A Framework for Quantitative Program Synthesis

Herbert Wiklicky
Imperial College London
h.wiklicky@imperial.ac.uk

QMUL, 4 May 2016
The terms **analysis** and **synthesis** are central in:

- Chemistry
- Mathematics
- Biology (Synthetic Biology)
- Computer Science (Program Synthesis)

**Dream**: Generate and maintain programs automatically (create and change to optimise performance, security, etc.). But…

*When someone says, "I want a programming language in which I need only say what I want done," give him a lollipop.* – Alan Perlis
The terms analysis and synthesis are central in:

- Chemistry
- Mathematics
- Biology (Synthetic Biology)
- Computer Science (Program Synthesis)

Dream: Generate and maintain programs automatically (create and change to optimise performance, security, etc.). But...

When someone says, "I want a programming language in which I need only say what I want done," give him a lollipop. – Alan Perlis
The terms **analysis** and **synthesis** are central in:

- Chemistry
- Mathematics
- Biology (Synthetic Biology)
- Computer Science (Program Synthesis)

**Dream:** Generate and maintain programs automatically (create and change to optimise performance, security, etc.). But...

*When someone says, "I want a programming language in which I need only say what I want done," give him a lollipop.* – Alan Perlis
Analysis and Synthesis

The terms analysis and synthesis are central in:

- Chemistry
- Mathematics
- Biology (Synthetic Biology)
- Computer Science (Program Synthesis)

Dream: Generate and maintain programs automatically (create and change to optimise performance, security, etc.). But...

When someone says, "I want a programming language in which I need only say what I want done," give him a lollipop. – Alan Perlis
The terms **analysis** and **synthesis** are central in:

- Chemistry
- Mathematics
- Biology (Synthetic Biology)
- Computer Science (Program Synthesis)

**Dream**: Generate and maintain programs automatically (create and change to optimise performance, security, etc.). But...

> When someone says, "I want a programming language in which I need only say what I want done," give him a lollipop. – Alan Perlis
The terms **analysis** and **synthesis** are central in:

- Chemistry
- Mathematics
- Biology (Synthetic Biology)
- Computer Science (Program Synthesis)

**Dream:** Generate and maintain programs automatically (create and change to optimise performance, security, etc.).

*When someone says, "I want a programming language in which I need only say what I want done," give him a lollipop.* — Alan Perlis
The terms **analysis** and **synthesis** are central in:

- Chemistry
- Mathematics
- Biology (Synthetic Biology)
- Computer Science (Program Synthesis)

**Dream**: Generate and maintain programs automatically (create and change to optimise performance, security, etc.). But...

When someone says, "I want a programming language in which I need only say what I want done," give him a lollipop. – Alan Perlis
Optimisation

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).
Allow for probabilistic choices and/or random assignments:

\[ S ::= [\text{skip}]^\ell \]
\[ [x := f(x_1, \ldots, x_n)]^\ell \]
\[ [x \?= \rho]^\ell \]
\[ S_1; S_2 \]
\[ [\text{choose}]^\ell p_1 : S_1 \text{ or } p_2 : S_2 \]
\[ \text{if } [b]^\ell \text{ then } S_1 \text{ else } S_2 \text{ fi} \]
\[ \text{while } [b]^\ell \text{ do } S \text{ od} \]

Make the probabilistic choices at runtime – not at design or compile time. But, …
A Probabilistic Language

Allow for probabilistic choices and/or random assignments:

\[
S ::= \begin{array}{l}
[\text{skip}]^\ell \\
[x := f(x_1, \ldots, x_n)]^\ell \\
[x \neq \rho]^\ell \\
S_1; S_2 \\
\text{choose}^\ell p_1 : S_1 \text{ or } p_2 : S_2 \text{ ro} \\
\text{if } [b]^\ell \text{ then } S_1 \text{ else } S_2 \text{ fi} \\
\text{while } [b]^\ell \text{ do } S \text{ od}
\end{array}
\]

Make the probabilistic choices at runtime – not at design or compile time. But, ...
A Probabilistic Language

Allow for probabilistic choices and/or random assignments:

\[ S ::= \begin{cases} 
[\text{skip}]^\ell \\
[x := f(x_1, \ldots, x_n)]^\ell \\
[x \?= \rho]^\ell \\
S_1; S_2 \\
[\text{choose}]^\ell p_1 : S_1 \text{ or } p_2 : S_2 \text{ ro} \\
\text{if } [b]^\ell \text{ then } S_1 \text{ else } S_2 \text{ fi} \\
\text{while } [b]^\ell \text{ do } S \text{ od} 
\end{cases} \]

Make the probabilistic choices at runtime – not at design or compile time. But, …
A Probabilistic Language

Allow for probabilistic choices and/or random assignments:

\[
S ::= \begin{array}{c}
[\text{skip}]^\ell \\
[x := f(x_1, \ldots, x_n)]^\ell \\
[x ?= \rho]^\ell \\
S_1; S_2 \\
[\text{choose}]^\ell p_1 : S_1 \text{ or } p_2 : S_2 \text{ ro} \\
\text{if } [b]^\ell \text{ then } S_1 \text{ else } S_2 \text{ fi} \\
\text{while } [b]^\ell \text{ do } S \text{ od}
\end{array}
\]

Make the probabilistic choices at runtime – not at design or compile time. But, …
Example: Cowboys Who Learn

\[ a := 0; \]
choose \( f : \{ t := 0 \} \text{ or } (1 - f) : \{ t := 1 \} \) ro;
\[ c := 1; \]
while \((c = 1)\) do
  if \((t = 0)\)
    then
      choose \( a : \{ c := 0 \} \text{ or } (1 - a) : \{ t := 1 \} \) ro
    else
      choose \( b : \{ c := 0 \} \text{ or } (1 - b) : \{ t := 0 \} \) ro
    fi;
  \[ a := \max(a + \frac{1}{10}, 1); \]
  od
Linear Operator Semantics

LOS

Linear Operator Semantics (LOS)

Program and program fragments are given a semantics in terms of linear operators on probabilistic state space.

We model state updates as well as control flow steps via program counter updates (labels), typically for $[S]^i$:

$$T_i = F_i \otimes E(i, f).$$

We get the generator of a (Discrete Time) Markov Chain:

$$T(P) = \sum_i \{T_i \mid T_i \in [P]\}$$

Linear Operator Semantics (LOS)

Program and program fragments are given a semantics in terms of linear operators on probabilistic state space.

We model state updates as well as control flow steps via program counter updates (labels), typically for $[S]^i$:

$$T_i = F_i \otimes E(i, f).$$

We get the generator of a (Discrete Time) Markov Chain:

$$T(P) = \sum_i \{T_i \mid T_i \in [P]\}$$

Linear Operator Semantics (LOS)

Program and program fragments are given a semantics in terms of linear operators on probabilistic state space.

We model state updates as well as control flow steps via program counter updates (labels), typically for $[S]^i$:

$$T_i = F_i \otimes E(i, f).$$

We get the generator of a (Discrete Time) Markov Chain:

$$T(P) = \sum_i \{T_i \mid T_i \in [P]\}$$

Linear Operator Semantics (LOS)

Program and program fragments are given a semantics in terms of linear operators on probabilistic state space.

We model state updates as well as control flow steps via program counter updates (labels), typically for \([S]^i\):

\[ T_i = F_i \otimes E(i, f). \]

We get the generator of a (Discrete Time) Markov Chain:

\[ T(P) = \sum_i \{ T_i \mid T_i \in \llbracket P \rrbracket \} \]

Linear Operator Semantics (LOS)

Program and program fragments are given a semantics in terms of linear operators on probabilistic state space.

We model state updates as well as control flow steps via program counter updates (labels), typically for $[S]^i$:

$$T_i = F_i \otimes E(i, f).$$

We get the generator of a (Discrete Time) Markov Chain:

$$T(P) = \sum_i \{T_i \mid T_i \in \llbracket P \rrbracket\}$$

Probabilistic State Space

We need a (compositional) representation of probabilistic states and configurations (which include also labels).

**Classical state** $\sigma \in \text{State}$ given by:

$$s \in \text{State} = \{\text{Var} \rightarrow \text{Value}\} = \text{Value}^\nu$$

**Probabilistic state** $\sigma \in \text{Dist}($State$) = \mathcal{D}($State$) \subseteq \nu($State$)$:

$$\sigma \in \nu(\text{Var} \rightarrow \text{Value}) =$$

$$= \nu(\text{Value}_1 \times \text{Value}_2 \times \ldots \times \text{Value}_v) =$$

$$= \nu(\text{Value}_1) \otimes \nu(\text{Value}_2) \otimes \ldots \otimes \nu(\text{Value}_v)$$

Essential for this treatment is the fact that:

$$\nu(S \times S) = \nu(S) \otimes \nu(S)$$
Probabilistic State Space

We need a (compositional) representation of probabilistic states and configurations (which include also labels).

