Program Analysis (470)
Further Topics
[Not for Exam]

Herbert Wiklicky
herbert@doc.ic.ac.uk

Department of Computing
Imperial College London

Spring 2015

Further Topics in Program Analysis

Nearly all aspects or properties of programs (in different kind of language) can be topic for program analysis.

Relational Analysis: How are values of variables related?

Pointer Analysis: Where do pointers refer to (alias)?

Termination Analysis: Does program (never) terminate?

Quantitative Anlysis: Estimate probabilities of properties.

Security Analysis: Guarantee security properties.

Groundness Analysis, Shape Analysis, Information Flow, etc.

There are also various other techniques (e.g. type based analysis) and combinations of static and dynamic analysis.

Further Topics in Program Analysis

Nearly all aspects or properties of programs (in different kind of language) can be topic for program analysis.

Relational Analysis: How are values of variables related?

Pointer Analysis: Where do pointers refer to (alias)?

Termination Analysis: Does program (never) terminate?

Quantitative Anlysis: Estimate probabilities of properties.

Security Analysis: Guarantee security properties.

Groundness Analysis, Shape Analysis, Information Flow, etc.

There are also various other techniques (e.g. type based analysis) and combinations of static and dynamic analysis.

Further Topics in Program Analysis

Nearly all aspects or properties of programs (in different kind of language) can be topic for program analysis.

Relational Analysis: How are values of variables related?

Pointer Analysis: Where do pointers refer to (alias)?

Termination Analysis: Does program (never) terminate?

Quantitative Anlysis: Estimate probabilities of properties.

Security Analysis: Guarantee security properties.

Groundness Analysis, Shape Analysis, Information Flow, etc.

There are also various other techniques (e.g. type based analysis) and combinations of static and dynamic analysis.

Further Topics in Program Analysis

Nearly all aspects or properties of programs (in different kind of language) can be topic for program analysis.

Relational Analysis: How are values of variables related?

Pointer Analysis: Where do pointers refer to (alias)?

Termination Analysis: Does program (never) terminate?

Quantitative Anlysis: Estimate probabilities of properties.

Security Analysis: Guarantee security properties.

Groundness Analysis, Shape Analysis, Information Flow, etc.

There are also various other techniques (e.g. type based analysis) and combinations of static and dynamic analysis.

Further Topics in Program Analysis

Nearly all aspects or properties of programs (in different kind of language) can be topic for program analysis.

Relational Analysis: How are values of variables related?

Pointer Analysis: Where do pointers refer to (alias)?

Termination Analysis: Does program (never) terminate?

Quantitative Anlysis: Estimate probabilities of properties.

Security Analysis: Guarantee security properties.

Groundness Analysis, Shape Analysis, Information Flow, etc.

There are also various other techniques (e.g. type based analysis) and combinations of static and dynamic analysis.

Further Topics in Program Analysis

Nearly all aspects or properties of programs (in different kind of language) can be topic for program analysis.

Relational Analysis: How are values of variables related?

Pointer Analysis: Where do pointers refer to (alias)?

Termination Analysis: Does program (never) terminate?

Quantitative Anlysis: Estimate probabilities of properties.

Security Analysis: Guarantee security properties.

Groundness Analysis, Shape Analysis, Information Flow, etc.

There are also various other techniques (e.g. type based analysis) and combinations of static and dynamic analysis.

Further Topics in Program Analysis

Nearly all aspects or properties of programs (in different kind of language) can be topic for program analysis.

Relational Analysis: How are values of variables related?

Pointer Analysis: Where do pointers refer to (alias)?

Termination Analysis: Does program (never) terminate?

Quantitative Anlysis: Estimate probabilities of properties.

Security Analysis: Guarantee security properties.

Groundness Analysis, Shape Analysis, Information Flow, etc.

There are also various other techniques (e.g. type based analysis) and combinations of static and dynamic analysis.

Further Topics in Program Analysis

Nearly all aspects or properties of programs (in different kind of language) can be topic for program analysis.

Relational Analysis: How are values of variables related?

Pointer Analysis: Where do pointers refer to (alias)?

Termination Analysis: Does program (never) terminate?

Quantitative Anlysis: Estimate probabilities of properties.

Security Analysis: Guarantee security properties.

Groundness Analysis, Shape Analysis, Information Flow, etc.

There are also various other techniques (e.g. type based analysis) and combinations of static and dynamic analysis.

Why Probabilistic Analysis?

Probabilistic Program Analysis aims in understanding probabilistic programs and probabilistic properties of (deterministic) programs (e.g. average properties).

- Introducing probability in the problem formulation allows us to get approximate answers rather that a 'yes' or 'no' answers from a data-flow analysis.
- Optimisations can then be performed *with some probability* by taking into account the various costs associated to the program. cf. speculative optimisation

Alessandra Di Pierro, Chris Hankin, and Herbert Wiklicky: *Probabilistic Semantics and Analysis*, p 1-42, LNCS Vol.6155, Springer, 2010.

Why Probabilistic Analysis?

Probabilistic Program Analysis aims in understanding probabilistic programs and probabilistic properties of (deterministic) programs (e.g. average properties).

- Introducing probability in the problem formulation allows us to get approximate answers rather that a ‘yes’ or ‘no’ answers from a data-flow analysis.
- Optimisations can then be performed *with some probability* by taking into account the various costs associated to the program. cf. speculative optimisation

Alessandra Di Pierro, Chris Hankin, and Herbert Wiklicky: *Probabilistic Semantics and Analysis*, p 1-42, LNCS Vol.6155, Springer, 2010.

Why Probabilistic Analysis?

Probabilistic Program Analysis aims in understanding probabilistic programs and probabilistic properties of (deterministic) programs (e.g. average properties).

- Introducing probability in the problem formulation allows us to get approximate answers rather that a ‘yes’ or ‘no’ answers from a data-flow analysis.
- Optimisations can then be performed *with some probability* by taking into account the various costs associated to the program. cf. speculative optimisation

Alessandra Di Pierro, Chris Hankin, and Herbert Wiklicky: *Probabilistic Semantics and Analysis*, p 1-42, LNCS Vol.6155, Springer, 2010.

Why Probabilistic Analysis?

Probabilistic Program Analysis aims in understanding probabilistic programs and probabilistic properties of (deterministic) programs (e.g. average properties).

- Introducing probability in the problem formulation allows us to get approximate answers rather that a 'yes' or 'no' answers from a data-flow analysis.
- Optimisations can then be performed *with some probability* by taking into account the various costs associated to the program. cf. speculative optimisation

Alessandra Di Pierro, Chris Hankin, and Herbert Wiklicky: *Probabilistic Semantics and Analysis*, p 1-42, LNCS Vol.6155, Springer, 2010.

Estimate - Conference Budget

Determine the necessary registration fee for 130 delegates.

	Costs	Overest.	Estimate	Underest.
Proceedings	3.345 £	4.000 £	3.500 £	3.000 £
Dinner	5.672 £	6.000 £	5.500 £	5.000 £
Excursion	1.813 £	2.000 £	2.000 £	1.000 £
Room Rent	2.000 £	2.000 £	2.000 £	2.000 £
Goodies	412 £	1.000 £	500 £	—
Total	?.???? £	15.000 £	13.500 £	11.000 £

Estimate - Conference Budget

Determine the necessary registration fee for 130 delegates.

	Costs	Overest.	Estimate	Underest.
Proceedings	3.345 £	4.000 £	3.500 £	3.000 £
Dinner	5.672 £	6.000 £	5.500 £	5.000 £
Excursion	1.813 £	2.000 £	2.000 £	1.000 £
Room Rent	2.000 £	2.000 £	2.000 £	2.000 £
Goodies	412 £	1.000 £	500 £	—
Total	?.???? £	15.000 £	13.500 £	11.000 £

Estimate - Conference Budget

Determine the necessary registration fee for 130 delegates.

