Probabilistic Program Analysis

Probablistic Abstract Interpretation

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

Herbert Wiklicky
Imperial College London, UK
herbert@doc.ic.uk

Approximation and Correctness

Data-flow analyses can be re-formulated in a different scenario where correctness is guaranteed by construction.

Classically, the theory of Abstract Interpretation allows us to

- construct simplified (computable) abstract semantics
- construct approximate solutions
- obtain the correctness of the approximate solution by construction.

Approximation and Correctness

Data-flow analyses can be re-formulated in a different scenario where correctness is guaranteed by construction.

Classically, the theory of Abstract Interpretation allows us to

- construct simplified (computable) abstract semantics
- construct approximate solutions
- obtain the correctness of the approximate solution by construction.

Approximation and Correctness

Data-flow analyses can be re-formulated in a different scenario where correctness is guaranteed by construction.

Classically, the theory of Abstract Interpretation allows us to

- construct simplified (computable) abstract semantics
- construct approximate solutions
- obtain the correctness of the approximate solution by construction.

Approximation and Correctness

Data-flow analyses can be re-formulated in a different scenario where correctness is guaranteed by construction.

Classically, the theory of Abstract Interpretation allows us to

- construct simplified (computable) abstract semantics
- construct approximate solutions
- obtain the correctness of the approximate solution by construction.

Approximation and Correctness

Data-flow analyses can be re-formulated in a different scenario where correctness is guaranteed by construction.

Classically, the theory of Abstract Interpretation allows us to

- construct simplified (computable) abstract semantics
- construct approximate solutions
- obtain the correctness of the approximate solution by construction.

Notions of Approximation

In order theoretic structures we are looking for
Safe Approximations

$$s^* \sqsubseteq s \quad \text{or} \quad s \sqsubseteq s^*$$

In quantitative, vector space structures we want
Close Approximations

$$\|s - s^*\| = \min_x \|s - x\|$$

Notions of Approximation

In order theoretic structures we are looking for
Safe Approximations

$$s^* \sqsubseteq s \quad \text{or} \quad s \sqsubseteq s^*$$

In quantitative, vector space structures we want
Close Approximations

$$\|s - s^*\| = \min_x \|s - x\|$$

Example: Function Approximation

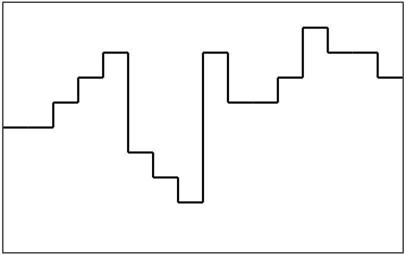
Concrete and abstract domain are step-functions on $[a, b]$.
The set of (real-valued) step-function $\mathcal{T}_n$ is based on the sub-division of the interval into n sub-intervals.

Example: Function Approximation

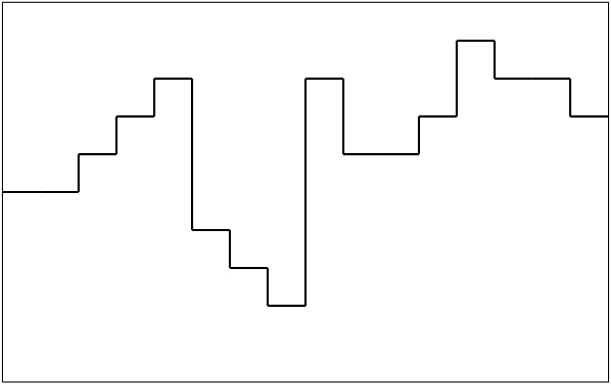
Concrete and abstract domain are step-functions on $[a, b]$.
The set of (real-valued) step-function $\mathcal{T}_n$ is based on the sub-division of the interval into n sub-intervals.

Example: Function Approximation

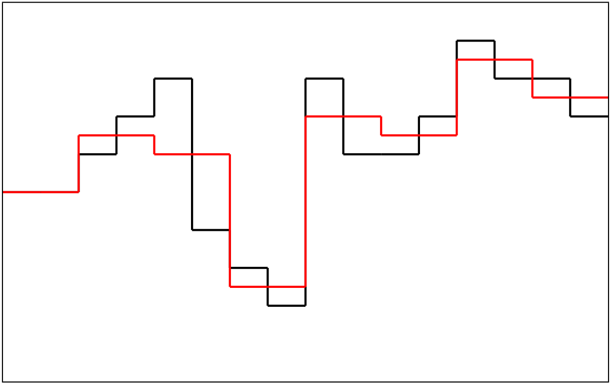
Concrete and abstract domain are step-functions on $[a, b]$.
The set of (real-valued) step-function $\mathcal{T}_n$ is based on the sub-division of the interval into n sub-intervals.



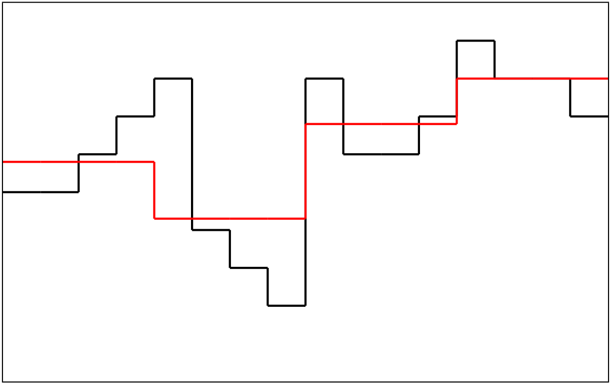
Close Approximations



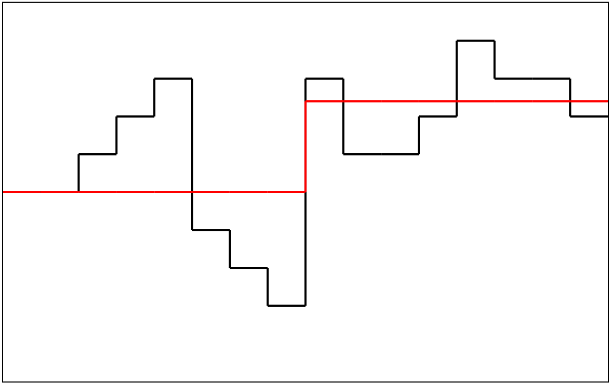
Close Approximations



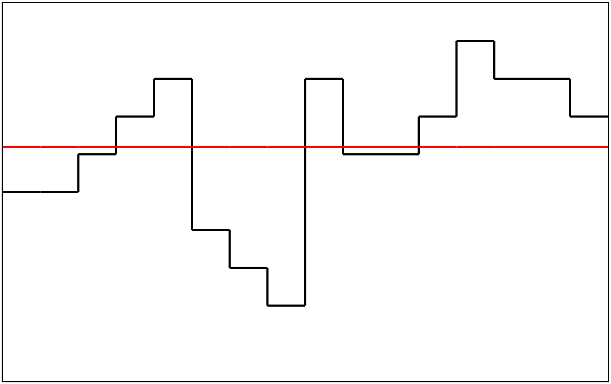
Close Approximations



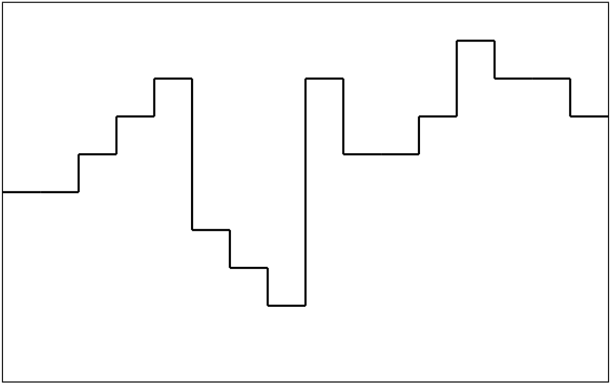
Close Approximations



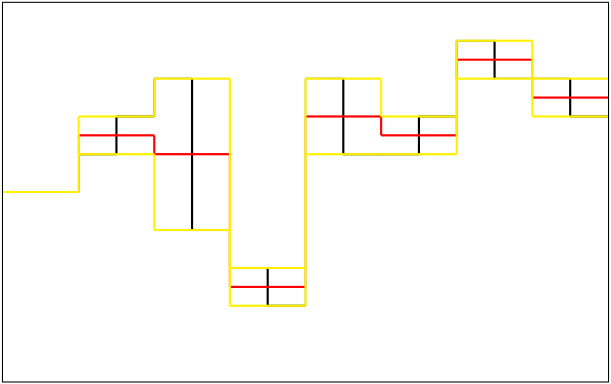
Close Approximations



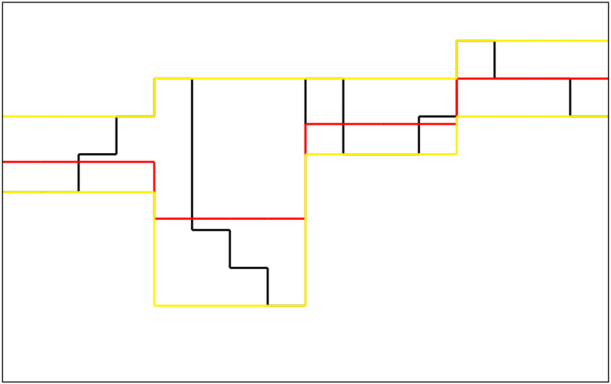
Close vs Correct Approximations



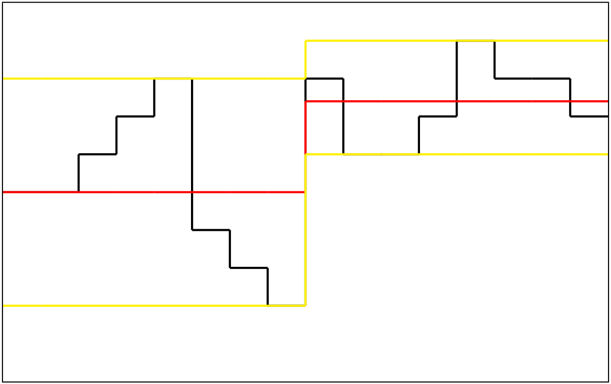
Close vs Correct Approximations



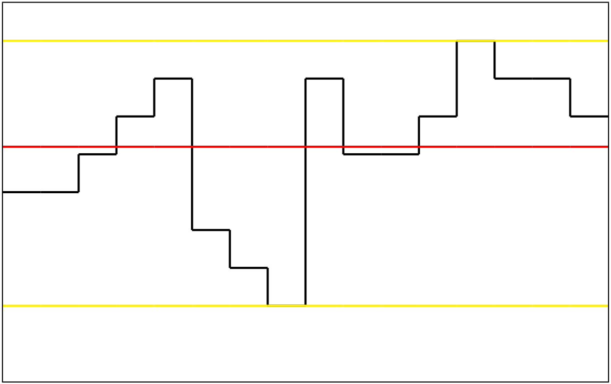
Close vs Correct Approximations



Close vs Correct Approximations



Close vs Correct Approximations



Abstract Interpretation

Some problems may be have too costly solutions or be uncomputable on a concrete space (complete lattice).

Find abstract descriptions on which computations are easier; then relate the concrete and abstract solutions.

Definition

Let $\mathcal{C} = (\mathcal{C}, \leq)$ and $\mathcal{D} = (\mathcal{D}, \sqsubseteq)$ be two partially ordered set. If there are two functions $\alpha : \mathcal{C} \to \mathcal{D}$ and $\gamma : \mathcal{D} \to \mathcal{C}$ such that for all $c \in \mathcal{C}$ and all $d \in \mathcal{D}$:

$$c \leq_{\mathcal{C}} \gamma(d) \text{ iff } \alpha(c) \sqsubseteq d,$$

then $(\mathcal{C}, \alpha, \gamma, \mathcal{D})$ form a Galois connection.

Abstract Interpretation

Some problems may be have too costly solutions or be uncomputable on a concrete space (complete lattice).
Find abstract descriptions on which computations are easier; then relate the concrete and abstract solutions.

Definition

Let $\mathcal{C} = (\mathcal{C}, \leq)$ and $\mathcal{D} = (\mathcal{D}, \sqsubseteq)$ be two partially ordered set. If there are two functions $\alpha : \mathcal{C} \rightarrow \mathcal{D}$ and $\gamma : \mathcal{D} \rightarrow \mathcal{C}$ such that for all $c \in \mathcal{C}$ and all $d \in \mathcal{D}$:

$$c \leq_{\mathcal{C}} \gamma(d) \text{ iff } \alpha(c) \sqsubseteq d,$$

then $(\mathcal{C}, \alpha, \gamma, \mathcal{D})$ form a Galois connection.

