 0.30 & 0.88 & 0.03 & 0.52 & 0.80 \\
0.59 & 0.49 & 0.01 & 0.69 & 0.22 & 0.23 & 0.10 & 0.01 & 0.10 & 0.22 & 0.03 & 0.55 & 0.11 
\end{pmatrix}
\]

For $\min \Phi_{11}$ (after 9 iterations) we get the optimal $\lambda_{ij}$’s:

\[
\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
\end{pmatrix}
\]

This corresponds to the program:

\[
[y := (y+x) \mod 2]^{10}; \ [x := (x+y) \mod 2]^{8}; \ [y := (y+x) \mod 2]^{10}
\]

For $\Phi_{00}$ we may also get: $[z := x]^6; \ [x := y]^2; \ [y := z]^5$. 
Some References

- Di Pierro, Wiklicky: A logico-algebraic approach to probabilistic program analysis Pre-Proceedings of LOPSTR’05.
Some References

- Di Pierro, Wiklicky: A logico-algebraic approach to probabilistic program analysis Pre-Proceedings of LOPSTR’05.
Some References

- Di Pierro, Wiklicky: A logico-algebraic approach to probabilistic program analysis Pre-Proceedings of LOPSTR’05.


Some References

- Di Pierro, Wiklicky: A logico-algebraic approach to probabilistic program analysis Pre-Proceedings of LOPSTR’05.