

A nondeterministic lattice of information

Carroll Morgan
University of New South Wales, and NICTA
Sydney, AU

The essence of information flow

Landauer and Redmond, 1993:

A ([deterministic](#)) lattice of information

Alvim, Chatzikokolakis, McIver, Morgan, Smith,
Palamidessi, since [2012](#):

A ([probabilistic](#)) lattice of information

This talk: filling the gap

- ① Landauer and Redmond, 1993:
A (deterministic) lattice of information
- ③ MPC 2015:
A (demonic) lattice of information
- ② Alvim, Chatzikokolakis, McIver, Morgan, Smith, Palamidessi, since 2012:
A (probabilistic) ~~lattice~~ of information
(not a lattice, actually: only a (C)PO)

A (deterministic) lattice of information

Here, programs are deterministic functions from secrets to observables; and such a function induces an equivalence relation on the secrets (of producing the same observable).

Over a fixed space of secrets X , those ER's (equivalently, partitions) have a well known lattice order (of partition refinement).

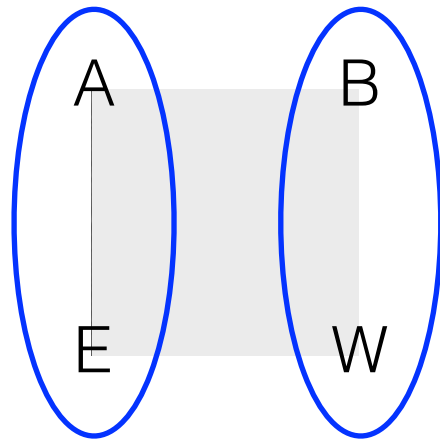
Landauer and Redmond explored the implications of this lattice for security.

... a long time ago



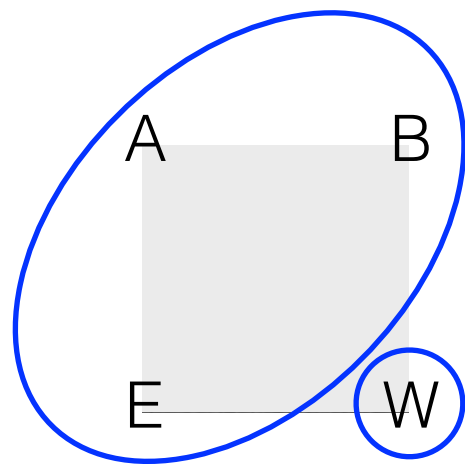
A (deterministic) **lattice** of information

This f is “vowel or consonant”.

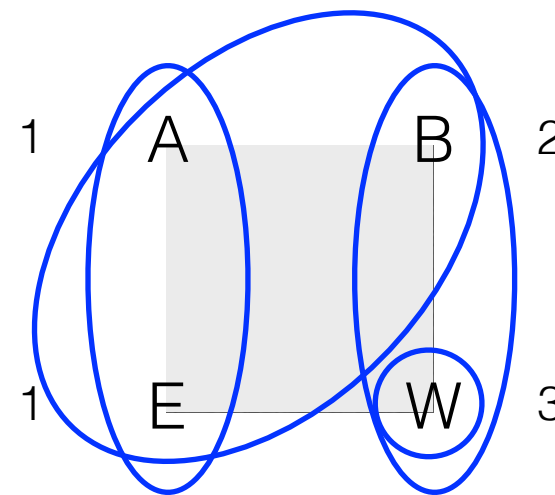


Let the deterministic observation-function be f .

cookie-cutter →



This f is “early or late”.

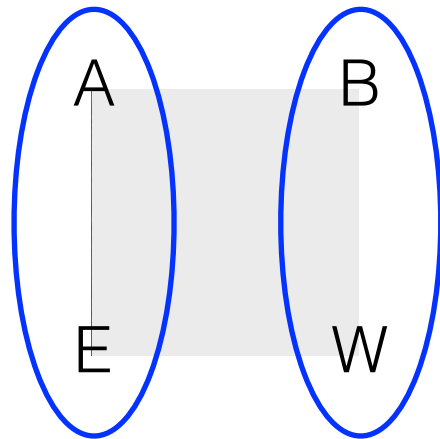


min

State space $\mathbf{X}$ is $\{A, B, E, W\}$.

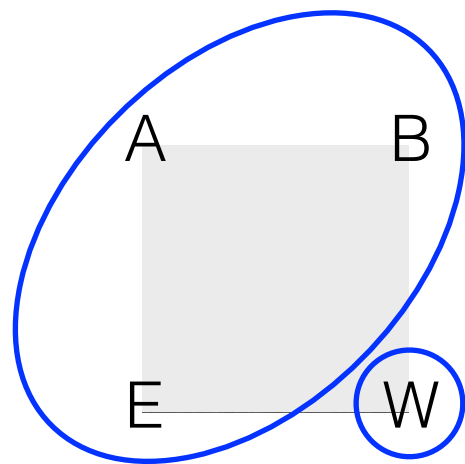
A (deterministic) **lattice** of information

This f is “vowel or consonant”.

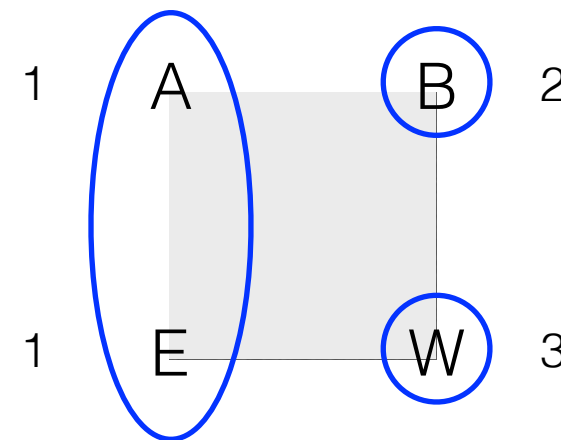


Let the deterministic observation-function be f .

cookie-cutter



This f is “early or late”.

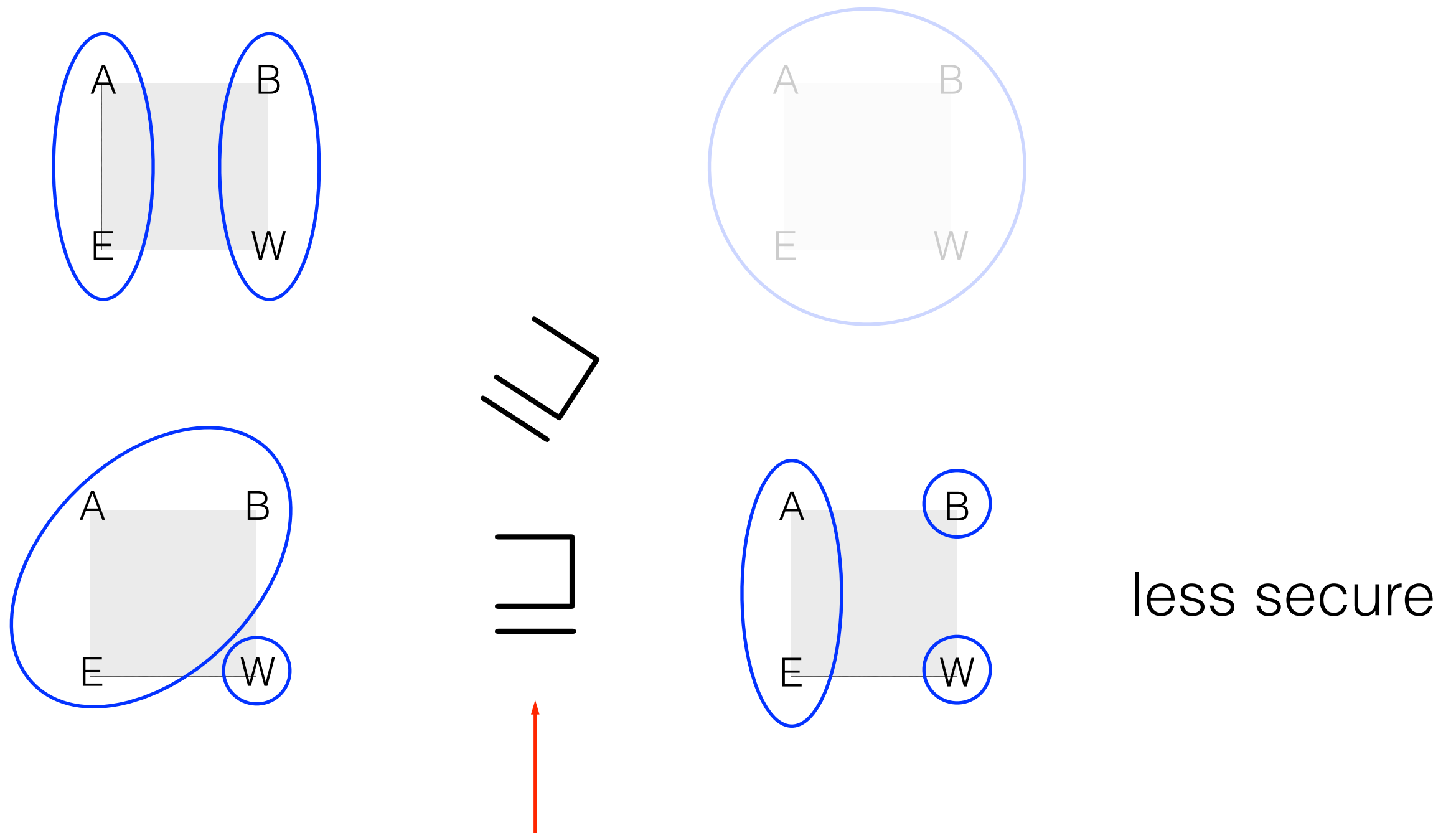


min

This f is (MOD 4).

