A Framework for Quantitative Program Synthesis

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

QMUL, 4 May 2016

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

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

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

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

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

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

Analysis and Synthesis

The terms analysis and synthesis are central in:

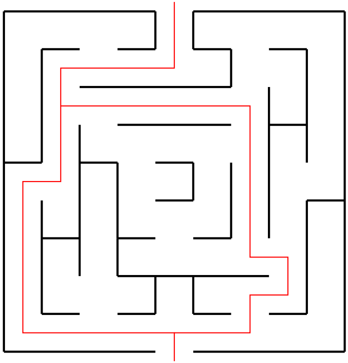
- Chemistry
- Mathematics
- Biology (Synthetic Biology)
- Computer Science (Program Synthesis)

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

Optimisation

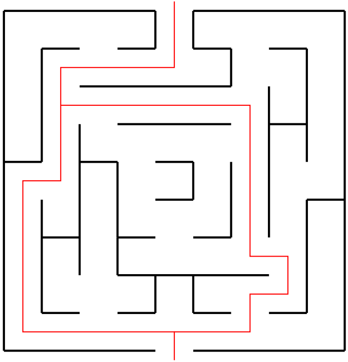
Finding the minimum length path vs minimum value of functions

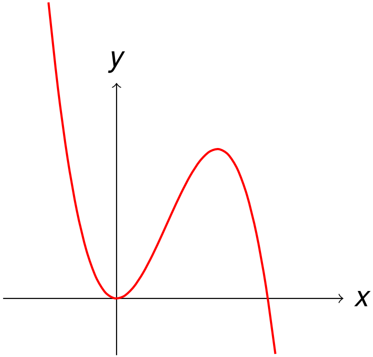


As usual (for now): Take the best non-linear optimisation tool money can't buy (leave it to "them" to make it work).

Optimisation

Finding the minimum length path vs minimum value of functions

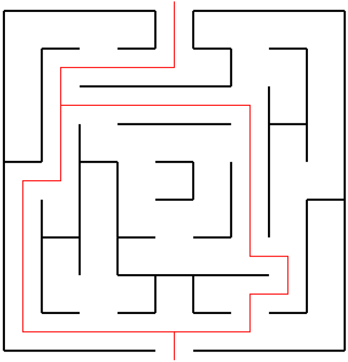


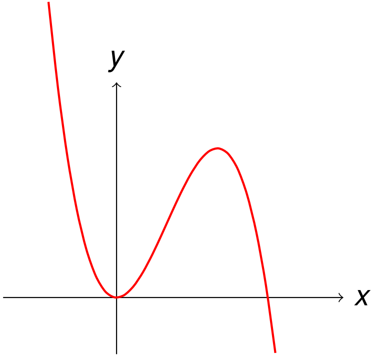


As usual (for now): Take the best non-linear optimisation tool money can't buy (leave it to "them" to make it work).

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).

A Probabilistic Language

Allow for probabilistic choices and/or random assignments:

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

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

A Probabilistic Language

Allow for probabilistic choices and/or random assignments:

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

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

A Probabilistic Language

Allow for probabilistic choices and/or random assignments:

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

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

A Probabilistic Language

Allow for probabilistic choices and/or random assignments:

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

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

Example: Cowboys Who Learn

$a := 0;$
$\texttt{choose}\ f : \{t := 0\}\ \texttt{or}\ (1-f) : \{t := 1\}\ \texttt{ro};$
$c := 1;$
$\texttt{while}\ (c = 1)\ \texttt{do}$
$\quad \texttt{if}\ (t = 0)$
$\quad \texttt{then}$
$\quad\quad \texttt{choose}\ a : \{c := 0\}\ \texttt{or}\ (1-a) : \{t := 1\}\ \texttt{ro}$
$\quad \texttt{else}$
$\quad\quad \texttt{choose}\ b : \{c := 0\}\ \texttt{or}\ (1-b) : \{t := 0\}\ \texttt{ro}$
$\quad \texttt{fi};$
$\quad a := \max(a + \frac{1}{10}, 1);$
$\texttt{od}$

Linear Operator Semantics
LOS

Di Pierro, Hankin, W.: *A Systematic Approach to Probabilistic Pointer Analysis*. APLAS 2007

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 $[S]^i$:

$$\mathbf{T}_i = \mathbf{F}_i \otimes \mathbf{E}(i, f).$$

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

$$\mathbf{T}(P) = \sum_i \{\mathbf{T}_i \mid \mathbf{T}_i \in [\![P]\!]\}$$

See also the Fixed-Point Semantics of Kozen: *Semantics of probabilistic programs*. J. Comput. Syst. 1981.

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 $[S]^i$:

$$\mathbf{T}_i = \mathbf{F}_i \otimes \mathbf{E}(i, f).$$

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

$$\mathbf{T}(P) = \sum_i \{\mathbf{T}_i \mid \mathbf{T}_i \in [\![P]\!]\}$$

See also the Fixed-Point Semantics of Kozen: *Semantics of probabilistic programs*. J. Comput. Syst. 1981.

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 $[S]^i$:

$$\mathbf{T}_i = \mathbf{F}_i \otimes \mathbf{E}(i, f).$$

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

$$\mathbf{T}(P) = \sum_i \{\mathbf{T}_i \mid \mathbf{T}_i \in [\![P]\!]\}$$

See also the Fixed-Point Semantics of Kozen: *Semantics of probabilistic programs*. J. Comput. Syst. 1981.

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 $[S]^i$:

$$\mathbf{T}_i = \mathbf{F}_i \otimes \mathbf{E}(i, f).$$

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

$$\mathbf{T}(P) = \sum_i \{\mathbf{T}_i \mid \mathbf{T}_i \in [\![P]\!]\}$$

See also the Fixed-Point Semantics of Kozen: *Semantics of probabilistic programs*. J. Comput. Syst. 1981.

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 $[S]^i$:

$$\mathbf{T}_i = \mathbf{F}_i \otimes \mathbf{E}(i, f).$$

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

$$\mathbf{T}(P) = \sum_i \{\mathbf{T}_i \mid \mathbf{T}_i \in [\![P]\!]\}$$

See also the Fixed-Point Semantics of Kozen: *Semantics of probabilistic programs*. J. Comput. Syst. 1981.

Probabilistic State Space

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

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

$$s \in \textbf{State} = (\textbf{Var} \rightarrow \textbf{Value}) = \textbf{Value}^v$$

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

$$\begin{aligned} \sigma &\in \mathcal{V}(\textbf{Var} \rightarrow \textbf{Value}) = \\ &= \mathcal{V}(\textbf{Value}_1 \times \textbf{Value}_2 \times \ldots \times \textbf{Value}_v) = \\ &= \mathcal{V}(\textbf{Value}_1) \otimes \mathcal{V}(\textbf{Value}_2) \otimes \ldots \otimes \mathcal{V}(\textbf{Value}_v) \end{aligned}$$

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 $\sigma \in \textbf{State}$ given by:

$$s \in \textbf{State} = (\textbf{Var} \rightarrow \textbf{Value}) = \textbf{Value}^{v}$$

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

$$\begin{aligned} \sigma &\in \mathcal{V}(\textbf{Var} \rightarrow \textbf{Value}) = \\ &= \mathcal{V}(\textbf{Value}_1 \times \textbf{Value}_2 \times \ldots \times \textbf{Value}_v) = \\ &= \mathcal{V}(\textbf{Value}_1) \otimes \mathcal{V}(\textbf{Value}_2) \otimes \ldots \otimes \mathcal{V}(\textbf{Value}_v) \end{aligned}$$

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 $\sigma \in$ **State** given by:

$$s \in \textbf{State} = (\textbf{Var} \rightarrow \textbf{Value}) = \textbf{Value}^v$$

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

$$\begin{aligned} \sigma &\in \mathcal{V}(\textbf{Var} \rightarrow \textbf{Value}) = \\ &= \mathcal{V}(\textbf{Value}_1 \times \textbf{Value}_2 \times \ldots \times \textbf{Value}_v) = \\ &= \mathcal{V}(\textbf{Value}_1) \otimes \mathcal{V}(\textbf{Value}_2) \otimes \ldots \otimes \mathcal{V}(\textbf{Value}_v) \end{aligned}$$