Abstract Interpretation

Some problems may be have too costly solutions or be uncomputable on a concrete space (complete lattice).
Find abstract descriptions on which computations are easier; then relate the concrete and abstract solutions.

Definition

Let $\mathcal{C} = (\mathcal{C}, \leq)$ and $\mathcal{D} = (\mathcal{D}, \sqsubseteq)$ be two partially ordered set. If there are two functions $\alpha : \mathcal{C} \to \mathcal{D}$ and $\gamma : \mathcal{D} \to \mathcal{C}$ such that for all $c \in \mathcal{C}$ and all $d \in \mathcal{D}$:

$$c \leq_{\mathcal{C}} \gamma(d) \text{ iff } \alpha(c) \sqsubseteq d,$$

then $(\mathcal{C}, \alpha, \gamma, \mathcal{D})$ form a Galois connection.

Galois Connections

Definition

Let $\mathcal{C} = (\mathcal{C}, \leq_{\mathcal{C}})$ and $\mathcal{D} = (\mathcal{D}, \leq_{\mathcal{D}})$ be two partially ordered sets with two order-preserving functions $\alpha : \mathcal{C} \mapsto \mathcal{D}$ and $\gamma : \mathcal{D} \mapsto \mathcal{C}$. Then $(\mathcal{C}, \alpha, \gamma, \mathcal{D})$ form a Galois connection iff

(i) $\alpha \circ \gamma$ is reductive i.e. $\forall d \in \mathcal{D}$, $\alpha \circ \gamma(d) \leq_{\mathcal{D}} d$,

(ii) $\gamma \circ \alpha$ is extensive i.e. $\forall c \in \mathcal{C}$, $c \leq_{\mathcal{C}} \gamma \circ \alpha(c)$.

Proposition

Let $(\mathcal{C}, \alpha, \gamma, \mathcal{D})$ be a Galois connection. Then α and γ are quasi-inverse, i.e.

(i) $\alpha \circ \gamma \circ \alpha = \alpha$

(ii) $\gamma \circ \alpha \circ \gamma = \gamma$

Galois Connections

Definition

Let $\mathcal{C} = (\mathcal{C}, \leq_{\mathcal{C}})$ and $\mathcal{D} = (\mathcal{D}, \leq_{\mathcal{D}})$ be two partially ordered sets with two order-preserving functions $\alpha : \mathcal{C} \mapsto \mathcal{D}$ and $\gamma : \mathcal{D} \mapsto \mathcal{C}$. Then $(\mathcal{C}, \alpha, \gamma, \mathcal{D})$ form a Galois connection iff

(i) $\alpha \circ \gamma$ is reductive i.e. $\forall d \in \mathcal{D}$, $\alpha \circ \gamma(d) \leq_{\mathcal{D}} d$,

(ii) $\gamma \circ \alpha$ is extensive i.e. $\forall c \in \mathcal{C}$, $c \leq_{\mathcal{C}} \gamma \circ \alpha(c)$.

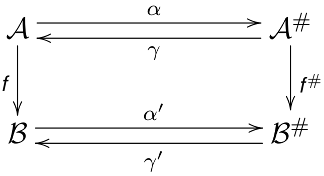
Proposition

Let $(\mathcal{C}, \alpha, \gamma, \mathcal{D})$ be a Galois connection. Then α and γ are quasi-inverse, i.e.

(i) $\alpha \circ \gamma \circ \alpha = \alpha$

(ii) $\gamma \circ \alpha \circ \gamma = \gamma$

General Construction



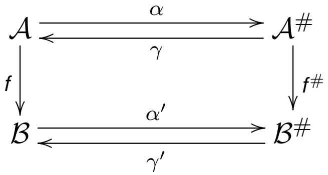
Correct approximation:

$$\alpha' \circ f \leq_\# f^\# \circ \alpha.$$

Induced semantics:

$$f^\# = \alpha \circ f \circ \gamma.$$

General Construction



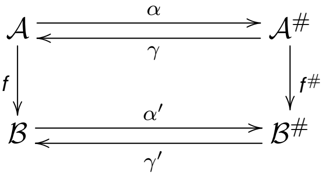
Correct approximation:

$$\alpha' \circ f \leq_{\#} f^{\#} \circ \alpha.$$

Induced semantics:

$$f^{\#} = \alpha \circ f \circ \gamma.$$

General Construction



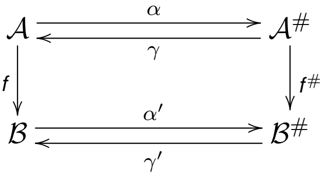
Correct approximation:

$$\alpha' \circ f \leq_{\#} f^{\#} \circ \alpha.$$

Induced semantics:

$$f^{\#} = \alpha \circ f \circ \gamma.$$

General Construction



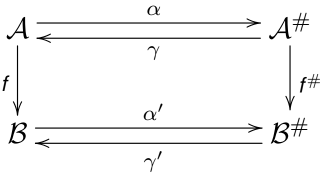
Correct approximation:

$$\alpha' \circ f \leq_\# f^\# \circ \alpha.$$

Induced semantics:

$$f^\# = \alpha \circ f \circ \gamma.$$

General Construction



Correct approximation:

$$\alpha' \circ f \leq_\# f^\# \circ \alpha.$$

Induced semantics:

$$f^\# = \alpha \circ f \circ \gamma.$$

Probabilistic Abstraction Domains

A probabilistic domain is essentially a vector space which represents the distributions $Dist(S)$ on the state space S of a probabilistic transition system, i.e. for finite state spaces

$$\mathcal{V}(S) = \{ (v_s)_{s \in S} \mid v_s \in \mathbb{R} \}.$$

In the finite setting we can identify $\mathcal{V}(S)$ with the Hilbert space $\ell^2(S)$.

The notion of *norm* is essential for our treatment; we will consider normed vector spaces.

Probabilistic Abstraction Domains

A probabilistic domain is essentially a vector space which represents the distributions $Dist(S)$ on the state space S of a probabilistic transition system, i.e. for finite state spaces

$$\mathcal{V}(S) = \{ (v_s)_{s \in S} \mid v_s \in \mathbb{R} \}.$$

In the finite setting we can identify $\mathcal{V}(S)$ with the Hilbert space $\ell^2(S)$.

The notion of *norm* is essential for our treatment; we will consider normed vector spaces.

Probabilistic Abstraction Domains

A probabilistic domain is essentially a vector space which represents the distributions $Dist(S)$ on the state space S of a probabilistic transition system, i.e. for finite state spaces

$$\mathcal{V}(S) = \{ (v_s)_{s \in S} \mid v_s \in \mathbb{R} \}.$$

In the finite setting we can identify $\mathcal{V}(S)$ with the Hilbert space $\ell^2(S)$.

The notion of *norm* is essential for our treatment; we will consider normed vector spaces.

Probabilistic Abstraction Domains

A probabilistic domain is essentially a vector space which represents the distributions *Dist*(S) on the state space S of a probabilistic transition system, i.e. for finite state spaces

$$\mathcal{V}(S) = \{ (v_s)_{s \in S} \mid v_s \in \mathbb{R}\}.$$

In the finite setting we can identify $\mathcal{V}(S)$ with the Hilbert space $\ell^2(S)$.

The notion of *norm* is essential for our treatment; we will consider normed vector spaces.

Norm and Operator Norm

A norm on a vector space $\mathcal{V}$ is a map $\|.\| : \mathcal{V} \mapsto \mathbb{R}$ such that for all $v, w \in \mathcal{V}$ and $c \in \mathbb{C}$:

- $\|v\| \geq 0$,
- $\|v\| = 0 \Leftrightarrow v = o$,
- $\|cv\| = |c|\|v\|$,
- $\|v + w\| \leq \|v\| + \|w\|$,

with $o \in \mathcal{V}$ the zero vector.

We can always use a norm to define a metric topology on a vector space via the distance function $d(v, w) = \|v - w\|$.

$$\|\mathbf{M}\| = \sup_{v \in \mathcal{V}} \frac{\|\mathbf{M}(v)\|}{\|v\|} = \sup_{\|v\|=1} \|\mathbf{M}(v)\|.$$

Norm and Operator Norm

A norm on a vector space $\mathcal{V}$ is a map $\|.\| : \mathcal{V} \mapsto \mathbb{R}$ such that for all $v, w \in \mathcal{V}$ and $c \in \mathbb{C}$:

- $\|v\| \geq 0$,
- $\|v\| = 0 \Leftrightarrow v = o$,
- $\|cv\| = |c|\|v\|$,
- $\|v + w\| \leq \|v\| + \|w\|$,

with $o \in \mathcal{V}$ the zero vector.

We can always use a norm to define a metric topology on a vector space via the distance function $d(v, w) = \|v - w\|$.

$$\|\mathbf{M}\| = \sup_{v \in \mathcal{V}} \frac{\|\mathbf{M}(v)\|}{\|v\|} = \sup_{\|v\|=1} \|\mathbf{M}(v)\|.$$

Norm and Operator Norm

A norm on a vector space $\mathcal{V}$ is a map $\|.\| : \mathcal{V} \mapsto \mathbb{R}$ such that for all $v, w \in \mathcal{V}$ and $c \in \mathbb{C}$:

- $\|v\| \geq 0$,
- $\|v\| = 0 \Leftrightarrow v = o$,
- $\|cv\| = |c|\|v\|$,
- $\|v + w\| \leq \|v\| + \|w\|$,

with $o \in \mathcal{V}$ the zero vector.

We can always use a norm to define a metric topology on a vector space via the distance function $d(v, w) = \|v - w\|$.

$$\|\mathbf{M}\| = \sup_{v \in \mathcal{V}} \frac{\|\mathbf{M}(v)\|}{\|v\|} = \sup_{\|v\|=1} \|\mathbf{M}(v)\|.$$

Norm and Operator Norm

A norm on a vector space $\mathcal{V}$ is a map $\|.\| : \mathcal{V} \mapsto \mathbb{R}$ such that for all $v, w \in \mathcal{V}$ and $c \in \mathbb{C}$:

- $\|v\| \geq 0$,
- $\|v\| = 0 \Leftrightarrow v = o$,
- $\|cv\| = |c|\|v\|$,
- $\|v + w\| \leq \|v\| + \|w\|$,

with $o \in \mathcal{V}$ the zero vector.

We can always use a norm to define a metric topology on a vector space via the distance function $d(v, w) = \|v - w\|$.

$$\|\mathbf{M}\| = \sup_{v \in \mathcal{V}} \frac{\|\mathbf{M}(v)\|}{\|v\|} = \sup_{\|v\|=1} \|\mathbf{M}(v)\|.$$

Norm and Operator Norm

A norm on a vector space $\mathcal{V}$ is a map $\|.\| : \mathcal{V} \mapsto \mathbb{R}$ such that for all $v, w \in \mathcal{V}$ and $c \in \mathbb{C}$:

- $\|v\| \geq 0$,
- $\|v\| = 0 \Leftrightarrow v = o$,
- $\|cv\| = |c|\|v\|$,
- $\|v + w\| \leq \|v\| + \|w\|$,

with $o \in \mathcal{V}$ the zero vector.

We can always use a norm to define a metric topology on a vector space via the distance function $d(v, w) = \|v - w\|$.

$$\|\mathbf{M}\| = \sup_{v \in \mathcal{V}} \frac{\|\mathbf{M}(v)\|}{\|v\|} = \sup_{\|v\|=1} \|\mathbf{M}(v)\|.$$

Norm and Operator Norm

A norm on a vector space $\mathcal{V}$ is a map $\|.\| : \mathcal{V} \mapsto \mathbb{R}$ such that for all $v, w \in \mathcal{V}$ and $c \in \mathbb{C}$:

- $\|v\| \geq 0$,
- $\|v\| = 0 \Leftrightarrow v = o$,
- $\|cv\| = |c|\|v\|$,
- $\|v + w\| \leq \|v\| + \|w\|$,

with $o \in \mathcal{V}$ the zero vector.

We can always use a norm to define a metric topology on a vector space via the distance function $d(v, w) = \|v - w\|$.

$$\|\mathbf{M}\| = \sup_{v \in \mathcal{V}} \frac{\|\mathbf{M}(v)\|}{\|v\|} = \sup_{\|v\|=1} \|\mathbf{M}(v)\|.$$

Norm and Operator Norm

A norm on a vector space $\mathcal{V}$ is a map $\|.\| : \mathcal{V} \mapsto \mathbb{R}$ such that for all $v, w \in \mathcal{V}$ and $c \in \mathbb{C}$:

- $\|v\| \geq 0$,
- $\|v\| = 0 \Leftrightarrow v = o$,
- $\|cv\| = |c|\|v\|$,
- $\|v + w\| \leq \|v\| + \|w\|$,

with $o \in \mathcal{V}$ the zero vector.