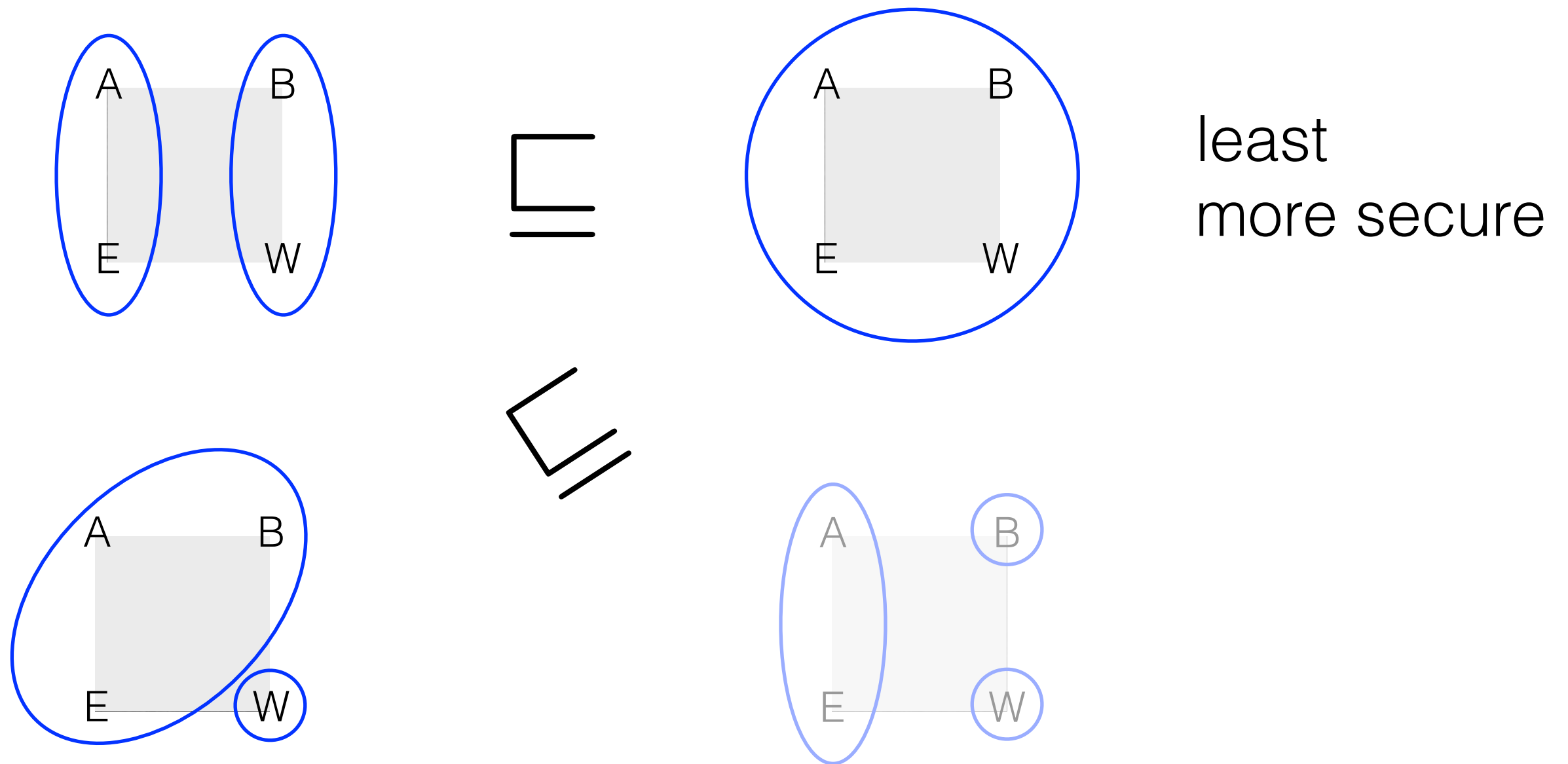
State space $\mathbf{X}$ is $\{A, B, E, W\}$.

A (deterministic) lattice of information



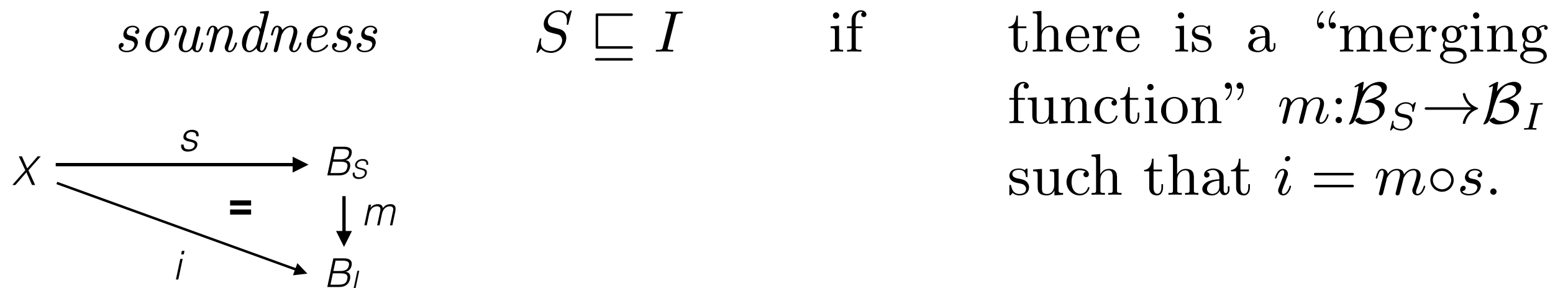
refinement order of increasing security

A (deterministic) lattice of information



Merging: soundness and completeness

Suppose we have two partitions S, I of state-space $\mathcal{X}$, and that S, I are generated by observation functions $s:\mathcal{X}\rightarrow\mathcal{B}_S$ and $i:\mathcal{X}\rightarrow\mathcal{B}_I$ respectively. Then



completeness $S \sqsubseteq I$ only if there is such an m .

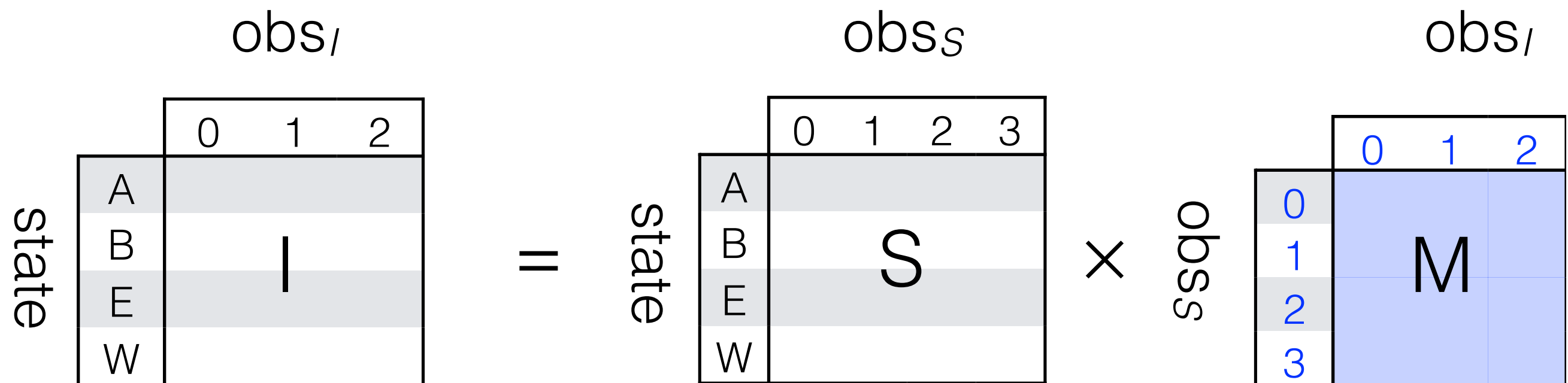
Observation as a (channel) matrix

		observation			
		0	1	2	3
state	A		✓		
	B			✓	
	E		✓		
	W				✓

partition cells

One tick per row; possibly many ticks in a column.

Refinement as matrix multiplication



Refinement is characterised by matrix post-multiplication with a “[merging matrix](#)”.

A (probabilistic) ~~lattice~~ of information

		observation			
		0	1	2	3
state	A	0.1	0.2	0.3	0.4
	B	0	0	1	0
	E	0.25	0.25	0.25	0.25
	W	0	0.5	0	0.5

$\Sigma = 1$

Each row's probabilities sum to one: a stochastic matrix.

A (probabilistic) lattice of information

outer probabilities

0.35 /4 0.95 /4 1.55 /4 1.15 /4

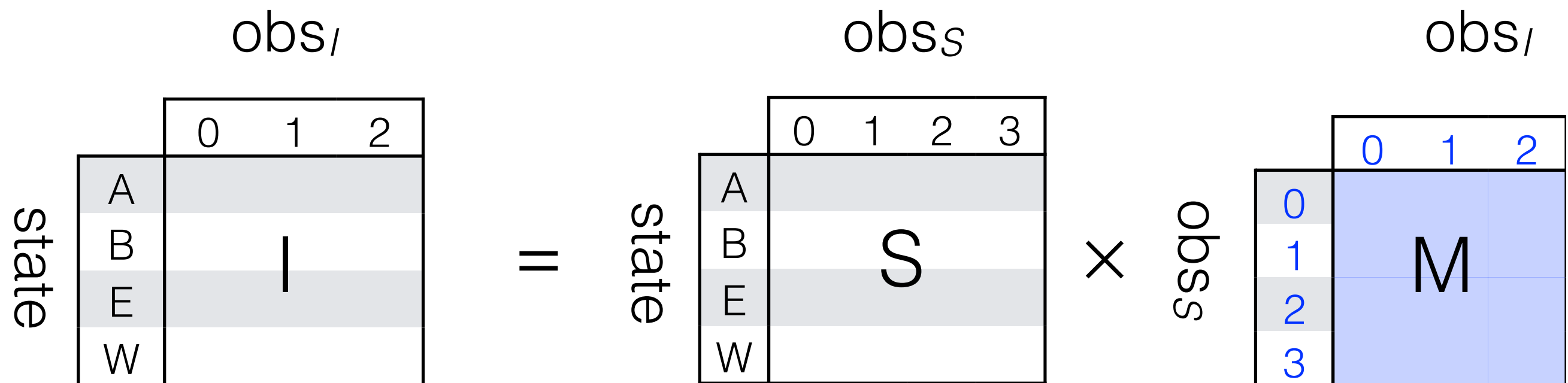
$$\Sigma = 1$$

	0.35 /4	0.95 /4	1.55 /4	1.15 /4
A	0.1/0.35	0.21	0.19	0.35
B	0	0	0.65	0
E	0.25/0.35	0.26	0.16	0.22
W	0	0.53	0	0.43

assume
incoming
state is
uniformly
distributed

inner, normalised conditional distributions

Refinement as matrix multiplication



Refinement is characterised by matrix post-multiplication with a (now quantitative) “**merging matrix**” — this time all three matrices are stochastic.