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 $\sigma \in \mathbf{State}$ given by:

$$s \in \mathbf{State} = (\mathbf{Var} \to \mathbf{Value}) = \mathbf{Value}^{v}$$

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

$$\begin{aligned} \sigma &\in \mathcal{V}(\mathbf{Var} \to \mathbf{Value}) = \\ &= \mathcal{V}(\mathbf{Value}_1 \times \mathbf{Value}_2 \times \ldots \times \mathbf{Value}_v) = \\ &= \mathcal{V}(\mathbf{Value}_1) \otimes \mathcal{V}(\mathbf{Value}_2) \otimes \ldots \otimes \mathcal{V}(\mathbf{Value}_v) \end{aligned}$$

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 $\sigma \in \textbf{State}$ given by:

$$s \in \textbf{State} = (\textbf{Var} \to \textbf{Value}) = \textbf{Value}^v$$

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

$$\begin{aligned} \sigma &\in \mathcal{V}(\textbf{Var} \to \textbf{Value}) = \\ &= \mathcal{V}(\textbf{Value}_1 \times \textbf{Value}_2 \times \ldots \times \textbf{Value}_v) = \\ &= \mathcal{V}(\textbf{Value}_1) \otimes \mathcal{V}(\textbf{Value}_2) \otimes \ldots \otimes \mathcal{V}(\textbf{Value}_v) \end{aligned}$$

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 $\sigma \in \mathbf{State}$ given by:

$$s \quad \in \quad \mathbf{State} = (\mathbf{Var} \to \mathbf{Value}) = \mathbf{Value}^v$$

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

$$\begin{aligned} \sigma \quad & \in \quad \mathcal{V}(\mathbf{Var} \to \mathbf{Value}) = \\ & = \quad \mathcal{V}(\mathbf{Value}_1 \times \mathbf{Value}_2 \times \ldots \times \mathbf{Value}_v) = \\ & = \quad \mathcal{V}(\mathbf{Value}_1) \otimes \mathcal{V}(\mathbf{Value}_2) \otimes \ldots \otimes \mathcal{V}(\mathbf{Value}_v) \end{aligned}$$

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 A** and a $k \times l$ matrix **B**:

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

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

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

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

Tensor or Kronecker Product

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

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

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

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

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

Tensor or Kronecker Product

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

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

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

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

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

Randomised Counting

$[c := 1]^1;\ [i := 0]^2;$
$\texttt{while}\ [c > 0]^3\ \texttt{do}$
$\quad [\texttt{choose}]^4\ \frac{1}{2} : [i := i + 1]^5\ \texttt{or}\ \frac{1}{2} : [c := 0]^6\ \texttt{ro}$
$\texttt{od};$
$[\texttt{skip}]^7$

LOS Semantics constructed from the set of operators:

$$
\begin{aligned}
[\![P]\!] = \{\ & \mathbf{F}_1 \otimes \mathbf{E}(1,2), \mathbf{F}_2 \otimes \mathbf{E}(2,3), \\
& \mathbf{P}(c > 0) \otimes \mathbf{E}(3,4), \mathbf{P}(c > 0)^{\perp} \otimes \mathbf{E}(3,7), \\
& \tfrac{1}{2} \cdot \mathbf{I} \otimes \mathbf{E}(4,5), \tfrac{1}{2} \cdot \mathbf{I} \otimes \mathbf{E}(4,6), \\
& \mathbf{F}_5 \otimes \mathbf{E}(5,3), \mathbf{F}_6 \otimes \mathbf{E}(6,3), \\
& \mathbf{I} \otimes \mathbf{E}(7,7)\ \}
\end{aligned}
$$

Example: Randomised Counting

Randomised Counting

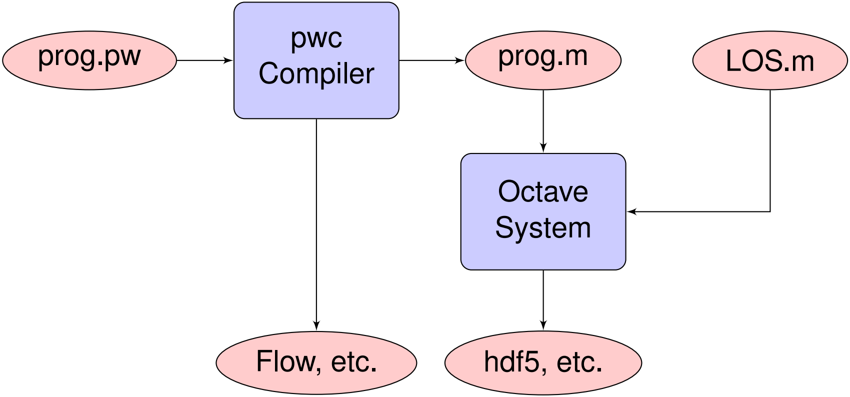
$$
\begin{array}{l}
[c := 1]^1;\ [i := 0]^2; \\
\texttt{while } [c > 0]^3 \texttt{ do} \\
\quad [\texttt{choose}]^4\ \frac{1}{2} : [i := i + 1]^5 \texttt{ or } \frac{1}{2} : [c := 0]^6 \texttt{ ro} \\
\texttt{od}; \\
[\texttt{skip}]^7
\end{array}
$$

LOS Semantics constructed from the set of operators:

$$
\begin{aligned}
[\![P]\!] = \{ \quad & \mathbf{F}_1 \otimes \mathbf{E}(1,2), \mathbf{F}_2 \otimes \mathbf{E}(2,3), \\
& \mathbf{P}(c > 0) \otimes \mathbf{E}(3,4), \mathbf{P}(c > 0)^{\perp} \otimes \mathbf{E}(3,7), \\
& \tfrac{1}{2} \cdot \mathbf{I} \otimes \mathbf{E}(4,5), \tfrac{1}{2} \cdot \mathbf{I} \otimes \mathbf{E}(4,6), \\
& \mathbf{F}_5 \otimes \mathbf{E}(5,3), \mathbf{F}_6 \otimes \mathbf{E}(6,3), \\
& \mathbf{I} \otimes \mathbf{E}(7,7) \quad \}
\end{aligned}
$$

A Tool: The pWhile "Compiler"

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



Probabilistic Abstract Interpretation
PAI

Di Pierro, W.: *Concurrent Constraint Programming: Towards Probabilistic Abstract Interpretation*, PPDP 2000.

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 $\mathbf{A} : \mathcal{C} \to \mathcal{D}$, construct concretisation $\mathbf{G} : \mathcal{D} \to \mathcal{C}$ to obtain a "smaller" abstract semantics:

$$\mathbf{T}(P)^{\#} = \mathbf{G}\mathbf{T}(P)\mathbf{A} = \mathbf{A}^{\dagger}\mathbf{T}(P)\mathbf{A}.$$

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

- Abstractions compatible with sum and tensor product: $(\bigotimes_i \mathbf{A}_i^{\dagger})(\sum_{i,j} \bigotimes_i \mathbf{F}_i)(\bigotimes_i \mathbf{A}_i) = \sum_{ij} \bigotimes_i (\mathbf{A}^{\dagger}\mathbf{F}_i\mathbf{A})$,
- Average of averages is not the average: $(\mathbf{A}_1\mathbf{A}_2)^{\dagger} \neq \mathbf{A}_2^{\dagger}\mathbf{A}_1^{\dagger}$.

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

$$\mathbf{T}(P)^{\#} = \mathbf{G}\mathbf{T}(P)\mathbf{A} = \mathbf{A}^{\dagger}\mathbf{T}(P)\mathbf{A}.$$

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

- Abstractions compatible with sum and tensor product: $(\bigotimes_i \mathbf{A}_i^{\dagger})(\sum_{i,j} \bigotimes_i \mathbf{F}_i)(\bigotimes_i \mathbf{A}_i) = \sum_{ij} \bigotimes_i (\mathbf{A}^{\dagger}\mathbf{F}_i\mathbf{A})$,
- Average of averages is not the average: $(\mathbf{A}_1\mathbf{A}_2)^{\dagger} \neq \mathbf{A}_2^{\dagger}\mathbf{A}_1^{\dagger}$.