We can always use a norm to define a metric topology on a vector space via the distance function $d(v, w) = \|v - w\|$.

$$\|\mathbf{M}\| = \sup_{v \in \mathcal{V}} \frac{\|\mathbf{M}(v)\|}{\|v\|} = \sup_{\|v\|=1} \|\mathbf{M}(v)\|.$$

Norm and Operator Norm

A norm on a vector space $\mathcal{V}$ is a map $\|.\| : \mathcal{V} \mapsto \mathbb{R}$ such that for all $v, w \in \mathcal{V}$ and $c \in \mathbb{C}$:

- $\|v\| \geq 0$,
- $\|v\| = 0 \Leftrightarrow v = o$,
- $\|cv\| = |c|\|v\|$,
- $\|v + w\| \leq \|v\| + \|w\|$,

with $o \in \mathcal{V}$ the zero vector.

We can always use a norm to define a metric topology on a vector space via the distance function $d(v, w) = \|v - w\|$.

$$\|\mathbf{M}\| = \sup_{v \in \mathcal{V}} \frac{\|\mathbf{M}(v)\|}{\|v\|} = \sup_{\|v\|=1} \|\mathbf{M}(v)\|.$$

Generalised Inverse

Definition

Let $\mathcal{C}$ and $\mathcal{D}$ be two finite-dimensional vector spaces and $\mathbf{A} : \mathcal{C} \to \mathcal{D}$ a linear map. Then the linear map $\mathbf{A}^\dagger = \mathbf{G} : \mathcal{D} \to \mathcal{C}$ is the Moore-Penrose pseudo-inverse of $\mathbf{A}$ iff

(i) $\mathbf{A} \circ \mathbf{G} = \mathbf{P}_A$,

(ii) $\mathbf{G} \circ \mathbf{A} = \mathbf{P}_G$,

where $\mathbf{P}_A$ and $\mathbf{P}_G$ denote orthogonal projections onto the ranges of $\mathbf{A}$ and $\mathbf{G}$.

Least Squares Solutions

Definition

Let $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{b} \in \mathbb{R}^m$. Then $\mathbf{u} \in \mathbb{R}^n$ is called a least squares solution to $\mathbf{Ax} = \mathbf{b}$ if

$$\|\mathbf{Au} - \mathbf{b}\| \leq \|\mathbf{Av} - \mathbf{b}\|, \text{ for all } \mathbf{v} \in \mathbb{R}^n.$$

Theorem

Let $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{b} \in \mathbb{R}^m$. Then $\mathbf{A}^\dagger \mathbf{b}$ is the minimal least squares solution to $\mathbf{Ax} = \mathbf{b}$.

Orthogonal Projections

Corollary

Let $\mathbf{P}$ *be a orthogonal projection on a finite dimensional vector space* $\mathcal{V}$. *Then for any* $\mathbf{x} \in \mathcal{V}$, $\mathbf{Px}$ *is the unique closest vector in* $\mathcal{V}$ *to* $\mathbf{x}$ *wrt the Euclidean norm.*

Extraction Functions

An extraction function $\eta : C \mapsto D$ is a mapping from a set of values to their descriptions in D.

It is easy to show that

Proposition

Given an extraction function $\eta : \mathcal{C} \mapsto \mathcal{D}$, the quadruple $(\mathcal{P}(\mathcal{C}), \alpha_\eta, \gamma_\eta, \mathcal{P}(\mathcal{D}))$ is a Galois connection with α_η and γ_η defined by:

$$\alpha_\eta(C') = \{\eta(c) \mid c \in C'\} \text{ and } \gamma_\eta(D') = \{v \mid \eta(v) \in D'\}$$

Extraction Functions

An extraction function $\eta : C \mapsto D$ is a mapping from a set of values to their descriptions in D.
It is easy to show that

Proposition

Given an extraction function $\eta : \mathcal{C} \mapsto \mathcal{D}$, the quadruple $(\mathcal{P}(\mathcal{C}), \alpha_\eta, \gamma_\eta, \mathcal{P}(\mathcal{D}))$ is a Galois connection with α_η and γ_η defined by:

$$\alpha_\eta(C') = \{\eta(c) \mid c \in C'\} \text{ and } \gamma_\eta(D') = \{v \mid \eta(v) \in D'\}$$

Extraction Functions

An extraction function $\eta : C \mapsto D$ is a mapping from a set of values to their descriptions in D.
It is easy to show that

Proposition

Given an extraction function $\eta : \mathcal{C} \mapsto \mathcal{D}$, the quadruple $(\mathcal{P}(\mathcal{C}), \alpha_\eta, \gamma_\eta, \mathcal{P}(\mathcal{D}))$ is a Galois connection with α_η and γ_η defined by:

$$\alpha_\eta(C') = \{\eta(c) \mid c \in C'\} \text{ and } \gamma_\eta(D') = \{v \mid \eta(v) \in D'\}$$

Vector Space Lifting

Free vector space construction on a set S:

$$\mathcal{V}(S) = \{\sum x_s s \mid x_s \in \mathbb{R}, s \in S\}$$

An obvious way to lift an extraction function to a linear map between vector spaces is to construct the free vector spaces on $\mathcal{C}$ and $\mathcal{D}$ and define:

Vector Space lifting: $\vec{\alpha} : \mathcal{V}(\mathcal{C}) \to \mathcal{V}(\mathcal{D})$

$$\vec{\alpha}(p_1 \cdot \vec{c}_1 + p_2 \cdot \vec{c}_2 + \ldots) = p_i \cdot \eta(c_1) + p_2 \cdot \eta(c_2) \ldots$$

Support Set: **supp** : $\mathcal{V}(\mathcal{C}) \to \mathcal{P}(\mathcal{C})$

$$\textbf{supp}(\vec{x}) = \{c_i \mid \langle c_i, p_i \rangle \in \vec{x} \text{ and } p_i \neq 0\}$$

Vector Space Lifting

Free vector space construction on a set S:

$$\mathcal{V}(S) = \{\sum x_s s \mid x_s \in \mathbb{R}, s \in S\}$$

An obvious way to lift an extraction function to a linear map between vector spaces is to construct the free vector spaces on $\mathcal{C}$ and $\mathcal{D}$ and define:

Vector Space lifting: $\vec{\alpha} : \mathcal{V}(\mathcal{C}) \to \mathcal{V}(\mathcal{D})$

$$\vec{\alpha}(p_1 \cdot \vec{c}_1 + p_2 \cdot \vec{c}_2 + \ldots) = p_i \cdot \eta(c_1) + p_2 \cdot \eta(c_2) \ldots$$

Support Set: **supp** : $\mathcal{V}(\mathcal{C}) \to \mathcal{P}(\mathcal{C})$

$$\textbf{supp}(\vec{x}) = \left\{c_i \mid \langle c_i, p_i \rangle \in \vec{x} \text{ and } p_i \neq 0\right\}$$

Vector Space Lifting

Free vector space construction on a set S:

$$\mathcal{V}(S) = \{\sum x_s s \mid x_s \in \mathbb{R}, s \in S\}$$

An obvious way to lift an extraction function to a linear map between vector spaces is to construct the free vector spaces on $\mathcal{C}$ and $\mathcal{D}$ and define:

Vector Space lifting: $\vec{\alpha} : \mathcal{V}(\mathcal{C}) \rightarrow \mathcal{V}(\mathcal{D})$

$$\vec{\alpha}(p_1 \cdot \vec{c}_1 + p_2 \cdot \vec{c}_2 + \ldots) = p_i \cdot \eta(c_1) + p_2 \cdot \eta(c_2) \ldots$$

Support Set: **supp** : $\mathcal{V}(\mathcal{C}) \rightarrow \mathcal{P}(\mathcal{C})$

$$\textbf{supp}(\vec{x}) = \{c_i \mid \langle c_i, p_i \rangle \in \vec{x} \text{ and } p_i \neq 0\}$$

Vector Space Lifting

Free vector space construction on a set S:

$$\mathcal{V}(S) = \{\sum x_s s \mid x_s \in \mathbb{R}, s \in S\}$$

An obvious way to lift an extraction function to a linear map between vector spaces is to construct the free vector spaces on $\mathcal{C}$ and $\mathcal{D}$ and define:

Vector Space lifting: $\vec{\alpha} : \mathcal{V}(\mathcal{C}) \to \mathcal{V}(\mathcal{D})$

$$\vec{\alpha}(p_1 \cdot \vec{c}_1 + p_2 \cdot \vec{c}_2 + \ldots) = p_i \cdot \eta(c_1) + p_2 \cdot \eta(c_2) \ldots$$

Support Set: $\textbf{supp} : \mathcal{V}(\mathcal{C}) \to \mathcal{P}(\mathcal{C})$

$$\textbf{supp}(\vec{x}) = \left\{ c_i \mid \langle c_i, p_i \rangle \in \vec{x} \text{ and } p_i \neq 0 \right\}$$

Relation with Classical Abstractions

Lemma

Let $\vec{\alpha}$ be a probabilistic abstraction function and let $\vec{\gamma}$ be its Moore-Penrose pseudo-inverse.

Then $\vec{\gamma} \circ \vec{\alpha}$ is extensive with respect to the inclusion on the support sets of vectors in $\mathcal{V}(\mathcal{C})$, i.e. $\forall \vec{x} \in \mathcal{V}(\mathcal{C})$,

$$\mathbf{supp}(\vec{x}) \subseteq \mathbf{supp}(\vec{\gamma} \circ \vec{\alpha}(\vec{x})).$$

Analogously we can show that $\vec{\alpha} \circ \vec{\gamma}$ is reductive. Therefore,

Proposition

$(\vec{\alpha}, \vec{\gamma})$ form a Galois connection wrt the support sets of $\mathcal{V}(\mathcal{C})$ and $\mathcal{V}(\mathcal{D})$, ordered by inclusion.

Relation with Classical Abstractions

Lemma

Let $\vec{\alpha}$ be a probabilistic abstraction function and let $\vec{\gamma}$ be its Moore-Penrose pseudo-inverse.

Then $\vec{\gamma} \circ \vec{\alpha}$ is extensive with respect to the inclusion on the support sets of vectors in $\mathcal{V}(\mathcal{C})$, i.e. $\forall \vec{x} \in \mathcal{V}(\mathcal{C})$,

$$\mathbf{supp}(\vec{x}) \subseteq \mathbf{supp}(\vec{\gamma} \circ \vec{\alpha}(\vec{x})).$$

Analogously we can show that $\vec{\alpha} \circ \vec{\gamma}$ is reductive. Therefore,

Proposition

$(\vec{\alpha}, \vec{\gamma})$ form a Galois connection wrt the support sets of $\mathcal{V}(\mathcal{C})$ and $\mathcal{V}(\mathcal{D})$, ordered by inclusion.

Relation with Classical Abstractions

Lemma

Let $\vec{\alpha}$ be a probabilistic abstraction function and let $\vec{\gamma}$ be its Moore-Penrose pseudo-inverse.

Then $\vec{\gamma} \circ \vec{\alpha}$ is extensive with respect to the inclusion on the support sets of vectors in $\mathcal{V}(\mathcal{C})$, i.e. $\forall \vec{x} \in \mathcal{V}(\mathcal{C})$,

$$\mathbf{supp}(\vec{x}) \subseteq \mathbf{supp}(\vec{\gamma} \circ \vec{\alpha}(\vec{x})).$$

Analogously we can show that $\vec{\alpha} \circ \vec{\gamma}$ is reductive. Therefore,

Proposition

$(\vec{\alpha}, \vec{\gamma})$ form a Galois connection wrt the support sets of $\mathcal{V}(\mathcal{C})$ and $\mathcal{V}(\mathcal{D})$, ordered by inclusion.

Examples of Lifted Abstractions

Parity Abstraction operator on $\mathcal{V}(\{1,\ldots,n\})$ (with n even):

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

Examples of Lifted Abstractions

Parity Abstraction operator on $\mathcal{V}(\{1,\ldots,n\})$ (with n even):

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

Examples of Lifted Abstractions

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

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

Examples of Lifted Abstractions

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

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

Example: Function Approximation (ctd.)

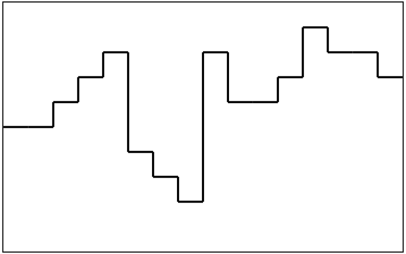
Concrete and abstract domain are step-functions on $[a, b]$.
The set of (real-valued) step-function $\mathcal{T}_n$ is based on the sub-division of the interval into n sub-intervals.