A (demonic) lattice of information

Missing, in between, is an opportunity — having nondeterministic matrices with not a single 1 per row but rather *at least one* 1 per row, and consequentially sets of sets that are **not necessarily partitions**, that can possibly overlap.

This is the *demonic lattice of information*: it generalises the deterministic case; it is generalised by the probabilistic case. Instead of cells (of a partition, disjoint), we speak of “shadows” — possibly overlapping subsets of the state space each representing knowledge that has escaped.

I will discuss highlights of this model.

A (demonic) lattice of information

Missing, in between, is an opportunity — having nondeterministic matrices with not a single 1 per row but rather *at least one* 1 per row, and consequentially sets of sets that are **not necessarily partitions**, that can possibly overlap.

This is the ***demonic lattice of information***: it generalises the deterministic case; it is generalised by the probabilistic case. Instead of cells (of a partition, disjoint), we speak of “shadows” — possibly overlapping subsets of the state space each representing knowledge that has escaped.

I will discuss highlights of this model.

A (demonic) lattice of information

The deterministic case generalises to the demonic case by allowing the partitions' **cells** to overlap.

The
nondeterministic
case

shadows

The probabilistic PO specialises to the demonic case by abstracting the conditional posteriors (**inners**) to their supports.

What's refinement here?
It's not shadow-superset.

This is surprising.

Each shadow (subset of the state space) represents a possible state of knowledge of the adversary. (The multiplicity of shadows represents externally visible demonic choice.)

Suppose there are three coins: coin A has two heads; coin C has two tails; and B has one of each.

The observation is the face that shows; the secret is “Which coin is it?”

Refinement is shadow *union*, not shadow-superset

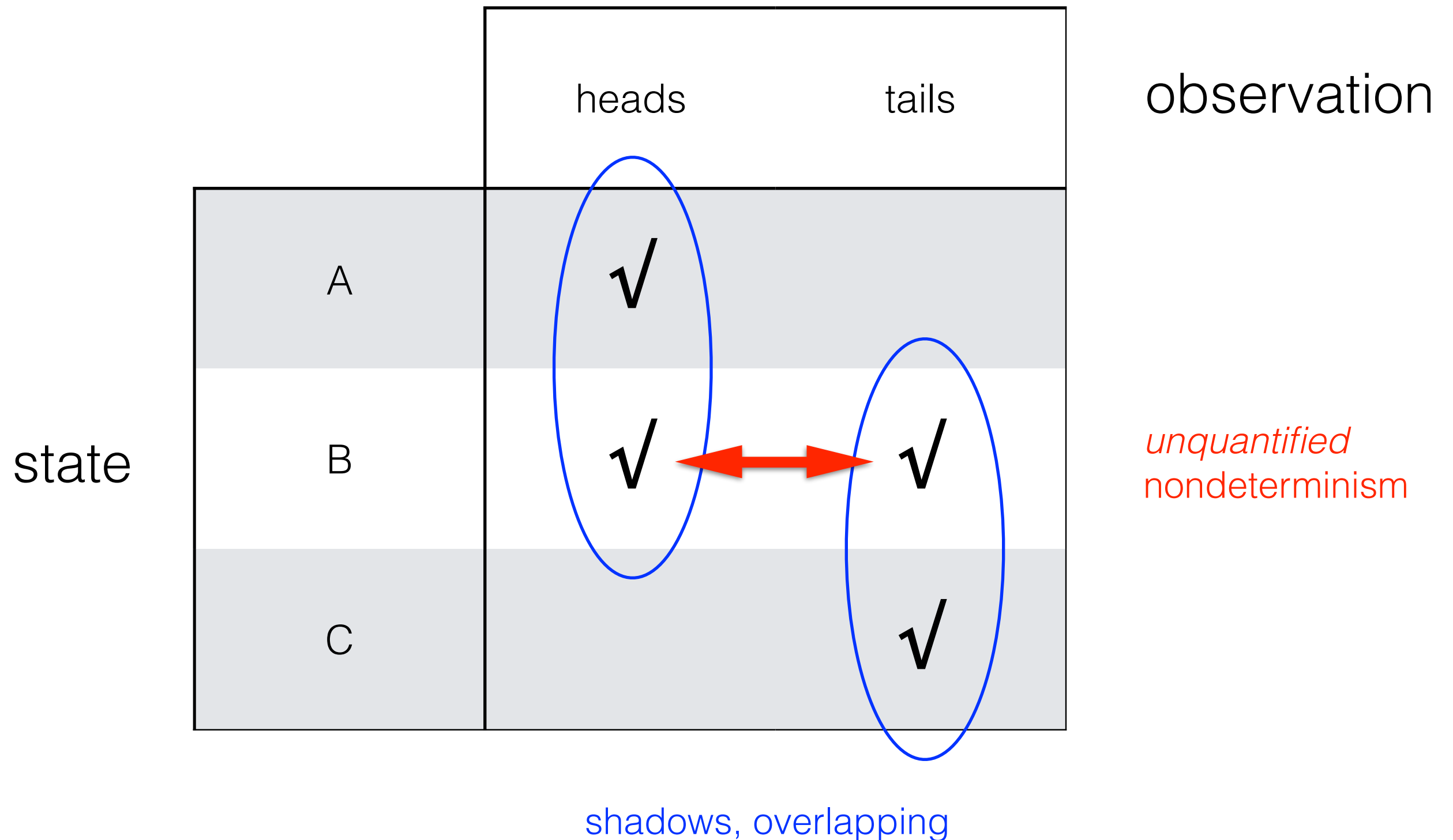
This is surprising.

Each shadow (subset of the state space) represents a possible state of knowledge of the adversary. (The multiplicity of shadows represents externally visible demonic choice.)

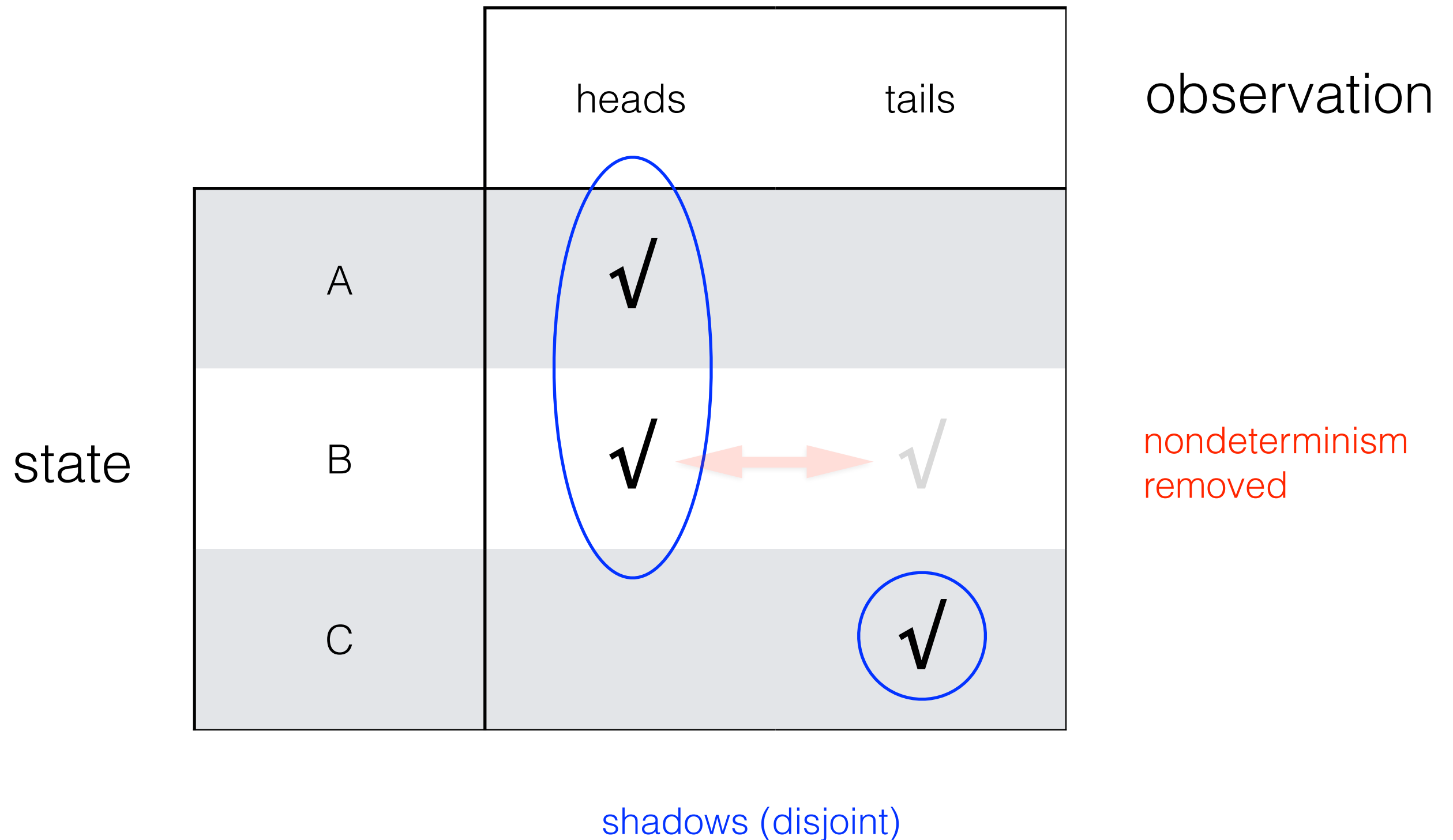
Suppose there are three coins: coin A has two heads; coin C has two tails; and B has one of each.

The observation is the face that shows; the secret is “Which coin is it?”

