

# *Formal Methods for Probabilistic Systems*

Annabelle McIver  
Carroll Morgan

- Probabilistic temporal logic: *qTL*
- Probabilistic sequential-programming logic: *pGCL*
  - Origins of (this) program logic
  - Syntax and semantics of *pGCL*
  - Geometric interpretation (informal)
  - Metatheorems (for iteration)

# Program logic

Hoare logic of  
sequential programs:  
 $\{pre\} prog \{post\}$

- What *kind* of Logic is it?
- *Where* did it come from?
- How does it *fit in*?

Hoare logic of  
sequential programs:  
 $\{pre\} prog \{post\}$

# Program logic

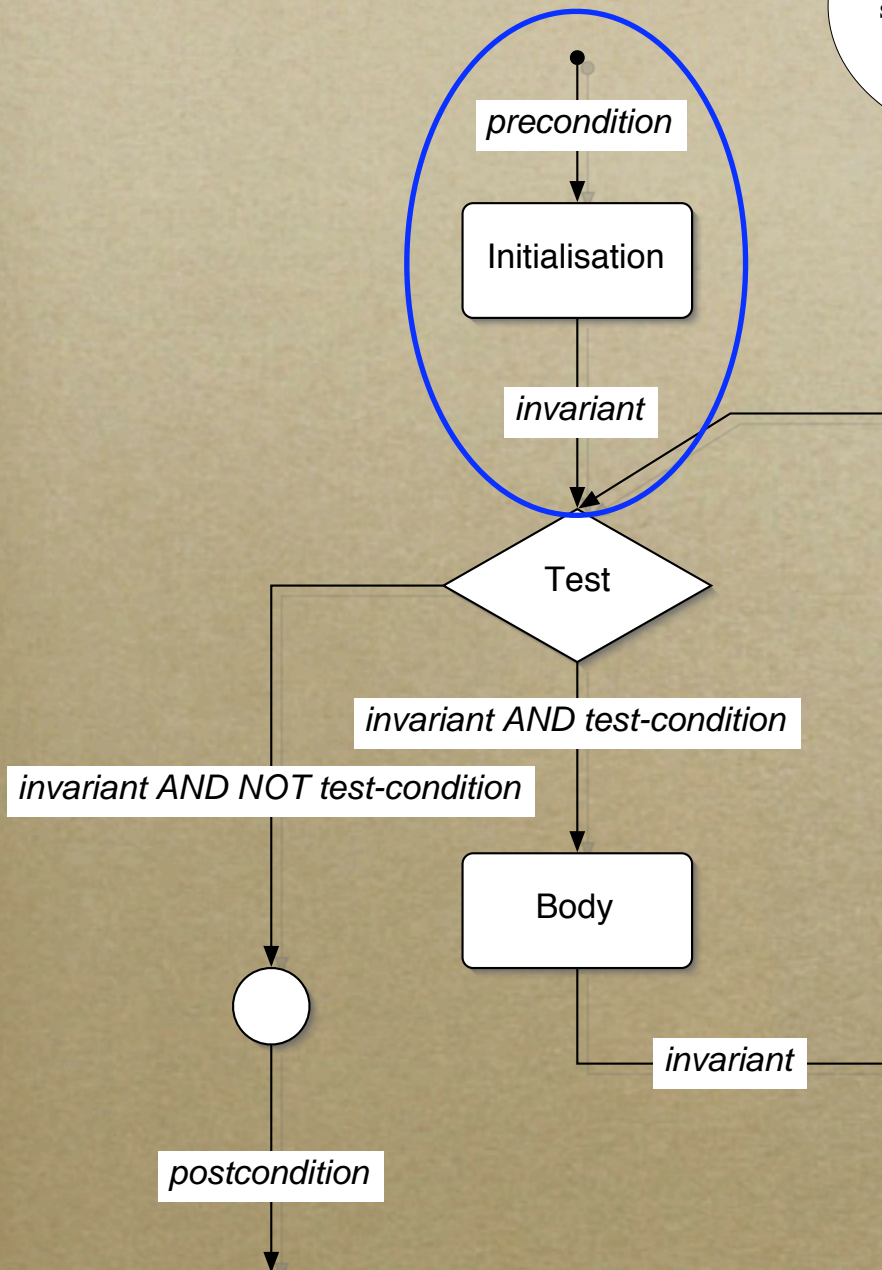
Hoare logic of  
sequential programs:  
 $\{pre\} prog \{post\}$

Floyd static annotations  
of flowchart programs

Inspired...