Example: Function Approximation (ctd.)

Concrete and abstract domain are step-functions on $[a, b]$. The set of (real-valued) step-function $\mathcal{T}_n$ is based on the sub-division of the interval into n sub-intervals.

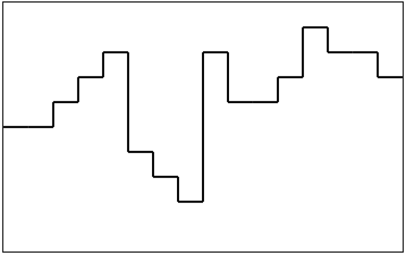
Example: Function Approximation (ctd.)

Concrete and abstract domain are step-functions on $[a, b]$.
The set of (real-valued) step-function $\mathcal{T}_n$ is based on the sub-division of the interval into n sub-intervals.



Example: Function Approximation (ctd.)

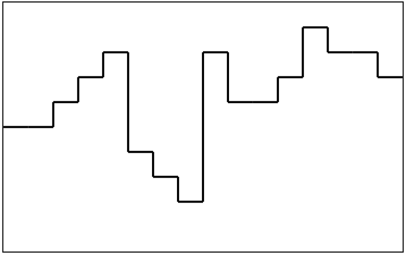
Concrete and abstract domain are step-functions on $[a, b]$. The set of (real-valued) step-function $\mathcal{T}_n$ is based on the sub-division of the interval into n sub-intervals.



Each step function in $\mathcal{T}_n$ corresponds to a vector in $\mathbb{R}^n$, e.g.

Example: Function Approximation (ctd.)

Concrete and abstract domain are step-functions on $[a, b]$. 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 $\mathbb{R}^n$, e.g.

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

Example: Abstraction Matrices

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

Compute the abstractions of f as $f\mathbf{A}_j$.

In a similar way we can also compute the over- and

Example: Abstraction Matrices

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

Compute the abstractions of f as $f\mathbf{A}_j$.

In a similar way we can also compute the over- and under-approximation of f in $\mathcal{T}_i$ based on the pointwise ordering and its reverse.

Example: Abstraction Matrices

Compute the abstractions of f as $f\mathbf{A}_j$.

In a similar way we can also compute the over- and under-approximation of f in $\mathcal{T}_i$ based on the pointwise ordering and its reverse.

Approximation Estimates

Compute the *least square error* as

$$\|f - f\mathbf{AG}\|.$$

$$\begin{aligned}
\|f - f\mathbf{A}_8\mathbf{G}_8\| &= 3.5355 \\
\|f - f\mathbf{A}_4\mathbf{G}_4\| &= 5.3151 \\
\|f - f\mathbf{A}_2\mathbf{G}_2\| &= 5.9896 \\
\|f - f\mathbf{A}_1\mathbf{G}_1\| &= 7.6444
\end{aligned}$$

Approximation Estimates

Compute the *least square error* as

$$\|f - f\mathbf{AG}\|.$$

$$\begin{aligned}
\|f - f\mathbf{A}_8\mathbf{G}_8\| &= 3.5355 \\
\|f - f\mathbf{A}_4\mathbf{G}_4\| &= 5.3151 \\
\|f - f\mathbf{A}_2\mathbf{G}_2\| &= 5.9896 \\
\|f - f\mathbf{A}_1\mathbf{G}_1\| &= 7.6444
\end{aligned}$$

Concrete Semantics (LOS)

$$\mathbf{T}(P) = \sum_{\langle i,p_{ij},j\rangle \in flow(P)} p_{ij} \cdot \mathbf{T}(\ell_i, \ell_j),$$

where

$$\mathbf{T}(\ell_i, \ell_j) = \mathbf{N} \otimes \mathbf{E}(\ell_i, \ell_j),$$

with **N** an operator representing a state update while the second factor realises the transfer of control from label ℓ_i to label ℓ_j.

Concrete Semantics (LOS)

$$\mathbf{T}(P) = \sum_{\langle i, p_{ij}, j\rangle \in \mathit{flow}(P)} p_{ij} \cdot \mathbf{T}(\ell_i, \ell_j),$$

where

$$\mathbf{T}(\ell_i, \ell_j) = \mathbf{N} \otimes \mathbf{E}(\ell_i, \ell_j),$$

with **N** an operator representing a state update while the second factor realises the transfer of control from label ℓ_i to label ℓ_j.

Concrete Semantics (LOS)

$$\mathbf{T}(P) = \sum_{\langle i, p_{ij}, j\rangle \in \mathit{flow}(P)} p_{ij} \cdot \mathbf{T}(\ell_i, \ell_j),$$

where

$$\mathbf{T}(\ell_i, \ell_j) = \mathbf{N} \otimes \mathbf{E}(\ell_i, \ell_j),$$

with **N** an operator representing a state update while the second factor realises the transfer of control from label ℓ_i to label ℓ_j.

Abstract Semantics

Moore-Penrose Pseudo-Inverse of a Tensor Product is:

$$(\mathbf{A}_1 \otimes \mathbf{A}_2 \otimes \ldots \otimes \mathbf{A}_n)^\dagger = \mathbf{A}_1^\dagger \otimes \mathbf{A}_2^\dagger \otimes \ldots \otimes \mathbf{A}_n^\dagger$$

Via linearity we can construct $\mathbf{T}^\#$ in the same way as $\mathbf{T}$, i.e

$$\mathbf{T}^\#(P) = \sum_{\langle i, p_{ij}, j\rangle \in \mathcal{F}(P)} p_{ij} \cdot \mathbf{T}^\#(\ell_i, \ell_j)$$

with local abstraction of individual variables:

$$\mathbf{T}^\#(\ell_i, \ell_j) = (\mathbf{A}_1^\dagger \mathbf{N}_{i1} \mathbf{A}_1) \otimes (\mathbf{A}_2^\dagger \mathbf{N}_{i2} \mathbf{A}_2) \otimes \ldots \otimes (\mathbf{A}_v^\dagger \mathbf{N}_{iv} \mathbf{A}_v) \otimes \mathbf{M}_{ij}$$

Abstract Semantics

Moore-Penrose Pseudo-Inverse of a Tensor Product is:

$$(\mathbf{A}_1 \otimes \mathbf{A}_2 \otimes \ldots \otimes \mathbf{A}_n)^\dagger = \mathbf{A}_1^\dagger \otimes \mathbf{A}_2^\dagger \otimes \ldots \otimes \mathbf{A}_n^\dagger$$

Via linearity we can construct $\mathbf{T}^\#$ in the same way as $\mathbf{T}$, i.e

$$\mathbf{T}^\#(P) = \sum_{\langle i, p_{ij}, j\rangle \in \mathcal{F}(P)} p_{ij} \cdot \mathbf{T}^\#(\ell_i, \ell_j)$$

with local abstraction of individual variables:

$$\mathbf{T}^\#(\ell_i, \ell_j) = (\mathbf{A}_1^\dagger \mathbf{N}_{i1} \mathbf{A}_1) \otimes (\mathbf{A}_2^\dagger \mathbf{N}_{i2} \mathbf{A}_2) \otimes \ldots \otimes (\mathbf{A}_v^\dagger \mathbf{N}_{iv} \mathbf{A}_v) \otimes \mathbf{M}_{ij}$$

Argument

$$
\begin{aligned}
\mathbf{T}^{\#} &= \mathbf{A}^{\dagger}\mathbf{T}\mathbf{A} \\
&= \mathbf{A}^{\dagger}\Big(\sum_{i,j}\mathbf{T}(i,j)\Big)\mathbf{A} \\
&= \sum_{i,j}\mathbf{A}^{\dagger}\mathbf{T}(i,j)\mathbf{A} \\
&= \sum_{i,j}\Big(\bigotimes_{k}\mathbf{A}_k\Big)^{\dagger}\mathbf{T}(i,j)\Big(\bigotimes_{k}\mathbf{A}_k\Big) \\
&= \sum_{i,j}\Big(\bigotimes_{k}\mathbf{A}_k\Big)^{\dagger}\Big(\bigotimes_{k}\mathbf{N}_{ik}\Big)\Big(\bigotimes_{k}\mathbf{A}_k\Big) \\
&= \sum_{i,j}\bigotimes_{k}\big(\mathbf{A}_k^{\dagger}\mathbf{N}_{ik}\mathbf{A}_k\big)
\end{aligned}
$$

Argument

$$\begin{aligned}
\mathbf{T}^{\#} &= \mathbf{A}^{\dagger}\mathbf{T}\mathbf{A} \\
&= \mathbf{A}^{\dagger}(\sum_{i,j} \mathbf{T}(i,j))\mathbf{A} \\
&= \sum_{i,j} \mathbf{A}^{\dagger}\mathbf{T}(i,j)\mathbf{A} \\
&= \sum_{i,j} (\bigotimes_k \mathbf{A}_k)^{\dagger}\mathbf{T}(i,j)(\bigotimes_k \mathbf{A}_k) \\
&= \sum_{i,j} (\bigotimes_k \mathbf{A}_k)^{\dagger}(\bigotimes_k \mathbf{N}_{ik})(\bigotimes_k \mathbf{A}_k) \\
&= \sum_{i,j} \bigotimes_k (\mathbf{A}_k^{\dagger}\mathbf{N}_{ik}\mathbf{A}_k)
\end{aligned}$$

Argument

$$
\begin{aligned}
\mathbf{T}^{\#} &= \mathbf{A}^{\dagger}\mathbf{T}\mathbf{A} \\
&= \mathbf{A}^{\dagger}(\sum_{i,j}\mathbf{T}(i,j))\mathbf{A} \\
&= \sum_{i,j}\mathbf{A}^{\dagger}\mathbf{T}(i,j)\mathbf{A} \\
&= \sum_{i,j}(\bigotimes_{k}\mathbf{A}_k)^{\dagger}\mathbf{T}(i,j)(\bigotimes_{k}\mathbf{A}_k) \\
&= \sum_{i,j}(\bigotimes_{k}\mathbf{A}_k)^{\dagger}(\bigotimes_{k}\mathbf{N}_{ik})(\bigotimes_{k}\mathbf{A}_k) \\
&= \sum_{i,j}\bigotimes_{k}(\mathbf{A}_k^{\dagger}\mathbf{N}_{ik}\mathbf{A}_k)
\end{aligned}
$$

Argument

$$
\begin{aligned}
\mathbf{T}^{\#} &= \mathbf{A}^{\dagger}\mathbf{T}\mathbf{A} \\
&= \mathbf{A}^{\dagger}(\sum_{i,j}\mathbf{T}(i,j))\mathbf{A} \\
&= \sum_{i,j}\mathbf{A}^{\dagger}\mathbf{T}(i,j)\mathbf{A} \\
&= \sum_{i,j}(\bigotimes_{k}\mathbf{A}_k)^{\dagger}\mathbf{T}(i,j)(\bigotimes_{k}\mathbf{A}_k) \\
&= \sum_{i,j}(\bigotimes_{k}\mathbf{A}_k)^{\dagger}(\bigotimes_{k}\mathbf{N}_{ik})(\bigotimes_{k}\mathbf{A}_k) \\
&= \sum_{i,j}\bigotimes_{k}(\mathbf{A}_k^{\dagger}\mathbf{N}_{ik}\mathbf{A}_k)
\end{aligned}
$$

Argument

$$
\begin{aligned}
\mathbf{T}^{\#} &= \mathbf{A}^{\dagger}\mathbf{T}\mathbf{A} \\
&= \mathbf{A}^{\dagger}(\sum_{i,j} \mathbf{T}(i,j))\mathbf{A} \\
&= \sum_{i,j} \mathbf{A}^{\dagger}\mathbf{T}(i,j)\mathbf{A} \\
&= \sum_{i,j} (\bigotimes_k \mathbf{A}_k)^{\dagger}\mathbf{T}(i,j)(\bigotimes_k \mathbf{A}_k) \\
&= \sum_{i,j} (\bigotimes_k \mathbf{A}_k)^{\dagger}(\bigotimes_k \mathbf{N}_{ik})(\bigotimes_k \mathbf{A}_k) \\
&= \sum_{i,j} \bigotimes_k (\mathbf{A}_k^{\dagger}\mathbf{N}_{ik}\mathbf{A}_k)
\end{aligned}
$$

Argument

$$
\begin{aligned}
\mathbf{T}^{\#} &= \mathbf{A}^{\dagger}\mathbf{T}\mathbf{A} \\
&= \mathbf{A}^{\dagger}\Big(\sum_{i,j}\mathbf{T}(i,j)\Big)\mathbf{A} \\
&= \sum_{i,j}\mathbf{A}^{\dagger}\mathbf{T}(i,j)\mathbf{A} \\
&= \sum_{i,j}\Big(\bigotimes_{k}\mathbf{A}_k\Big)^{\dagger}\mathbf{T}(i,j)\Big(\bigotimes_{k}\mathbf{A}_k\Big) \\
&= \sum_{i,j}\Big(\bigotimes_{k}\mathbf{A}_k\Big)^{\dagger}\Big(\bigotimes_{k}\mathbf{N}_{ik}\Big)\Big(\bigotimes_{k}\mathbf{A}_k\Big) \\
&= \sum_{i,j}\bigotimes_{k}\Big(\mathbf{A}_k^{\dagger}\mathbf{N}_{ik}\mathbf{A}_k\Big)
\end{aligned}
$$

Parity Analysis

Determine at each program point whether a variable is *even* or *odd*.