**Classical state** $\sigma \in \text{State}$ given by:

$$s \in \text{State} = (\text{Var} \rightarrow \text{Value}) = \text{Value}^V$$

**Probabilistic state** $\sigma \in \text{Dist(State)} = \mathcal{D}(\text{State}) \subseteq \mathcal{V}(\text{State})$:

$$\sigma \in \mathcal{V}(\text{Var} \rightarrow \text{Value}) =$$

$$= \mathcal{V}(\text{Value}_1 \times \text{Value}_2 \times \ldots \times \text{Value}_v) =$$

$$= \mathcal{V}(\text{Value}_1) \otimes \mathcal{V}(\text{Value}_2) \otimes \ldots \otimes \mathcal{V}(\text{Value}_v)$$

Essential for this treatment is the fact that:

$$\mathcal{V}(S \times S) = \mathcal{V}(S) \otimes \mathcal{V}(S)$$
Probabilistic State Space

We need a (compositional) representation of probabilistic states and configurations (which include also labels).

Classical state $\sigma \in \text{State}$ given by:

$$s \in \text{State} = (\text{Var} \rightarrow \text{Value}) = \text{Value}^V$$

Probabilistic state $\sigma \in \text{Dist(State)} = \mathcal{D}(\text{State}) \subseteq \mathcal{V}(\text{State})$:

$$\sigma \in \mathcal{V}(\text{Var} \rightarrow \text{Value}) =$$

$$= \mathcal{V}(\text{Value}_1 \times \text{Value}_2 \times \ldots \times \text{Value}_v) =$$

$$= \mathcal{V}(\text{Value}_1) \otimes \mathcal{V}(\text{Value}_2) \otimes \ldots \otimes \mathcal{V}(\text{Value}_v)$$

Essential for this treatment is the fact that:

$$\mathcal{V}(S \times S) = \mathcal{V}(S) \otimes \mathcal{V}(S)$$
Probabilistic State Space

We need a (compositional) representation of probabilistic states and configurations (which include also labels).

**Classical state** \( \sigma \in \text{State} \) given by:

\[
s \in \text{State} = (\text{Var} \rightarrow \text{Value}) = \text{Value}^V
\]

**Probabilistic state** \( \sigma \in \text{Dist(State)} = \mathcal{D}(\text{State}) \subseteq \mathcal{V}(\text{State}) \):

\[
\sigma \in \mathcal{V}(\text{Var} \rightarrow \text{Value}) = \\
= \mathcal{V}(\text{Value}_1 \times \text{Value}_2 \times \ldots \times \text{Value}_V) = \\
= \mathcal{V}(\text{Value}_1) \otimes \mathcal{V}(\text{Value}_2) \otimes \ldots \otimes \mathcal{V}(\text{Value}_V)
\]

Essential for this treatment is the fact that:

\[
\mathcal{V}(S \times S) = \mathcal{V}(S) \otimes \mathcal{V}(S)
\]
Probabilistic State Space

We need a (compositional) representation of probabilistic states and configurations (which include also labels).

Classical state $\sigma \in \text{State}$ given by:

$$s \in \text{State} = (\text{Var} \rightarrow \text{Value}) = \text{Value}^V$$

Probabilistic state $\sigma \in \text{Dist(State)} = \mathcal{D} (\text{State}) \subseteq \mathcal{V} (\text{State})$:

$$\sigma \in \mathcal{V} (\text{Var} \rightarrow \text{Value}) =$$

$$= \mathcal{V} (\text{Value}_1 \times \text{Value}_2 \times \ldots \times \text{Value}_v) =$$

$$= \mathcal{V} (\text{Value}_1) \otimes \mathcal{V} (\text{Value}_2) \otimes \ldots \otimes \mathcal{V} (\text{Value}_v)$$

Essential for this treatment is the fact that:

$$\mathcal{V} (S \times S) = \mathcal{V} (S) \otimes \mathcal{V} (S)$$
Probabilistic State Space

We need a (compositional) representation of probabilistic states and configurations (which include also labels).

Classical state \( \sigma \in \text{State} \) given by:

\[
\sigma \in \text{State} = (\text{Var} \rightarrow \text{Value}) = \text{Value}^V
\]

Probabilistic state \( \sigma \in \text{Dist}(\text{State}) = \mathcal{D}(\text{State}) \subseteq \mathcal{V}(\text{State}) \):

\[
\sigma \in \mathcal{V}(\text{Var} \rightarrow \text{Value}) = \\
= \mathcal{V}(\text{Value}_1 \times \text{Value}_2 \times \ldots \times \text{Value}_v) = \\
= \mathcal{V}(\text{Value}_1) \otimes \mathcal{V}(\text{Value}_2) \otimes \ldots \otimes \mathcal{V}(\text{Value}_v)
\]

Essential for this treatment is the fact that:

\[
\mathcal{V}(S \times S) = \mathcal{V}(S) \otimes \mathcal{V}(S)
\]
Tensor or Kronecker Product

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

$$
\mathbf{A} = \begin{pmatrix}
a_{11} & \ldots & a_{1m} \\
\vdots & \ddots & \vdots \\
a_{n1} & \ldots & a_{nm}
\end{pmatrix} \quad \mathbf{B} = \begin{pmatrix}
b_{11} & \ldots & b_{1l} \\
\vdots & \ddots & \vdots \\
b_{k1} & \ldots & b_{kl}
\end{pmatrix}
$$

The tensor or Kronecker product $\mathbf{A} \otimes \mathbf{B}$ is a $nk \times ml$ matrix:

$$
\mathbf{A} \otimes \mathbf{B} = \begin{pmatrix}
a_{11} \mathbf{B} & \ldots & a_{1m} \mathbf{B} \\
\vdots & \ddots & \vdots \\
a_{n1} \mathbf{B} & \ldots & a_{nm} \mathbf{B}
\end{pmatrix}
$$

Special cases are square matrices ($n = m$ and $k = l$) and vectors (row $n = k = 1$, column $m = l = 1$).
Given a $n \times m$ matrix $A$ and a $k \times l$ matrix $B$:

$$
A = \begin{pmatrix}
  a_{11} & \cdots & a_{1m} \\
  \vdots & \ddots & \vdots \\
  a_{n1} & \cdots & a_{nm}
\end{pmatrix} \quad \quad \quad
B = \begin{pmatrix}
  b_{11} & \cdots & b_{1l} \\
  \vdots & \ddots & \vdots \\
  b_{k1} & \cdots & b_{kl}
\end{pmatrix}
$$

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

$$
A \otimes B = \begin{pmatrix}
  a_{11}B & \cdots & a_{1m}B \\
  \vdots & \ddots & \vdots \\
  a_{n1}B & \cdots & a_{nm}B
\end{pmatrix}
$$

Special cases are square matrices ($n = m$ and $k = l$) and vectors (row $n = k = 1$, column $m = l = 1$).
Given a $n \times m$ matrix $A$ and a $k \times l$ matrix $B$:

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nm} \end{pmatrix} \quad \quad B = \begin{pmatrix} b_{11} & \cdots & b_{1l} \\ \vdots & \ddots & \vdots \\ b_{k1} & \cdots & b_{kl} \end{pmatrix}$$

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

$$A \otimes B = \begin{pmatrix} a_{11}B & \cdots & a_{1m}B \\ \vdots & \ddots & \vdots \\ a_{n1}B & \cdots & a_{nm}B \end{pmatrix}$$

Special cases are square matrices ($n = m$ and $k = l$) and vectors (row $n = k = 1$, column $m = l = 1$).
Example: Randomised Counting

Randomised Counting

\[ \begin{align*}
[c := 1]^{1} & ; [i := 0]^{2} ; \\
\text{while } [c > 0]^{3} \text{ do } & \\
[\text{choose}]^{4} \frac{1}{2} : [i := i + 1]^{5} \text{ or } \frac{1}{2} : [c := 0]^{6} \text{ od;} \\
[\text{skip}]^{7}
\end{align*} \]