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

$$\mathbf{T}(P)^{\#} = \mathbf{G}\mathbf{T}(P)\mathbf{A} = \mathbf{A}^{\dagger}\mathbf{T}(P)\mathbf{A}.$$

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

- Abstractions compatible with sum and tensor product: $(\bigotimes_i \mathbf{A}_i^{\dagger})(\sum_{i,j} \bigotimes_i \mathbf{F}_i)(\bigotimes_i \mathbf{A}_i) = \sum_{ij} \bigotimes_i (\mathbf{A}^{\dagger}\mathbf{F}_i\mathbf{A})$,
- Average of averages is not the average: $(\mathbf{A}_1\mathbf{A}_2)^{\dagger} \neq \mathbf{A}_2^{\dagger}\mathbf{A}_1^{\dagger}$.

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 $\mathbf{A} : \mathcal{C} \to \mathcal{D}$, construct concretisation $\mathbf{G} : \mathcal{D} \to \mathcal{C}$ to obtain a "smaller" abstract semantics:

$$\mathbf{T}(P)^{\#} = \mathbf{G}\mathbf{T}(P)\mathbf{A} = \mathbf{A}^{\dagger}\mathbf{T}(P)\mathbf{A}.$$

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

- Abstractions compatible with sum and tensor product: $(\bigotimes_i \mathbf{A}_i^{\dagger})(\sum_{i,j} \bigotimes_i \mathbf{F}_i)(\bigotimes_i \mathbf{A}_i) = \sum_{ij} \bigotimes_i (\mathbf{A}^{\dagger}\mathbf{F}_i\mathbf{A})$,
- Average of averages is not the average: $(\mathbf{A}_1\mathbf{A}_2)^{\dagger} \neq \mathbf{A}_2^{\dagger}\mathbf{A}_1^{\dagger}$.

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

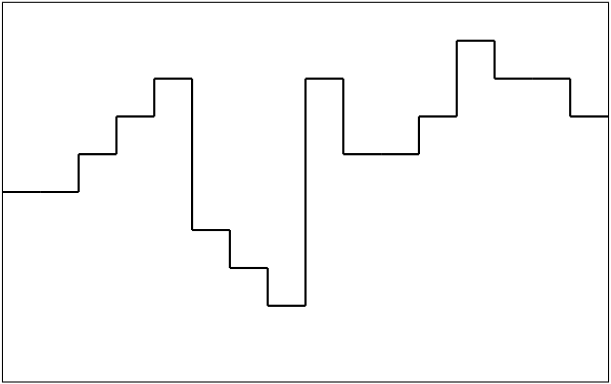
Take an abstraction $\mathbf{A} : \mathcal{C} \to \mathcal{D}$, construct concretisation $\mathbf{G} : \mathcal{D} \to \mathcal{C}$ to obtain a "smaller" abstract semantics:

$$\mathbf{T}(P)^{\#} = \mathbf{G}\mathbf{T}(P)\mathbf{A} = \mathbf{A}^{\dagger}\mathbf{T}(P)\mathbf{A}.$$

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

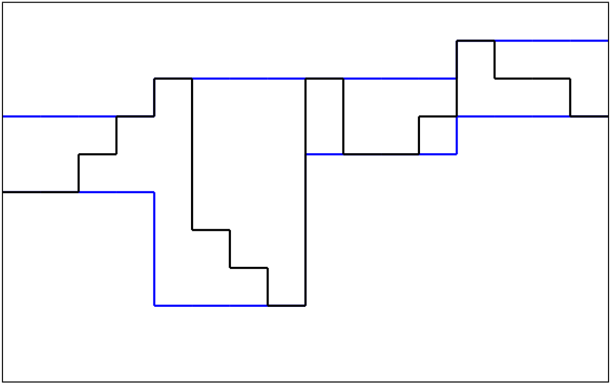
- Abstractions compatible with sum and tensor product: $(\bigotimes_i \mathbf{A}_i^{\dagger})(\sum_{i,j} \bigotimes_i \mathbf{F}_i)(\bigotimes_i \mathbf{A}_i) = \sum_{ij} \bigotimes_i (\mathbf{A}^{\dagger}\mathbf{F}_i\mathbf{A})$,
- Average of averages is not the average: $(\mathbf{A}_1\mathbf{A}_2)^{\dagger} \neq \mathbf{A}_2^{\dagger}\mathbf{A}_1^{\dagger}$.

Approximations: Over, Under, Least Square



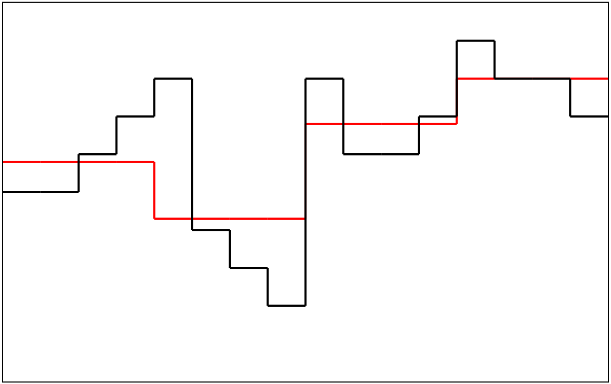
Abstraction: $\mathcal{T}_{16} \rightarrow \mathcal{T}_4$

Approximations: Over, Under, Least Square



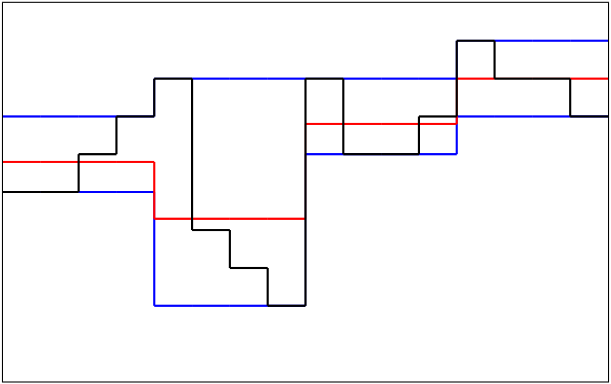
Abstraction: $\mathcal{T}_{16} \rightarrow \mathcal{T}_4$

Approximations: Over, Under, Least Square



Abstraction: $\mathcal{T}_{16} \to \mathcal{T}_4$

Approximations: Over, Under, Least Square



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

Example: Forgetful Abstraction

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

$$\mathbf{A}_f = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix} \qquad \mathbf{A}_f^\dagger = \begin{pmatrix} \frac{1}{n} & \frac{1}{n} & \frac{1}{n} & \frac{1}{n} & \ldots & \frac{1}{n} \end{pmatrix}$$

Example: Forgetful Abstraction

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

$$\mathbf{A}_f = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix} \qquad \mathbf{A}_f^\dagger = \begin{pmatrix} \frac{1}{n} & \frac{1}{n} & \frac{1}{n} & \frac{1}{n} & \ldots & \frac{1}{n} \end{pmatrix}$$

Example: Parity Abstraction

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

$$\mathbf{A}_p = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \\ \vdots & \vdots \\ 0 & 1 \end{pmatrix} \qquad \mathbf{A}_p^\dagger = \begin{pmatrix} \frac{2}{n} & 0 & \frac{2}{n} & 0 & \ldots & 0 \\ 0 & \frac{2}{n} & 0 & \frac{2}{n} & \ldots & \frac{2}{n} \end{pmatrix}$$