	Costs	Overest.	Estimate	Underest.
Proceedings	3.345 £	4.000 £	3.500 £	3.000 £
Dinner	5.672 £	6.000 £	5.500 £	5.000 £
Excursion	1.813 £	2.000 £	2.000 £	1.000 £
Room Rent	2.000 £	2.000 £	2.000 £	2.000 £
Goodies	412 £	1.000 £	500 £	—
Total	?.???? £	15.000 £	13.500 £	11.000 £

Estimate - Conference Budget

Determine the necessary registration fee for 130 delegates.

	Costs	Overest.	Estimate	Underest.
Proceedings	3.345 £	4.000 £	3.500 £	3.000 £
Dinner	5.672 £	6.000 £	5.500 £	5.000 £
Excursion	1.813 £	2.000 £	2.000 £	1.000 £
Room Rent	2.000 £	2.000 £	2.000 £	2.000 £
Goodies	412 £	1.000 £	500 £	—
Total	?.???? £	15.000 £	13.500 £	11.000 £

Estimate - Conference Budget

Determine the necessary registration fee for 130 delegates.

	Costs	Overest.	Estimate	Underest.
Proceedings	3.345 £	4.000 £	3.500 £	3.000 £
Dinner	5.672 £	6.000 £	5.500 £	5.000 £
Excursion	1.813 £	2.000 £	2.000 £	1.000 £
Room Rent	2.000 £	2.000 £	2.000 £	2.000 £
Goodies	412 £	1.000 £	500 £	—
Total	?.???? £	15.000 £	13.500 £	11.000 £

Estimate - Conference Budget

Determine the necessary registration fee for 130 delegates.

	Costs	Overest.	Estimate	Underest.
Proceedings	3.345 £	4.000 £	3.500 £	3.000 £
Dinner	5.672 £	6.000 £	5.500 £	5.000 £
Excursion	1.813 £	2.000 £	2.000 £	1.000 £
Room Rent	2.000 £	2.000 £	2.000 £	2.000 £
Goodies	412 £	1.000 £	500 £	—
Total	?.???? £	15.000 £	13.500 £	11.000 £

Estimate - Conference Budget

Determine the necessary registration fee for 130 delegates.

	Costs	Overest.	Estimate	Underest.
Proceedings	3.345 £	4.000 £	3.500 £	3.000 £
Dinner	5.672 £	6.000 £	5.500 £	5.000 £
Excursion	1.813 £	2.000 £	2.000 £	1.000 £
Room Rent	2.000 £	2.000 £	2.000 £	2.000 £
Goodies	412 £	1.000 £	500 £	—
Total	?.???? £	15.000 £	13.500 £	11.000 £

Estimate - Conference Budget

Determine the necessary registration fee for 130 delegates.

	Costs	Overest.	Estimate	Underest.
Proceedings	3.345 £	4.000 £	3.500 £	3.000 £
Dinner	5.672 £	6.000 £	5.500 £	5.000 £
Excursion	1.813 £	2.000 £	2.000 £	1.000 £
Room Rent	2.000 £	2.000 £	2.000 £	2.000 £
Goodies	412 £	1.000 £	500 £	—
Total	13.242 £	15.000 £	13.500 £	11.000 £

Relational Program Analysis

Factorial – Input/Output behaviour (I/O Semantics):

$$\begin{aligned} 1! &= 1 \\ n! &= n \cdot (n-1)! \end{aligned}$$

For random input $n \in \{1,2,3\}$ – i.e. $P(n=1) = P(n=2) = P(n=3) = \frac{1}{3}$ – determine the probability that $n!$ is **even** or **odd**:

$$P(\text{parity}(n!) = \textbf{even}) = \frac{2}{3} \text{ and } P(\text{parity}(n!) = \textbf{odd}) = \frac{1}{3}.$$

However – the probabilities are not independent – we have, e.g.

$$\begin{aligned} P(\text{parity}(n!) = \textbf{even} \ \wedge \ n=1) &= 0 \\ P(\text{parity}(n!) = \textbf{even}) \cdot P(n=1) &= \frac{2}{3} \cdot \frac{1}{3} = \frac{2}{9} \end{aligned}$$

Relational Program Analysis

Factorial – Input/Output behaviour (I/O Semantics):

$$\begin{aligned} 1! &= 1 \\ n! &= n \cdot (n-1)! \end{aligned}$$

For random input $n \in \{1,2,3\}$ – i.e. $P(n=1) = P(n=2) = P(n=3) = \frac{1}{3}$ – determine the probability that $n!$ is **even** or **odd**:

$$P(\text{parity}(n!) = \textbf{even}) = \frac{2}{3} \quad \text{and} \quad P(\text{parity}(n!) = \textbf{odd}) = \frac{1}{3}.$$

However – the probabilities are not independent – we have, e.g.

$$\begin{aligned} P(\text{parity}(n!) = \textbf{even} \ \wedge \ n = 1) &= 0 \\ P(\text{parity}(n!) = \textbf{even}) \cdot P(n=1) &= \frac{2}{3} \cdot \frac{1}{3} = \frac{2}{9} \end{aligned}$$

Relational Program Analysis

Factorial – Input/Output behaviour (I/O Semantics):

$$\begin{aligned} 1! &= 1 \\ n! &= n \cdot (n-1)! \end{aligned}$$

For random input $n \in \{1,2,3\}$ – i.e. $P(n=1) = P(n=2) = P(n=3) = \frac{1}{3}$ – determine the probability that $n!$ is **even** or **odd**:

$$P(\text{parity}(n!) = \textbf{even}) = \frac{2}{3} \text{ and } P(\text{parity}(n!) = \textbf{odd}) = \frac{1}{3}.$$

However – the probabilities are not independent – we have, e.g.

$$\begin{aligned} P(\text{parity}(n!) = \textbf{even} \ \wedge \ n = 1) &= 0 \\ P(\text{parity}(n!) = \textbf{even}) \cdot P(n=1) &= \frac{2}{3} \cdot \frac{1}{3} = \frac{2}{9} \end{aligned}$$

Expressing Dependencies/Correlations/etc.

Some joint probability distributions can be expressed as tensor product of two (independent) probability distributions $\mathbf{d}_1$ and $\mathbf{d}_2$:

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

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

$$\begin{pmatrix} 0 & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & 0 & 0 \end{pmatrix} = \frac{1}{3}(\mathbf{e}_2 \otimes \mathbf{f}_1) + \frac{1}{3}(\mathbf{e}_1 \otimes \mathbf{f}_2) + \frac{1}{3}(\mathbf{e}_1 \otimes \mathbf{f}_3)$$

with $\mathbf{e}_i \in \mathbb{R}^2$ and $\mathbf{f}_j \in \mathbb{R}^3$ (column and row) basis vectors

Expressing Dependencies/Correlations/etc.

Some joint probability distributions can be expressed as tensor product of two (independent) probability distributions $\mathbf{d}_1$ and $\mathbf{d}_2$:

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

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

$$\begin{pmatrix} 0 & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & 0 & 0 \end{pmatrix} = \frac{1}{3}(\mathbf{e}_2 \otimes \mathbf{f}_1) + \frac{1}{3}(\mathbf{e}_1 \otimes \mathbf{f}_2) + \frac{1}{3}(\mathbf{e}_1 \otimes \mathbf{f}_3)$$

with $\mathbf{e}_i \in \mathbb{R}^2$ and $\mathbf{f}_j \in \mathbb{R}^3$ (column and row) basis vectors

Expressing Dependencies/Correlations/etc.