Floyd static annotations  
of flowchart programs

# Program logic



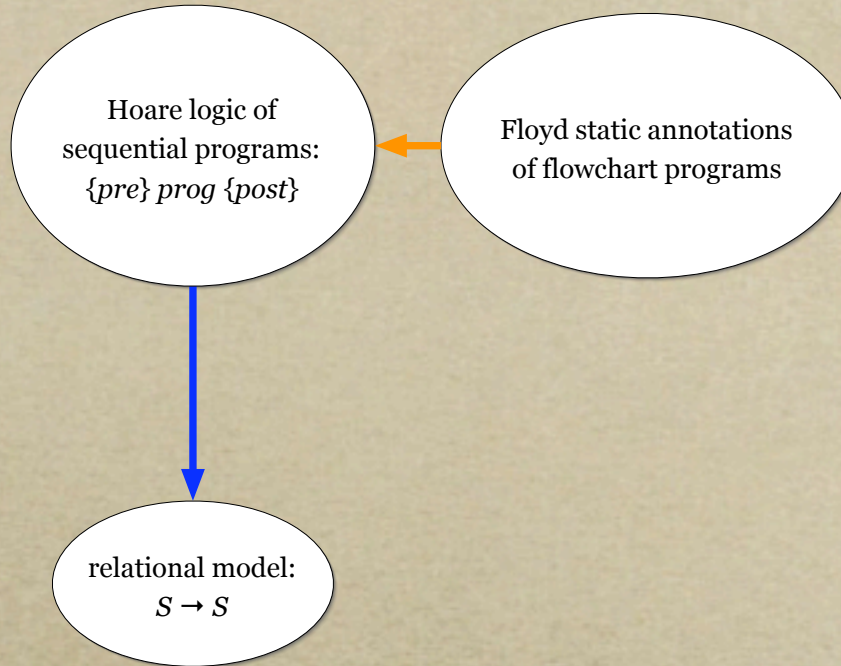
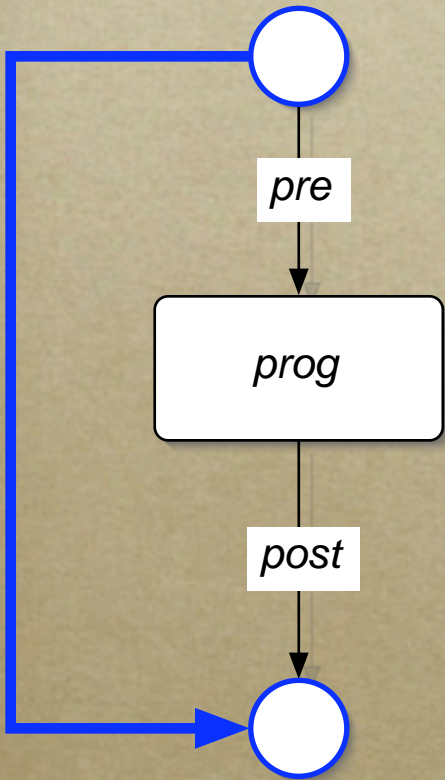
Hoare logic of sequential programs:  
 $\{pre\} prog \{post\}$

Floyd static annotations of flowchart programs

Inspired...

R.W. Floyd. Assigning meanings to programs.  
*Mathematical Aspects of Computer Science*,  
J.T. Schwartz (ed.)  
American Mathematical Society, 1967

# Program logic



Inspired...  
Is modelled by...

relational model:  
 $S \rightarrow S$

C.A.R. Hoare.  
An axiomatic basis for computer programming.  
*Comm. A.C.M.* 12(10), 1969