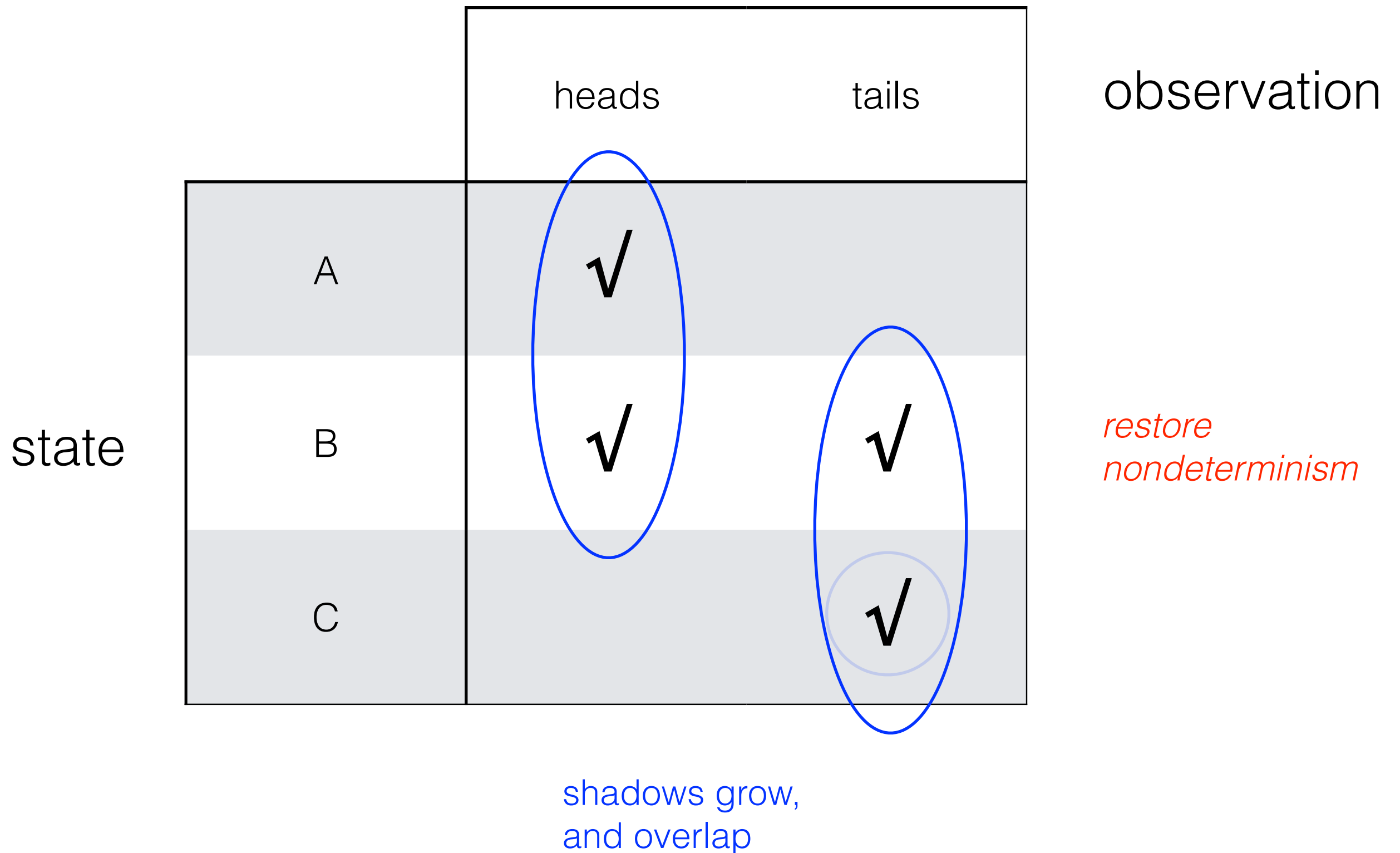
Observation as a (channel) matrix



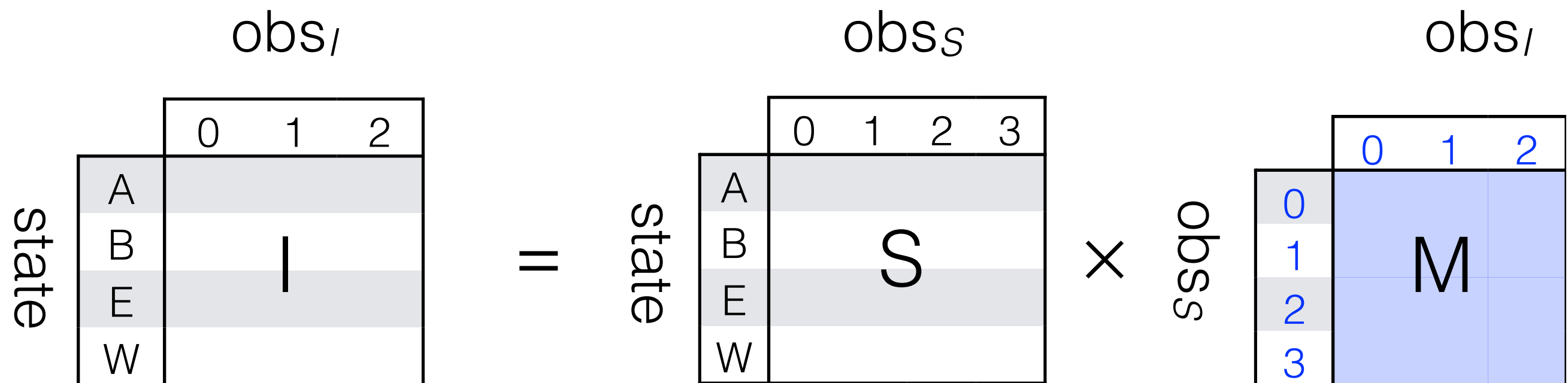
Is that system more secure than this?



Is this system more secure?



Refinement as matrix multiplication



Refinement is characterised by matrix post-multiplication with a “[merging matrix](#)”, but this time all three matrices are demonic: all 0’s or 1’s; at least one 1 in each row.

A (demonic) lattice of information: the story so far

- The state-space is some non-empty $\mathcal{X}$.
- The *semantic space* is then sets of subsets, “shadows” of $\mathcal{X}$. They do not have to cover all of $\mathcal{X}$; they do not have to be disjoint.
- The “healthiness condition” for these sets of shadows is *union closure* (not superset closure).
- Refinement is then merely reverse inclusion wrt. these healthy sets of sets (as with other powerdomains).

A tale of two families: Smyth and Jones

Mr and Mrs Smyth and Mr and Mrs Jones are next-door neighbours. Each family has a locked mailbox by the street.

Adversary Albert knows that Mrs Smyth occasionally receives money in the post, and he would like to steal it.

A tale of two families: Smyth and Jones

Because the mailboxes are locked, however, he has to steal the whole box and then break it open in his garage, at home.

Albert watches the postman every day, waiting for his chance. The observable is the mailbox; the secret is the recipient.

The shadows for each delivered letter are $\{S,s\}$ and $\{J,j\}$. When the shadow contains s , he will steal the Smyth-box. Except...

Mr Smyth is a **Mafia** boss

Except... If Albert happens to steal any of Mr Smyth's mail,

he will be **killed**.

Thus Albert dares not risk stealing the Smyth's mailbox, ever. And, as a result, **this system is secure against Albert**.

*The significance of reward/punishment for the adversary is something we have learned **only recently** from Quantitative Information Flow, i.e. from the probabilistic case: **security wrt a particular adversary depends on his circumstances**.*

Mr Smyth is a Mafia boss

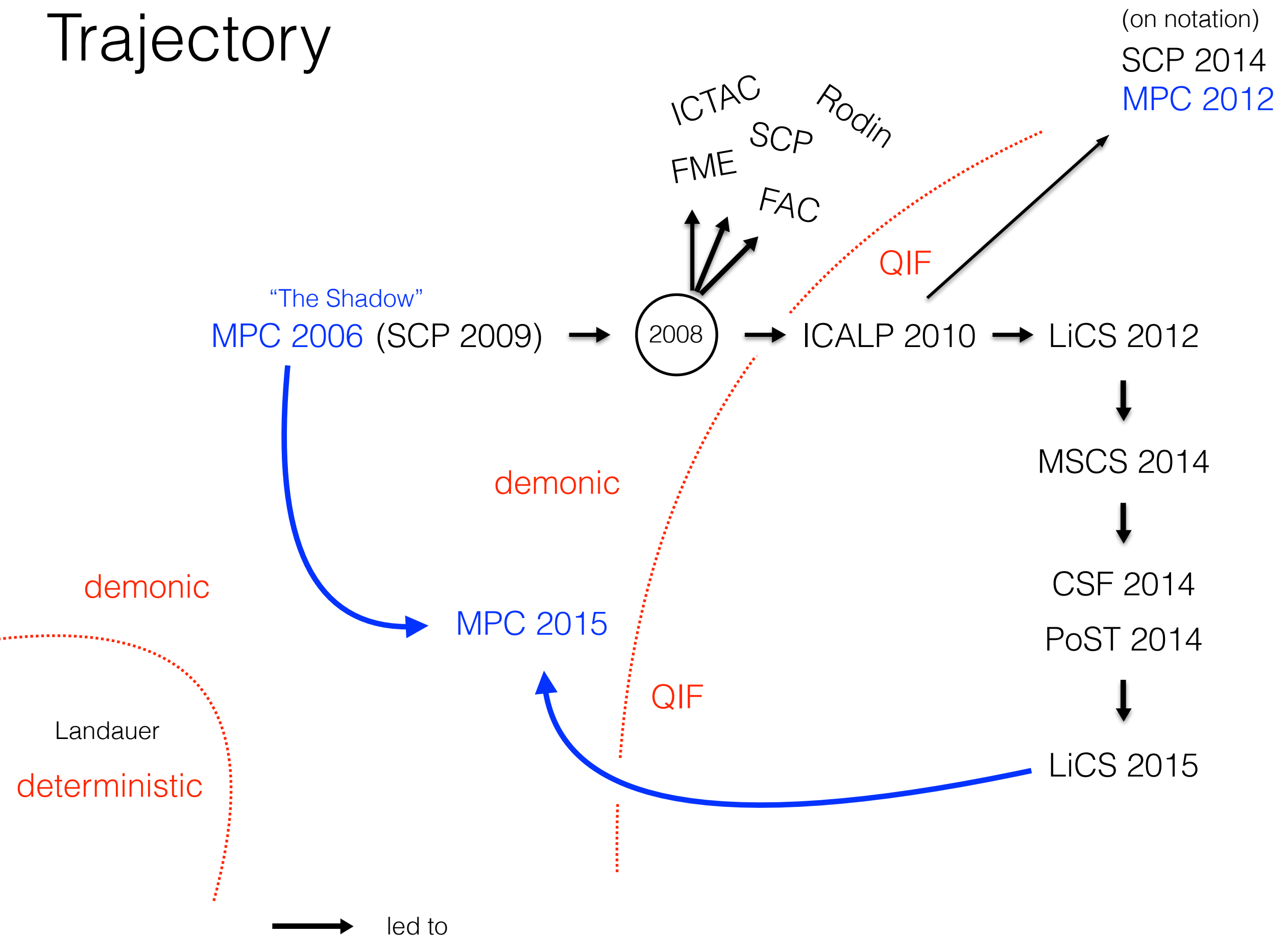
Except... If Albert happens to steal any of Mr Smyth's mail,

he will be killed.