Some joint probability distributions can be expressed as tensor product of two (independent) probability distributions $\mathbf{d}_1$ and $\mathbf{d}_2$:

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

But there are no two vectors $\mathbf{d}_1$ and $\mathbf{d}_2$ such that for example

$$\begin{pmatrix} 0 & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & 0 & 0 \end{pmatrix} = \mathbf{d}_1 \otimes \mathbf{d}_2$$

Expressing Dependencies/Correlations/etc.

Some joint probability distributions can be expressed as tensor product of two (independent) probability distributions $\mathbf{d}_1$ and $\mathbf{d}_2$:

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

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

$$\begin{pmatrix} 0 & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & 0 & 0 \end{pmatrix} = \frac{1}{3}(\mathbf{e}_2 \otimes \mathbf{f}_1) + \frac{1}{3}(\mathbf{e}_1 \otimes \mathbf{f}_2) + \frac{1}{3}(\mathbf{e}_1 \otimes \mathbf{f}_3)$$

with $\mathbf{e}_i \in \mathbb{R}^2$ and $\mathbf{f}_j \in \mathbb{R}^3$ (column and row) basis vectors

Tensor (Kronecker) Product

Every vector space has an (algebraic) base $\{\mathbf{e}_i\}$ such that

$$\mathbf{x} = x_1\mathbf{e}_1 + x_2\mathbf{e}_2 + \dots$$

This allows to specify vectors via coordinates $\mathbf{x} = (x_1, x_2, \dots)$.
Base vectors are in this context simply of the form

$$\mathbf{e}_i = (e_{i1}, e_{i2}, \dots) \quad \text{with } e_{ij} = \begin{cases} 1 & \text{for } i = j \\ 0 & \text{otherwise} \end{cases}$$

The tensor product space $\mathcal{V} \otimes \mathcal{W}$ can be seen as generated by (formal) tensors of the form $\mathbf{v}_i \otimes \mathbf{w}_j$ with $\mathbf{v}_i \in \mathcal{V}$ and $\mathbf{w} \in \mathcal{W}$ base vectors, i.e. every $\mathbf{x} \in \mathcal{V} \otimes \mathcal{W}$ is of the form

$$\mathbf{x} = \sum_{ij} x_{ij} \cdot (\mathbf{v}_i \otimes \mathbf{w}_j).$$

Tensor (Kronecker) Product

Every vector space has an (algebraic) base $\{\mathbf{e}_i\}$ such that

$$\mathbf{x} = x_1\mathbf{e}_1 + x_2\mathbf{e}_2 + \dots$$

This allows to specify vectors via coordinates $\mathbf{x} = (x_1, x_2, \dots)$.
Base vectors are in this context simply of the form

$$\mathbf{e}_i = (e_{i1}, e_{i2}, \dots) \quad \text{with } e_{ij} = \begin{cases} 1 & \text{for } i = j \\ 0 & \text{otherwise} \end{cases}$$

The tensor product space $\mathcal{V} \otimes \mathcal{W}$ can be seen as generated by (formal) tensors of the form $\mathbf{v}_i \otimes \mathbf{w}_j$ with $\mathbf{v}_i \in \mathcal{V}$ and $\mathbf{w} \in \mathcal{W}$ base vectors, i.e. every $\mathbf{x} \in \mathcal{V} \otimes \mathcal{W}$ is of the form

$$\mathbf{x} = \sum_{ij} x_{ij} \cdot (\mathbf{v}_i \otimes \mathbf{w}_j).$$

Tensor (Kronecker) Product

Every vector space has an (algebraic) base $\{\mathbf{e}_i\}$ such that

$$\mathbf{x} = x_1\mathbf{e}_1 + x_2\mathbf{e}_2 + \dots$$

This allows to specify vectors via coordinates $\mathbf{x} = (x_1, x_2, \dots)$.
Base vectors are in this context simply of the form

$$\mathbf{e}_i = (e_{i1}, e_{i2}, \dots) \quad \text{with } e_{ij} = \begin{cases} 1 & \text{for } i = j \\ 0 & \text{otherwise} \end{cases}$$

The tensor product space $\mathcal{V} \otimes \mathcal{W}$ can be seen as generated by (formal) tensors of the form $\mathbf{v}_i \otimes \mathbf{w}_j$ with $\mathbf{v}_i \in \mathcal{V}$ and $\mathbf{w} \in \mathcal{W}$ base vectors, i.e. every $\mathbf{x} \in \mathcal{V} \otimes \mathcal{W}$ is of the form

$$\mathbf{x} = \sum_{ij} x_{ij} \cdot (\mathbf{v}_i \otimes \mathbf{w}_j).$$

Tensor (Kronecker) Product

Every vector space has an (algebraic) base $\{\mathbf{e}_i\}$ such that

$$\mathbf{x} = x_1\mathbf{e}_1 + x_2\mathbf{e}_2 + \dots$$

This allows to specify vectors via coordinates $\mathbf{x} = (x_1, x_2, \dots)$. Base vectors are in this context simply of the form

$$\mathbf{e}_i = (e_{i1}, e_{i2}, \dots) \quad \text{with } e_{ij} = \begin{cases} 1 & \text{for } i = j \\ 0 & \text{otherwise} \end{cases}$$

The tensor product space $\mathcal{V} \otimes \mathcal{W}$ can be seen as generated by (formal) tensors of the form $\mathbf{v}_i \otimes \mathbf{w}_j$ with $\mathbf{v}_i \in \mathcal{V}$ and $\mathbf{w} \in \mathcal{W}$ base vectors, i.e. every $\mathbf{x} \in \mathcal{V} \otimes \mathcal{W}$ is of the form

$$\mathbf{x} = \sum_{ij} x_{ij} \cdot (\mathbf{v}_i \otimes \mathbf{w}_j).$$

Markov Chain Semantics

The collecting semantics of a program P is given by:

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

i.e. as a (stochastic) linear operator (i.e. matrix) on the space of distributions over configurations.

Local effects $\mathbf{T}(\ell_i, \ell_j)$: Data Update $\mathbf{N}$ + Control Step $\mathbf{M}$

$$\mathbf{T}(\ell_i, \ell_j) = \mathbf{N}_i \otimes \mathbf{M}_{ij} = \mathbf{N}_{i1} \otimes \mathbf{N}_{i2} \otimes \ldots \otimes \mathbf{N}_{iv} \otimes \mathbf{M}_{ij}$$

Describes the effect of changes to the probabilistic state via a Discrete Time Markov Chain (DTMC)

Markov Chain Semantics

The collecting semantics of a program P is given by:

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

i.e. as a (stochastic) linear operator (i.e. matrix) on the space of distributions over configurations.

Local effects $\mathbf{T}(\ell_i, \ell_j)$: Data Update $\mathbf{N}$ + Control Step $\mathbf{M}$

$$\mathbf{T}(\ell_i, \ell_j) = \mathbf{N}_i \otimes \mathbf{M}_{ij} = \mathbf{N}_{i1} \otimes \mathbf{N}_{i2} \otimes \ldots \otimes \mathbf{N}_{iv} \otimes \mathbf{M}_{ij}$$

Describes the effect of changes to the probabilistic state via a Discrete Time Markov Chain (DTMC)

Markov Chain Semantics

The collecting semantics of a program P is given by:

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

i.e. as a (stochastic) linear operator (i.e. matrix) on the space of distributions over configurations.