LOS Semantics constructed from the set of operators:

\[ \begin{align*}
\llbracket P \rrbracket & = \{ \ F_{1} \otimes E(1, 2), F_{2} \otimes E(2, 3), \\
& P(c > 0) \otimes E(3, 4), P(c > 0) \bot \otimes E(3, 7), \\
& \frac{1}{2} \cdot I \otimes E(4, 5), \frac{1}{2} \cdot I \otimes E(4, 6), \\
& F_{5} \otimes E(5, 3), F_{6} \otimes E(6, 3), \\
& I \otimes E(7, 7) \} \}
\end{align*} \]
Example: Randomised Counting

Randomised Counting

\[
\begin{align*}
&[c := 1]; [i := 0]; \\
&\text{while } [c > 0] \text{ do} \\
&\quad \left[\text{choose}\right] \frac{1}{2} : [i := i + 1] \text{ or } \frac{1}{2} : [c := 0] \text{ od;}
\end{align*}
\]

[skip]

LOS Semantics constructed from the set of operators:

\[
\begin{align*}
\sem{P} &= \{ \\
&F_1 \otimes E(1, 2), F_2 \otimes E(2, 3), \\
&P(c > 0) \otimes E(3, 4), P(c > 0) \bot \otimes E(3, 7), \\
&\frac{1}{2} \cdot I \otimes E(4, 5), \frac{1}{2} \cdot I \otimes E(4, 6), \\
&F_5 \otimes E(5, 3), F_6 \otimes E(6, 3), \\
&I \otimes E(7, 7) \}
\end{align*}
\]
A Tool: The pWhile "Compiler"

The \texttt{pwc} tool – implemented using \texttt{ocaml} – produces \texttt{octave} scripts which allow the explicit construction of the concrete LOS operators (as quite large sparse matrices).

\begin{center}
\begin{tikzpicture}
  \node [circle, fill=red!30] (1) at (0,0) {prog.pw};
  \node [circle, fill=blue!30] (2) at (2,0) {pwc Compiler};
  \node [circle, fill=red!30] (3) at (4,0) {prog.m};
  \node [circle, fill=red!30] (4) at (4,-2) {hdf5, etc.};
  \node [circle, fill=red!30] (5) at (4,2) {LOS.m};
  \node [circle, fill=red!30] (6) at (2,-2) {Flow, etc.};
  \node [circle, fill=blue!30] (7) at (2,2) {Octave System};
  \draw [->] (1) -- (2);
  \draw [->] (2) -- (3);
  \draw [->] (3) -- (7);
  \draw [->] (7) -- (4);
  \draw [->] (7) -- (5);
\end{tikzpicture}
\end{center}
Probabilistic Abstract Interpretation
PAI

Probabilistic Abstract Interpretation (PAI)

There is the need to “simplify” or “abstract” the semantics in order to have a feasible object to investigate and analyse.

Take an abstraction $A : C \rightarrow D$, construct concretisation $G : D \rightarrow C$ to obtain a “smaller” abstract semantics:

$$T(P)\# = GT(P)A = A^\dagger T(P)A.$$

With $A^\dagger$ the Moore-Penrose Pseudo-Inverse of $A$ we get a Least Square Approximation. Instances: Graph Isomorphism for $A$ permutations; Bisimulation for $A$ classifications.

- Abstractions compatible with sum and tensor product:
  $$(\bigotimes_i A_i^\dagger)(\sum_{i,j} \bigotimes_i F_i)(\bigotimes_i A_i) = \sum_{ij} \bigotimes_i (A_{i}^\dagger F_i A),$$
- Average of averages is not the average: $(A_1 A_2)^\dagger \neq A_2^\dagger A_1^\dagger$. 
Probabilistic Abstract Interpretation (PAI)

There is the need to “simplify” or “abstract” the semantics in order to have a feasible object to investigate and analyse.

Take an abstraction $A : \mathcal{C} \rightarrow \mathcal{D}$, construct concretisation $G : \mathcal{D} \rightarrow \mathcal{C}$ to obtain a “smaller” abstract semantics:

$$T(P)^\# = GT(P)A = A^\dagger T(P)A.$$  

With $A^\dagger$ the Moore-Penrose Pseudo-Inverse of $A$ we get a Least Square Approximation. Instances: Graph Isomorphism for $A$ permutations; Bisimulation for $A$ classifications.

- Abstractions compatible with sum and tensor product:
  $$(\bigotimes_i A_{i}^\dagger) (\sum_{i,j} \bigotimes_i F_i) (\bigotimes_i A_i) = \sum_{i,j} \bigotimes_i (A_i^\dagger F_i A),$$

- Average of averages is not the average: $(A_1 A_2)^\dagger \neq A_2^\dagger A_1^\dagger$. 

h.wiklicky@imperial.ac.uk

Quantitative Synthesis
There is the need to “simplify” or “abstract” the semantics in order to have a feasible object to investigate and analyse.

Take an abstraction $A : C \rightarrow D$, construct concretisation $G : D \rightarrow C$ to obtain a “smaller” abstract semantics:

$$T(P)\# = GT(P)A = A^\dagger T(P)A.$$ 

With $A^\dagger$ the Moore-Penrose Pseudo-Inverse of $A$ we get a Least Square Approximation. Instances: Graph Isomorphism for $A$ permutations; Bisimulation for $A$ classifications.

- Abstractions compatible with sum and tensor product:
  $$\bigotimes_i A_i^\dagger \left( \sum_{i,j} \bigotimes_i F_{ij} \right) \bigotimes_i A_i = \sum_{ij} \bigotimes_i (A_i^\dagger F_{ij} A_i),$$
- Average of averages is not the average:
  $$(A_1 A_2)^\dagger \neq A_2^\dagger A_1^\dagger.$$
There is the need to “simplify” or “abstract” the semantics in order to have a feasible object to investigate and analyse.

Take an abstraction $A : \mathcal{C} \to \mathcal{D}$, construct concretisation $G : \mathcal{D} \to \mathcal{C}$ to obtain a “smaller” abstract semantics:

$$T(P)^\# = GT(P)A = A^\dagger T(P)A.$$ 

With $A^\dagger$ the Moore-Penrose Pseudo-Inverse of $A$ we get a Least Square Approximation. Instances: Graph Isomorphism for $A$ permutations; Bisimulation for $A$ classifications.

- Abstractions compatible with sum and tensor product:
  $$\left( \bigotimes_i A_i^\dagger \right) \left( \sum_{i,j} \bigotimes_i F_i \right) \left( \bigotimes_i A_i \right) = \sum_{ij} \bigotimes_i (A^\dagger F_i A),$$

- Average of averages is not the average: $(A_1 A_2)^\dagger \neq A_2^\dagger A_1^\dagger$. 

h.wiklicky@imperial.ac.uk  
Quantitative Synthesis
Probabilistic Abstract Interpretation (PAI)

There is the need to “simplify” or “abstract” the semantics in order to have a feasible object to investigate and analyse.

Take an abstraction $A : \mathcal{C} \rightarrow \mathcal{D}$, construct concretisation $G : \mathcal{D} \rightarrow \mathcal{C}$ to obtain a “smaller” abstract semantics:

$$T(P)^\# = GT(P)A = A^\dagger T(P)A.$$ 

With $A^\dagger$ the Moore-Penrose Pseudo-Inverse of $A$ we get a Least Square Approximation. Instances: Graph Isomorphism for $A$ permutations; Bisimulation for $A$ classifications.

- Abstractions compatible with sum and tensor product:
  $$\bigotimes_i (A_i^\dagger) \bigotimes_i (F_i \bigotimes_i A_i) = \bigotimes_{ij} (A_i^\dagger F_i A_i),$$

- Average of averages is not the average: $(A_1 A_2)^\dagger \neq A_2^\dagger A_1^\dagger$. 
Approximations: Over, Under, Least Square

Abstraction: $\mathcal{T}_{16} \rightarrow \mathcal{T}_4$
Approximations: Over, Under, Least Square

Abstraction: $\mathcal{T}_{16} \rightarrow \mathcal{T}_4$
Approximations: Over, Under, Least Square

Abstraction: $T_{16} \rightarrow T_{4}$
Approximations: Over, Under, Least Square

Abstraction: $\mathcal{T}_{16} \rightarrow \mathcal{T}_4$
Example: Forgetful Abstraction

Forgetful Abstraction operator on $\mathcal{V}([1, \ldots, n]) \rightarrow \mathcal{V}([\ast])$:

$$A_f = \begin{pmatrix} 1 \\ 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix} \quad A_f^\dagger = \left( \frac{1}{n} \quad \frac{1}{n} \quad \frac{1}{n} \quad \frac{1}{n} \quad \ldots \quad \frac{1}{n} \right)$$
Example: Forgetful Abstraction