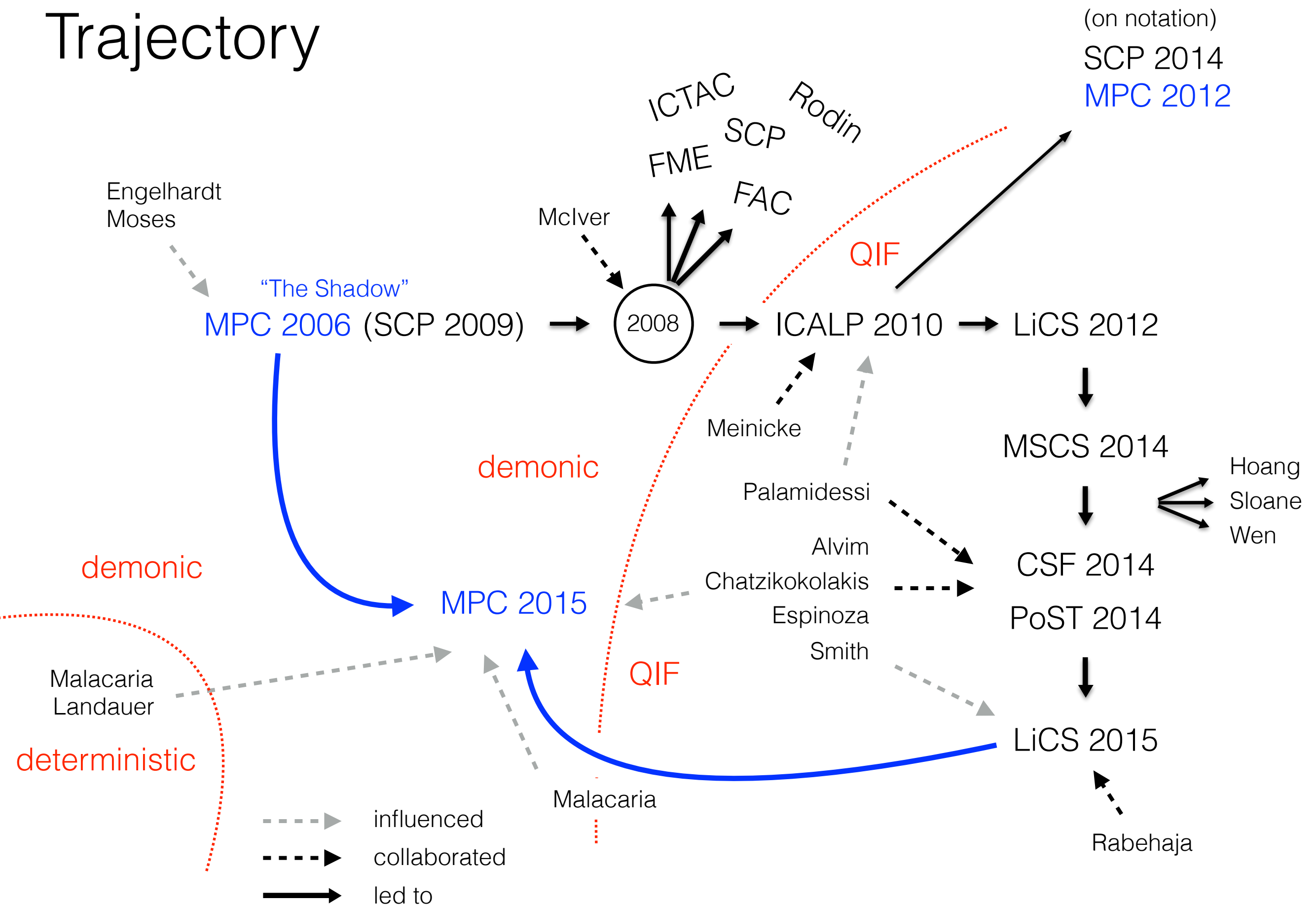
Thus Albert dares not risk stealing the Smyth's mailbox, ever. And, as a result, this system is secure against Albert.

The significance of reward/punishment for the adversary is something we have learned only recently from Quantitative Information Flow, i.e. from the probabilistic case: security wrt a particular adversary depends on his circumstances.

Trajectory



Trajectory



Mrs Smyth finds out about Mr Smyth

When Mrs Smyth discovers Mr Smyth's Mafia connections, she divorces him and resumes her maiden name: she becomes Ms Jones.

For a while, her mail continues to be delivered to her old address: the mailbox now reads

Mr Smyth and Ms Jones.

Unfortunately, the postman sometimes gets confused, and puts Ms Jones' mail in the wrong box.

The shadows are now $\{S,s\}$ and $\{s,J,j\}$, representing an increase of ignorance. Yet the system has become insecure against Albert, because he is prepared to steal the Jones's mailbox. Earlier, he was not.

Mrs Smyth finds out about Mr Smyth

When Mrs Smyth discovers Mr Smyth's Mafia connections, she divorces him and resumes her maiden name: she becomes Ms Jones.

For a while, her mail continues to be delivered to her old address: the mailbox now reads

Mr Smyth and Ms Jones.

Unfortunately, the postman sometimes gets confused, and puts Ms Jones' mail in the wrong box.

The shadows are now $\{S,s\}$ and $\{s,J,j\}$, representing an increase of ignorance. Yet the system has become insecure against Albert, because he is prepared to steal the Jones's mailbox. Earlier, he was not.

And in case you don't believe in fairytales

Let program *Spec* be $x:\in\{0,1\} \sqcap x:\in\{2,3\}$, where $(:\in)$ is “internal demonic choice”, not observable (like choosing a recipient) and $(\sqcap)$ is “external demonic choice”, observable (like choosing a mailbox).

Let program *Kludge* be $x:\in\{0,1\} \sqcap x:\in\{1,2,3\}$, so that every shadow of *Kludge* is a superset of some shadow of *Spec*.

Then *Kludge*; print $x\div 2$ might reveal that $x=1$; but *Spec*; print $x\div 2$ never can.

And in case you don't believe in fairytales

Let program *Spec* be $x:\in\{0,1\} \sqcap x:\in\{2,3\}$, where $(:\in)$ is “internal demonic choice”, not observable (like flipping a coin or rolling a die) and $(\sqcap)$ is “external demonic choice”, observable (like choosing between the coin and the die in the first place).

Let program *Kludge* be $x:\in\{0,1\} \sqcap x:\in\{1,2,3\}$, so that every shadow of *Kludge* is a superset of some shadow of *Spec*.

Then *Kludge*; print $x\div 2$ might reveal that $x=1$; but *Spec*; print $x\div 2$ never can.

That's no fairytale: it's real. It's an example of the failure of compositionality, equivalently a failure of monotonicity wrt refinement.

What is refinement, then?

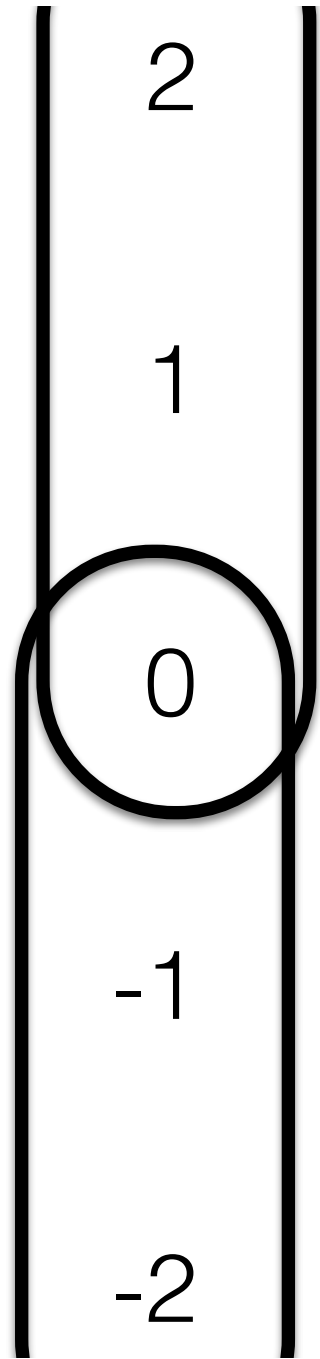
Refinement is done in two (optional) steps:

- i. (possibly) Add shadows each of which is the union of shadows that are already there; and
- ii. (possibly) Remove shadows.

With this partial order (of refinement) we have a lattice.

An easier way of looking at this (eventually) is to impose *union-closure*.

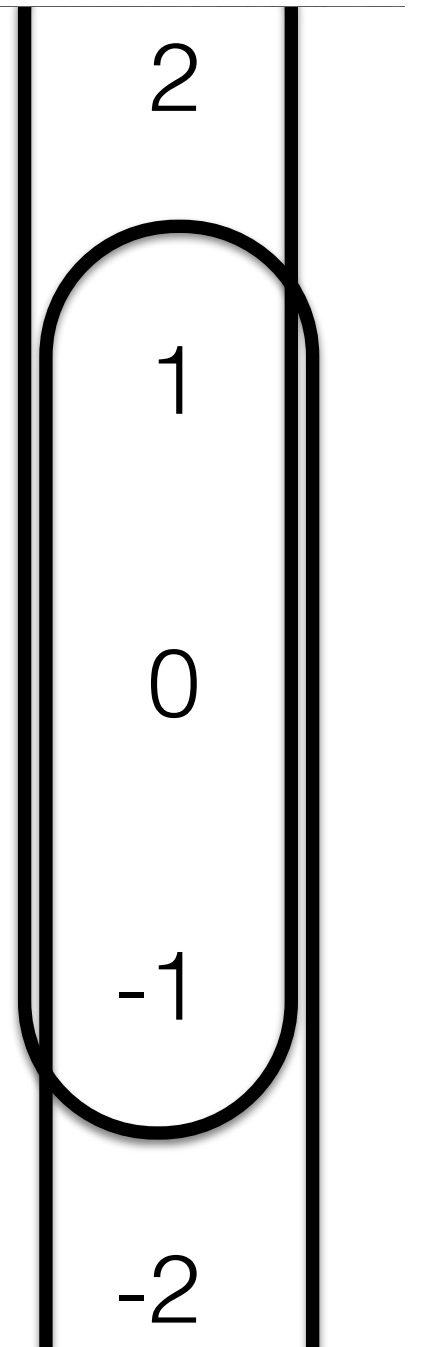
What is refinement, then?