Local effects $\mathbf{T}(\ell_i, \ell_j)$: Data Update $\mathbf{N}$ + Control Step $\mathbf{M}$

$$\mathbf{T}(\ell_i, \ell_j) = \mathbf{N}_i \otimes \mathbf{M}_{ij} = \mathbf{N}_{i1} \otimes \mathbf{N}_{i2} \otimes \ldots \otimes \mathbf{N}_{iv} \otimes \mathbf{M}_{ij}$$

Describes the effect of changes to the probabilistic state via a Discrete Time Markov Chain (DTMC)

Modelling Tests

Consider conditional jumps or statements, e.g.

$$\textbf{if } \mathit{even}(x) \textbf{ then } x := x/2 \textbf{ else } y + 1 \textbf{ fi}$$

NB To avoid errors $a/b = \lceil a/b \rceil$ and $a + b = a + b \bmod n$.

Modelling Tests

Consider conditional jumps or statements, e.g.

$$\textbf{if } \mathit{even}(x) \textbf{ then } x := x/2 \textbf{ else } y + 1 \textbf{ fi}$$

NB To avoid errors $a/b = \lceil a/b \rceil$ and $a + b = a + b \bmod n$.

Modelling Tests

Consider conditional jumps or statements, e.g.

$$\textbf{if } \mathit{even}(x) \textbf{ then } x := x/2 \textbf{ else } y + 1 \textbf{ fi}$$

NB To avoid errors $a/b = \lceil a/b \rceil$ and $a + b = a + b \bmod n$.

$$[\![x := x/2]\!] = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \otimes \mathbf{I}$$

Modelling Tests

Consider conditional jumps or statements, e.g.

$$\textbf{if } \mathit{even}(x) \textbf{ then } x := x/2 \textbf{ else } y + 1 \textbf{ fi}$$

NB To avoid errors $a/b = \lceil a/b \rceil$ and $a + b = a + b \bmod n$.

$$[\![y := y + 1]\!] = \mathbf{I} \otimes \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

Tests and Distribution Splitting

We represent the filter for testing if x is even by a projection:

$$\mathbf{P}(\mathit{even}(x)) = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \otimes \mathbf{I}$$

Its negation is represented by:

$$\mathbf{P}(\neg \mathit{even}(x)) = \mathbf{P}(\mathit{even}(x))^{\perp} = \mathbf{I} - \mathbf{P}(\mathit{even}(x)).$$

Conditional Statements

The semantics of a conditional is given by applying the semantics of the branches to the filtered (probabilistic) states and to combine the results. In our example:

$$[\![\textbf{if } \mathit{even}(x) \textbf{ then } x := x/2 \textbf{ else } y + 1 \textbf{ fi}]\!] =$$
$$= \quad \mathbf{P}(\mathit{even}(x)) \cdot [\![x := x/2]\!] + \mathbf{P}(\mathit{even}(x))^{\perp} \cdot [\![y := y + 1]\!]$$

Given state where x has with probability $\frac{1}{2}$ values 3 and 6, and y value 2, i.e. $\sigma_0 = (0, 0, \frac{1}{2}, 0, 0, \frac{1}{2}, 0, 0) \otimes (0, 1, 0, 0)$ then

$$\sigma_0 \cdot \mathbf{P}(\mathit{even}(x)) \quad = \quad (0, 0, 0, 0, 0, \frac{1}{2}, 0, 0) \otimes (0, 1, 0, 0)$$
$$\sigma_0 \cdot \mathbf{P}(\mathit{even}(x))^{\perp} \quad = \quad (0, 0, \frac{1}{2}, 0, 0, 0, 0, 0) \otimes (0, 1, 0, 0)$$

Semantics of Conditionals

Applying the semantics of both branches gives us:

$$\begin{aligned}
&\sigma_0 \cdot \mathbf{P}(\mathit{even}(x)) \cdot [\![x := x/2]\!] = \\
&\quad = \quad (0, 0, \frac{1}{2}, 0, 0, 0, 0) \otimes (0, 1, 0, 0) \\
&\sigma_0 \cdot \mathbf{P}(\mathit{even}(x))^{\perp} \cdot [\![y := y + 1]\!] = \\
&\quad = \quad (0, 0, \frac{1}{2}, 0, 0, 0, 0, 0) \otimes (0, 0, 1, 0)
\end{aligned}$$

The sum of both branches is now, maybe somewhat surprising:

$$\sigma = (0, 0, 1, 0, 0, 0, 0, 0) \otimes (0, \frac{1}{2}, \frac{1}{2}, 0)$$

Though we have started with a definitive value for y and a distribution for x, the opposite is now the case

Semantical Abstraction

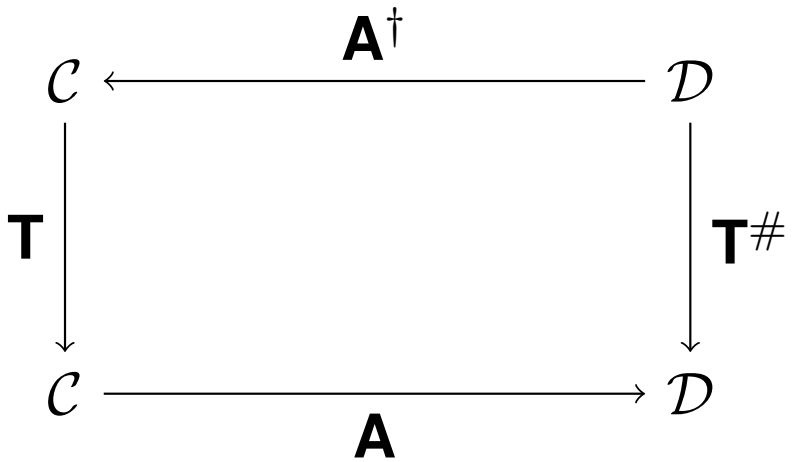
Consider a concrete domain $\mathcal{C}$ and an abstract domain $\mathcal{D}$:

With an abstraction $\mathbf{A} : \mathbf{C} \to \mathbf{D}$ and a concretisation $\mathbf{G} : \mathbf{D} \to \mathbf{C}$:

$$\mathbf{T}^\# = \mathbf{GTA}$$

Semantical Abstraction

Consider a concrete domain $\mathcal{C}$ and an abstract domain $\mathcal{D}$:

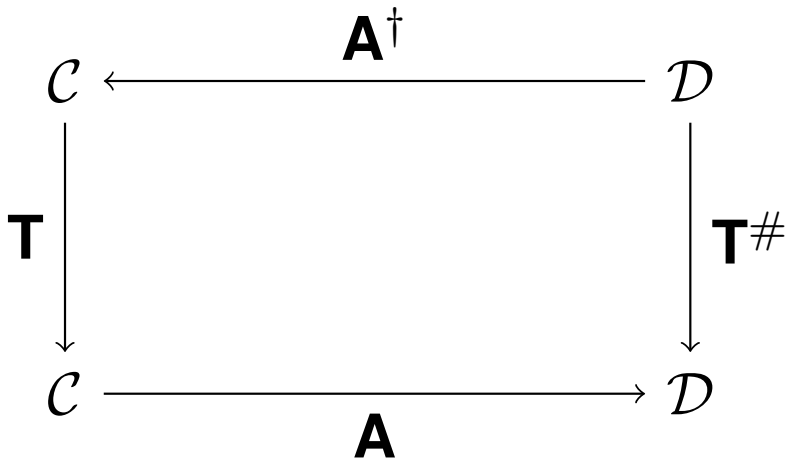


With an abstraction $\mathbf{A} : \mathbf{C} \to \mathbf{D}$ and a concretisation $\mathbf{G} : \mathbf{D} \to \mathbf{C}$:

$$\mathbf{T}^\# = \mathbf{GTA}$$

Semantical Abstraction

Consider a concrete domain $\mathcal{C}$ and an abstract domain $\mathcal{D}$:

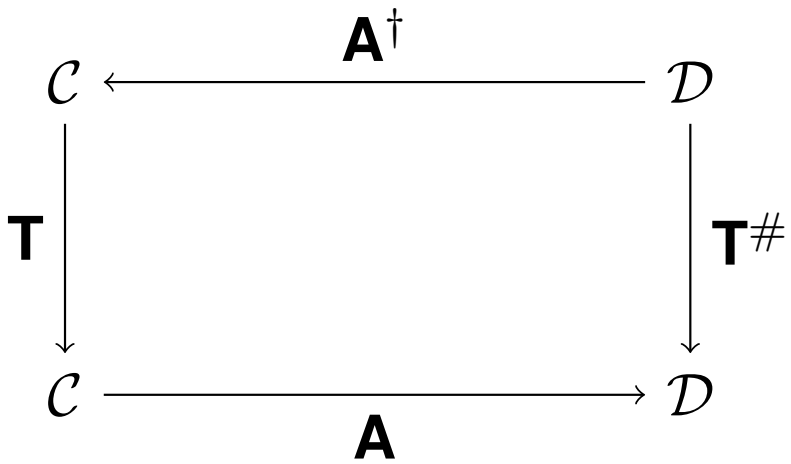


With an abstraction $\mathbf{A} : \mathbf{C} \to \mathbf{D}$ and a concretisation $\mathbf{G} : \mathbf{D} \to \mathbf{C}$:

$$\mathbf{T}^\# = \mathbf{GTA}$$

Semantical Abstraction

Consider a concrete domain $\mathcal{C}$ and an abstract domain $\mathcal{D}$:



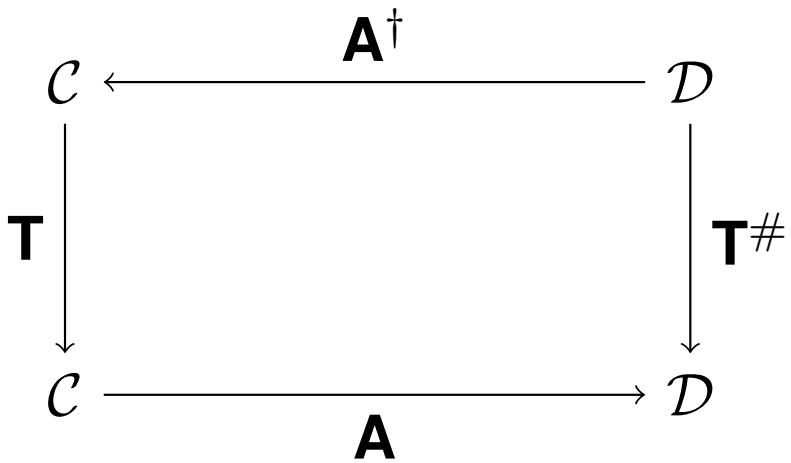
With an abstraction $\mathbf{A} : \mathbf{C} \to \mathbf{D}$ and a concretisation $\mathbf{G} : \mathbf{D} \to \mathbf{C}$:

$$\mathbf{T}^\# = \mathbf{GTA}$$

Abstract Interpretation: $(\mathbf{A}, \mathbf{G})$ form a Galois Connection.

Semantical Abstraction

Consider a concrete domain $\mathcal{C}$ and an abstract domain $\mathcal{D}$:



With an abstraction $\mathbf{A} : \mathbf{C} \to \mathbf{D}$ and a concretisation $\mathbf{G} : \mathbf{D} \to \mathbf{C}$:

$$\mathbf{T}^\# = \mathbf{GTA}$$

Probabilistic Abst.Int.: $(\mathbf{A}, \mathbf{G})$ Moore-Penrose Pseudo-Inverse.

Moore-Penrose Pseudo-Inverse

Definition

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

$$\text{(i)} \quad \mathbf{A} \circ \mathbf{G} = \mathbf{P}_A,$$
$$\text{(ii)} \quad \mathbf{G} \circ \mathbf{A} = \mathbf{P}_G,$$

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

For a concrete linear operator $\mathbf{T}$ describing the semantics of a statement or test we have the induced abstract semantics as $\mathbf{T}^\# = \mathbf{A}^\dagger \mathbf{T} \mathbf{A}$ (post-multiplication).

Moore-Penrose Pseudo-Inverse

Definition

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

$$\text{(i)} \quad \mathbf{A} \circ \mathbf{G} = \mathbf{P}_A,$$
$$\text{(ii)} \quad \mathbf{G} \circ \mathbf{A} = \mathbf{P}_G,$$

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

For a concrete linear operator $\mathbf{T}$ describing the semantics of a statement or test we have the induced abstract semantics as $\mathbf{T}^\# = \mathbf{A}^\dagger\mathbf{T}\mathbf{A}$ (post-multiplication).

Examples of Abstractions

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

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

Examples of 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 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 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: Factorial

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

Example: Factorial

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

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

Abstract Semantics

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 – Matrices

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

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

Abstract Semantics – Matrices

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

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

Abstract Semantics – Matrices

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

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

Abstract Semantics – Matrices

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

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

Abstract Semantics – Matrices

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

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

Implementation

Implementation of concrete and abstract semantics of Factorial using **octave**. Ranges: $n \in \{1, 2, \max\}$ and $m \in \{1, 2, \max!\}$.

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

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

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

Implementation

Implementation of concrete and abstract semantics of Factorial using **octave**. Ranges: $n \in \{1, 2, \max\}$ and $m \in \{1, 2, \max!\}$.

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

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

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

Implementation

Implementation of concrete and abstract semantics of Factorial using **octave**. Ranges: $n \in \{1, 2, \max\}$ and $m \in \{1, 2, \max!\}$.

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

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

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

Secure Information Flow

We investigate the problem of (guaranteeing) secure information flow, i.e. look at the question which objects can be handled by which subjects securely.

Subjects: People, officials, processes, ...

Objects: Papers, files, variables, ...

In order to simplify things we consider only two different types of interaction:

Read and Write.

Secure Information Flow

We investigate the problem of (guaranteeing) secure information flow, i.e. look at the question which objects can be handled by which subjects securely.

Subjects: People, officials, processes, ...

Objects: Papers, files, variables, ...

In order to simplify things we consider only two different types of interaction:

Read and Write.

Secure Information Flow

We investigate the problem of (guaranteeing) secure information flow, i.e. look at the question which objects can be handled by which subjects securely.

Subjects: People, officials, processes, ...

Objects: Papers, files, variables, ...

In order to simplify things we consider only two different types of interaction:

Read and Write.

Security Policy

In order to avoid non-authorised information flow:

Discretional Approach: The “owner” of sensitive information controls access.

Non-Discretional Approach: There are formal rules which guide information access.

For the non-discretional approach one has thus to formulate a security policy which determines which subjects can access which documents in which way.

Security Policy

In order to avoid non-authorised information flow:

Discretional Approach: The “owner” of sensitive information controls access.

Non-Discretional Approach: There are formal rules which guide information access.

For the non-discretional approach one has thus to formulate a security policy which determines which subjects can access which documents in which way.