Example: Parity Abstraction

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

$$\mathbf{A}_p = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \\ \vdots & \vdots \\ 0 & 1 \end{pmatrix} \qquad \mathbf{A}_p^{\dagger} = \begin{pmatrix} \frac{2}{n} & 0 & \frac{2}{n} & 0 & \ldots & 0 \\ 0 & \frac{2}{n} & 0 & \frac{2}{n} & \ldots & \frac{2}{n} \end{pmatrix}$$

Example: Sign Abstraction

Sign Abstraction $\mathcal{V}(\{-n,\ldots,0,\ldots,n\}) \to \mathcal{V}(\{-,0,+\})$:

$$\mathbf{A}_s = \begin{pmatrix} 1 & 0 & 0 \\ \vdots & \vdots & \vdots \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ \vdots & \vdots & \vdots \\ 0 & 0 & 1 \end{pmatrix} \qquad \mathbf{A}_s^\dagger = \begin{pmatrix} \frac{1}{n} & \ldots & \frac{1}{n} & 0 & 0 & \ldots & 0 \\ 0 & \ldots & 0 & 1 & 0 & \ldots & 0 \\ 0 & \ldots & 0 & 0 & \frac{1}{n} & \ldots & \frac{1}{n} \end{pmatrix}$$

Example: Sign Abstraction

Sign Abstraction $\mathcal{V}(\{-n,\ldots,0,\ldots,n\}) \to \mathcal{V}(\{-,0,+\})$:

$$\mathbf{A}_s = \begin{pmatrix} 1 & 0 & 0 \\ \vdots & \vdots & \vdots \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ \vdots & \vdots & \vdots \\ 0 & 0 & 1 \end{pmatrix} \qquad \mathbf{A}_s^\dagger = \begin{pmatrix} \frac{1}{n} & \ldots & \frac{1}{n} & 0 & 0 & \ldots & 0 \\ 0 & \ldots & 0 & 1 & 0 & \ldots & 0 \\ 0 & \ldots & 0 & 0 & \frac{1}{n} & \ldots & \frac{1}{n} \end{pmatrix}$$

Synthesis and Optimisation

W.: *Program Synthesis and Linear Operator Semantics* in SYNT'14, EPTCS 157, arXiv:1407.5393

Sketches and Templates

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

```
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;
}
```

Armando Solar Lezama, PhD Thesis 2008, p26. (see also: Int. J. Softw. Tools. Technol. Transfer, 2013)

A Probabilistic "Sketching" Framework

Implementation Language: see above

Specification Language: for probabilistic sketches

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

Assertions and Requirements: via abstract properties (PAI).

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

A Probabilistic "Sketching" Framework

Implementation Language: see above

Specification Language: for probabilistic sketches

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

Assertions and Requirements: via abstract properties (PAI).

Implementation: p_i constants (or program variables)

Specification: p_i (mathematical) variables, i.e. λ_i

A Probabilistic “Sketching” Framework

Implementation Language: see above

Specification Language: for probabilistic sketches

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

Assertions and Requirements: via abstract properties (PAI).

Implementation: p_i constants (or program variables)

Specification: p_i (mathematical) variables, i.e. λ_i

A Probabilistic "Sketching" Framework

Implementation Language: see above

Specification Language: for probabilistic sketches

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

Assertions and Requirements: via abstract properties (PAI).

Implementation: p_i constants (or program variables)

Specification: p_i (mathematical) variables, i.e. λ_i

A General Approach

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

$$\ldots [\texttt{choose}]^{\ell}\ p_1 : S_1\ \texttt{or}\ \ldots\ \texttt{or}\ p_n : S_n\ \texttt{ro};\ \ldots$$

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

$$[\![P(\lambda_1, \lambda_2, \ldots, \lambda_n)]\!] = \mathbf{T}(\lambda_1, \lambda_2, \ldots, \lambda_n)$$

- Establish constraints on functional behaviour, e.g.

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

- Additional non-functional (performance) objectives

$$\min_{\lambda_1, \lambda_2, \ldots, \lambda_n} \Phi(\mathbf{T}(\lambda_1, \lambda_2, \ldots, \lambda_n))$$

A General Approach

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

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

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

$$[\![P(\lambda_1, \lambda_2, \ldots, \lambda_n)]\!] = \mathbf{T}(\lambda_1, \lambda_2, \ldots, \lambda_n)$$

- Establish constraints on functional behaviour, e.g.

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

- Additional non-functional (performance) objectives

$$\min_{\lambda_1, \lambda_2, \ldots, \lambda_n} \Phi(\mathbf{T}(\lambda_1, \lambda_2, \ldots, \lambda_n))$$

A General Approach

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

$$\ldots [\mathtt{opt}]^\ell \; S_1 \; \mathtt{or} \; \ldots \; \mathtt{or} \; S_n \; \mathtt{top}; \; \ldots$$

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

$$[\![P(\lambda_1, \lambda_2, \ldots, \lambda_n)]\!] = \mathbf{T}(\lambda_1, \lambda_2, \ldots, \lambda_n)$$

- Establish constraints on functional behaviour, e.g.

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

- Additional non-functional (performance) objectives

$$\min_{\lambda_1, \lambda_2, \ldots, \lambda_n} \Phi(\mathbf{T}(\lambda_1, \lambda_2, \ldots, \lambda_n))$$

A General Approach

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

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

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

$$[\![P(\lambda_1, \lambda_2, \ldots, \lambda_n)]\!] = \mathbf{T}(\lambda_1, \lambda_2, \ldots, \lambda_n)$$

- Establish constraints on functional behaviour, e.g.

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

- Additional non-functional (performance) objectives

$$\min_{\lambda_1, \lambda_2, \ldots, \lambda_n} \Phi(\mathbf{T}(\lambda_1, \lambda_2, \ldots, \lambda_n))$$

A General Approach

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

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

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

$$[\![P(\lambda_1, \lambda_2, \ldots, \lambda_n)]\!] = \mathbf{T}(\lambda_1, \lambda_2, \ldots, \lambda_n)$$

- Establish constraints on functional behaviour, e.g.

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

- Additional non-functional (performance) objectives

$$\min_{\lambda_1, \lambda_2, \ldots, \lambda_n} \Phi(\mathbf{T}(\lambda_1, \lambda_2, \ldots, \lambda_n))$$

A General Approach

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

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

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

$$[\![P(\lambda_1, \lambda_2, \ldots, \lambda_n)]\!] = \mathbf{T}(\lambda_1, \lambda_2, \ldots, \lambda_n)$$

- Establish constraints on functional behaviour, e.g.

$$\mathbf{A}^\dagger \mathbf{T}(\lambda_1, \lambda_2, \ldots, \lambda_n)\mathbf{A} = [\![S]\!]$$

- Additional non-functional (performance) objectives

$$\min_{\lambda_1, \lambda_2, \ldots, \lambda_n} \Phi(\mathbf{T}(\lambda_1, \lambda_2, \ldots, \lambda_n))$$

A General Approach

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

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

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

$$[\![P(\lambda_1, \lambda_2, \ldots, \lambda_n)]\!] = \mathbf{T}(\lambda_1, \lambda_2, \ldots, \lambda_n)$$

- Establish constraints on functional behaviour, e.g.

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

- Additional non-functional (performance) objectives

$$\min_{\lambda_1, \lambda_2, \ldots, \lambda_n} \Phi(\mathbf{T}(\lambda_1, \lambda_2, \ldots, \lambda_n))$$

Demo: Mini Example

Take the following simple program or probabilistic sketch:

```
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:

`[choose]`[1] λ_1 `: {x:=0} or` λ_2 `: {x:=1} or` λ_3 `: {x:=-1} ro.`

The aim is to minimise

$$\Phi(\lambda_1, \lambda_2, \lambda_3) = \|\mathbf{A}_s^\dagger \cdot \mathbf{F}_1(\lambda_1, \lambda_2, \lambda_3) \cdot \mathbf{A}_s - \mathbf{S}\|$$

$$\text{with } \mathbf{S} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} \text{ on } \mathcal{V}(\{-, 0, +\}).$$

Demo: Mini Example

Take the following simple program or probabilistic sketch:

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

The aim is to minimise

$$\Phi(\lambda_1, \lambda_2, \lambda_3) = \|\mathbf{A}_s^\dagger \cdot \mathbf{F}_1(\lambda_1, \lambda_2, \lambda_3) \cdot \mathbf{A}_s - \mathbf{S}\|$$

$$\text{with}\ \ \mathbf{S} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}\ \ \text{on}\ \ \mathcal{V}(\{-, 0, +\}).$$

Demo: Mini Example

Take the following simple program or probabilistic sketch:

```
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:
[`choose`][1] λ_1 : `{x:=0} or` λ_2 : `{x:=1} or` λ_3 : `{x:=-1} ro`.