Refinements are:

- say nothing
- say non-neg or nothing
- say non-pos or nothing
- say *nn* or *np* or nothing

But this one is not a refinement.



The demonic lattice: a very quick tour

- i. Union-closed sets of shadows can be taken the model; but what are the tests?
- ii. Given the tests, what's *soundness* and *completeness* for testing in this case?
- iii. Can the tests be expressed in terms of program variables?
- iv. Can we achieve source-level reasoning for this kind of non-interference security?

The demonic lattice: a very quick tour

Union-closed sets of shadows is the model; but what are the tests?

A test is a pair of sets A, C — and a shadow S passes such a test just if

S is a subset of C whenever S is a subset of A .

a shadow S satisfies (A, C)
just when $S \subseteq A \Rightarrow S \subseteq C$.

Any A, C are allowed. For example if A and C don't intersect, then it simply means that the pair (A, C) is satisfied only by the empty S .

“Synthesis” of the A, C -test

$$Spc \not\sqsubseteq Kld$$

$$\equiv (\exists \text{ shadow } K:Kld \cdot K \notin Spc)$$

$$\equiv \text{“} Spc \text{ union-closed”}$$

$$(\exists K:Kld \cdot K \text{ not union of any subset of } Spc)$$

$$\equiv (\exists K:Kld \cdot (\exists k:K \cdot (\forall \text{ shadow } S:SpC \cdot k \in S \Rightarrow S \not\subseteq K)))$$

$$\equiv (\exists K:Kld; k \cdot k \in K \wedge (\forall \text{ shadow } S:SpC \cdot k \in S \Rightarrow S \not\subseteq K))$$

$$\equiv \text{“define } \Phi_{A,C}(X) := X \subseteq A \Rightarrow X \subseteq C \text{”}$$

$$(\exists K:Kld; k \cdot \neg \Phi_{K, \bar{k}}(K) \wedge (\forall S:SpC \cdot \Phi_{K, \bar{k}}(S)))$$

$$\equiv (\exists K:Kld; \text{test } \Phi \cdot \neg \Phi(K) \wedge (\forall S:SpC \cdot \Phi(S)))$$

complement of {k}

“if” here requires a moment’s thought.

The demonic lattice: a very quick tour

Given the tests, what's soundness and completeness for testing in this case?

- If *Spec* is refined by *Imp*, then every test passed by all shadows of *Spec* must also be passed by all shadows of *Imp*.
- If *Kludge* does not refine *Spec*, then there is a test that all shadows of *Spec* pass, but some shadow of *Imp* fails.
- Any collection of tests characterises a union-closed set of shadows.
- Any union-closed set of shadows is characterised by some collection of tests.

“Synthesis” of the A, C -test

$$Spc \not\sqsubseteq Kld$$

$$\equiv (\exists \text{ shadow } K:Kld \cdot K \notin Spc)$$

$$\equiv \text{“} Spc \text{ union-closed”}$$

$$(\exists K:Kld \cdot K \text{ not union of any subset of } Spc)$$

$$\equiv (\exists K:Kld \cdot (\exists k:K \cdot (\forall \text{ shadow } S:SpC \cdot k \in S \Rightarrow S \not\subseteq K)))$$

$$\equiv (\exists K:Kld; k \cdot k \in K \wedge (\forall \text{ shadow } S:SpC \cdot k \in S \Rightarrow S \not\subseteq K))$$

$$\equiv \text{“define } \Phi_{A,C}(X) := X \subseteq A \Rightarrow X \subseteq C\text{”}$$

$$(\exists K:Kld; k \cdot \neg \Phi_{K, \bar{k}}(K) \wedge (\forall S:SpC \cdot \Phi_{K, \bar{k}}(S)))$$

$$\equiv (\exists K:Kld; \text{test } \Phi \cdot \neg \Phi(K) \wedge (\forall S:SpC \cdot \Phi(S)))$$

complement of {k}

“if” here requires a moment’s thought.

The demonic lattice: a very quick tour

Can the tests be expressed in terms of program variables?

When A and C are given as predicates over the program variables, say formulae Φ and Ψ rather than set expressions, then “being a subset” is implication, universally quantified over those program variables.

The demonic lattice: a very quick tour

Can the tests be expressed in terms of program variables?

And that is a a very **familiar paradigm** for those used to assertional reasoning over program texts.

The shadow's being a “subset” of some predicate Φ can be thought of as a modal formula $\mathbf{K}\Phi$, that is that “ Φ is known to hold for all states in the shadow”.

Its dual is $\mathbf{P}\Phi$, that “It is not known that Φ doesn't hold for some state in the shadow.”

Idioms and examples

We use state variables $x, y, \dots$ and a state-space $\mathcal{X} \times \mathcal{Y} \times \dots$ as appropriate.

x 's exact value is unknown $(\forall x:\mathcal{X} \cdot \mathbf{K}(x=x) \Rightarrow \mathbf{K}\perp)$

nothing is known about x $(\forall x:\mathcal{X} \cdot \mathbf{K}(x \neq x) \Rightarrow \mathbf{K}\perp)$

neither x 's nor y 's exact value is known

$(\forall x:\mathcal{X}; y:\mathcal{Y} \cdot (\mathbf{K}(x=x) \Rightarrow \mathbf{K}\perp) \wedge (\mathbf{K}(y=y) \Rightarrow \mathbf{K}\perp))$

learning x 's value does not reveal y 's value

$(\forall x:\mathcal{X}; y:\mathcal{Y} \cdot \mathbf{K}(x=x \Rightarrow y=y) \Rightarrow \mathbf{K}(x \neq x))$

learning x 's value does not reveal anything about y

$(\forall x:\mathcal{X}; y:\mathcal{Y} \cdot \mathbf{K}(x=x \Rightarrow y \neq y) \Rightarrow \mathbf{K}(x \neq x))$

We use $\top$ for true and $\perp$ for false.

Idioms and examples

We use state variables $x, y, \dots$ and a state-space $\mathcal{X} \times \mathcal{Y} \times \dots$ as appropriate.

x 's exact value is unknown $(\forall x:\mathcal{X} \cdot \mathbf{K}(x=x) \Rightarrow \mathbf{K}\perp)$
nothing is known about x $(\forall x:\mathcal{X} \cdot \mathbf{K}(x \neq x) \Rightarrow \mathbf{K}\perp)$

neither x 's nor y 's exact value is known
 $(\forall x:\mathcal{X}; y:\mathcal{Y} \cdot (\mathbf{K}(x=x) \Rightarrow \mathbf{K}\perp) \wedge (\mathbf{K}(y=y) \Rightarrow \mathbf{K}\perp))$

learning x 's value does not reveal y 's value
 $(\forall x:\mathcal{X}; y:\mathcal{Y} \cdot \mathbf{K}(x=x \Rightarrow y=y) \Rightarrow \mathbf{K}(x \neq x))$

learning x 's value does not reveal anything about y
 $(\forall x:\mathcal{X}; y:\mathcal{Y} \cdot \mathbf{K}(x=x \Rightarrow y \neq y) \Rightarrow \mathbf{K}(x \neq x))$

Idioms and examples

We use state variables $x, y, \dots$ and a state-space $\mathcal{X} \times \mathcal{Y} \times \dots$ as appropriate.

x 's exact value is unknown $(\forall x:\mathcal{X} \cdot \mathbf{K}(x=x) \Rightarrow \mathbf{K}\perp)$

nothing is known about x $(\forall x:\mathcal{X} \cdot \mathbf{K}(x \neq x) \Rightarrow \mathbf{K}\perp)$

neither x 's nor y 's exact value is known

$(\forall x:\mathcal{X}; y:\mathcal{Y} \cdot (\mathbf{K}(x=x) \Rightarrow \mathbf{K}\perp) \wedge (\mathbf{K}(y=y) \Rightarrow \mathbf{K}\perp))$

learning x 's value does not reveal y 's value

$(\forall x:\mathcal{X}; y:\mathcal{Y} \cdot \mathbf{K}(x=x \Rightarrow y=y) \Rightarrow \mathbf{K}(x \neq x))$

learning x 's value does not reveal anything about y

$(\forall x:\mathcal{X}; y:\mathcal{Y} \cdot \mathbf{K}(x=x \Rightarrow y \neq y) \Rightarrow \mathbf{K}(x \neq x))$

Idioms and examples

We use state variables $x, y, \dots$ and a state-space $\mathcal{X} \times \mathcal{Y} \times \dots$ as appropriate.

x 's exact value is unknown $(\forall x:\mathcal{X} \cdot \mathbf{K}(x=x) \Rightarrow \mathbf{K}\perp)$

nothing is known about x $(\forall x:\mathcal{X} \cdot \mathbf{K}(x \neq x) \Rightarrow \mathbf{K}\perp)$

neither x 's nor y 's exact value is known

$(\forall x:\mathcal{X}; y:\mathcal{Y} \cdot (\mathbf{K}(x=x) \Rightarrow \mathbf{K}\perp) \wedge (\mathbf{K}(y=y) \Rightarrow \mathbf{K}\perp))$

learning x 's value does not reveal y 's value

$(\forall x:\mathcal{X}; y:\mathcal{Y} \cdot \mathbf{K}(x=x \Rightarrow y=y) \Rightarrow \mathbf{K}(x \neq x))$

learning x 's value does not reveal anything about y

$(\forall x:\mathcal{X}; y:\mathcal{Y} \cdot \mathbf{K}(x=x \Rightarrow y \neq y) \Rightarrow \mathbf{K}(x \neq x))$

Idioms and examples

We use state variables $x, y, \dots$ and a state-space $\mathcal{X} \times \mathcal{Y} \times \dots$ as appropriate.

x 's exact value is unknown $(\forall x:\mathcal{X} \cdot \mathbf{K}(x=x) \Rightarrow \mathbf{K}\perp)$

nothing is known about x $(\forall x:\mathcal{X} \cdot \mathbf{K}(x \neq x) \Rightarrow \mathbf{K}\perp)$

neither x 's nor y 's exact value is known

$(\forall x:\mathcal{X}; y:\mathcal{Y} \cdot (\mathbf{K}(x=x) \Rightarrow \mathbf{K}\perp) \wedge (\mathbf{K}(y=y) \Rightarrow \mathbf{K}\perp))$

learning x 's value does not reveal y 's value

$(\forall x:\mathcal{X}; y:\mathcal{Y} \cdot \mathbf{K}(x=x \Rightarrow y=y) \Rightarrow \mathbf{K}(x \neq x))$

learning x 's value does not reveal anything about y

$(\forall x:\mathcal{X}; y:\mathcal{Y} \cdot \mathbf{K}(x=x \Rightarrow y \neq y) \Rightarrow \mathbf{K}(x \neq x))$

Do we really understand the English here?