Forgetful Abstraction operator on $\mathcal{V}(\{1, \ldots, n\}) \rightarrow \mathcal{V}(\{\ast\})$:

$$A_f = \begin{pmatrix} 1 \\ 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix} \quad A_f^\dagger = \begin{pmatrix} 1/n \\ 1/n \\ 1/n \\ \ldots \\ 1/n \end{pmatrix}$$
Example: Parity Abstraction

Parity Abstraction $\mathcal{V}(\{1, \ldots, n\}) \rightarrow \mathcal{V}(\{o, e\})$ (with $n$ even):

$A_p = \begin{pmatrix}
1 & 0 \\
0 & 1 \\
1 & 0 \\
0 & 1 \\
\vdots & \vdots \\
0 & 1 \\
\end{pmatrix}$

$A_p^\dagger = \begin{pmatrix}
\frac{2}{n} & 0 & \frac{2}{n} & 0 & \cdots & 0 \\
0 & \frac{2}{n} & 0 & \frac{2}{n} & \cdots & 0 \\
\end{pmatrix}$
Parity Abstraction $\mathcal{V}(\{1, \ldots, n\}) \rightarrow \mathcal{V}(\{o, e\})$ (with $n$ even):

$$A_p = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \\ \vdots & \vdots \\ 0 & 1 \end{pmatrix} \quad A_p^\dagger = \begin{pmatrix} 2/n & 0 & 2/n & 0 & \ldots & 0 \\ 0 & 2/n & 0 & 2/n & \ldots & 0 \end{pmatrix}$$
Example: Sign Abstraction

Sign Abstraction \( \nu(\{-n, \ldots, 0, \ldots, n\}) \rightarrow \nu(\{-, 0, +\}) \):

\[
A_s = \begin{pmatrix}
1 & 0 & 0 \\
\vdots & \vdots & \vdots \\
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1 \\
\vdots & \vdots & \vdots \\
0 & 0 & 1
\end{pmatrix}
\]

\[
A^\dagger_s = \begin{pmatrix}
\frac{1}{n} & \cdots & \frac{1}{n} & 0 & 0 & \cdots & 0 \\
0 & \cdots & 0 & 1 & 0 & \cdots & 0 \\
0 & \cdots & 0 & 0 & \frac{1}{n} & \cdots & \frac{1}{n}
\end{pmatrix}
\]
Example: Sign Abstraction

Sign Abstraction $\nu(\{-n, \ldots, 0, \ldots, n\}) \rightarrow \nu(\{-, 0, +\})$:

\[
A_s = \begin{pmatrix}
1 & 0 & 0 \\
\vdots & \vdots & \vdots \\
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1 \\
\vdots & \vdots & \vdots \\
0 & 0 & 1
\end{pmatrix}
\]

\[
A_s^\dagger = \begin{pmatrix}
\frac{1}{n} & \ldots & \frac{1}{n} & 0 & 0 & \ldots & 0 \\
0 & \ldots & 0 & 1 & 0 & \ldots & 0 \\
0 & \ldots & 0 & 0 & \frac{1}{n} & \ldots & \frac{1}{n}
\end{pmatrix}
\]
Synthesis and Optimisation

Approach: Don’t try to generate a whole program, introduce ‘design choices’ (holes) into a sketch of a program.

```c
int W = 32;
void main(bit[W] x, bit[W] y) {
    bit[W] xold = x;
    bit[W] yold = y;
    if(??) { x = x ^ y; } else { y = x ^ y; } 
    if(??) { x = x ^ y; } else { y = x ^ y; } 
    if(??) { x = x ^ y; } else { y = x ^ y; } 
    assert y == xold && x == yold; 
}
```

A Probabilistic “Sketching” Framework

Implementation Language: see above

Specification Language: for probabilistic sketches

\[ S ::= [\text{skip}]^\ell \]
\[ [x := f(x_1, \ldots, x_n)]^\ell \]
\[ [x \mathbin{=} \rho]^\ell \]
\[ S_1 ; S_2 \]
\[ [\text{choose}]^\ell p_1 : S_1 \text{ or } p_2 : S_2 \text{ ro} \]
\[ \text{if } [b]^\ell \text{ then } S_1 \text{ else } S_2 \text{ fi} \]
\[ \text{while } [b]^\ell \text{ do } S \text{ od} \]

Assertions and Requirements: via abstract properties (PAI).

Implementation: \( p_i \) constants (or program variables)
Specification: \( p_i \) (mathematical) variables, i.e. \( \lambda_i \)
A Probabilistic “Sketching” Framework

Implementation Language: see above
Specification Language: for probabilistic sketches

\[
S ::= \begin{array}{l}
[\text{skip}]^l \\
[x := f(x_1, \ldots, x_n)]^l \\
[x := \rho]^l \\
S_1; S_2 \\
[\text{choose}]^l p_1 : S_1 \text{ or } p_2 : S_2 \\
\text{if } [b]^l \text{ then } S_1 \text{ else } S_2 \\
\text{while } [b]^l \text{ do } S \text{ od}
\end{array}
\]

Assertions and Requirements: via abstract properties (PAI).

Implementation: \( p_i \) constants (or program variables)
Specification: \( p_i \) (mathematical) variables, i.e. \( \lambda_i \)

h.wiklicky@imperial.ac.uk

Quantitative Synthesis
A Probabilistic “Sketching” Framework

Implementation Language: see above
Specification Language: for probabilistic sketches

\[
S ::= \begin{align*}
& [\text{skip}]^\ell \\
& [x := f(x_1, \ldots, x_n)]^\ell \\
& [x \neq \rho]^\ell \\
& S_1; S_2 \\
& [\text{choose}]^\ell p_1 : S_1 \text{ or } p_2 : S_2 \text{ ro} \\
& \text{if } [b]^\ell \text{ then } S_1 \text{ else } S_2 \text{ fi} \\
& \text{while } [b]^\ell \text{ do } S \text{ od}
\end{align*}
\]

Assertions and Requirements: via abstract properties (PAI).

Implementation: \( p_i \) constants (or program variables)
Specification: \( p_i \) (mathematical) variables, i.e. \( \lambda_i \)
A Probabilistic “Sketching” Framework

Implementation Language: see above

Specification Language: for probabilistic sketches

\[ S ::= \begin{align*}
& \text{[skip]}^\ell \\
& \text{[x := } f(x_1, \ldots, x_n)]^\ell \\
& \text{[x ?= } \rho]^\ell \\
& S_1 ; S_2 \\
& \text{[choose]}^\ell p_1 : S_1 \text{ or } p_2 : S_2 \text{ ro} \\
& \text{if } [b]^\ell \text{ then } S_1 \text{ else } S_2 \text{ fi} \\
& \text{while } [b]^\ell \text{ do } S \text{ od}
\end{align*} \]

Assertions and Requirements: via abstract properties (PAI).

Implementation: \( p_i \) constants (or program variables)

Specification: \( p_i \) (mathematical) variables, i.e. \( \lambda_i \)
A General Approach

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

\[
...[\text{choose}]^\ell p_1 : S_1 \text{ or } \ldots \text{ or } p_n : S_n \text{ or } \ldots
\]

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

- 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

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

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

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

- Establish constraints on functional behaviour, e.g.

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

- 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

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

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

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

- 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.

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

- 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

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

- 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

$$\min_{\lambda_1, \lambda_2, \ldots, \lambda_n} \Phi(T(\lambda_1, \lambda_2, \ldots, \lambda_n))$$
Demo: Mini Example

Take the following simple program or probabilistic sketch:

\[
\text{var } x : \{-8..8\};
\begin{align*}
\text{begin} & \quad \text{opt } \{x := 0\} \text{ or } \{x := 1\} \text{ or } \{x := -1\} \text{ top; }
\text{end}
\end{align*}
\]

Synthesis requirement: \(x > 0\) at the end of the execution.

This probabilistic sketch is essentially equivalent to:

\[
\text{[choose]}^{\dagger} \lambda_1 : \{x:=0\} \text{ or } \lambda_2 : \{x:=1\} \text{ or } \lambda_3 : \{x:=-1\} \text{ ro.}
\]

The aim is to minimise

\[
\Phi(\lambda_1, \lambda_2, \lambda_3) = \|A_s^\dagger \cdot F_1(\lambda_1, \lambda_2, \lambda_3) \cdot A_s - S\|
\]

with \(S = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}\) on \(\mathcal{V}(\{-, 0, +\})\).
Take the following simple program or probabilistic sketch:

```plaintext
var x :{-8..8};
begin
  opt \{x := 0\} or \{x := 1\} or \{x := -1\} top;
end
```

Synthesis requirement: \( x > 0 \) at the end of the execution.