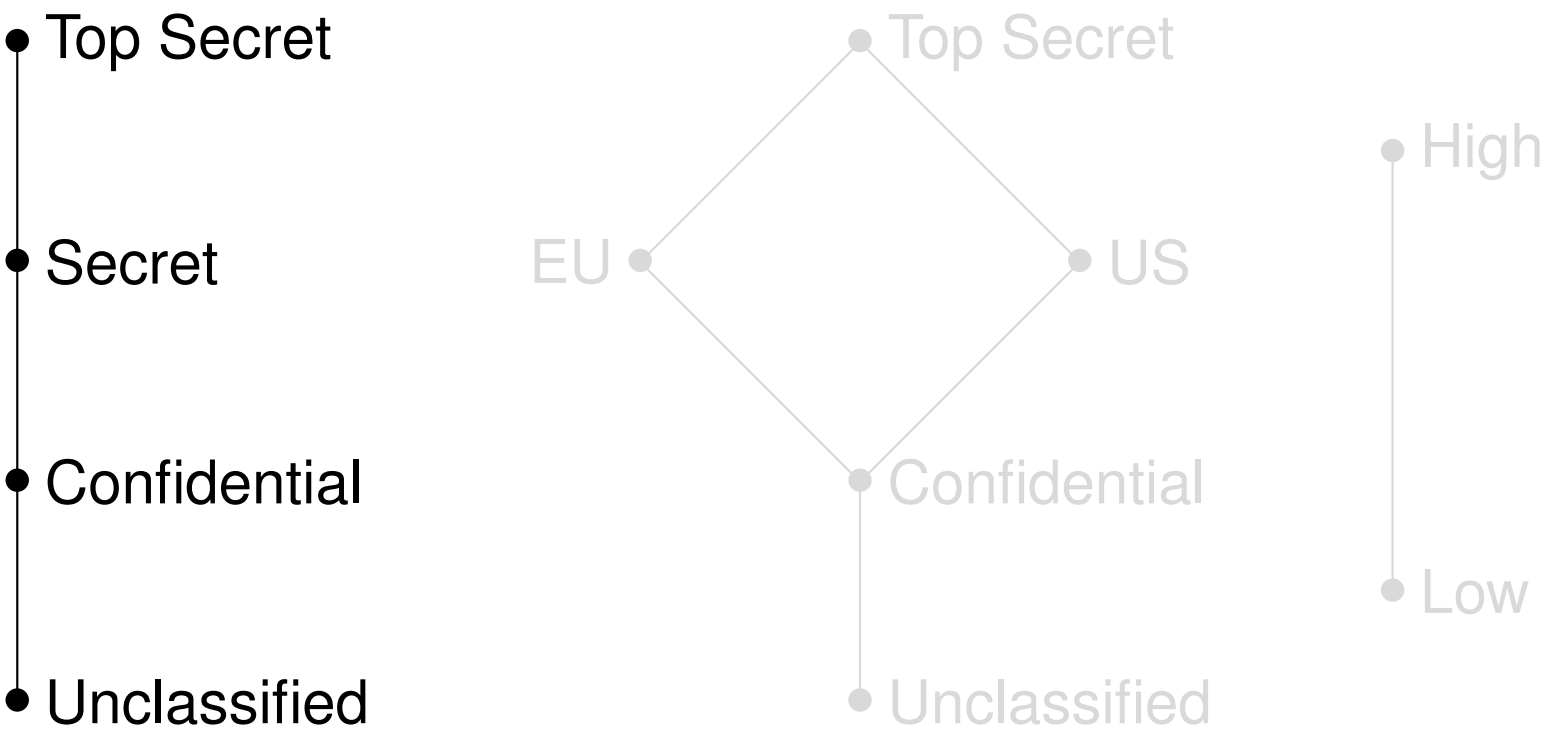
Bell & LaPadula — Denning

Security policies are often formulated in terms of security classes or clearance levels for subjects and objects.

The security classes form a lattice $(SC, \sqsubseteq)$.

Bell & LaPadula — Denning

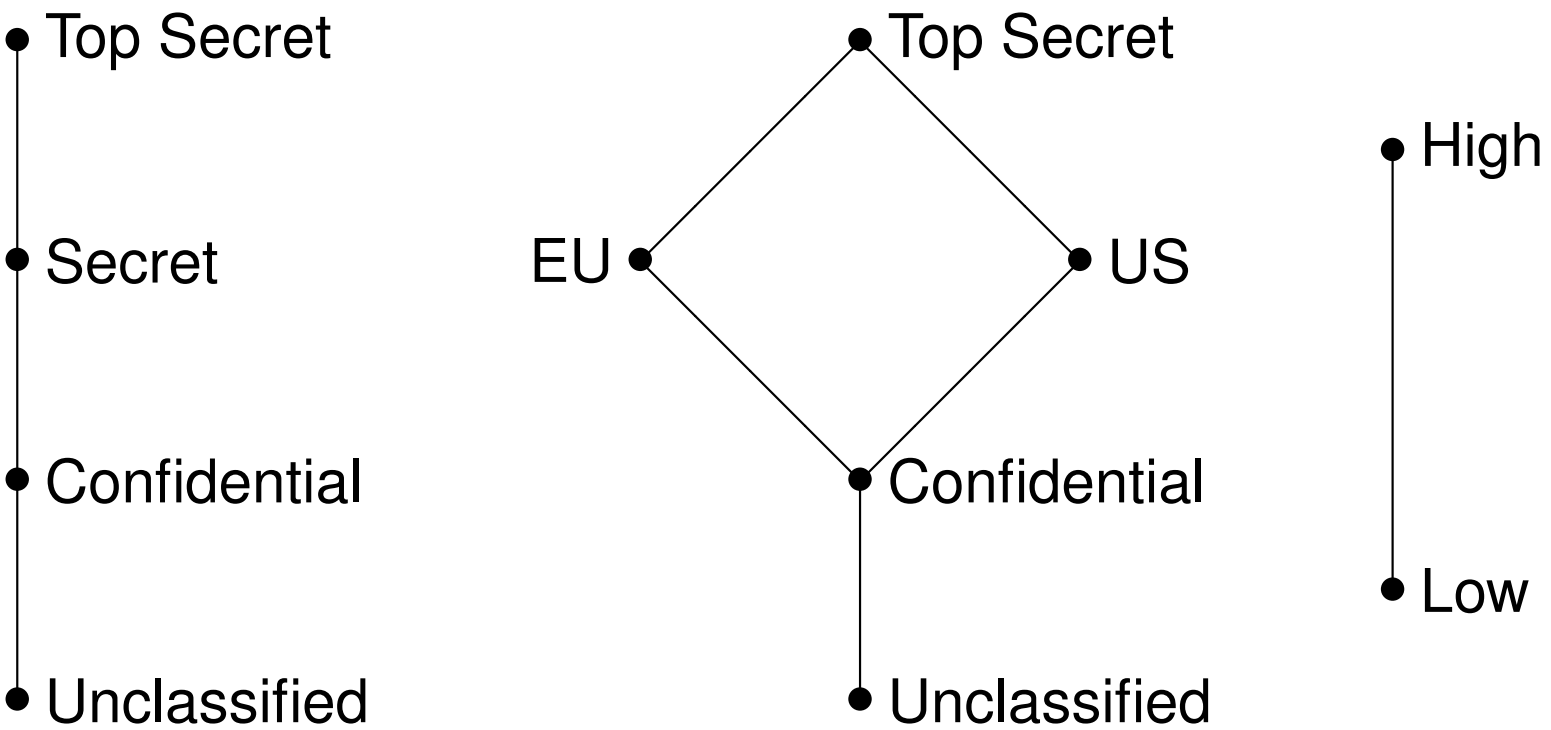
Security policies are often formulated in terms of security classes or clearance levels for subjects and objects.



The security classes form a lattice $(SC, \sqsubseteq)$.