Parity Abstraction operator on $\mathcal{V}(\{0, \ldots, n\})$ (with n even):

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

Parity Analysis

Determine at each program point whether a variable is *even* or *odd*.

Parity Abstraction operator on $\mathcal{V}(\{0, \ldots, n\})$ (with n even):

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

Example

$[m \leftarrow i]^1$;
while $[n > 1]^2$ **do**
    $[m \leftarrow m \times n]^3$;
    $[n \leftarrow n - 1]^4$
od
$[\textbf{stop}]^5$

Example

1: $[m \leftarrow i]^1$;
2: **while** $[n > 1]^2$ **do**
3: $\quad [m \leftarrow m \times n]^3$;
4: $\quad [n \leftarrow n - 1]^4$
5: **od**
6: $[\textbf{stop}]^5$

$$
\begin{aligned}
\mathbf{T} \;=\;& \mathbf{U}(\mathtt{m} \leftarrow i) \otimes \mathbf{E}(1,2) \\
+\;& \mathbf{P}(n > 1) \otimes \mathbf{E}(2,3) \\
+\;& \mathbf{P}(n \leq 1) \otimes \mathbf{E}(2,5) \\
+\;& \mathbf{U}(\mathtt{m} \leftarrow m \times n) \otimes \mathbf{E}(3,4) \\
+\;& \mathbf{U}(\mathtt{n} \leftarrow n - 1) \otimes \mathbf{E}(4,2) \\
+\;& \mathbf{I} \otimes \mathbf{E}(5,5)
\end{aligned}
$$

Example

1: $[m \leftarrow i]^1$;
2: **while** $[n > 1]^2$ **do**
3: $\quad [m \leftarrow m \times n]^3$;
4: $\quad [n \leftarrow n - 1]^4$
5: **od**
6: $[\textbf{stop}]^5$

$$
\begin{aligned}
\mathbf{T}^\# \;=\;& \mathbf{U}^\#(\mathtt{m} \leftarrow i) \otimes \mathbf{E}(1,2) \\
+\;& \mathbf{P}^\#(n > 1) \otimes \mathbf{E}(2,3) \\
+\;& \mathbf{P}^\#(n \leq 1) \otimes \mathbf{E}(2,5) \\
+\;& \mathbf{U}^\#(\mathtt{m} \leftarrow m \times n) \otimes \mathbf{E}(3,4) \\
+\;& \mathbf{U}^\#(\mathtt{n} \leftarrow n - 1) \otimes \mathbf{E}(4,2) \\
+\;& \mathbf{I}^\# \otimes \mathbf{E}(5,5)
\end{aligned}
$$

Abstract Semantics

Abstraction: $\mathbf{A} = \mathbf{A}_p \otimes \mathbf{I}$, i.e. m abstract (parity) but n concrete.

$$
\begin{aligned}
\mathbf{T}^{\#} &= \mathbf{U}^{\#}(m \leftarrow 1) \otimes \mathbf{E}(1,2) \\
&+ \mathbf{P}^{\#}(n > 1) \otimes \mathbf{E}(2,3) \\
&+ \mathbf{P}^{\#}(n \leq 1) \otimes \mathbf{E}(2,5) \\
&+ \mathbf{U}^{\#}(m \leftarrow m \times n) \otimes \mathbf{E}(3,4) \\
&+ \mathbf{U}^{\#}(n \leftarrow n - 1) \otimes \mathbf{E}(4,2) \\
&+ \mathbf{I}^{\#} \otimes \mathbf{E}(5,5)
\end{aligned}
$$

Abstract Semantics

$$\mathbf{U}^{\#}(m \leftarrow 1) =$$

$$= \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & & \dots & 1 \end{pmatrix}$$

Abstract Semantics

$$\mathbf{U}^{\#}(n \leftarrow n-1) =$$

$$= \quad \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 & 0 & 0 & \dots & 0 \\ 1 & 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 0 \end{pmatrix}$$

Abstract Semantics

$$\mathbf{P}^{\#}(n > 1) =$$

$$= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 \end{pmatrix}$$

Abstract Semantics

$$\mathbf{P}^{\#}(n \leq 1) =$$

$$= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 0 \end{pmatrix}$$

Abstract Semantics

$$\mathbf{U}^{\#}(m \leftarrow m \times n) = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 \end{pmatrix} +$$

$$+ \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \dots & \ddots \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \dots & \ddots \end{pmatrix}$$

Implementation

Implementation of concrete and abstract semantics of Factorial using **octave**. Ranges: $n \in \{1, \ldots, d\}$ and $m \in \{1, \ldots, d!\}$.

d	$\dim(\mathbf{T}(F))$	$\dim(\mathbf{T}^{\#}(F))$
2	45	30
3	140	40
4	625	50
5	3630	60
6	25235	70
7	201640	80
8	1814445	90
9	18144050	100

Using uniform initial distributions $\mathbf{d_0}$ for n and m.

Implementation

Implementation of concrete and abstract semantics of Factorial using **octave**. Ranges: $n \in \{1, \ldots, d\}$ and $m \in \{1, \ldots, d!\}$.

d	$\dim(\mathbf{T}(F))$	$\dim(\mathbf{T}^{\#}(F))$
2	45	30
3	140	40
4	625	50
5	3630	60
6	25235	70
7	201640	80
8	1814445	90
9	18144050	100

Using uniform initial distributions $\mathbf{d_0}$ for n and m.

Scalablity

The abstract probabilities for *m* being **even** or **odd** when we execute the abstract program for various *d* values are:

d	**even**	**odd**
10	0.81818	0.18182
100	0.98019	0.019802
1000	0.99800	0.0019980
10000	0.99980	0.00019998

Live Variable Analysis

$[\textbf{skip}]^1$
if $[odd(y)]^2$ **then**
    $[x \leftarrow x + 1]^3$
else
    $[y \leftarrow y + 1]^4$
fi
$[y \leftarrow y + 1]^5$

Classical Analysis: $\text{LV}_{entry}(2) = \{x, y\}$

Probabilistic Analysis:

Live Variable Analysis

[**skip**]^1
if $[odd(y)]^2$ **then**
  $[x \leftarrow x + 1]^3$
else
  $[y \leftarrow y + 1]^4$
fi
$[y \leftarrow y + 1]^5$

Classical Analysis: $\mathrm{LV}_{entry}(2) = \{x, y\}$

Probabilistic Analysis:

Live Variable Analysis

$[\textbf{skip}]^1$
$\textbf{if } [odd(y)]^2 \textbf{ then}$
    $[x \leftarrow x + 1]^3$
$\textbf{else}$
    $[y \leftarrow y + 1]^4$
$\textbf{fi}$
$[y \leftarrow y + 1]^5$

Classical Analysis: $\mathrm{LV}_{entry}(2) = \{x, y\}$

Probabilistic Analysis: $\mathrm{LV}_{entry}(2) = \{\langle x, \frac{1}{2}\rangle, \langle y, 1\rangle\}$

Live Variable Analysis

$[y \leftarrow 2 \times x]^1$
if $[odd(y)]^2$ **then**
    $[x \leftarrow x + 1]^3$
else
    $[y \leftarrow y + 1]^4$
fi
$[y \leftarrow y + 1]^5$

Classical Analysis: $\mathrm{LV}_{entry}(2) = \{x, y\}$

Probabilistic Analysis: $\mathrm{LV}_{entry}(2) =$

Live Variable Analysis

$$
\begin{array}{l}
[y \leftarrow 2 \times x]^1 \\
\textbf{if } [odd(y)]^2 \textbf{ then} \\
\qquad [x \leftarrow x + 1]^3 \\
\textbf{else} \\
\qquad [y \leftarrow y + 1]^4 \\
\textbf{fi} \\
[y \leftarrow y + 1]^5
\end{array}
$$

Classical Analysis: $\mathrm{LV}_{entry}(2) = \{x, y\}$

Probabilistic Analysis: $\mathrm{LV}_{entry}(2) = \{\langle y, 1\rangle\}$

Program “Transformation”

$[y \leftarrow 2 \times x]^1$
if $[odd(y)]^2$ **then**
    $[x \leftarrow x + 1]^3$
else
    $[y \leftarrow y + 1]^4$
fi
$[y \leftarrow y + 1]^5$

$[y \leftarrow 2 \times x]^1$
[**choose**]^2
    $p_\top : [x \leftarrow x + 1]^3$
or
    $p_\bot : [y \leftarrow y + 1]^4$
$[y \leftarrow y + 1]^5$

Determine branching probabilities in a first-phase analysis and utilise this information to perform the actual analysis:

$$p^\top = \mathbf{A}^\dagger \cdot \mathbf{P}(b = \mathbf{true}) \cdot \mathbf{A} \text{ and } p^\bot = \mathbf{A}^\dagger \cdot \mathbf{P}(b = \mathbf{false}) \cdot \mathbf{A}$$

Program “Transformation”

$[y \leftarrow 2 \times x]^1$
if $[odd(y)]^2$ **then**
  $[x \leftarrow x + 1]^3$
else
  $[y \leftarrow y + 1]^4$
fi
$[y \leftarrow y + 1]^5$

$[y \leftarrow 2 \times x]^1$
$[\textbf{choose}]^2$
  $p_\top : [x \leftarrow x + 1]^3$
or
  $p_\perp : [y \leftarrow y + 1]^4$
$[y \leftarrow y + 1]^5$

Determine branching probabilities in a first-phase analysis and utilise this information to perform the actual analysis:

$$p^\top = \mathbf{A}^\dagger \cdot \mathbf{P}(b = \textbf{true}) \cdot \mathbf{A} \text{ and } p^\perp = \mathbf{A}^\dagger \cdot \mathbf{P}(b = \textbf{false}) \cdot \mathbf{A}$$

Syntax of pWhile with Pointers

$$
\begin{array}{rcl}
S & ::= & [\textbf{skip}]^{\ell} \\
& | & [\textbf{stop}]^{\ell} \\
& | & [p \leftarrow e]^{\ell} \\
& | & S_1;\ S_2 \\
& | & [\textbf{choose}]^{\ell}\ p_1 : S_1\ \textbf{or}\ p_2 : S_2 \\
& | & \textbf{if}\ [b]^{\ell}\ \textbf{then}\ S_1\ \textbf{else}\ S_2 \\
& | & \textbf{while}\ [b]^{\ell}\ \textbf{do}\ S
\end{array}
$$

$$p ::= *^{r}\mathrm{x} \text{ with } \mathrm{x} \in \textbf{Var} \qquad e ::= a \mid b \mid l$$

$$a ::= n \mid p \mid a_1 \odot a_2 \qquad l ::= \mathrm{NIL} \mid p \mid \&p$$

$$b ::= \textbf{true} \mid \textbf{false} \mid p \mid \neg b \mid b_1 \between b_2 \mid a_1 \lessgtr a_2$$

Syntax of pWhile with Pointers

$$
\begin{array}{rcl}
S & ::= & [\textbf{skip}]^\ell \\
& \mid & [\textbf{stop}]^\ell \\
& \mid & [p \leftarrow e]^\ell \\
& \mid & S_1;\ S_2 \\
& \mid & [\textbf{choose}]^\ell\ p_1 : S_1\ \textbf{or}\ p_2 : S_2 \\
& \mid & \textbf{if}\ [b]^\ell\ \textbf{then}\ S_1\ \textbf{else}\ S_2 \\
& \mid & \textbf{while}\ [b]^\ell\ \textbf{do}\ S
\end{array}
$$

$$
\begin{array}{rclcrcl}
p & ::= & *^r \mathrm{x} \text{ with } \mathrm{x} \in \textbf{Var} & & e & ::= & a \mid b \mid l \\
a & ::= & n \mid p \mid a_1 \odot a_2 & & l & ::= & \mathrm{NIL} \mid p \mid \&p
\end{array}
$$

$$
b \ ::= \ \textbf{true} \mid \textbf{false} \mid p \mid \neg b \mid b_1 \between b_2 \mid a_1 \lessgtr a_2
$$