This probabilistic sketch is essentially equivalent to:

\[
\text{[choose]}^1 \lambda_1 : \{x := 0\} \text{ or } \lambda_2 : \{x := 1\} \text{ or } \lambda_3 : \{x := -1\} \text{ ro.}
\]

The aim is to minimise

\[
\Phi(\lambda_1, \lambda_2, \lambda_3) = \|A^\dagger_s \cdot F_1(\lambda_1, \lambda_2, \lambda_3) \cdot A_s - S\|
\]

with

\[
S = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}
\]
on \( \mathcal{V}(\{-, 0, +\}) \).
Demo: Mini Example

Take the following simple program or probabilistic sketch:

```plaintext
var x :{-8..8};
begin
  opt \{x := 0\} or \{x := 1\} or \{x := -1\} top;
end
```

Synthesis requirement: $x > 0$ at the end of the execution.

This probabilistic sketch is essentially equivalent to:

$$\begin{align*}
\text{[choose]} &\lambda_1 : \{x := 0\} \text{ or } \lambda_2 : \{x := 1\} \text{ or } \lambda_3 : \{x := -1\} \text{ ro.}
\end{align*}$$

The aim is to minimise

$$\Phi(\lambda_1, \lambda_2, \lambda_3) = \|A_s^\dagger \cdot F_1(\lambda_1, \lambda_2, \lambda_3) \cdot A_s - S\|$$

with

$$S = \begin{pmatrix}
0 & 0 & 1 \\
0 & 0 & 1 \\
0 & 0 & 1
\end{pmatrix} \text{ on } V(\{-, 0, +\}).$$
Swapping: The XOR Trick

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

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

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

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

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

\[
\text{[choose]}^1 \lambda_{1,1} : S_1 \text{ or } \ldots \text{ or } \lambda_{1,n} : S_n \text{ ro;}
\]

\[
\text{[choose]}^2 \lambda_{2,1} : S_1 \text{ or } \ldots \text{ or } \lambda_{2,n} : S_n \text{ ro;}
\]

\[
\text{[choose]}^3 \lambda_{3,1} : S_1 \text{ or } \ldots \text{ or } \lambda_{3,n} : S_n \text{ ro;}
\]

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

\[
\text{[skip]}^1
\]

\[
[x := y]^2 \quad [x := z]^3
\]

\[
[y := x]^4 \quad [y := z]^5
\]

\[
[z := x]^6 \quad [z := y]^7
\]

\[
[x := (x + y) \mod 2]^8 \quad [x := (x + z) \mod 2]^9
\]

\[
[y := (y + x) \mod 2]^10 \quad [y := (y + z) \mod 2]^11
\]

\[
[z := (z + x) \mod 2]^12 \quad [z := (z + y) \mod 2]^13
\]
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 function

$$\Phi(\lambda_{ij}) = \|A^\dagger T(\lambda_{ij})A - S\|_2$$

with $A = I_{(4)} \otimes A_{f(2)} = \text{diag}(1, 1, 1, 1) \otimes (1, 1)^t$. 

h.wiklicky@imperial.ac.uk

Quantitative Synthesis
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 $\mathbb{R}^2 \otimes \mathbb{R}^2$):

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

Objective function

$$\Phi(\lambda_{ij}) = \| A^\dagger T(\lambda_{ij}) A - S \|_2$$

with $A = I_{(4)} \otimes A_{f(2)} = \text{diag}(1, 1, 1, 1) \otimes (1, 1)^t$. 
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 $\mathbb{R}^2 \otimes \mathbb{R}^2$):

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

Objective function

$$\Phi(\lambda_{ij}) = \| A^\dagger T(\lambda_{ij}) A - S \|_2$$

with $A = I_{(4)} \otimes A_{f(2)} = \text{diag}(1, 1, 1, 1) \otimes (1, 1)^t$. 

h.wiklicky@imperial.ac.uk

Quantitative Synthesis
More general, we will penalises for reading or writing to $z$:

$$\Phi_{\rho\omega}(\lambda_{ij}) = \|A^\dagger T(\lambda_{ij})A - S\|_2 + \rho R(\lambda_{ij}) + \omega W(\lambda_{ij})$$

The optimisation problem we thus have to solve is given by

$$\min_{\lambda_{ij}} \Phi_{\rho\omega}(\lambda_{ij})$$

subject to

$$\sum_{j=1}^{13} \lambda_{ij} = 1 \text{ for } i = 1, 2, 3$$

and

$$0 \leq \lambda_{ij} \leq 1 \text{ for } i = 1, 2, 3 \text{ and } j = 1, \ldots, 13$$
More general, we will penalise for reading or writing to $z$:

$$\Phi_{\rho \omega}(\lambda_{ij}) = \|A^\dagger T(\lambda_{ij})A - S\|_2 + \rho R(\lambda_{ij}) + \omega W(\lambda_{ij})$$

The optimisation problem we thus have to solve is given by

$$\min_{\lambda_{ij}} \Phi_{\rho \omega}(\lambda_{ij})$$

subject to

$$\sum_{j=1}^{13} \lambda_{ij} = 1 \text{ for } i = 1, 2, 3$$

and

$$0 \leq \lambda_{ij} \leq 1 \text{ for } i = 1, 2, 3 \text{ and } j = 1, \ldots, 13$$
Penalising Read/Writes to Variable $z$

The functions $R$ and $W$ determine the probability that in each step of our program the variable $z$ is read or written to.

To do this, define two projections on $\mathbb{R}^{13}$, the label space:

\[ P_r = \text{diag}(0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1) \]
\[ P_w = \text{diag}(0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1) \]

then we can define the (ad hoc) penalising functions as:

\[ R(\lambda_{ij}) = \| \sum_{i=1}^{3} (\lambda_{ij})_j P_r \|_1 \quad \text{and} \quad W(\lambda_{ij}) = \| \sum_{i=1}^{3} (\lambda_{ij})_j P_w \|_1 \]
Penalising Read/Writes to Variable $z$

The functions $R$ and $W$ determine the probability that in each step of our program the variable $z$ is read or written to.

To do this, define two projections on $\mathbb{R}^{13}$, the label space:

$$P_r = \text{diag}(0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1)$$
$$P_w = \text{diag}(0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1)$$

then we can define the (ad hoc) penalising functions as:

$$R(\lambda_{ij}) = \| \sum_{i=1}^{3} (\lambda_{ij})_j P_r \|_1$$
$$W(\lambda_{ij}) = \| \sum_{i=1}^{3} (\lambda_{ij})_j P_w \|_1$$
Penalising Read/Writes to Variable $z$

The functions $R$ and $W$ determine the probability that in each step of our program the variable $z$ is read or written to.

To do this, define two projections on $\mathbb{R}^{13}$, the label space:

$$P_r = \text{diag}(0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1)$$
$$P_w = \text{diag}(0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1)$$

then we can define the (ad hoc) penalising functions as:

$$R(\lambda_{ij}) = \| \sum_{i=1}^{3} (\lambda_{ij})_j P_r \|_1$$
$$W(\lambda_{ij}) = \| \sum_{i=1}^{3} (\lambda_{ij})_j P_w \|_1$$
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:

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

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:

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

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 **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:

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

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:

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

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:

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

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:

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

This corresponds to the program:

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

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

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:

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

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

\[
\begin{align*}
[z := x]^6; \ [x := y]^2; \ [y := z]^5
\end{align*}
\]

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

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

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:

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

This corresponds to the program:

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

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

This corresponds to the program:

$$[y := (y+x) \mod 2]^{10}; [x := (x+y) \mod 2]^8; [y := (y+x) \mod 2]^9$$

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:

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

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

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

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 \( \lambda_{ij} \):

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

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

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

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: \([z := x]^{6}; [x := y]^{2}; [y := z]^{5}\).
begin [x := 0]^1 ; [y := 0]^2 ;
  do [choose]^3
    pxxx: [x := @goInc3(x, 18)]^4 or
    pxx: [x := @goInc2(x, 18)]^5 or
    px: [x := @goInc1(x, 18)]^6 or
    p: [skip]^7 or
    py: [y := @goInc1(y, 18)]^8 or
    pyy: [y := @goInc2(y, 18)]^9 or
    pyyy: [y := @goInc3(y, 18)]^10
  od
until [((x == 18) && (y == 18))]^11 od;
[stop]^12 end

@goInc1(x, b) is defined as \( x = \min(\max(x + 1, 0), b) \). In every iteration \( x \) or \( y \) are incremented by 0, 1, 2 or 3.
Random Walk on a Go Board: Optimisation

- **Average running time** $c_0 = (0, 0) \rightarrow (18, 18)$ or penalty

$$\Phi_0(\lambda_i) = \sum_{n=1}^{200} n \cdot \|c_0 \cdot (T(\lambda_i)^n - T(\lambda_i)^{n-1}) \cdot A_{\ell_12}\|_1 + 1000 \cdot (1 - \|c_0 \cdot T(\lambda_i)^{200} \cdot A_{\ell_12}\|_1)$$

Optimum for $\lambda_1 = pxxx = 0.5$ and $\lambda_7 = pyyy = 0.5$.