Bell & LaPadula — Denning

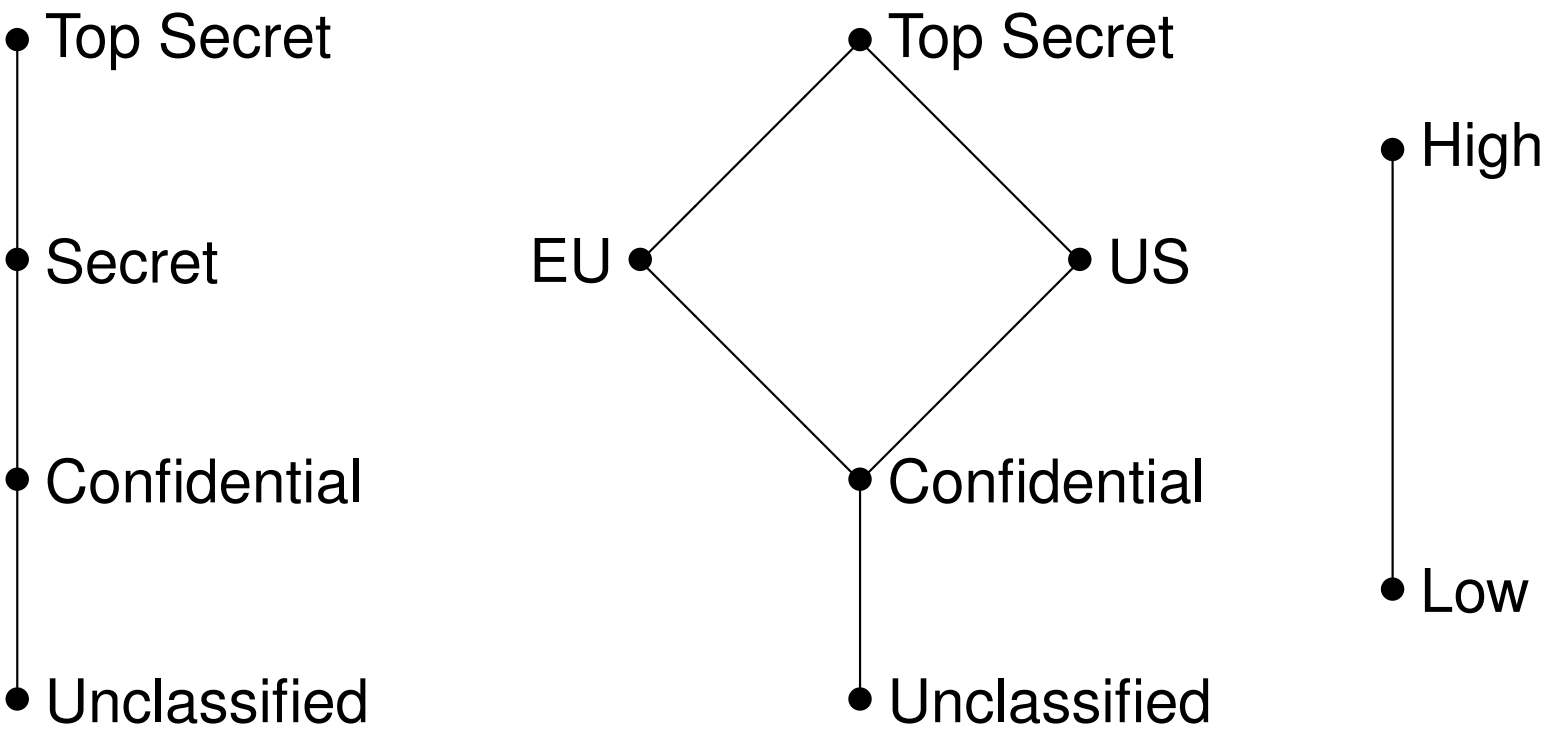
Security policies are often formulated in terms of security classes or clearance levels for subjects and objects.



The security classes form a lattice $(SC, \sqsubseteq)$.

Bell & LaPadula — Denning

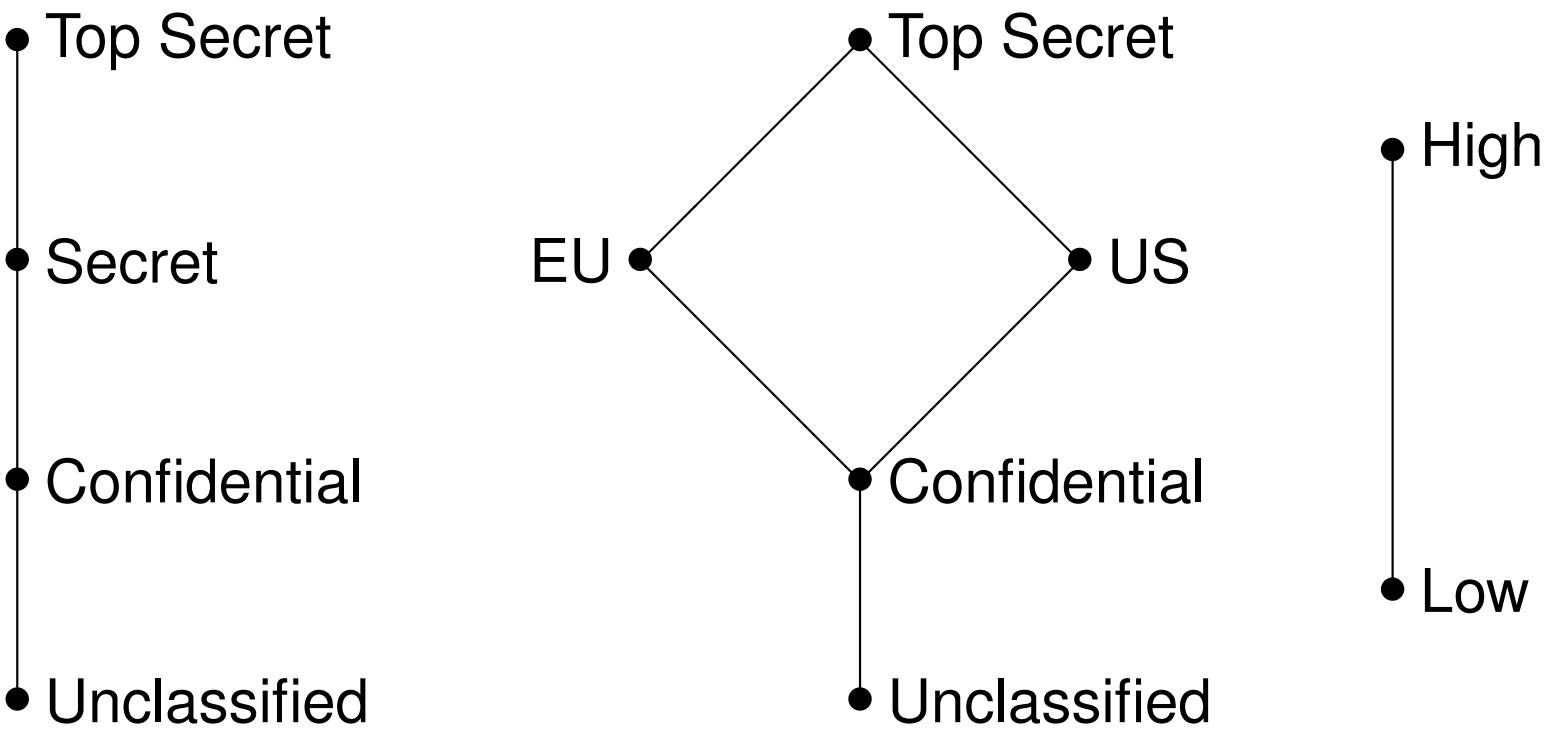
Security policies are often formulated in terms of security classes or clearance levels for subjects and objects.