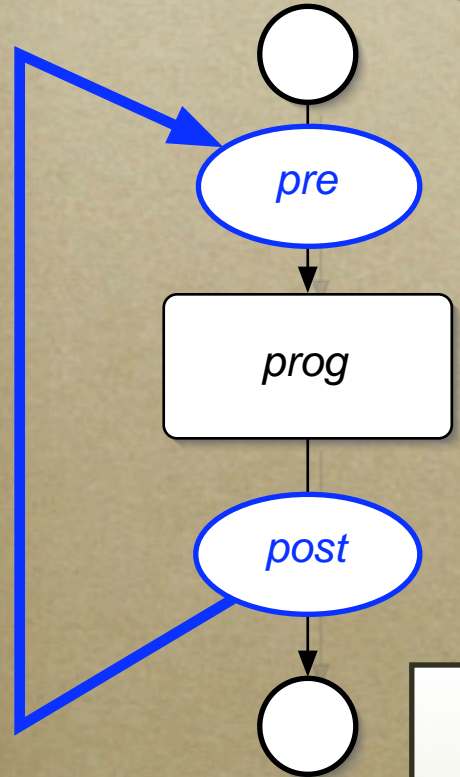
# Program 1

add non-determinism

Dijkstra logic of weakest preconditions:  
 $pre \Rightarrow wp(prog, post)$

Hoare logic of sequential programs:  
 $\{pre\} prog \{post\}$

Floyd static annotations of flowchart programs



relational model:  
 $S \rightarrow S$

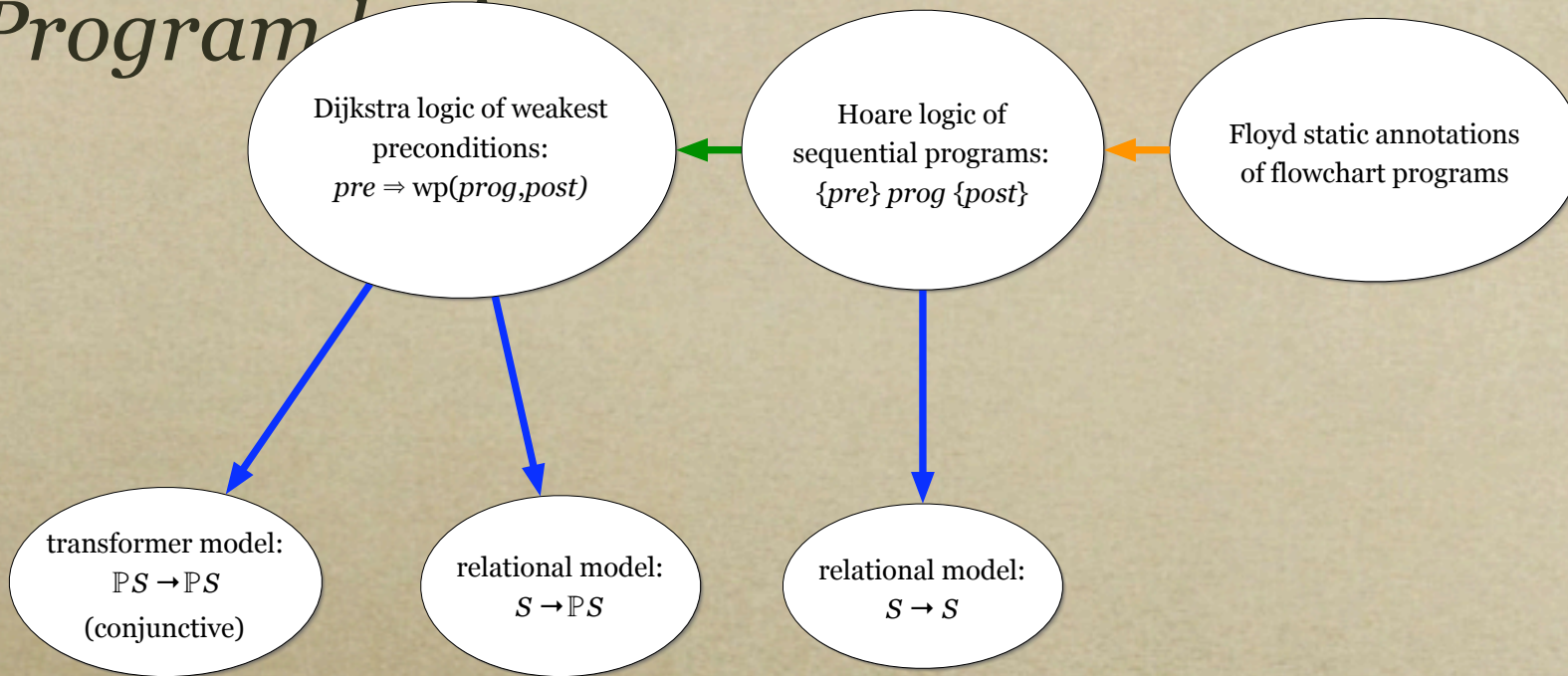
add non-determinism

Dijkstra logic of weakest preconditions:  
 $pre \Rightarrow wp(prog, post)$

Inspired...  
Is modelled by...  
Generalises to...