- **Additional**: Close to diagonal, symmetric $x$ and $y$ moves

$$\Phi_1(\lambda_i) = \Phi_0(\lambda_i) + 10000 \cdot (|(\lambda_1 - \lambda_7)| + |(\lambda_2 - \lambda_6)| + |(\lambda_5 - \lambda_3)|)$$

Optimum for $\lambda_1 = pxxx = 0.5$ and $\lambda_7 = pyyy = 0.5$.

- **Additional**: Avoid odd coordinates, visit only $(2k, 2k)$

$$\Phi_2(\lambda_i) = \Phi_1(\lambda_i) + 1000 \cdot \sum_{n=1}^{200} \langle c_0 T(\lambda_i)^n (A_{\rho} \otimes A_{\rho}), (0, 1, 1, 1) \rangle$$

Optimum for $\lambda_2 = pxx = 0.5 = pyy = \lambda_6$. 

h.wiklicky@imperial.ac.uk
Quantitative Synthesis
Random Walk on a Go Board: Optimisation

- **Average running time** $c_0 = (0, 0) \rightarrow (18, 18)$ or penalty

$$\Phi_0(\lambda_i) = \sum_{n=1}^{200} n \cdot \|c_0 \cdot (T(\lambda_i)^n - T(\lambda_i)^{n-1}) \cdot A_{\ell_1} \|_1 + 1000 \cdot (1 - \|c_0 \cdot T(\lambda_i)^{200} \cdot A_{\ell_1} \|_1)$$

Optimum for $\lambda_1 = pxxx = 0.5$ and $\lambda_7 = pyyyy = 0.5$.

- **Additional**: Close to diagonal, symmetric $x$ and $y$ moves

$$\Phi_1(\lambda_i) = \Phi_0(\lambda_i) + 10000 \cdot (|\lambda_1 - \lambda_7| + |\lambda_2 - \lambda_6| + |\lambda_5 - \lambda_3|).$$

Optimum for $\lambda_1 = pxxx = 0.5$ and $\lambda_7 = pyyyy = 0.5$.

- **Additional**: Avoid odd coordinates, visit only $(2k, 2k)$

$$\Phi_2(\lambda_i) = \Phi_1(\lambda_i) + 1000 \cdot \sum_{n=1}^{200} \langle c_0 T(\lambda_i)^n (A_p \otimes A_p), (0, 1, 1, 1) \rangle$$

Optimum for $\lambda_2 = pxx = 0.5 = pyy = \lambda_6$. 

h.wiklicky@imperial.ac.uk    Quantitative Synthesis
Random Walk on a Go Board: Optimisation

- **Average running time** $c_0 = (0, 0) \rightarrow (18, 18)$ or penalty

  $$\Phi_0(\lambda_i) = \sum_{n=1}^{200} n \cdot \|c_0 \cdot (T(\lambda_i)^n - T(\lambda_i)^{n-1}) \cdot A_{\ell_1} \|_1 + 1000 \cdot (1 - \|c_0 \cdot T(\lambda_i)^{200} \cdot A_{\ell_1} \|_1)$$

  Optimum for $\lambda_1 = px\!xx = 0.5$ and $\lambda_7 = pyy\!y = 0.5$.

- **Additional**: Close to diagonal, symmetric $x$ and $y$ moves

  $$\Phi_1(\lambda_i) = \Phi_0(\lambda_i) + 10000 \cdot (\|\lambda_1 - \lambda_7\| + \|\lambda_2 - \lambda_6\| + \|\lambda_5 - \lambda_3\|).$$

  Optimum for $\lambda_1 = px\!xx = 0.5$ and $\lambda_7 = pyy\!y = 0.5$.

- **Additional**: Avoid odd coordinates, visit only $(2k, 2k)$

  $$\Phi_2(\lambda_i) = \Phi_1(\lambda_i) + 1000 \sum_{n=1}^{200} \langle c_0 T(\lambda_i)^n (A_p \otimes A_p), (0, 1, 1, 1) \rangle$$

  Optimum for $\lambda_2 = px\!x = 0.5 = p\!y\!y = \lambda_6$. 

h.wiklicky@imperial.ac.uk  
Quantitative Synthesis
We used similar ideas in investigating the problem of fixing time leaks (Kocher, Agat, etc.). [ICICS08]

The issues like scaleability (utilising PAI) and choice of optimisation algorithms need further investigation.

In order to improve optimisation it could be useful to use PAI for approximation and symmetry breaking.

Comparison with ideal, abstract behaviour $S$ as objective function, i.e. $\Phi(\lambda_i) = \|A^\dagger T(P(\lambda_i))A - S\|$. 

Formal translation of constraints/assertions (e.g. PCTL) into objective function, i.e. $\Phi(\lambda_i) = \|T(P(\lambda_i)) - PCTL\|$. 

Comparison with other approaches in program synthesis (e.g. discrete, quantitative)
Conclusions

- We used similar ideas in investigating the problem of fixing time leaks (Kocher, Agat, etc.). [ICICS08]
- The issues like scaleability (utilising PAI) and choice of optimisation algorithms need further investigation.
- In order to improve optimisation it could be useful to use PAI for approximation and symmetry breaking.
- Comparison with ideal, abstract behaviour $S$ as objective function, i.e. $\Phi(\lambda_i) = \|A^\dagger T(P(\lambda_i))A - S\|$.  
- Formal translation of constraints/assertions (e.g. PCTL) into objective function, i.e. $\Phi(\lambda_i) = \|T(P(\lambda_i)) - PCTL\|$.  
- Comparison with other approaches in program synthesis (e.g. discrete, quantitative)
We used similar ideas in investigating the problem of fixing time leaks (Kocher, Agat, etc.). [ICICS08]

The issues like scaleability (utilising PAI) and choice of optimisation algorithms need further investigation.

In order to improve optimisation it could be useful to use PAI for approximation and symmetry breaking.

Comparison with ideal, abstract behaviour $S$ as objective function, i.e. $\Phi(\lambda_i) = \|A^\dagger T(P(\lambda_i))A - S\|$.  

Formal translation of constraints/assertions (e.g. PCTL) into objective function, i.e. $\Phi(\lambda_i) = \|T(P(\lambda_i)) - PCTL\|$.  

Comparison with other approaches in program synthesis (e.g. discrete, quantitative)
Conclusions

- We used similar ideas in investigating the problem of fixing time leaks (Kocher, Agat, etc.). [ICICS08]
- The issues like scaleability (utilising PAI) and choice of optimisation algorithms need further investigation.
- In order to improve optimisation it could be useful to use PAI for approximation and symmetry breaking.
- Comparison with ideal, abstract behaviour $S$ as objective function, i.e. $\Phi(\lambda_i) = \|A^\dagger T(P(\lambda_i))A - S\|$.
- Formal translation of constraints/assertions (e.g. PCTL) into objective function, i.e. $\Phi(\lambda_i) = \|T(P(\lambda_i)) - PCTL\|$.
- Comparison with other approaches in program synthesis (e.g. discrete, quantitative)
Conclusions

- We used similar ideas in investigating the problem of fixing time leaks (Kocher, Agat, etc.). [ICICS08]
- The issues like scaleability (utilising PAI) and choice of optimisation algorithms need further investigation.
- In order to improve optimisation it could be useful to use PAI for approximation and symmetry breaking.
- Comparison with ideal, abstract behaviour $S$ as objective function, i.e. $\Phi(\lambda_i) = \|A^{\dagger}T(P(\lambda_i))A - S\|.$
- Formal translation of constraints/assertions (e.g. PCTL) into objective function, i.e. $\Phi(\lambda_i) = \|T(P(\lambda_i)) - PCTL\|.$
- Comparison with other approaches in program synthesis (e.g. discrete, quantitative)
Conclusions