The security classes form a lattice $(SC, \sqsubseteq)$.

Bell & LaPadula — Denning

Security policies are often formulated in terms of security classes or clearance levels for subjects and objects.



The security classes form a lattice $(SC, \sqsubseteq)$.

Basic Security Principles

There are two principles in the Bell-LaPadula approach which determine the security level of objects which can be accessed by subjects with a certain security clearance.

No Read-Up: A subject is allowed a read access to an object only if the security class of the subject dominates the security class of the object.

No Write-Down: A subject is allowed a write access to an object only if the security class of the subject is dominated by the security class of the object.

Basic Security Principles

There are two principles in the Bell-LaPadula approach which determine the security level of objects which can be accessed by subjects with a certain security clearance.

No Read-Up: A subject is allowed a read access to an object only if the security class of the subject dominates the security class of the object.

No Write-Down: A subject is allowed a write access to an object only if the security class of the subject is dominated by the security class of the object.

Basic Security Principles

There are two principles in the Bell-LaPadula approach which determine the security level of objects which can be accessed by subjects with a certain security clearance.

No Read-Up: A subject is allowed a read access to an object only if the security class of the subject dominates the security class of the object.

No Write-Down: A subject is allowed a write access to an object only if the security class of the subject is dominated by the security class of the object.

Information Flow

Direct Information Flow

```
high := low   vs   low := high
```

Indirect Information Flow

```
if high=0 then low:=0 else low:=1
```

```
low:=0; while low<high do low:=low+1
```

Information Flow

Direct Information Flow

```
high := low   vs   low := high
```

Indirect Information Flow

```
if high=0 then low:=0 else low:=1
```

```
low:=0; while low<high do low:=low+1
```

Information Flow

Direct Information Flow

```
high := low   vs   low := high
```

Indirect Information Flow

```
if high=0 then low:=0 else low:=1
```

```
low:=0; while low<high do low:=low+1
```

Type System for Security

Assume a lattice of *security classes* $(SC, \sqsubseteq)$.

$$\sigma \in SC$$

(data types) $\quad \tau ::= \sigma$

(phrase types) $\quad \rho ::= \tau \mid \tau\ var \mid \tau\ cmd$

Identifier Typing

$$SC : \mathbf{Var}_\star \to SC$$

Denis Volpano, Geoffrey Smith and Cynthia Irvine:
A Sound Type System for Secure Flow Analysis, Journal of Computer Security 4(3), 167–187, 1996.

Type System for Security

Assume a lattice of *security classes* $(SC, \sqsubseteq)$.

$$\sigma \in SC$$

(data types)	τ	$::=$	σ
(phrase types)	ρ	$::=$	$\tau \mid \tau\ var \mid \tau\ cmd$

Identifier Typing

$$SC : \mathbf{Var}_\star \to SC$$

Denis Volpano, Geoffrey Smith and Cynthia Irvine:
A Sound Type System for Secure Flow Analysis, Journal of Computer Security 4(3), 167–187, 1996.

Type System for Security

Assume a lattice of *security classes* $(SC, \sqsubseteq)$.

$$\sigma \in SC$$

$$\begin{array}{lrcl} \text{(data types)} & \tau & ::= & \sigma \\ \text{(phrase types)} & \rho & ::= & \tau \mid \tau\ \mathit{var} \mid \tau\ \mathit{cmd} \end{array}$$

Identifier Typing

$$SC : \mathbf{Var}_\star \to SC$$

Denis Volpano, Geoffrey Smith and Cynthia Irvine:
A Sound Type System for Secure Flow Analysis, Journal of Computer Security 4(3), 167–187, 1996.

Type System for Security

Assume a lattice of *security classes* $(SC, \sqsubseteq)$.

$$\sigma \in SC$$

$$\begin{array}{lrcl} \text{(data types)} & \tau & ::= & \sigma \\ \text{(phrase types)} & \rho & ::= & \tau \mid \tau\ \mathit{var} \mid \tau\ \mathit{cmd} \end{array}$$

Identifier Typing

$$SC : \mathbf{Var}_\star \to SC$$

Denis Volpano, Geoffrey Smith and Cynthia Irvine:
A Sound Type System for Secure Flow Analysis, Journal of Computer Security 4(3), 167–187, 1996.

Security Typing I

$$\text{(CONST)} \quad SC \vdash n : \tau$$

$$\text{(VAR)} \quad SC \vdash x : \tau\ var \quad \text{if } SC(x) = \tau$$

$$\text{(EXPR)} \quad \frac{SC \vdash e_1 : \tau \qquad SC \vdash e_2 : \tau}{SC \vdash e_1\ op\ e_2 : \tau}$$

$$\text{(VAL)} \quad \frac{SC \vdash e : \tau\ var}{SC \vdash e : \tau}$$

$$\text{(ASSIGN)} \quad \frac{SC \vdash x : \tau\ var \qquad SC \vdash e : \tau}{SC \vdash x := e : \tau\ cmd}$$

Security Typing II

$$(\textsc{skp}) \quad SC \vdash \texttt{skip} : \tau\ cmd$$

$$(\textsc{seq}) \quad \frac{SC \vdash S_1 : \tau\ cmd \qquad SC \vdash S_2 : \tau\ cmd}{SC \vdash S_1;\ S_2 : \tau\ cmd}$$

$$(\textsc{if}) \quad \frac{SC \vdash e : \tau \qquad SC \vdash S_1 : \tau\ cmd \qquad SC \vdash S_2 : \tau\ cmd}{SC \vdash \texttt{if}\ e\ \texttt{then}\ S_1\ \texttt{else}\ S_2 : \tau\ cmd}$$

$$(\textsc{while}) \quad \frac{SC \vdash e : \tau \qquad SC \vdash S : \tau\ cmd}{SC \vdash \texttt{while}\ e\ \texttt{do}\ S : \tau\ cmd}$$

Type Soundness = Security

Theorem

If

- $SC \vdash S : \rho$
- $\langle S, s_1 \rangle \Rightarrow^* \langle S_1, s_1' \rangle$ *or* $\langle S, s_1 \rangle \Rightarrow^* s_1'$
- $\langle S, s_2 \rangle \Rightarrow^* \langle S_2, s_2' \rangle$ *or* $\langle S, s_2 \rangle \Rightarrow^* s_2'$
- $s_1(x) = s_2(x)$ *for all* x *with* $SC(x) \sqsubseteq \tau$

then

- $s_1'(x) = s_2'(x)$ *for all* x *with* $SC(x) \sqsubseteq \tau$.

For all typeable programs S and any given security level τ: If the initial states agree on “low” variables they will also be indistinguishable on “low” variables in the end; no leakage.

Type Soundness = Security

Theorem

If

- $SC \vdash S : \rho$
- $\langle S, s_1 \rangle \Rightarrow^* \langle S_1, s_1' \rangle$ *or* $\langle S, s_1 \rangle \Rightarrow^* s_1'$
- $\langle S, s_2 \rangle \Rightarrow^* \langle S_2, s_2' \rangle$ *or* $\langle S, s_2 \rangle \Rightarrow^* s_2'$
- $s_1(x) = s_2(x)$ *for all* x *with* $SC(x) \sqsubseteq \tau$

then

- $s_1'(x) = s_2'(x)$ *for all* x *with* $SC(x) \sqsubseteq \tau$.

For all typeable programs S and any given security level τ: If the initial states agree on “low” variables they will also be indistinguishable on “low” variables in the end; no leakage.