The aim is to minimise

$$\Phi(\lambda_1, \lambda_2, \lambda_3) = \|\mathbf{A}_s^\dagger \cdot \mathbf{F}_1(\lambda_1, \lambda_2, \lambda_3) \cdot \mathbf{A}_s - \mathbf{S}\|$$

$$\text{with } \mathbf{S} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} \text{ on } \mathcal{V}(\{-, 0, +\}).$$

Swapping: The XOR Trick

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

$$
\begin{array}{c}
[\texttt{choose}]^1\ \lambda_{1,1} : S_1\ \texttt{or} \ldots \texttt{or}\ \lambda_{1,n} : S_n\ \texttt{ro};\\
[\texttt{choose}]^2\ \lambda_{2,1} : S_1\ \texttt{or} \ldots \texttt{or}\ \lambda_{2,n} : S_n\ \texttt{ro};\\
[\texttt{choose}]^3\ \lambda_{3,1} : S_1\ \texttt{or} \ldots \texttt{or}\ \lambda_{3,n} : S_n\ \texttt{ro};
\end{array}
$$

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

$$
\begin{array}{rl}
\multicolumn{2}{c}{[\texttt{skip}]^1} \\
[x := y]^2 & [x := z]^3 \\
[y := x]^4 & [y := z]^5 \\
[z := x]^6 & [z := y]^7 \\
[x := (x+y) \bmod 2]^8 & [x := (x+z) \bmod 2]^9 \\
[y := (y+x) \bmod 2]^{10} & [y := (y+z) \bmod 2]^{11} \\
[z := (z+x) \bmod 2]^{12} & [z := (z+y) \bmod 2]^{13}
\end{array}
$$

Swapping: The XOR Trick

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

$$
\begin{array}{c}
[\texttt{choose}]^1\ \lambda_{1,1} : S_1\ \texttt{or}\ \ldots\ \texttt{or}\ \lambda_{1,n} : S_n\ \texttt{ro};\\
[\texttt{choose}]^2\ \lambda_{2,1} : S_1\ \texttt{or}\ \ldots\ \texttt{or}\ \lambda_{2,n} : S_n\ \texttt{ro};\\
[\texttt{choose}]^3\ \lambda_{3,1} : S_1\ \texttt{or}\ \ldots\ \texttt{or}\ \lambda_{3,n} : S_n\ \texttt{ro};
\end{array}
$$

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

$$
\begin{array}{rl}
\multicolumn{2}{c}{[\texttt{skip}]^1}\\
[x := y]^2 & [x := z]^3\\
[y := x]^4 & [y := z]^5\\
[z := x]^6 & [z := y]^7\\
[x := (x + y) \bmod 2]^8 & [x := (x + z) \bmod 2]^9\\
[y := (y + x) \bmod 2]^{10} & [y := (y + z) \bmod 2]^{11}\\
[z := (z + x) \bmod 2]^{12} & [z := (z + y) \bmod 2]^{13}
\end{array}
$$

Swapping: Parameterised LOS and Objective

Using 13 transfer functions $\mathbf{F}_1 \ldots \mathbf{F}_{13}$ to define

$$\mathbf{T}(\lambda_{ij}) = \prod_{i=1}^{3} \mathbf{T}_i(\lambda_{ij}) \quad \text{with} \quad \mathbf{T}_i(\lambda_{ij}) = \sum_{j=1}^{13} \lambda_{ij} \mathbf{F}_j$$

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

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

Objective function

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

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

Swapping: Parameterised LOS and Objective

Using 13 transfer functions $\mathbf{F}_1 \ldots \mathbf{F}_{13}$ to define

$$\mathbf{T}(\lambda_{ij}) = \prod_{i=1}^{3} \mathbf{T}_i(\lambda_{ij}) \quad \text{with} \quad \mathbf{T}_i(\lambda_{ij}) = \sum_{j=1}^{13} \lambda_{ij} \mathbf{F}_j$$

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

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

Objective function

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

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

Swapping: Parameterised LOS and Objective

Using 13 transfer functions $\mathbf{F}_1 \ldots \mathbf{F}_{13}$ to define

$$\mathbf{T}(\lambda_{ij}) = \prod_{i=1}^{3} \mathbf{T}_i(\lambda_{ij}) \quad \text{with} \quad \mathbf{T}_i(\lambda_{ij}) = \sum_{j=1}^{13} \lambda_{ij} \mathbf{F}_j$$

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

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

Objective function

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

with $\mathbf{A} = \mathbf{I}_{(4)} \otimes \mathbf{A}_{f(2)} = \mathrm{diag}(1,1,1,1) \otimes (1,1)^t$.

Swapping: Optimised LOS

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

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

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

$$0 \leq \lambda_{ij} \leq 1 \text{ for } i = 1, 2, 3 \text{ and } j = 1, \ldots, 13$$

Swapping: Optimised LOS

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

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

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

$$0 \leq \lambda_{ij} \leq 1 \text{ for } i = 1, 2, 3 \text{ and } j = 1, \dots, 13$$

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:

$$\begin{aligned} \mathbf{P}_r &= \mathrm{diag}(0,0,1,0,1,0,0,0,1,0,1,1,1) \\ \mathbf{P}_w &= \mathrm{diag}(0,0,0,0,0,1,1,0,0,0,0,1,1) \end{aligned}$$

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

$$R(\lambda_{ij}) = \| \sum_{i=1}^{3} (\lambda_{ij})_j \mathbf{P}_r \|_1 \text{ and } W(\lambda_{ij}) = \| \sum_{i=1}^{3} (\lambda_{ij})_j \mathbf{P}_w \|_1$$

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:

$$\begin{aligned} \mathbf{P}_r &= \mathrm{diag}(0,0,1,0,1,0,0,0,1,0,1,1,1) \\ \mathbf{P}_w &= \mathrm{diag}(0,0,0,0,0,1,1,0,0,0,0,1,1) \end{aligned}$$

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

$$R(\lambda_{ij}) = \| \sum_{i=1}^{3} (\lambda_{ij})_j \mathbf{P}_r \|_1 \text{ and } W(\lambda_{ij}) = \| \sum_{i=1}^{3} (\lambda_{ij})_j \mathbf{P}_w \|_1$$

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:

$$\begin{aligned}
\mathbf{P}_r &= \mathrm{diag}(0,0,1,0,1,0,0,0,1,0,1,1,1) \\
\mathbf{P}_w &= \mathrm{diag}(0,0,0,0,0,1,1,0,0,0,0,1,1)
\end{aligned}$$

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

$$R(\lambda_{ij}) = \|\sum_{i=1}^{3} (\lambda_{ij})_j \mathbf{P}_r\|_1 \quad \text{and} \quad W(\lambda_{ij}) = \|\sum_{i=1}^{3} (\lambda_{ij})_j \mathbf{P}_w\|_1$$

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 λ_{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 `octave` the optimal λ_{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:

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

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 λ_{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 `octave` the optimal λ_{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:

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

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 λ_{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 `octave` the optimal λ_{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:

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

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 λ_{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 `octave` the optimal λ_{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:

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

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 λ_{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 `octave` the optimal λ_{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:

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

Swapping: Test Runs

For randomly chosen initial values for λ_{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 λ_{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:

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

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

Swapping: Test Runs

For randomly chosen initial values for λ_{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 λ_{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:

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

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

Swapping: Test Runs

For randomly chosen initial values for λ_{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 λ_{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:

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

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

Swapping: Test Runs

For randomly chosen initial values for λ_{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 λ_{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:

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

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

Random Walk on a Go Board $(0 \ldots 18) \times (0 \ldots 18)$

```
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
  ro
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 $c_0 = (0,0) \to (18,18)$ or penalty

$$\Phi_0(\lambda_i) = \sum_{n=1}^{200} n \cdot \|c_0 \cdot (\mathbf{T}(\lambda_i)^n - \mathbf{T}(\lambda_i)^{n-1}) \cdot \mathbf{A}_{\ell_{12}}\|_1 + 1000 \cdot (1 - \|c_0 \cdot \mathbf{T}(\lambda_i)^{200} \cdot \mathbf{A}_{\ell_{12}}\|_1)$$

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

- Additional: Close to diagonal, symmetric $\mathtt{x}$ and $\mathtt{y}$ 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 $\lambda_1 = \mathtt{pxxx} = 0.5$ and $\lambda_7 = \mathtt{pyyy} = 0.5$.

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

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

  Optimum for $\lambda_2 = \mathtt{pxx} = 0.5 = \mathtt{pyy} = \lambda_6$.

Random Walk on a Go Board: Optimisation

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

$$\Phi_0(\lambda_i) = \sum_{n=1}^{200} n \cdot \|c_0 \cdot (\mathbf{T}(\lambda_i)^n - \mathbf{T}(\lambda_i)^{n-1}) \cdot \mathbf{A}_{\ell_{12}}\|_1 + 1000 \cdot (1 - \|c_0 \cdot \mathbf{T}(\lambda_i)^{200} \cdot \mathbf{A}_{\ell_{12}}\|_1)$$

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

- Additional: Close to diagonal, symmetric $\mathtt{x}$ and $\mathtt{y}$ 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 $\lambda_1 = \mathtt{pxxx} = 0.5$ and $\lambda_7 = \mathtt{pyyy} = 0.5$.

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

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

  Optimum for $\lambda_2 = \mathtt{pxx} = 0.5 = \mathtt{pyy} = \lambda_6$.

Random Walk on a Go Board: Optimisation

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

$$\begin{aligned}\Phi_0(\lambda_i) = \; & \textstyle\sum_{n=1}^{200} n \cdot \|c_0 \cdot (\mathbf{T}(\lambda_i)^n - \mathbf{T}(\lambda_i)^{n-1}) \cdot \mathbf{A}_{\ell_{12}}\|_1 + \\ & 1000 \cdot (1 - \|c_0 \cdot \mathbf{T}(\lambda_i)^{200} \cdot \mathbf{A}_{\ell_{12}}\|_1)\end{aligned}$$

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

- Additional: Close to diagonal, symmetric $\mathtt{x}$ and $\mathtt{y}$ 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 $\lambda_1 = \mathtt{pxxx} = 0.5$ and $\lambda_7 = \mathtt{pyyy} = 0.5$.

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

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

  Optimum for $\lambda_2 = \mathtt{pxx} = 0.5 = \mathtt{pyy} = \lambda_6$.

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. $\Phi(\lambda_i) = \|\mathbf{A}^\dagger\mathbf{T}(P(\lambda_i))\mathbf{A} - \mathbf{S}\|$.
- Formal translation of constraints/assertions (e.g. PCTL) into objective function, i.e. $\Phi(\lambda_i) = \|\mathbf{T}(P(\lambda_i)) - PCTL\|$.
- 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. $\Phi(\lambda_i) = \|\mathbf{A}^\dagger\mathbf{T}(P(\lambda_i))\mathbf{A} - \mathbf{S}\|$.
- Formal translation of constraints/assertions (e.g. PCTL) into objective function, i.e. $\Phi(\lambda_i) = \|\mathbf{T}(P(\lambda_i)) - PCTL\|$.
- 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. $\Phi(\lambda_i) = \|\mathbf{A}^\dagger \mathbf{T}(P(\lambda_i))\mathbf{A} - \mathbf{S}\|$.
- Formal translation of constraints/assertions (e.g. PCTL) into objective function, i.e. $\Phi(\lambda_i) = \|\mathbf{T}(P(\lambda_i)) - PCTL\|$.
- 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. $\Phi(\lambda_i) = \|\mathbf{A}^\dagger \mathbf{T}(P(\lambda_i))\mathbf{A} - \mathbf{S}\|$.
- Formal translation of constraints/assertions (e.g. PCTL) into objective function, i.e. $\Phi(\lambda_i) = \|\mathbf{T}(P(\lambda_i)) - PCTL\|$.
- 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. $\Phi(\lambda_i) = \|\mathbf{A}^\dagger \mathbf{T}(P(\lambda_i))\mathbf{A} - \mathbf{S}\|$.
- Formal translation of constraints/assertions (e.g. PCTL) into objective function, i.e. $\Phi(\lambda_i) = \|\mathbf{T}(P(\lambda_i)) - PCTL\|$.
- 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. $\Phi(\lambda_i) = \|\mathbf{A}^\dagger \mathbf{T}(P(\lambda_i))\mathbf{A} - \mathbf{S}\|$.
- Formal translation of constraints/assertions (e.g. PCTL) into objective function, i.e. $\Phi(\lambda_i) = \|\mathbf{T}(P(\lambda_i)) - PCTL\|$.
- Comparison with other approaches in program synthesis (e.g. discrete, quantitative)

Extras: LOS

Structure of LOS (1)

Semantics of statements are sets of global and local operators:

$$[\![.]\!] : \textbf{Stmt} \to \mathcal{P}(\Gamma \cup \Lambda)$$