- We used similar ideas in investigating the problem of fixing time leaks (Kocher, Agat, etc.). [ICICS08]
- The issues like scaleability (utilising PAI) and choice of optimisation algorithms need further investigation.
- In order to improve optimisation it could be useful to use PAI for approximation and symmetry breaking.
- Comparison with ideal, abstract behaviour $S$ as objective function, i.e. $\Phi(\lambda_i) = \|A^\dagger T(P(\lambda_i))A - S\|$.
- Formal translation of constraints/assertions (e.g. PCTL) into objective function, i.e. $\Phi(\lambda_i) = \|T(P(\lambda_i)) - PCTL\|$.
- Comparison with other approaches in program synthesis (e.g. discrete, quantitative)
Extras: LOS
Semantics of statements are sets of global and local operators:

\[
\llbracket \cdot \rrbracket : \text{Stmt} \rightarrow \mathcal{P}(\Gamma \cup \Lambda)
\]

\[
\Gamma = \mathcal{L}(\mathcal{V}(\text{Value}) \otimes \mathcal{V}(\text{Label}))
\]

\[
\Lambda = \mathcal{L}(\mathcal{V}(\text{Value})) \times \text{Label}
\]

\[
\llbracket \text{skip} \rrbracket^\ell = \{ \langle I, \ell \rangle \}
\]

\[
\llbracket x := e \rrbracket^\ell = \{ \langle \text{U}(x \leftarrow e), \ell \rangle \}
\]

\[
\llbracket x =\rho \rrbracket^\ell = \{ \langle \sum_{\langle p, r \rangle \in \rho} p \cdot \text{U}(x \leftarrow r), \ell \rangle \}
\]

\[
[ S_1 ; S_2 ] = ([ S_1 ] \triangleright \text{init}(S2)) \cup [ S_2 ]
\]

where we use the “connection” operator defined as

\[
\langle F, \ell \rangle \triangleright \ell' = F \otimes \text{E}(\ell, \ell')
\]
Semantics of statements are sets of global and local operators:

\[ \begin{align*}
\llbracket \cdot \rrbracket : \text{Stmt} & \rightarrow \mathcal{P}(\Gamma \cup \Lambda) \\
\Gamma &= \mathcal{L}(\mathcal{V}(\text{Value}^\nu) \otimes \mathcal{V}(\text{Label})) \\
\Lambda &= \mathcal{L}(\mathcal{V}(\text{Value}^\nu)) \times \text{Label}
\end{align*} \]

\[
\llbracket \text{skip} \rrbracket^\ell = \{ \langle I, \ell \rangle \}
\]

\[
\llbracket x := e \rrbracket^\ell = \{ \langle U(x \leftarrow e), \ell \rangle \}
\]

\[
\llbracket x \neq \rho \rrbracket^\ell = \{ \langle \sum_{(p,r) \in \rho} p \cdot U(x \leftarrow r), \ell \rangle \}
\]

\[
\llbracket S_1; S_2 \rrbracket = (\llbracket S_1 \rrbracket \triangleright \text{init}(S2)) \cup \llbracket S_2 \rrbracket
\]

where we use the “connection” operator defined as

\[ \langle F, \ell \rangle \triangleright \ell' = F \otimes E(\ell, \ell') \]
\[
[[\text{choose}] \ell \ p_1 : S_1 \text{ or } p_2 : S_2 \text{ ro}] = \{\tilde{p}_1 \cdot I \otimes E(\ell, \text{init}(S_1))\} \cup \{\tilde{p}_2 \cdot I \otimes E(\ell, \text{init}(S_2))\} \cup [S_1] \cup [S_2]
\]
\[
[[\text{if } [b] \ell \text{ then } S_1 \text{ else } S_2 \text{ fi}] = \{\langle P(b), \ell \rangle \} \triangleright \text{init}(S_1)\} \cup \{\langle P(b)^\bot, \ell \rangle \} \triangleright \text{init}(S_2)\} \cup [S_1] \cup [S_2]
\]
\[
[[\text{while } [b] \ell \text{ do } S \text{ od}] = \{\langle P(b), \ell \rangle \} \triangleright \text{init}(S)\} \cup [S] \cup \{\langle P(b)^\bot, \ell \rangle \}
\]

With normalisation (at compile time) \( \tilde{p}_i = \frac{p_i}{p_1 + p_2} \) and adding a final loop \( \ell^* \), i.e. \( [P] \triangleright \ell^* \), when we consider a full program.
Basic Operators

If we have a single variable $x$ (with finite set of values) then its probabilistic state is a (row) vector $\sigma$ and basic $T_i$'s are:

Assigning a variable a constant value $c$:

$$
(U(c))_{ij} = \begin{cases} 
1 & \text{if } \xi(c) = i \\
0 & \text{otherwise.}
\end{cases}
$$

Testing if a variable fulfills a (boolean) test $b$:

$$
(P(b))_{ij} = \begin{cases} 
1 & \text{if } b(n) \text{ holds, and } \xi(c) = i \\
0 & \text{otherwise.}
\end{cases}
$$

We also use the identity operator/matrix $I$ and matrix unites $E_{ij}$. 
If we have a single variable $x$ (with finite set of values) then its probabilistic state is a (row) vector $\sigma$ and basic $T_i$’s are:

Assigning a variable a constant value $c$:

$$(U(c))_{ij} = \begin{cases} 1 & \text{if } \xi(c) = i \\ 0 & \text{otherwise} \end{cases}.$$ 

Testing if a variable fulfills a (boolean) test $b$:

$$(P(b))_{ij} = \begin{cases} 1 & \text{if } b(n) \text{ holds, and } \xi(c) = i \\ 0 & \text{otherwise} \end{cases}.$$ 

We also use the identity operator/matrix $I$ and matrix unites $E_{ij}$. 

h.wiklicky@imperial.ac.uk

Quantitative Synthesis
Basic Operators

If we have a single variable $x$ (with finite set of values) then its probabilistic state is a (row) vector $\sigma$ and basic $T_i$’s are:

Assigning a variable a constant value $c$:

$$(U(c))_{ij} = \begin{cases} 
1 & \text{if } \xi(c) = i \\
0 & \text{otherwise.}
\end{cases}$$

Testing if a variable fulfills a (boolean) test $b$:

$$(P(b))_{ij} = \begin{cases} 
1 & \text{if } b(n) \text{ holds, and } \xi(c) = i \\
0 & \text{otherwise.}
\end{cases}$$

We also use the identity operator/matrix $I$ and matrix unites $E_{ij}$. 

quantitative synthesis
Multi-Variable Operations

Testing if a variable fulfils a (boolean) test $b$, equality to state $s$, and with respect to expression $e$:

\[(P(b))_{ij} = \begin{cases} 
1 & \text{if } b(n) \text{ holds, and } \xi(c) = i \\
0 & \text{otherwise.} 
\end{cases}\]

\[P(s) = \bigotimes_{i=1}^{v} P(s(x_i))\]

\[P(e = c) = \sum_{\mathcal{E}(e)s=c} P(s)\]
Assigning a variable a constant value \( c \) and an expression \( e \):

\[
(U(c))_{ij} = \begin{cases} 
  1 & \text{if } \xi(c) = i \\
  0 & \text{otherwise.}
\end{cases}
\]

\[
U(x_k \leftarrow c) = \bigotimes_{i=1}^{k-1} I \otimes U(c) \otimes \bigotimes_{i=k+1}^{v} I
\]

\[
U(x_k \leftarrow e) = \sum_c P(e = c)U(x_k \leftarrow c)
\]
Relation to Kozen’s Semantics

We recover Kozen’s Input/Output semantics in the time limit.

Proposition

Given a program $P$ and an initial probabilistic state $s_0$ as a distribution over the program variables, let $\left[P\right]_K$ be Kozen’s semantics of $P$ and $T(P)$ the LOS. Then

$$\left(s_0 \otimes e_0\right) \cdot \left(\lim_{n \to \infty} T(P)^n\right) \cdot S_{\ell^*} = s_0 \cdot \left[P\right]_K.$$

$S_{\ell^*}$ abstracts the state at the final label $\ell^*$, i.e. $S_{\ell^*} = I \otimes e_{\ell^*}^t$.

Note that the tensor product construction in LOS gives a more structured semantics than in the case of Kozen’s semantics.
Relation to Kozen’s Semantics

We recover Kozen’s Input/Output semantics in the time limit.

**Proposition**

Given a program $P$ and an initial probabilistic state $s_0$ as a distribution over the program variables, let $[P]_K$ be Kozen’s semantics of $P$ and $T(P)$ the LOS. Then

$$
(s_0 \otimes e_0) \cdot \left( \lim_{n \to \infty} T(P)^n \right) \cdot S_{\ell^*} = s_0 \cdot [P]_K.
$$

$S_{\ell^*}$ abstracts the state at the final label $\ell^*$, i.e. $S_{\ell^*} = I \otimes e_{\ell^*}^t$.