Syntax of pWhile with Pointers

$$
\begin{array}{rcl}
S & ::= & [\mathbf{skip}]^{\ell} \\
 & | & [\mathbf{stop}]^{\ell} \\
 & | & [p \leftarrow e]^{\ell} \\
 & | & S_1;\ S_2 \\
 & | & [\mathbf{choose}]^{\ell}\ p_1 : S_1\ \mathbf{or}\ p_2 : S_2 \\
 & | & \mathbf{if}\ [b]^{\ell}\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2 \\
 & | & \mathbf{while}\ [b]^{\ell}\ \mathbf{do}\ S
\end{array}
$$

$$p ::= *^{r}\mathrm{x} \text{ with } \mathrm{x} \in \mathbf{Var} \qquad e ::= a \mid b \mid l$$

$$a ::= n \mid p \mid a_1 \odot a_2 \qquad l ::= \mathrm{NIL} \mid p \mid \&p$$

$$b ::= \mathbf{true} \mid \mathbf{false} \mid p \mid \neg b \mid b_1 \between b_2 \mid a_1 \lessgtr a_2$$

Syntax of pWhile with Pointers

$$
\begin{array}{rcl}
S & ::= & [\mathbf{skip}]^{\ell} \\
 & | & [\mathbf{stop}]^{\ell} \\
 & | & [p \leftarrow e]^{\ell} \\
 & | & S_1;\ S_2 \\
 & | & [\mathbf{choose}]^{\ell}\ p_1 : S_1\ \mathbf{or}\ p_2 : S_2 \\
 & | & \mathbf{if}\ [b]^{\ell}\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2 \\
 & | & \mathbf{while}\ [b]^{\ell}\ \mathbf{do}\ S
\end{array}
$$

$$
p ::= *^{r}\mathrm{x} \text{ with } \mathrm{x} \in \mathbf{Var} \qquad e ::= a \mid b \mid l
$$

$$
a ::= n \mid p \mid a_1 \odot a_2 \qquad l ::= \mathrm{NIL} \mid p \mid \&p
$$

$$
b ::= \mathbf{true} \mid \mathbf{false} \mid p \mid \neg b \mid b_1 \between b_2 \mid a_1 \lessgtr a_2
$$

Syntax of pWhile with Pointers

$$
\begin{array}{lcl}
S & ::= & [\textbf{skip}]^{\ell} \\
 & | & [\textbf{stop}]^{\ell} \\
 & | & [p \leftarrow e]^{\ell} \\
 & | & S_1;\ S_2 \\
 & | & [\textbf{choose}]^{\ell}\ p_1 : S_1\ \textbf{or}\ p_2 : S_2 \\
 & | & \textbf{if}\ [b]^{\ell}\ \textbf{then}\ S_1\ \textbf{else}\ S_2 \\
 & | & \textbf{while}\ [b]^{\ell}\ \textbf{do}\ S
\end{array}
$$

$$
\begin{array}{lcl}
p & ::= & *^{r}\mathrm{x} \text{ with } \mathrm{x} \in \textbf{Var} \qquad e ::= a \mid b \mid l \\
a & ::= & n \mid p \mid a_1 \odot a_2 \qquad l ::= \mathrm{NIL} \mid p \mid \&p \\
b & ::= & \textbf{true} \mid \textbf{false} \mid p \mid \neg b \mid b_1 \between b_2 \mid a_1 \lessgtr a_2
\end{array}
$$

Syntax of pWhile with Pointers

$$
\begin{array}{rcl}
S & ::= & [\mathbf{skip}]^{\ell} \\
& \mid & [\mathbf{stop}]^{\ell} \\
& \mid & [p \leftarrow e]^{\ell} \\
& \mid & S_1;\ S_2 \\
& \mid & [\mathbf{choose}]^{\ell}\ p_1 : S_1\ \mathbf{or}\ p_2 : S_2 \\
& \mid & \mathbf{if}\ [b]^{\ell}\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2 \\
& \mid & \mathbf{while}\ [b]^{\ell}\ \mathbf{do}\ S
\end{array}
$$

$$
\begin{array}{rclcrcl}
p & ::= & *^{r}\mathrm{x} \text{ with } \mathrm{x} \in \mathbf{Var} & \quad & e & ::= & a \mid b \mid l \\
a & ::= & n \mid p \mid a_1 \odot a_2 & & l & ::= & \mathrm{NIL} \mid p \mid \&p
\end{array}
$$

$$
b \ ::= \ \mathbf{true} \mid \mathbf{false} \mid p \mid \neg b \mid b_1 \between b_2 \mid a_1 \lessgtr a_2
$$

Syntax of pWhile with Pointers

$$
\begin{array}{lcl}
S & ::= & [\mathbf{skip}]^{\ell} \\
& | & [\mathbf{stop}]^{\ell} \\
& | & [p \leftarrow e]^{\ell} \\
& | & S_1;\ S_2 \\
& | & [\mathbf{choose}]^{\ell}\ p_1 : S_1 \ \mathbf{or}\ p_2 : S_2 \\
& | & \mathbf{if}\ [b]^{\ell}\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2 \\
& | & \mathbf{while}\ [b]^{\ell}\ \mathbf{do}\ S
\end{array}
$$

$$
\begin{array}{lcl}
p & ::= & *^{r}\mathrm{x} \text{ with } \mathrm{x} \in \mathbf{Var} \\
a & ::= & n \mid p \mid a_1 \odot a_2 \\
b & ::= & \mathbf{true} \mid \mathbf{false} \mid p \mid \neg b \mid b_1 \between b_2 \mid a_1 \lessgtr a_2
\end{array}
\qquad
\begin{array}{lcl}
e & ::= & a \mid b \mid l \\
l & ::= & \mathrm{NIL} \mid p \mid \&p
\end{array}
$$

Syntax of pWhile with Pointers

$$
\begin{array}{rcl}
S & ::= & [\textbf{skip}]^{\ell} \\
 & | & [\textbf{stop}]^{\ell} \\
 & | & [p \leftarrow e]^{\ell} \\
 & | & S_1;\ S_2 \\
 & | & [\textbf{choose}]^{\ell}\ p_1 : S_1\ \textbf{or}\ p_2 : S_2 \\
 & | & \textbf{if}\ [b]^{\ell}\ \textbf{then}\ S_1\ \textbf{else}\ S_2 \\
 & | & \textbf{while}\ [b]^{\ell}\ \textbf{do}\ S
\end{array}
$$

$p ::= *^r\mathtt{x}$ with $\mathtt{x} \in \textbf{Var}$ $\quad e ::= a \mid b \mid l$

$a ::= n \mid p \mid a_1 \odot a_2 \quad l ::= \text{NIL} \mid p \mid \&p$

$b ::= \textbf{true} \mid \textbf{false} \mid p \mid \neg b \mid b_1 \between b_2 \mid a_1 \lessgtr a_2$

Syntax of pWhile with Pointers

$$
\begin{array}{rcl}
S & ::= & [\mathbf{skip}]^{\ell} \\
& | & [\mathbf{stop}]^{\ell} \\
& | & [p \leftarrow e]^{\ell} \\
& | & S_1;\ S_2 \\
& | & [\mathbf{choose}]^{\ell}\ p_1 : S_1\ \mathbf{or}\ p_2 : S_2 \\
& | & \mathbf{if}\ [b]^{\ell}\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2 \\
& | & \mathbf{while}\ [b]^{\ell}\ \mathbf{do}\ S
\end{array}
$$

$$
p ::= *^{r}\mathrm{x} \text{ with } \mathrm{x} \in \mathbf{Var} \qquad e ::= a \mid b \mid l
$$

$$
a ::= n \mid p \mid a_1 \odot a_2 \qquad l ::= \mathrm{NIL} \mid p \mid \&p
$$

$$
b ::= \mathbf{true} \mid \mathbf{false} \mid p \mid \neg b \mid b_1 \between b_2 \mid a_1 \lessgtr a_2
$$

Syntax of pWhile with Pointers

$$
\begin{array}{rcl}
S & ::= & [\textbf{skip}]^\ell \\
& | & [\textbf{stop}]^\ell \\
& | & [p \leftarrow e]^\ell \\
& | & S_1;\ S_2 \\
& | & [\textbf{choose}]^\ell\ p_1 : S_1\ \textbf{or}\ p_2 : S_2 \\
& | & \textbf{if}\ [b]^\ell\ \textbf{then}\ S_1\ \textbf{else}\ S_2 \\
& | & \textbf{while}\ [b]^\ell\ \textbf{do}\ S
\end{array}
$$

$$p \quad ::= \quad *^r\mathtt{x} \text{ with } \mathtt{x} \in \textbf{Var} \qquad e \quad ::= \quad a \mid b \mid l$$

$$a \quad ::= \quad n \mid p \mid a_1 \odot a_2 \qquad l \quad ::= \quad \mathrm{NIL} \mid p \mid \&p$$

$$b \quad ::= \quad \textbf{true} \mid \textbf{false} \mid p \mid \neg b \mid b_1 \between b_2 \mid a_1 \lessgtr a_2$$

Syntax of pWhile with Pointers

$$
\begin{array}{rcl}
S & ::= & [\mathbf{skip}]^\ell \\
& | & [\mathbf{stop}]^\ell \\
& | & [p \leftarrow e]^\ell \\
& | & S_1;\ S_2 \\
& | & [\mathbf{choose}]^\ell\ p_1 : S_1\ \mathbf{or}\ p_2 : S_2 \\
& | & \mathbf{if}\ [b]^\ell\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2 \\
& | & \mathbf{while}\ [b]^\ell\ \mathbf{do}\ S
\end{array}
$$

$$
p \ ::= \ *^r \mathrm{x} \text{ with } \mathrm{x} \in \mathbf{Var} \qquad e \ ::= \ a \mid b \mid l
$$

$$
a \ ::= \ n \mid p \mid a_1 \odot a_2 \qquad l \ ::= \ \mathrm{NIL} \mid p \mid \&p
$$

$$
b \ ::= \ \mathbf{true} \mid \mathbf{false} \mid p \mid \neg b \mid b_1 \between b_2 \mid a_1 \lessgtr a_2
$$

Syntax of pWhile with Pointers

$$
\begin{array}{rcl}
S & ::= & [\textbf{skip}]^\ell \\
& | & [\textbf{stop}]^\ell \\
& | & [p \leftarrow e]^\ell \\
& | & S_1;\ S_2 \\
& | & [\textbf{choose}]^\ell\ p_1 : S_1\ \textbf{or}\ p_2 : S_2 \\
& | & \textbf{if}\ [b]^\ell\ \textbf{then}\ S_1\ \textbf{else}\ S_2 \\
& | & \textbf{while}\ [b]^\ell\ \textbf{do}\ S
\end{array}
$$

$$p ::= *^r\mathtt{x} \text{ with } \mathtt{x} \in \textbf{Var} \qquad e ::= a \mid b \mid l$$

$$a ::= n \mid p \mid a_1 \odot a_2 \qquad l ::= \mathrm{NIL} \mid p \mid \&p$$

$$b ::= \textbf{true} \mid \textbf{false} \mid p \mid \neg b \mid b_1 \between b_2 \mid a_1 \lessgtr a_2$$

Example

if $[(z_0 \text{ mod } 2 = 0)]^1$ **then**
$[x \leftarrow \&z_1]^2;\ [y \leftarrow \&z_2]^3$
else
$[x \leftarrow \&z_2]^4;\ [y \leftarrow \&z_1]^5$
fi
$[\textbf{stop}]^6$

$[\textbf{choose}]^1$
$\frac{1}{2} : ([x \leftarrow \&z_1]^2;\ [y \leftarrow \&z_2]^3)$
or
$\frac{1}{2} : ([x \leftarrow \&z_2]^4;\ [y \leftarrow \&z_1]^5)$
$[\textbf{stop}]^6$

Example

if $[(\mathtt{z}_0 \bmod 2 = 0)]^1$ **then**
$[\mathtt{x} \leftarrow \& \mathtt{z}_1]^2; \; [\mathtt{y} \leftarrow \& \mathtt{z}_2]^3$
else
$[\mathtt{x} \leftarrow \& \mathtt{z}_2]^4; \; [\mathtt{y} \leftarrow \& \mathtt{z}_1]^5$
fi
$[\textbf{stop}]^6$