$$\Gamma = \mathcal{L}(\mathcal{V}(\textbf{Value}^v) \otimes \mathcal{V}(\textbf{Label}))$$

$$\Lambda = \mathcal{L}(\mathcal{V}(\textbf{Value}^v)) \times \textbf{Label}$$

$$\begin{aligned}
[\![[\texttt{skip}]^\ell]\!] &= \{\langle \mathbf{I}, \ell\rangle\} \\
[\![[x := e]^\ell]\!] &= \{\langle \mathbf{U}(x \leftarrow e), \ell\rangle\} \\
[\![[x \mathrel{?}= \rho]^\ell]\!] &= \{\langle \sum_{\langle p,r\rangle \in \rho} p \cdot \mathbf{U}(x \leftarrow r), \ell\rangle\} \\
[\![S_1;\ S_2]\!] &= ([\![S_1]\!] \triangleright init(S2)) \cup [\![S_2]\!]
\end{aligned}$$

where we use the "connection" operator defined as

$$\langle \mathbf{F}, \ell\rangle \triangleright \ell' = \mathbf{F} \otimes \mathbf{E}(\ell, \ell')$$

Structure of LOS (1)

Semantics of statements are sets of global and local operators:

$$[\![.]\!] : \textbf{Stmt} \to \mathcal{P}(\Gamma \cup \Lambda)$$

$$\Gamma = \mathcal{L}(\mathcal{V}(\textbf{Value}^v) \otimes \mathcal{V}(\textbf{Label}))$$

$$\Lambda = \mathcal{L}(\mathcal{V}(\textbf{Value}^v)) \times \textbf{Label}$$

$$\begin{array}{rcl}
[\![[\texttt{skip}]^\ell]\!] & = & \{\langle \mathbf{I}, \ell \rangle\} \\
[\![[x \texttt{ := } e]^\ell]\!] & = & \{\langle \mathbf{U}(x \leftarrow e), \ell \rangle\} \\
[\![[x \texttt{ ?= } \rho]^\ell]\!] & = & \{\langle \sum_{\langle p,r \rangle \in \rho} p \cdot \mathbf{U}(x \leftarrow r), \ell \rangle\} \\
[\![S_1;\ S_2]\!] & = & ([\![S_1]\!] \rhd init(S2)) \cup [\![S_2]\!]
\end{array}$$

where we use the “connection” operator defined as

$$\langle \mathbf{F}, \ell \rangle \rhd \ell' = \mathbf{F} \otimes \mathbf{E}(\ell, \ell')$$

The Structure of LOS (2)

$$
\begin{aligned}
[\![[\texttt{choose}]^\ell\ p_1 : S_1\ \texttt{or}\ p_2 : S_2\ \texttt{ro}]\!] &= \{\tilde{p}_1 \cdot \mathbf{I} \otimes \mathbf{E}(\ell, init(S_1)\} \cup \\
&\quad \{\tilde{p}_2 \cdot \mathbf{I} \otimes \mathbf{E}(\ell, init(S_2)\} \cup \\
&\quad [\![S_1]\!] \cup [\![S_2]\!] \\
[\![\texttt{if}\ [b]^\ell\ \texttt{then}\ S_1\ \texttt{else}\ S_2\ \texttt{fi}]\!] &= \{\langle \mathbf{P}(b), \ell\rangle\} \rhd init(S_1)\} \cup \\
&\quad \{\langle \mathbf{P}(b)^\perp, \ell\rangle\} \rhd init(S_2)\} \cup \\
&\quad [\![S_1]\!] \cup [\![S_2]\!] \\
[\![\texttt{while}\ [b]^\ell\ \texttt{do}\ S\ \texttt{od}]\!] &= \{\langle \mathbf{P}(b), \ell\rangle\} \rhd init(S)\} \cup \\
&\quad [\![S]\!] \cup \{\langle \mathbf{P}(b)^\perp, \ell\rangle\}
\end{aligned}
$$

With normalisation (at compile time) $\tilde{p}_i = \frac{p_i}{p_1+p_2}$ and adding a final loop ℓ^*, i.e. $[\![P]\!] \rhd \ell^*$, 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 σ and basic $\mathbf{T}_i$'s are:

Assigning a variable a constant value c:

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

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

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

We also use the identity operator/matrix $\mathbf{I}$ and matrix unites $\mathbf{E}_{ij}$.

Basic Operators

If we have a single variable x (with finite set of values) then its probabilistic state is a (row) vector σ and basic $\mathbf{T}_i$'s are:

Assigning a variable a constant value c:

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

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

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

We also use the identity operator/matrix $\mathbf{I}$ and matrix unites $\mathbf{E}_{ij}$.

Basic Operators

If we have a single variable x (with finite set of values) then its probabilistic state is a (row) vector σ and basic $\mathbf{T}_i$'s are:

Assigning a variable a constant value c:

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

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

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

We also use the identity operator/matrix $\mathbf{I}$ and matrix unites $\mathbf{E}_{ij}$.

Multi-Variable Operations

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

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

$$\mathbf{P}(s) = \bigotimes_{i=1}^{v} \mathbf{P}(s(\mathtt{x}_i))$$

$$\mathbf{P}(e = c) = \sum_{\mathcal{E}(e)s=c} \mathbf{P}(s)$$

Multi-Variable Operations

Assigning a variable a constant value c and an expression e:

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

$$\mathbf{U}(\mathtt{x}_k \leftarrow c) = \bigotimes_{i=1}^{k-1} \mathbf{I} \otimes \mathbf{U}(c) \otimes \bigotimes_{i=k+1}^{v} \mathbf{I}$$

$$\mathbf{U}(\mathtt{x}_k \leftarrow e) = \sum_{c} \mathbf{P}(e = c)\mathbf{U}(\mathtt{x}_k \leftarrow c)$$

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 $[\![P]\!]_K$ be Kozen's semantics of P and $\mathbf{T}(P)$ the LOS. Then

$$(s_0 \otimes e_0) \cdot (\lim_{n\to\infty} \mathbf{T}(P)^n) \cdot \mathbf{S}_{\ell*} = s_0 \cdot [\![P]\!]_K.$$

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

Note that the tensor product construction in LOS gives a more structured semantics than in the case of Kozen's semantics.