Note that the tensor product construction in LOS gives a more structured semantics than in the case of Kozen’s semantics.
Relation to Kozen’s Semantics

We recover Kozen’s Input/Output semantics in the time limit.

Proposition

Given a program $P$ and an initial probabilistic state $s_0$ as a distribution over the program variables, let $\llbracket P \rrbracket_K$ be Kozen’s semantics of $P$ and $T(P)$ the LOS. Then

$$\left( s_0 \otimes e_0 \right) \cdot \left( \lim_{n \to \infty} T(P)^n \right) \cdot S_{\ell^*} = s_0 \cdot \llbracket P \rrbracket_K.$$

$S_{\ell^*}$ abstracts the state at the final label $\ell^*$, i.e. $S_{\ell^*} = I \otimes e^{t_{\ell^*}}$.

Note that the tensor product construction in LOS gives a more structured semantics than in the case of Kozen’s semantics.
Kozen’s and Weakest Precondition Semantics

Input/Output Semantics
Kozen’s semantics only captures probabilities for terminating computations, thus same $[P]$ but different $T(P)$ for:

\[
\text{while true do } x \triangleq \{0, 1\} \text{ od}
\]
\[
\text{while true do } x \leftarrow x + 1 \text{ od}
\]

Reason: Usage of labels/PC

Forward vs Backward Semantics
Transform (probabilistic) state or transform observable (strongly related to weakest pre-condition). Consider adjoint operator:

\[
\langle x \cdot \llbracket P \rrbracket, y \rangle = \langle x, y \cdot \llbracket P \rrbracket^* \rangle
\]

Reason: Self-duality of Hilbert space $\ell_2$
Kozen’s and Weakest Precondition Semantics

Input/Output Semantics
Kozen’s semantics only captures probabilities for terminating computations, thus same $[P]$ but different LOS $T(P)$ for:

while true do $x := \{0, 1\}$ od
while true do $x := x + 1$ od

Reason: Usage of labels/PC

Forward vs Backward Semantics
Transform (probabilistic) state or transform observable (strongly related to weakest pre-condition). Consider adjoint operator:

$$\langle x \cdot [P], y \rangle = \langle x, y \cdot [P]^* \rangle$$

Reason: Self-duality of Hilbert space $\ell_2$
Kozen’s and Weakest Precondition Semantics

Input/Output Semantics
Kozen’s semantics only captures probabilities for terminating computations, thus same $\llbracket P \rrbracket$ but different LOS $T(P)$ for:

\[
\begin{align*}
\text{while } \textbf{true} & \text{ do } x \equiv \{0, 1\} \od \text{ od} \\
\text{while } \textbf{true} & \text{ do } x := x + 1 \od \text{ od}
\end{align*}
\]

Reason: Usage of labels/PC

Forward vs Backward Semantics
Transform (probabilistic) state or transform observable (strongly related to weakest pre-condition). Consider adjoint operator:

\[
\langle x \cdot \llbracket P \rrbracket, y \rangle = \langle x, y \cdot \llbracket P \rrbracket^* \rangle
\]

Reason: Self-duality of Hilbert space $\ell_2$
Kozen’s and Weakest Precondition Semantics

Input/Output Semantics
Kozen’s semantics only captures probabilities for terminating computations, thus same $\mathbb{[P]}$ but different LOS $\mathbf{T}(P)$ for:

\[
\begin{align*}
\text{while } \textbf{true} \text{ do } & \ x \ = \ \{0, 1\} \ \text{od} \\
\text{while } \textbf{true} \text{ do } & \ x \ := \ x + 1 \ \text{od}
\end{align*}
\]

Reason: Usage of labels/PC

Forward vs Backward Semantics
Transform (probabilistic) state or transform observable (strongly related to weakest pre-condition). Consider adjoint operator:

\[
\langle x \cdot \mathbb{[P]}, y \rangle = \langle x, y \cdot \mathbb{[P]}^* \rangle
\]

Reason: Self-duality of Hilbert space $\ell_2$
Extras: PAI
Concrete and abstract domain are sets of step-functions on an interval \([a, b]\). The set of (real-valued) step-function \(T_n\) is based on the sub-division of the interval into \(n\) sub-intervals. Each step function in \(T_n\) corresponds to a vector in \(\mathbb{R}^n\).
Concrete and abstract domain are sets of **step-functions** on an interval \([a, b]\). The set of (real-valued) step-function \(T_n\) is based on the sub-division of the interval into \(n\) sub-intervals. Each step function in \(T_n\) corresponds to a vector in \(\mathbb{R}^n\).
Concrete and abstract domain are sets of step-functions on an interval \([a, b]\). The set of (real-valued) step-function \( \mathcal{T}_n \) is based on the sub-division of the interval into \( n \) sub-intervals. Each step function in \( \mathcal{T}_n \) corresponds to a vector in \( \mathbb{R}^n \).
Concrete and abstract domain are sets of step-functions on an interval $[a, b]$. The set of (real-valued) step-function $\mathcal{T}_n$ is based on the sub-division of the interval into $n$ sub-intervals. Each step function in $\mathcal{T}_n$ corresponds to a vector in $\mathbb{R}^n$. 

$$f = (5, 5, 6, 7, 8, 4, 3, 2, 8, 6, 6, 7, 9, 8, 8, 7)$$
Concrete and abstract domain are sets of step-functions on an interval \([a, b]\). The set of (real-valued) step-function \(\mathcal{T}_n\) is based on the sub-division of the interval into \(n\) sub-intervals. Each step function in \(\mathcal{T}_n\) corresponds to a vector in \(\mathbb{R}^n\).

\[
f = ( 5 \ 5 \ 6 \ 7 \ 8 \ 4 \ 3 \ 2 \ 8 \ 6 \ 6 \ 7 \ 9 \ 8 \ 8 \ 7 )
\]
Example: Abstractions $\mathcal{T}_{16} \rightarrow \mathcal{T}_{8}$

$$A_8 = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}$$
Example: Abstractions $T_{16} \rightarrow T_8$

$$G_8 = \begin{pmatrix}
\frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2}
\end{pmatrix}$$

h.wiklicky@imperial.ac.uk

Quantitative Synthesis
Example: Abstractions $\mathcal{T}_{16} \rightarrow \mathcal{T}_4$

\[
A_4 = \begin{pmatrix}
1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 \\
\end{pmatrix}
\]
Example: Abstractions $\mathcal{T}_{16} \rightarrow \mathcal{T}_{4}$

$$G_4 = \begin{pmatrix}
\frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4}
\end{pmatrix}$$
Example: Abstractions $\mathcal{T}_{16} \rightarrow \mathcal{T}_2$

$$A_2 = \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{pmatrix}$$
Example: Abstractions $\mathcal{T}_{16} \rightarrow \mathcal{T}_2$

$$G_2 = \begin{pmatrix}
\frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8}
\end{pmatrix}$$
Example: Abstractions $\mathcal{T}_{16} \rightarrow \mathcal{T}_1$

$$A_1 = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

h.wiklicky@imperial.ac.uk  Quantitative Synthesis
Example: Abstractions $\mathcal{T}_{16} \rightarrow \mathcal{T}_1$

$$G_1 = \left( \begin{array}{cccccccccccccccc}
\frac{1}{16} & \frac{1}{16} & \frac{1}{16} & \frac{1}{16} & \frac{1}{16} & \frac{1}{16} & \frac{1}{16} & \frac{1}{16} & \frac{1}{16} & \frac{1}{16} & \frac{1}{16} & \frac{1}{16} & \frac{1}{16} & \frac{1}{16} & \frac{1}{16} & \frac{1}{16}
\end{array} \right)$$
Example: Geometric Interpretation
Example: Geometric Interpretation

\[ \mathcal{T}_{16} \rightarrow \mathcal{T}_{8} \]
Example: Geometric Interpretation

\( \mathcal{T}_{16} \rightarrow \mathcal{T}_8 \)
Example: Geometric Interpretation

$\mathcal{T}_{16} \rightarrow \mathcal{T}_4$
Example: Geometric Interpretation

$\mathcal{T}_{16} \rightarrow \mathcal{T}_{4}$
Example: Geometric Interpretation

\[ \mathcal{T}_{16} \rightarrow \mathcal{T}_2 \]
Example: Geometric Interpretation

$T_{16} \rightarrow T_{2}$
Example: Geometric Interpretation

\[ \mathcal{T}_{16} \rightarrow \mathcal{T}_1 \]