Idioms and examples

if x is not 0, then in fact it's not 1 (either)
—(ditto)—

$$K(x \neq 0) \Rightarrow K(x \neq 1)$$
$$K\{1, 2, 3\} \Rightarrow K\{2, 3\}$$

if we know the letter was not for Mr. Smyth,
then we know it was not for Mrs. Smyth

$$\begin{aligned} & K(\text{not for Mr. Smyth}) \\ \Rightarrow & K(\text{not for Mrs. Smyth}) \\ \text{equivalently} & \\ \Rightarrow & P(\text{for Mrs. Smyth}) \\ & P(\text{for Mr. Smyth}) \end{aligned}$$

The Mr/Mrs Smyth assertion is satisfied by the Mrs. Smyth system but not by the Ms. Jones system: it shows therefore that the latter does not refine the former.

Hey, there's no idiom for that...

Only properties that are preserved by refinement can be expressed in this form, that is as conjunctions of ***K***-implications (equiv. of ***P***-implications).

And **every** property that is refinement preserving can be expressed that way. **expressively complete**

As with all formal methods, our hope here is that this rigour is limiting our vocabulary to statements that actually make sense for refinement. And any property not preserved by refinement does **not make sense**, at least not for serious program development.

And every failed refinement can be caught by a single ***K***-pair or ***P***-pair.

The demonic lattice: a very quick tour

Can we achieve source-level reasoning for this kind of non-interference security?

Yes. Although it can be intricate, at least it's mechanical.

- For `wp.assignment.Phi` replace every $\mathbf{K}(\text{body})$ in Φ with $\mathbf{K}(\text{wp.assignment.body})$
- For `external demonic choice`, take the conjunction.
- For `wp.(print exp).Phi` replace every $\mathbf{K}(\text{body})$ in Φ with $\mathbf{K}(\text{exp}=\mathbf{x} \Rightarrow \text{body})$ and put $(\forall \mathbf{x} \dots)$ on the outside.

$$\mathbf{P}\Phi = \neg \mathbf{K}(\neg \Phi) \text{ becomes } \neg \mathbf{K}(\Psi \Rightarrow \neg \Phi) = \mathbf{P}(\Psi \wedge \Phi)$$

Refinements... and not

$$Prog_1 \sqcap Prog_2 \sqsubseteq Prog_1$$

$$\begin{aligned} \text{because} \quad & wp.(Prog_1 \sqcap Prog_2).\Phi \\ &= wp.Prog_1.\Phi \wedge wp.Prog_2.\Phi \\ &\Rightarrow wp.Prog_1.\Phi \end{aligned}$$

so that e.g. we have $x:=1 \sqcap x:=2 \sqsubseteq x:=1$.

But we never have $x:\in S_1 \sqsubseteq x:\in S_2$ unless $S_1=S_2$.

Why not?

External- vs. internal demonic choice

External demonic choice represents (as usual) implementation freedom, deliberate looseness in a specification, run-time unpredictability...

and it can be “refined away” to a more constrained implementation, a tighter specification, or more predictable behaviour at run-time.

Refinement can make *EDC* smaller, but never larger.

External- vs. **internal** demonic choice

Internal demonic choice represents deliberate obfuscation. It can't be reduced, because that would reduce security as well (e.g. fewer possible passwords).

But in general it can't be increased either, because that might consequentially cause an increase in external nondeterminism. For example, sticking a “more secure” 10-character password into an 8-byte field actually decreases security: **buffer overrun**.

Refinement in general cannot change *IDC* in either direction: not smaller, not larger.

External- vs. **internal** demonic choice

Internal demonic choice represents deliberate obfuscation. It can't be reduced, because that would reduce security as well (e.g. fewer possible passwords).

But in general it can't be increased either, because that might consequentially cause an increase in external nondeterminism. For example, sticking a “more secure” 10-character password into an 8-byte field actually decreases security: **buffer overrun**.

Refinement in general cannot change **IDC** in either direction: not smaller, not larger.

How changing IDC is prevented

Suppose $\text{wp.}(x:\in S).(\mathbf{K}\Psi \Rightarrow \mathbf{K}\Phi)$ holds. That is by definition

$$\mathbf{K}(\text{wp.}(x:\in S).\Psi) \Rightarrow \mathbf{K}(\text{wp.}(x:\in S).\Phi) .$$

If S is made larger, the antecedent and the consequent both become stronger; if S is made smaller, the antecedent and the consequent both become weaker. Neither of those is implication in general.

How changing IDC is detected

Programs $x:\in S_1$ and $x:\in S_2$ have as results the single shadows S_1 and S_2 resp. What single test distinguishes them when $S_1 \neq S_2$?

If $S_2 \not\subseteq S_1$ then S_1 passes $\mathbf{K}\top \Rightarrow \mathbf{K}S_1$ but S_2 does not.
If $S_2 \subset S_1$ then S_1 passes $\mathbf{K}S_2 \Rightarrow \mathbf{K}\perp$ but S_2 does not.

Either way we have $x:\in S_1 \not\subseteq x:\in S_2$.

Examples of source-level reasoning

$$\begin{aligned} & \text{wp.}(x:=x+1).\text{“don’t know } x \text{ is even”} \\ = & \text{wp.}(x:=x+1).(\mathbf{K}(x \text{ is even}) \Rightarrow \mathbf{K}\perp) \\ = & \mathbf{K}(\text{wp.}(x:=x+1).(x \text{ is even})) \Rightarrow \mathbf{K}(\text{wp.}(x:=x+1).\perp) \\ = & \mathbf{K}(x \text{ is odd}) \Rightarrow \mathbf{K}\perp \\ = & \text{“don’t know } x \text{ is odd”} . \\ & \text{beforehand} \end{aligned}$$

Deterministic.

Examples of source-level reasoning

$$\begin{aligned} & \text{wp.}(x:=x+1 \sqcap \mathbf{skip}). \text{“don’t know } x \text{ is even”} \\ = & \quad \text{wp.}(x:=x+1).(\mathbf{K}(x \text{ is even}) \Rightarrow \mathbf{K}\perp) \\ & \wedge \quad \text{wp.}\mathbf{skip}.(\mathbf{K}(x \text{ is even}) \Rightarrow \mathbf{K}\perp) \\ = & \quad \mathbf{K}(\text{wp.}(x:=x+1).(x \text{ is even})) \Rightarrow \mathbf{K}(\text{wp.}(x:=x+1).\perp) \\ & \wedge \quad \mathbf{K}(\text{wp.}\mathbf{skip}.(x \text{ is even})) \Rightarrow \mathbf{K}(\text{wp.}\mathbf{skip}.\perp) \\ = & \quad (\mathbf{K}(x \text{ is odd}) \Rightarrow \mathbf{K}\perp) \wedge (\mathbf{K}(x \text{ is even}) \Rightarrow \mathbf{K}\perp) \\ = & \quad \mathbf{K}(x \text{ is odd}) \vee \mathbf{K}(x \text{ is even}) \Rightarrow \mathbf{K}\perp \\ = & \quad (\exists z:\{0, 1\} \cdot \mathbf{K}(x \bmod 2 = z)) \Rightarrow \mathbf{K}\perp \\ = & \quad \text{“don’t know } x\text{’s parity”} . \end{aligned}$$

External nondeterminism is visible to the adversary.

Examples of source-level reasoning

Between statements this is *EDC*; within an expression it is *IDC*.


$$\begin{aligned} & \text{wp.}(x := (x+1) \sqcap x). \text{“don’t know } x \text{ is even”} \\ = & \text{wp.}(x := (x+1) \sqcap x). (\mathbf{K}(x \text{ is even}) \Rightarrow \mathbf{K}\perp) \\ = & \mathbf{K}(\text{wp.}(x := (x+1) \sqcap x). (x \text{ is even})) \\ \Rightarrow & \mathbf{K}(\text{wp.}(x := (x+1) \sqcap x). \perp) \\ = & \mathbf{K}\perp \Rightarrow \mathbf{K}\perp \\ = & \top. \end{aligned}$$

Internal nondeterminism is hidden from the adversary.

Examples of source-level reasoning

$$\begin{aligned} & \text{wp.}(\text{print } x). \text{“don’t know } x=0\text{”} \quad \text{“}x \text{ might not be 0”} \\ = & \text{wp.}(\text{print } x). (\mathbf{K}(x=0) \Rightarrow \mathbf{K}\perp) \\ = & (\forall x \cdot \mathbf{K}(x=x \Rightarrow x=0) \Rightarrow \mathbf{K}(x=x \Rightarrow \perp)) \\ = & (\forall x \cdot \mathbf{K}(x=x \Rightarrow x=0) \Rightarrow \mathbf{K}(x \neq x)) \\ = & \mathbf{K}(x \neq 0) \wedge (\forall x \cdot \mathbf{K}(x=x \Rightarrow x=0) \Rightarrow \mathbf{K}(x \neq x)) \\ = & \mathbf{K}(x \neq 0) \wedge (\forall x \cdot \mathbf{K}(x \neq x) \Rightarrow \mathbf{K}(x \neq x)) \\ = & \mathbf{K}(x \neq 0) \quad \text{“}x \text{ can’t be 0”} \end{aligned}$$

To ensure that an attacker will think *x might not* be 0 afterwards,
you must ensure that *x is definitely not* 0 before.