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 $\mathbf{T}(P)$ the LOS. Then

$$(s_0 \otimes e_0) \cdot (\lim_{n\to\infty} \mathbf{T}(P)^n) \cdot \mathbf{S}_{\ell*} = s_0 \cdot [\![P]\!]_K.$$

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

Note that the tensor product construction in LOS gives a more structured semantics than in the case of Kozen's semantics.

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 $\mathbf{T}(P)$ the LOS. Then

$$(s_0 \otimes e_0) \cdot (\lim_{n \to \infty} \mathbf{T}(P)^n) \cdot \mathbf{S}_{\ell^*} = s_0 \cdot [\![P]\!]_K.$$

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

Note that the tensor product construction in LOS gives a more structured semantics than in the case of Kozen's semantics.

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

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

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 ℓ_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 $\mathbf{T}(P)$ for:

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

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 ℓ_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 $\mathbf{T}(P)$ for:

$$\texttt{while } \mathbf{true} \texttt{ do } x \mathrel{?=} \{0,1\} \texttt{ od}$$
$$\texttt{while } \mathbf{true} \texttt{ do } x := x+1 \texttt{ od}$$

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 ℓ_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 $\mathbf{T}(P)$ for:

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

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 ℓ_2

Extras: PAI

Example: Function Approximation

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 $\mathbb{R}^n$.

Example: Function Approximation

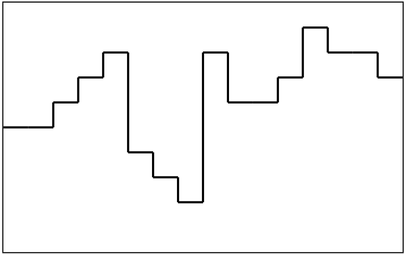
Concrete and abstract domain are sets of step-functions on an interval $[a, b]$. The set of (real-valued) step-function $\mathcal{T}_n$ is based on the sub-division of the interval into n sub-intervals. Each step function in $\mathcal{T}_n$ corresponds to a vector in $\mathbb{R}^n$.

Example: Function Approximation

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 $\mathbb{R}^n$.

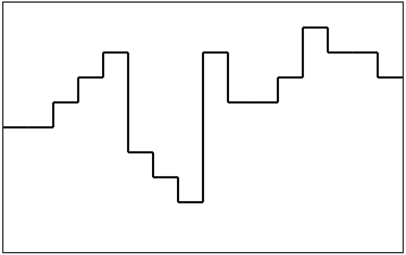
Example: Function Approximation

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 $\mathbb{R}^n$.



Example: Function Approximation

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 $\mathbb{R}^n$.



$$f = \begin{pmatrix} 5 & 5 & 6 & 7 & 8 & 4 & 3 & 2 & 8 & 6 & 6 & 7 & 9 & 8 & 8 & 7 \end{pmatrix}$$

Example: Abstractions $\mathcal{T}_{16} \rightarrow \mathcal{T}_8$

$$
\mathbf{A}_8 = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}
$$

Example: Abstractions $\mathcal{T}_{16} \to \mathcal{T}_8$

$$\mathbf{G}_8 = \begin{pmatrix}
\frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2}
\end{pmatrix}$$

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

$$\mathbf{A}_4 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Example: Abstractions $\mathcal{T}_{16} \to \mathcal{T}_4$

$$\mathbf{G}_4 = \begin{pmatrix} \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \end{pmatrix}$$

Example: Abstractions $\mathcal{T}_{16} \rightarrow \mathcal{T}_2$

$$\mathbf{A}_2 = \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{pmatrix}$$

Example: Abstractions $\mathcal{T}_{16} \rightarrow \mathcal{T}_2$

$$\mathbf{G}_2 = \begin{pmatrix} \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} \end{pmatrix}$$

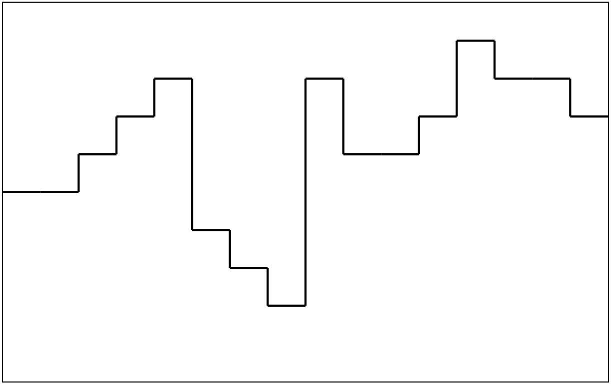
Example: Abstractions $\mathcal{T}_{16} \to \mathcal{T}_1$

$$\mathbf{A}_1 = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

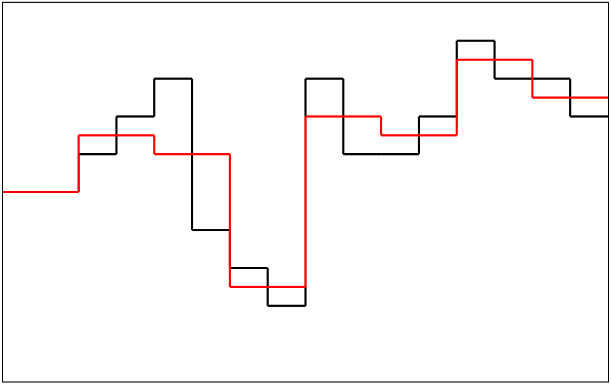
Example: Abstractions $\mathcal{T}_{16} \rightarrow \mathcal{T}_1$

$$\mathbf{G}_1 = \begin{pmatrix} \frac{1}{16} & \frac{1}{16} & \frac{1}{16} & \frac{1}{16} & \frac{1}{16} & \frac{1}{16} & \frac{1}{16} & \frac{1}{16} & \frac{1}{16} & \frac{1}{16} & \frac{1}{16} & \frac{1}{16} & \frac{1}{16} & \frac{1}{16} & \frac{1}{16} & \frac{1}{16} \end{pmatrix}$$

Example: Geometric Interpretation

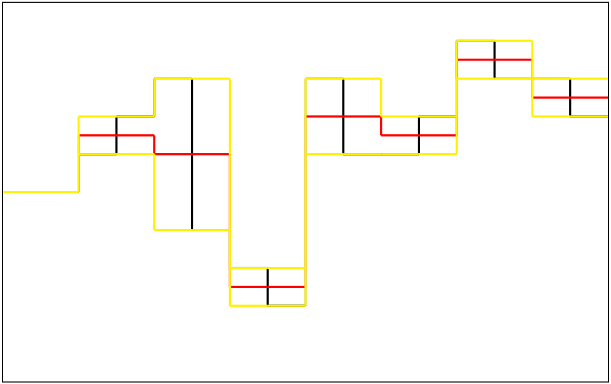


Example: Geometric Interpretation



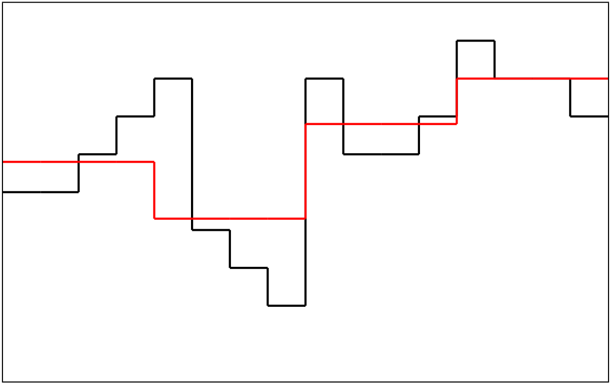
$\mathcal{T}_{16} \to \mathcal{T}_8$

Example: Geometric Interpretation



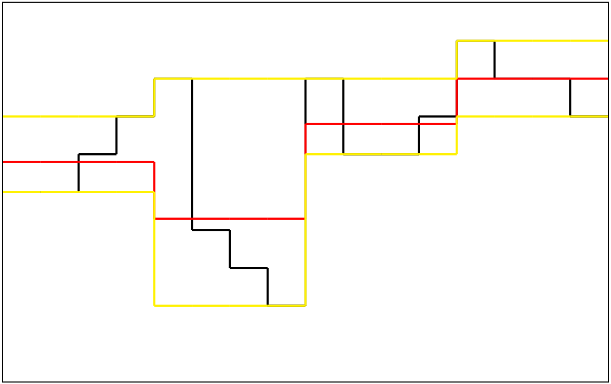
$\mathcal{T}_{16} \rightarrow \mathcal{T}_{8}$

Example: Geometric Interpretation



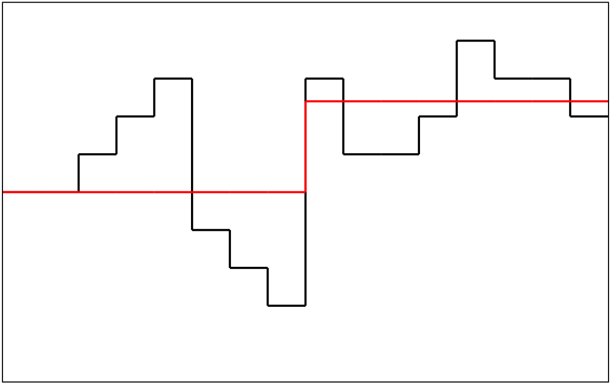
$\mathcal{T}_{16} \rightarrow \mathcal{T}_4$

Example: Geometric Interpretation



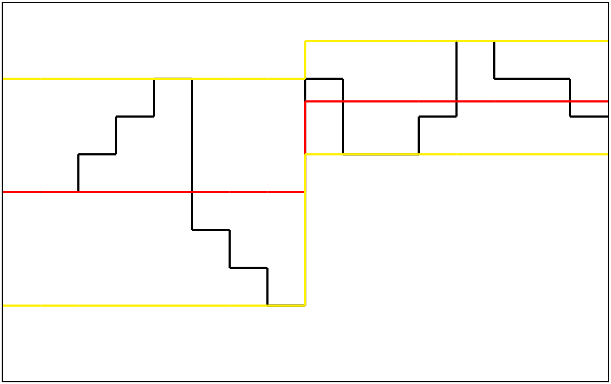
$\mathcal{T}_{16} \to \mathcal{T}_4$

Example: Geometric Interpretation



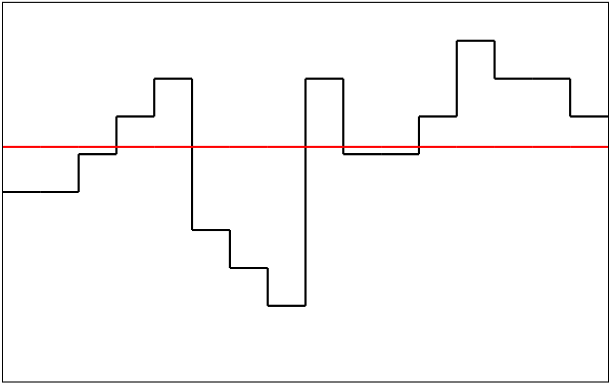
$\mathcal{T}_{16} \rightarrow \mathcal{T}_{2}$

Example: Geometric Interpretation



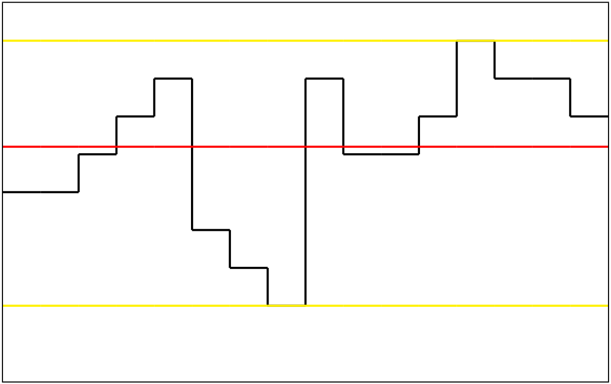
$\mathcal{T}_{16} \to \mathcal{T}_2$

Example: Geometric Interpretation



$\mathcal{T}_{16} \to \mathcal{T}_1$

Example: Geometric Interpretation



$\mathcal{T}_{16} \rightarrow \mathcal{T}_1$