$[\textbf{choose}]^1$
$\frac{1}{2} : ([\mathtt{x} \leftarrow \& \mathtt{z}_1]^2; \; [\mathtt{y} \leftarrow \& \mathtt{z}_2]^3)$
or
$\frac{1}{2} : ([\mathtt{x} \leftarrow \& \mathtt{z}_2]^4; \; [\mathtt{y} \leftarrow \& \mathtt{z}_1]^5)$
$[\textbf{stop}]^6$

Test Operators and Filters

Select a certain value $c \in \textbf{Value}$:

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

Test Operators and Filters

Select a certain value $c \in \mathbf{Value}$:

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

$$\mathbf{P}(2) = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Test Operators and Filters

Select a certain value $c \in \textbf{Value}$:

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

Select a certain classical state $\sigma \in \textbf{State}$:

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

Test Operators and Filters

Select a certain value $c \in \textbf{Value}$:

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

Select a certain classical state $\sigma \in \textbf{State}$:

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

$$\mathbf{P}(\sigma(\mathrm{x}_1 \mapsto 2, \mathrm{x}_2 \mapsto 4)) = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Test Operators and Filters

Select a certain value $c \in \textbf{Value}$:

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

Select a certain classical state $\sigma \in \textbf{State}$:

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

Select states where expression $e = a \mid b \mid l$ evaluates to c:

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

Selection via Projections

Filtering out *relevant* configurations, i.e. only those which fulfill a certain condition. Use diagonal matrix **P**:

$$(\mathbf{P})_{ii} = \begin{cases} 1 & \text{if condition holds for } c_i \in \textbf{Value} \\ 0 & \text{otherwise.} \end{cases}$$

$$\begin{pmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \end{pmatrix}^t \cdot \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 \\ d_2 \\ d_3 \\ 0 \\ d_5 \\ 0 \end{pmatrix}^t$$

Selection via Projections

Filtering out *relevant* configurations, i.e. only those which fulfill a certain condition. Use diagonal matrix **P**:

$$(\mathbf{P})_{ii} = \begin{cases} 1 & \text{if condition holds for } c_i \in \textbf{Value} \\ 0 & \text{otherwise.} \end{cases}$$

$$\begin{pmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \end{pmatrix}^t \cdot \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 \\ d_2 \\ d_3 \\ 0 \\ d_5 \\ 0 \end{pmatrix}^t$$

Example

if $[(z_0 \text{ mod } 2 = 0)]^1$ **then**
$[x \leftarrow \& z_1]^2; \; [y \leftarrow \& z_2]^3$
else
$[x \leftarrow \& z_2]^4; \; [y \leftarrow \& z_1]^5$
fi
$[\textbf{stop}]^6$

$\textbf{Var} = \{x, y, z_0, z_1, z_2\}$

$$\mathbf{P}(z_0 \text{ mod } 2 \neq 0) = \mathbf{I} \otimes \mathbf{I} \otimes \begin{pmatrix} 1 & 0 & 0 & 0 & \dots \\ 0 & 0 & 0 & 0 & \dots \\ 0 & 0 & 1 & 0 & \dots \\ 0 & 0 & 0 & 0 & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix} \otimes \mathbf{I} \otimes \mathbf{I}$$

Example

if $[(\mathtt{z}_0 \bmod 2 = 0)]^1$ **then**
$[\mathtt{x} \leftarrow \&\mathtt{z}_1]^2;\ [\mathtt{y} \leftarrow \&\mathtt{z}_2]^3$
else
$[\mathtt{x} \leftarrow \&\mathtt{z}_2]^4;\ [\mathtt{y} \leftarrow \&\mathtt{z}_1]^5$
fi
[**stop**]^6

$$\textbf{Var} = \{\mathtt{x}, \mathtt{y}, \mathtt{z}_0, \mathtt{z}_1, \mathtt{z}_2\}$$

$$\mathbf{P}(\mathtt{z}_0 \bmod 2 = 0) = \mathbf{I} \otimes \mathbf{I} \otimes \begin{pmatrix} 0 & 0 & 0 & 0 & \dots \\ 0 & 1 & 0 & 0 & \dots \\ 0 & 0 & 0 & 0 & \dots \\ 0 & 0 & 0 & 1 & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix} \otimes \mathbf{I} \otimes \mathbf{I}$$

Example

if $[(\mathtt{z}_0 \bmod 2 = 0)]^1$ **then**
$[\mathtt{x} \leftarrow \&\mathtt{z}_1]^2;\ [\mathtt{y} \leftarrow \&\mathtt{z}_2]^3$
else
$[\mathtt{x} \leftarrow \&\mathtt{z}_2]^4;\ [\mathtt{y} \leftarrow \&\mathtt{z}_1]^5$
fi
[**stop**]^6

$$\textbf{Var} = \{\mathtt{x}, \mathtt{y}, \mathtt{z}_0, \mathtt{z}_1, \mathtt{z}_2\}$$

$$\mathbf{P}(\mathtt{z}_0 \bmod 2 \neq 0) = \mathbf{I} \otimes \mathbf{I} \otimes \begin{pmatrix} 1 & 0 & 0 & 0 & \dots \\ 0 & 0 & 0 & 0 & \dots \\ 0 & 0 & 1 & 0 & \dots \\ 0 & 0 & 0 & 0 & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix} \otimes \mathbf{I} \otimes \mathbf{I}$$

Update Operators

For all initial values change to constant $c \in \mathbf{Value}$:

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

Set value of variable $\mathtt{x}_k \in \mathbf{Var}$ to constant $c \in \mathbf{Value}$:

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

Set variable $\mathtt{x}_k \in \mathbf{Var}$ to value given by expression $e = a \mid b \mid l$:

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

Update Operators

For all initial values change to constant $c \in \textbf{Value}$:

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

$$\mathbf{U}(3) = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & \dots \\ 0 & 0 & 1 & 0 & 0 & \dots \\ 0 & 0 & 1 & 0 & 0 & \dots \\ 0 & 0 & 1 & 0 & 0 & \dots \\ 0 & 0 & 1 & 0 & 0 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

Set value of variable $\mathtt{x}_k \in \textbf{Var}$ to constant $c \in \textbf{Value}$:

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

Set variable $\mathtt{x}_k \in \textbf{Var}$ to value given by expression $e = a \mid b \mid l$:

Update Operators

For all initial values change to constant $c \in \textbf{Value}$:

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

Set value of variable $\mathtt{x}_k \in \textbf{Var}$ to constant $c \in \textbf{Value}$:

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

Set variable $\mathtt{x}_k \in \textbf{Var}$ to value given by expression $e = a \mid b \mid l$:

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

Update Operators

For all initial values change to constant $c \in \mathbf{Value}$:

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

Set value of variable $\mathtt{x}_k \in \mathbf{Var}$ to constant $c \in \mathbf{Value}$:

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

Set variable $\mathtt{x}_k \in \mathbf{Var}$ to value given by expression $e = a \mid b \mid l$:

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

Update for Pointers

For an assignment with a pointer on the l.h.s. we need to determine recursevly the actual variable p is pointing to:

$$\mathbf{U}(*^r \mathrm{x}_k \leftarrow e) = \sum_{\mathrm{x}_i} \mathbf{P}(\mathrm{x}_k = \&\mathrm{x}_i)\mathbf{U}(*^{r-1}\mathrm{x}_i \leftarrow e)$$

Update for Pointers

For an assignment with a pointer on the l.h.s. we need to determine recursevly the actual variable p is pointing to:

$$\mathbf{U}(*^r \mathrm{x}_k \leftarrow e) = \sum_{\mathrm{x}_i} \mathbf{P}(\mathrm{x}_k = \& \mathrm{x}_i)\mathbf{U}(*^{r-1} \mathrm{x}_i \leftarrow e)$$

Update for Pointers

For an assignment with a pointer on the l.h.s. we need to determine recursevly the actual variable p is pointing to:

$$\mathbf{U}(*^r \mathtt{x}_k \leftarrow e) = \sum_{\mathtt{x}_i} \mathbf{P}(\mathtt{x}_k = \& \mathtt{x}_i) \mathbf{U}(*^{r-1} \mathtt{x}_i \leftarrow e)$$

Note that we always get eventually to the base case, i.e. where p refers to a concrete variable $\mathtt{x}_k$ and thus the update operator $\mathbf{U}(\mathtt{x}_k \leftarrow e)$ from before.

Update for Pointers

For an assignment with a pointer on the l.h.s. we need to determine recursevly the actual variable p is pointing to:

$$\mathbf{U}(*^r \mathrm{x}_k \leftarrow e) = \sum_{\mathrm{x}_i} \mathbf{P}(\mathrm{x}_k = \&\mathrm{x}_i)\mathbf{U}(*^{r-1}\mathrm{x}_i \leftarrow e)$$

For a pointer of second order with $\mathrm{x}_2 \to \mathrm{x}_1 \to \mathrm{x}_0$ we get:

$$\begin{aligned}
\mathbf{U}(**\mathrm{x}_2 \leftarrow 4) &= \sum_{\mathrm{x}_i} \mathbf{P}(\mathrm{x}_2 = \&\mathrm{x}_i)\mathbf{U}(*\mathrm{x}_i \leftarrow 4) \\
\mathbf{U}(*\mathrm{x}_1 \leftarrow 4) &= \sum_{\mathrm{x}_i} \mathbf{P}(\mathrm{x}_1 = \&\mathrm{x}_i)\mathbf{U}(\mathrm{x}_i \leftarrow 4) \\
\mathbf{U}(\mathrm{x}_0 \leftarrow 4) &
\end{aligned}$$

Example

if $[(z_0 \bmod 2 = 0)]^1$ **then**
$[x \leftarrow \&z_1]^2;\ [y \leftarrow \&z_2]^3$
else
$[x \leftarrow \&z_2]^4;\ [y \leftarrow \&z_1]^5$
fi
$[\textbf{stop}]^6$

$$\mathbf{P}(\mathit{even}(z_0)) \otimes \mathbf{E}(1,2) +$$
$$\mathbf{P}(\mathit{odd}(z_0)) \otimes \mathbf{E}(1,4) +$$
$$\mathbf{U}(x \leftarrow \&z_1) \otimes \mathbf{E}(2,3) +$$
$$\mathbf{U}(y \leftarrow \&z_2) \otimes \mathbf{E}(3,6) +$$
$$\mathbf{U}(x \leftarrow \&z_2) \otimes \mathbf{E}(4,5) +$$
$$\mathbf{U}(y \leftarrow \&z_1) \otimes \mathbf{E}(5,6) +$$
$$\mathbf{I} \otimes \mathbf{E}(6,6)$$

Example

$$[\textbf{choose}]^1 \quad \tfrac{1}{2} : ([\mathtt{x} \leftarrow \&\mathtt{z}_1]^2;\ [\mathtt{y} \leftarrow \&\mathtt{z}_2]^3) \quad \textbf{or} \quad \tfrac{1}{2} : ([\mathtt{x} \leftarrow \&\mathtt{z}_2]^4;\ [\mathtt{y} \leftarrow \&\mathtt{z}_1]^5) \quad [\textbf{stop}]^6$$

$$\begin{aligned}
&\tfrac{1}{2} \cdot (\mathbf{I} \otimes \mathbf{E}(1,2)) + \\
&\tfrac{1}{2} \cdot (\mathbf{I} \otimes \mathbf{E}(1,4)) + \\
&\mathbf{U}(\mathtt{x} \leftarrow \&\mathtt{z}_1) \otimes \mathbf{E}(2,3) + \\
&\mathbf{U}(\mathtt{y} \leftarrow \&\mathtt{z}_2) \otimes \mathbf{E}(3,6) + \\
&\mathbf{U}(\mathtt{x} \leftarrow \&\mathtt{z}_2) \otimes \mathbf{E}(4,5) + \\
&\mathbf{U}(\mathtt{y} \leftarrow \&\mathtt{z}_1) \otimes \mathbf{E}(5,6) + \\
&\mathbf{I} \otimes \mathbf{E}(6,6)
\end{aligned}$$

Abstract Branching Probabilities

The abstract tests $\mathbf{P}^\#$ describe the branching probabilities depending on abstract values.

For example, consider $\mathbf{P}(n)$ testing if a variable with values $1, \ldots, n$ is a prime number.

Abstract Branching Probabilities

The abstract tests $\mathbf{P}^{\#}$ describe the branching probabilities depending on abstract values.

For example, consider $\mathbf{P}(n)$ testing if a variable with values $1, \ldots, n$ is a prime number.

$$\mathbf{P}(8) = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Abstract Branching Probabilities

The abstract tests $\mathbf{P}^{\#}$ describe the branching probabilities depending on abstract values.