Examples of source-level reasoning

$$\begin{aligned} & \text{wp.}(\text{print } x). \text{“don’t know } x\text{’s exact value”} \\ = & \text{wp.}(\text{print } x). (\forall x \cdot \mathbf{K}(x=x) \Rightarrow \mathbf{K}\perp) \\ = & (\forall x' \cdot (\forall x \cdot \mathbf{K}(x=x' \Rightarrow x=x) \Rightarrow \mathbf{K}(x=x' \Rightarrow \perp))) \\ = & (\forall x, x' \cdot \mathbf{K}(x=x' \Rightarrow x=x) \Rightarrow \mathbf{K}(x \neq x')) \\ \Rightarrow & (\forall x \cdot \mathbf{K}(x=x \Rightarrow x=x) \Rightarrow \mathbf{K}(x \neq x)) \\ = & (\forall x \cdot \mathbf{K}(x \neq x)) \\ = & \mathbf{K}(\forall x \cdot x \neq x) \quad \text{“}x \text{ can’t be any value } \mathbf{x} \text{ before.”} \\ = & \mathbf{K}\perp. \end{aligned}$$

It would indeed be a miracle if printing x ’s value did not reveal x ’s exact value.

Examples of source-level reasoning

Either x or y is printed, but we do not know which one it was.
When can we figure out y 's value even so?

$$\begin{aligned} & \text{wp.}(\text{print } x \sqcap y). \text{“do not know } y\text{'s exact value”} \\ = & \text{wp.}(\text{print } x \sqcap y). (\forall y \cdot \mathbf{K}(y=y) \Rightarrow \mathbf{K}\perp) \\ = & (\forall y, z \cdot \mathbf{K}((x=z \vee y=z) \Rightarrow y=y) \Rightarrow \mathbf{K}(x \neq z \wedge y \neq z)) , \end{aligned}$$

which is mysteriously complex. Does it have to be?

The nondeterminism above is *internal*, within the print statement: it is not observable.

Examples of source-level reasoning

$$\begin{aligned} & \text{wp.}(\text{print } x \sqcap y). \text{“do not know } y\text{’s exact value”} \\ = & \text{wp.}(\text{print } x \sqcap y). (\forall y \cdot \mathbf{K}(y=y) \Rightarrow \mathbf{K}\perp) \\ = & (\forall y, z \cdot \mathbf{K}((x=z \vee y=z) \Rightarrow y=y) \Rightarrow \mathbf{K}(x \neq z \wedge y \neq z)) \end{aligned}$$

Either x or y is printed, but we do not know which one it was.
When can we figure out y ’s value even so?

It should at least imply “we don’t know y ’s value beforehand.” It does that.

It should imply “there is no value that y can take but x cannot.” It does that, too.

It even implies “there is no value that x can take but not y and which x -value determines y uniquely.

Examples of source-level reasoning

$$\begin{aligned} & \text{wp.}(\text{print } x \sqcap y). \text{“do not know } y\text{’s exact value”} \\ = & \text{wp.}(\text{print } x \sqcap y). (\forall y \cdot \mathbf{K}(y=y) \Rightarrow \mathbf{K}\perp) \\ = & (\forall y, z \cdot \mathbf{K}((x=z \vee y=z) \Rightarrow y=y) \Rightarrow \mathbf{K}(x \neq z \wedge y \neq z)) \end{aligned}$$

It should at least imply “we don’t know y ’s value beforehand.” It does that.

It should imply “there is no value that y can take but x cannot.” It does that, too.

It even implies “there is no value that x can take but not y and which x -value determines y uniquely.

Examples of source-level reasoning

$$\begin{aligned} & \text{wp.}(\text{print } x \sqcap y). \text{“do not know } y\text{’s exact value”} \\ = & \text{wp.}(\text{print } x \sqcap y). (\forall y \cdot \mathbf{K}(y=y) \Rightarrow \mathbf{K}\perp) \\ = & (\forall y, z \cdot \mathbf{K}((x=z \vee y=z) \Rightarrow y=y) \Rightarrow \mathbf{K}(x \neq z \wedge y \neq z)) \end{aligned}$$

It should at least imply “we don’t know y ’s value beforehand.” It does that.

It should imply “there is no value that y can take but x cannot.” It does that, too.

It even implies “there is no value that x can take but not y and which x -value determines y uniquely.

Examples of source-level reasoning

$$\begin{aligned} & \text{wp.}(\text{print } x \sqcap y). \text{“do not know } y\text{’s exact value”} \\ = & \text{wp.}(\text{print } x \sqcap y). (\forall y \cdot \mathbf{K}(y=y) \Rightarrow \mathbf{K}\perp) \\ = & (\forall y, z \cdot \mathbf{K}((x=z \vee y=z) \Rightarrow y=y) \Rightarrow \mathbf{K}(x \neq z \wedge y \neq z)) \end{aligned}$$

It should at least imply “we don’t know y ’s value beforehand.” It does that.

It should imply “there is no value that y can take but x cannot.” It does that, too.

It even implies “there is no value that x can take but not y and which x -value determines y uniquely.

But all that is still not enough.

Examples of source-level reasoning

$$\begin{aligned} & \text{wp.}(\text{print } x \sqcap y). \text{“do not know } y\text{’s exact value”} \\ = & \text{wp.}(\text{print } x \sqcap y). (\forall y \cdot \mathbf{K}(y=y) \Rightarrow \mathbf{K}\perp) \\ = & (\forall y, z \cdot \mathbf{K}((x=z \vee y=z) \Rightarrow y=y) \Rightarrow \mathbf{K}(x \neq z \wedge y \neq z)) \end{aligned}$$

It should at least imply “we don’t know y ’s value beforehand.” It does that.

It should imply “there is no value that y can take but x cannot.” It does that, too.

It even implies “there is no value that x can take but not y and which x -value determines y uniquely.

What if we know nothing about x, y except that they are equal?

Examples of source-level reasoning

$$\begin{aligned} & \text{wp.}(\text{print } x \sqcap y). \text{“do not know } y\text{’s exact value”} \\ = & \text{wp.}(\text{print } x \sqcap y). (\forall y \cdot \mathbf{K}(y=y) \Rightarrow \mathbf{K}\perp) \\ = & (\forall y, z \cdot \mathbf{K}((x=z \vee y=z) \Rightarrow y=y) \Rightarrow \mathbf{K}(x \neq z \wedge y \neq z)) \\ = & (\forall z \cdot (\exists y \cdot \mathbf{K}((x=z \vee y=z) \Rightarrow y=y)) \Rightarrow \mathbf{K}(x \neq z \wedge y \neq z)) \\ = & (\forall z \cdot \mathbf{P}(x=z \vee y=z) \Rightarrow (\nexists y \cdot \mathbf{K}((x=z \vee y=z) \Rightarrow y=y))) \\ = & (\forall z \cdot \mathbf{P}(z \in \{x, y\}) \Rightarrow (\nexists y \cdot \mathbf{K}(z \in \{x, y\} \Rightarrow y=y))) , \end{aligned}$$

i.e. “For every value (z) that x or y might take, there is no single value (y) determined for y .”

This is not rocket science — it’s just fiddling with sets. But it is very tricky!
Security in general is very tricky.

$x \ y$
00
01
11
12
22
20

Conclusions

healthiness condition for the model

- **Union-closed sets** of shadows is our model for demonic non-interference security.
- It is a **lattice**: it generalises L&R's lattice; it is generalised by hyper-distributions.
- Any non-refinement can be demonstrated with a single **primitive test** "If we know A then we (also) know C."
- Conjunctions of primitive tests capture **all and only** union-closed sets of shadows.
- There is a **wp-calculus** for conjunctions of primitive tests.
- The wp-generated preconditions can be complex! Is that intrinsic to security, viz. inescapable?

Conclusions

- **Union-closed sets** of shadows is our model for demonic non-interference security.
- It is a **lattice**: it generalises L&R's lattice; it is generalised by hyper-distributions. located in a hierarchy of models
- Any non-refinement can be demonstrated with a single **primitive test** "If we know A then we (also) know C."
- Conjunctions of primitive tests capture **all and only** union-closed sets of shadows.
- There is a **wp-calculus** for conjunctions of primitive tests.
- The wp-generated preconditions can be complex! Is that intrinsic to security, viz. inescapable?

Conclusions

- **Union-closed sets** of shadows is our model for demonic non-interference security.
- It is a **lattice**: it generalises L&R's lattice; it is generalised by hyper-distributions.
- Any non-refinement can be demonstrated with a single **primitive test** "If we know A then we (also) know C." expressively complete
- Conjunctions of primitive tests capture **all and only** union-closed sets of shadows.
- There is a **wp-calculus** for conjunctions of primitive tests.
- The wp-generated preconditions can be complex! Is that intrinsic to security, viz. inescapable?

Conclusions

- **Union-closed sets** of shadows is our model for demonic non-interference security.
- It is a **lattice**: it generalises L&R's lattice; it is generalised by hyper-distributions.
- Any non-refinement can be demonstrated with a single **primitive test** “If we know A then we (also) know C.”
- Conjunctions of primitive tests capture **all and only** union-closed sets of shadows.
- There is a **wp-calculus** for conjunctions of primitive tests.
can calculate with it
- The wp-generated preconditions can be complex! Is that intrinsic to security, viz. inescapable?

Conclusions

- **Union-closed sets** of shadows is our model for demonic non-interference security.
- It is a **lattice**: it generalises L&R's lattice; it is generalised by hyper-distributions.
- Any non-refinement can be demonstrated with a single **primitive test** "If we know A then we (also) know C."
- Conjunctions of primitive tests capture **all and only** union-closed sets of shadows.
- There is a **wp-calculus** for conjunctions of primitive tests.
- The wp-generated preconditions can be complex! Is that intrinsic to security, viz. inescapable? **Hmm...**

On principles...

deterministic **Landauer and Redmond's** (1993) original work provided the motivation. Their context was simple enough that principles were not explicitly necessary.

probabilistic **Hyper-distributions** (2010–5) are sufficiently complex that principles are indispensable.

demonic **Shadows** (2006) were complex enough to benefit from explicit principles, but still simple enough that ad-hoc progress could be made without.

Using principles from 2010–5 and motivation from 1993, we have put **Shadows** at their place in the hierarchy.