For example, consider $\mathbf{P}(n)$ testing if a variable with values $1, \ldots, n$ is a prime number.

$$\mathbf{I} - \mathbf{P}(8) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Abstract Branching Probabilities

The abstract tests $\mathbf{P}^{\#}$ describe the branching probabilities depending on abstract values.

For example, consider $\mathbf{P}(n)$ testing if a variable with values $1, \ldots, n$ is a prime number.

Abstraction used could be parity testing for even/odd-ness.

$$\mathbf{A}_p^{\dagger}\mathbf{P}(5)\mathbf{A}_p = \begin{pmatrix} 0.50000 & 0.00000 \\ 0.00000 & 0.66667 \end{pmatrix}$$

$$\mathbf{A}_p^{\dagger}(\mathbf{I} - \mathbf{P}(5))\mathbf{A}_p = \begin{pmatrix} 0.50000 & 0.00000 \\ 0.00000 & 0.33333 \end{pmatrix}$$

Abstract Branching Probabilities

The abstract tests $\mathbf{P}^{\#}$ describe the branching probabilities depending on abstract values.

For example, consider $\mathbf{P}(n)$ testing if a variable with values $1, \ldots, n$ is a prime number.

Abstraction used could be parity testing for even/odd-ness.

$$\mathbf{A}_p^\dagger \mathbf{P}(5) \mathbf{A}_p = \begin{pmatrix} 0.50000 & 0.00000 \\ 0.00000 & 0.66667 \end{pmatrix}$$

$$\mathbf{A}_p^\dagger (\mathbf{I} - \mathbf{P}(5)) \mathbf{A}_p = \begin{pmatrix} 0.50000 & 0.00000 \\ 0.00000 & 0.33333 \end{pmatrix}$$

Abstract Branching Probabilities

The abstract tests $\mathbf{P}^{\#}$ describe the branching probabilities depending on abstract values.

For example, consider $\mathbf{P}(n)$ testing if a variable with values $1, \ldots, n$ is a prime number.

Abstraction used could be parity testing for even/odd-ness.

$$\mathbf{A}_p^{\dagger}\mathbf{P}(10)\mathbf{A}_p = \begin{pmatrix} 0.20000 & 0.00000 \\ 0.00000 & 0.60000 \end{pmatrix}$$

$$\mathbf{A}_p^{\dagger}(\mathbf{I} - \mathbf{P}(10))\mathbf{A}_p = \begin{pmatrix} 0.80000 & 0.00000 \\ 0.00000 & 0.40000 \end{pmatrix}$$

Abstract Branching Probabilities

The abstract tests $\mathbf{P}^{\#}$ describe the branching probabilities depending on abstract values.

For example, consider $\mathbf{P}(n)$ testing if a variable with values $1, \ldots, n$ is a prime number.

Abstraction used could be parity testing for even/odd-ness.

$$\mathbf{A}_p^{\dagger}\mathbf{P}(100)\mathbf{A}_p = \begin{pmatrix} 0.02000 & 0.00000 \\ 0.00000 & 0.48000 \end{pmatrix}$$

$$\mathbf{A}_p^{\dagger}(\mathbf{I} - \mathbf{P}(100))\mathbf{A}_p = \begin{pmatrix} 0.98000 & 0.00000 \\ 0.00000 & 0.52000 \end{pmatrix}$$

Abstract Branching Probabilities

The abstract tests $\mathbf{P}^{\#}$ describe the branching probabilities depending on abstract values.

For example, consider $\mathbf{P}(n)$ testing if a variable with values $1, \ldots, n$ is a prime number.

Abstraction used could be parity testing for even/odd-ness.

$$\mathbf{A}_p^{\dagger}\mathbf{P}(1000)\mathbf{A}_p = \begin{pmatrix} 0.00200 & 0.00000 \\ 0.00000 & 0.33400 \end{pmatrix}$$

$$\mathbf{A}_p^{\dagger}(\mathbf{I} - \mathbf{P}(1000))\mathbf{A}_p = \begin{pmatrix} 0.99800 & 0.00000 \\ 0.00000 & 0.66600 \end{pmatrix}$$

Abstract Branching Probabilities

The abstract tests $\mathbf{P}^{\#}$ describe the branching probabilities depending on abstract values.

For example, consider $\mathbf{P}(n)$ testing if a variable with values $1, \ldots, n$ is a prime number.

Abstraction used could be parity testing for even/odd-ness.

$$\mathbf{A}_p^{\dagger}\mathbf{P}(10000)\mathbf{A}_p = \begin{pmatrix} 0.00020 & 0.00000 \\ 0.00000 & 0.24560 \end{pmatrix}$$

$$\mathbf{A}_p^{\dagger}(\mathbf{I} - \mathbf{P}(10000))\mathbf{A}_p = \begin{pmatrix} 0.99980 & 0.00000 \\ 0.00000 & 0.75440 \end{pmatrix}$$

Transforming **if** into **choose**

Based on the abstract branching probabilities we can replace tests, e.g. in **if**'s, by probabilistic choices. In a a first phase, we need to determine the probabilities of abstract values.

If we have the probabilities of z_0 being even or odd we can compute the probabilities of the **then** and **else** branch using $\mathbf{P}^{\#}$. For z_0 being even and odd with the same probability:

$$[\textbf{choose}]^1$$
$$\quad \tfrac{1}{2} : ([x \leftarrow \& z_1]^2;\ [y \leftarrow \& z_2]^3)$$
$$\textbf{or}$$
$$\quad \tfrac{1}{2} : ([x \leftarrow \& z_2]^4;\ [y \leftarrow \& z_1]^5)$$
$$[\textbf{stop}]^6$$

Transforming **if** into **choose**

Based on the abstract branching probabilities we can replace tests, e.g. in **if**'s, by probabilistic choices. In a a first phase, we need to determine the probabilities of abstract values.

If we have the probabilities of z_0 being even or odd we can compute the probabilities of the **then** and **else** branch using $\mathbf{P}^{\#}$. For z_0 being even and odd with the same probability:

[choose]^1

$\frac{1}{2} : ([x \leftarrow \&z_1]^2;\ [y \leftarrow \&z_2]^3)$

or

$\frac{1}{2} : ([x \leftarrow \&z_2]^4;\ [y \leftarrow \&z_1]^5)$

[stop]^6

Transforming **if** into **choose**

Based on the abstract branching probabilities we can replace tests, e.g. in **if**'s, by probabilistic choices. In a a first phase, we need to determine the probabilities of abstract values.

If we have the probabilities of z_0 being even or odd we can compute the probabilities of the **then** and **else** branch using $\mathbf{P}^{\#}$. For z_0 being even and odd with the same probability:

[**choose**]^1

$\frac{1}{2} : ([x \leftarrow \& z_1]^2;\ [y \leftarrow \& z_2]^3)$

or

$\frac{1}{2} : ([x \leftarrow \& z_2]^4;\ [y \leftarrow \& z_1]^5)$

[**stop**]^6

Transforming **if** into **choose**

Based on the abstract branching probabilities we can replace tests, e.g. in **if**'s, by probabilistic choices. In a a first phase, we need to determine the probabilities of abstract values.

If we have the probabilities of $\mathtt{z}_0$ being even or odd we can compute the probabilities of the **then** and **else** branch using $\mathbf{P}^{\#}$. For $\mathtt{z}_0$ being even and odd with the same probability:

if $[(\mathtt{z}_0 \text{ mod } 2 = 0)]^1$ **then**
$[\mathtt{x} \leftarrow \&\mathtt{z}_1]^2$; $[\mathtt{y} \leftarrow \&\mathtt{z}_2]^3$
else
$[\mathtt{x} \leftarrow \&\mathtt{z}_2]^4$; $[\mathtt{y} \leftarrow \&\mathtt{z}_1]^5$
fi
[**stop**]^6

Transforming **if** into **choose**

Based on the abstract branching probabilities we can replace tests, e.g. in **if**'s, by probabilistic choices. In a a first phase, we need to determine the probabilities of abstract values.

If we have the probabilities of z_0 being even or odd we can compute the probabilities of the **then** and **else** branch using $\mathbf{P}^{\#}$. For z_0 being even and odd with the same probability:

[**choose**]^1
$\frac{1}{2} : ([x \leftarrow \& z_1]^2;\ [y \leftarrow \& z_2]^3)$
or
$\frac{1}{2} : ([x \leftarrow \& z_2]^4;\ [y \leftarrow \& z_1]^5)$
[**stop**]^6

Probabilistic Pointer Analysis

The typical result of a probabilistic pointer analysis is a so-called points-to matrix: records for every program point the probability that a pointer refers to particular (other) variable.

Consider again our standard example.

if $[(z_0 \bmod 2 = 0)]^1$ **then**
    $[x \leftarrow \&z_1]^2$; $[y \leftarrow \&z_2]^3$
else
    $[x \leftarrow \&z_2]^4$; $[y \leftarrow \&z_1]^5$
fi
$[$**stop**$]^6$

Where do x and y point to with what probabilities?

Probabilistic Pointer Analysis

The typical result of a probabilistic pointer analysis is a so-called points-to matrix: records for every program point the probability that a pointer refers to particular (other) variable.

Consider again our standard example.

if $[(z_0 \bmod 2 = 0)]^1$ **then**
$[x \leftarrow \&z_1]^2;\ [y \leftarrow \&z_2]^3$
else
$[x \leftarrow \&z_2]^4;\ [y \leftarrow \&z_1]^5$
fi
$[\textbf{stop}]^6$

Where do x and y point to with what probabilities?

Probabilistic Pointer Analysis

The typical result of a probabilistic pointer analysis is a so-called points-to matrix: records for every program point the probability that a pointer refers to particular (other) variable.

Consider again our standard example.

$$
\begin{array}{l}
\textbf{if } [(\mathtt{z}_0 \bmod 2 = 0)]^1 \textbf{ then} \\
\quad [\mathtt{x} \leftarrow \&\mathtt{z}_1]^2;\ [\mathtt{y} \leftarrow \&\mathtt{z}_2]^3 \\
\textbf{else} \\
\quad [\mathtt{x} \leftarrow \&\mathtt{z}_2]^4;\ [\mathtt{y} \leftarrow \&\mathtt{z}_1]^5 \\
\textbf{fi} \\
[\textbf{stop}]^6
\end{array}
$$

Where do $\mathtt{x}$ and $\mathtt{y}$ point to with what probabilities?

Points-To Matrix vs Points-To Tensor

if $[(z_0 \bmod 2 = 0)]^1$ **then**
$[x \leftarrow \&z_1]^2;\ [y \leftarrow \&z_2]^3$
else
$[x \leftarrow \&z_2]^4;\ [y \leftarrow \&z_1]^5$
fi
$[\textbf{stop}]^6$

Points-To Matrix

	$\&x$	$\&y$	$\&z_0$	$\&z_1$	$\&z_2$
x	0	0	0	$\frac{1}{2}$	$\frac{1}{2}$
y	0	0	0	$\frac{1}{2}$	$\frac{1}{2}$

Points-To Matrix vs Points-To Tensor

if $[(\mathtt{z}_0 \text{ mod } 2 = 0)]^1$ **then**
$[\mathtt{x} \leftarrow \&\mathtt{z}_1]^2$; $[\mathtt{y} \leftarrow \&\mathtt{z}_2]^3$
else
$[\mathtt{x} \leftarrow \&\mathtt{z}_2]^4$; $[\mathtt{y} \leftarrow \&\mathtt{z}_1]^5$
fi
$[\textbf{stop}]^6$

Points-To Matrix

$$(0,0,0,\frac{1}{2},\frac{1}{2}) \quad — \quad (0,0,0,\frac{1}{2},\frac{1}{2}).$$

Points-To Matrix vs Points-To Tensor

if $[(\mathtt{z}_0 \bmod 2 = 0)]^1$ **then**
$[\mathtt{x} \leftarrow \&\mathtt{z}_1]^2;\ [\mathtt{y} \leftarrow \&\mathtt{z}_2]^3$
else
$[\mathtt{x} \leftarrow \&\mathtt{z}_2]^4;\ [\mathtt{y} \leftarrow \&\mathtt{z}_1]^5$
fi
$[\textbf{stop}]^6$

Points-To Matrix

$$(0, 0, 0, \frac{1}{2}, \frac{1}{2}) \quad — \quad (0, 0, 0, \frac{1}{2}, \frac{1}{2}).$$

Points-To Tensor

$$\frac{1}{2} \cdot (0, 0, 0, 1, 0) \otimes (0, 0, 0, 0, 1) + \frac{1}{2} \cdot (0, 0, 0, 0, 1) \otimes (0, 0, 0, 1, 0)$$