# Program Logic

add non-determinism



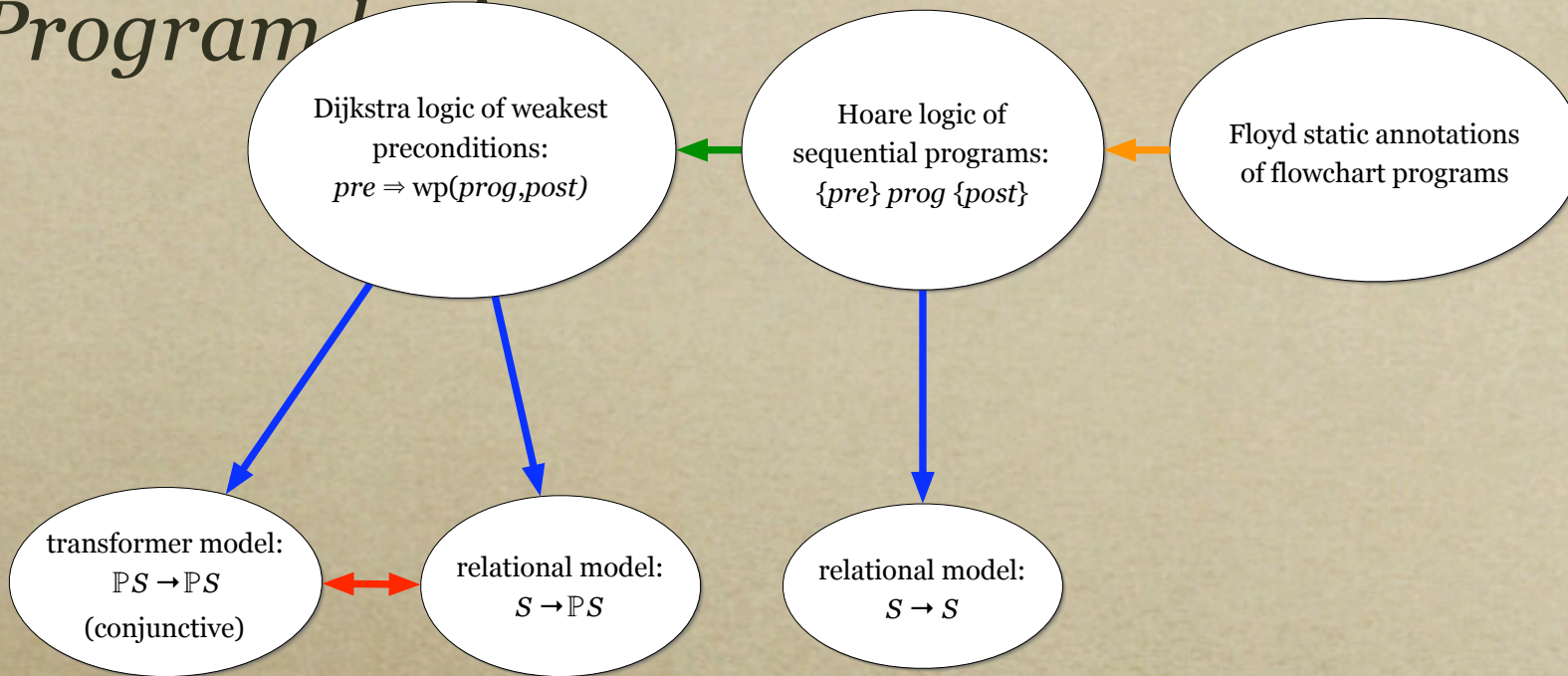
Inspired...  
Is modelled by...  
Generalises to...

transformer model:  
 $\mathbb{P}S \rightarrow \mathbb{P}S$   
(conjunctive)

relational model:  
 $S \rightarrow \mathbb{P}S$

add non-determinism

# Program Logic



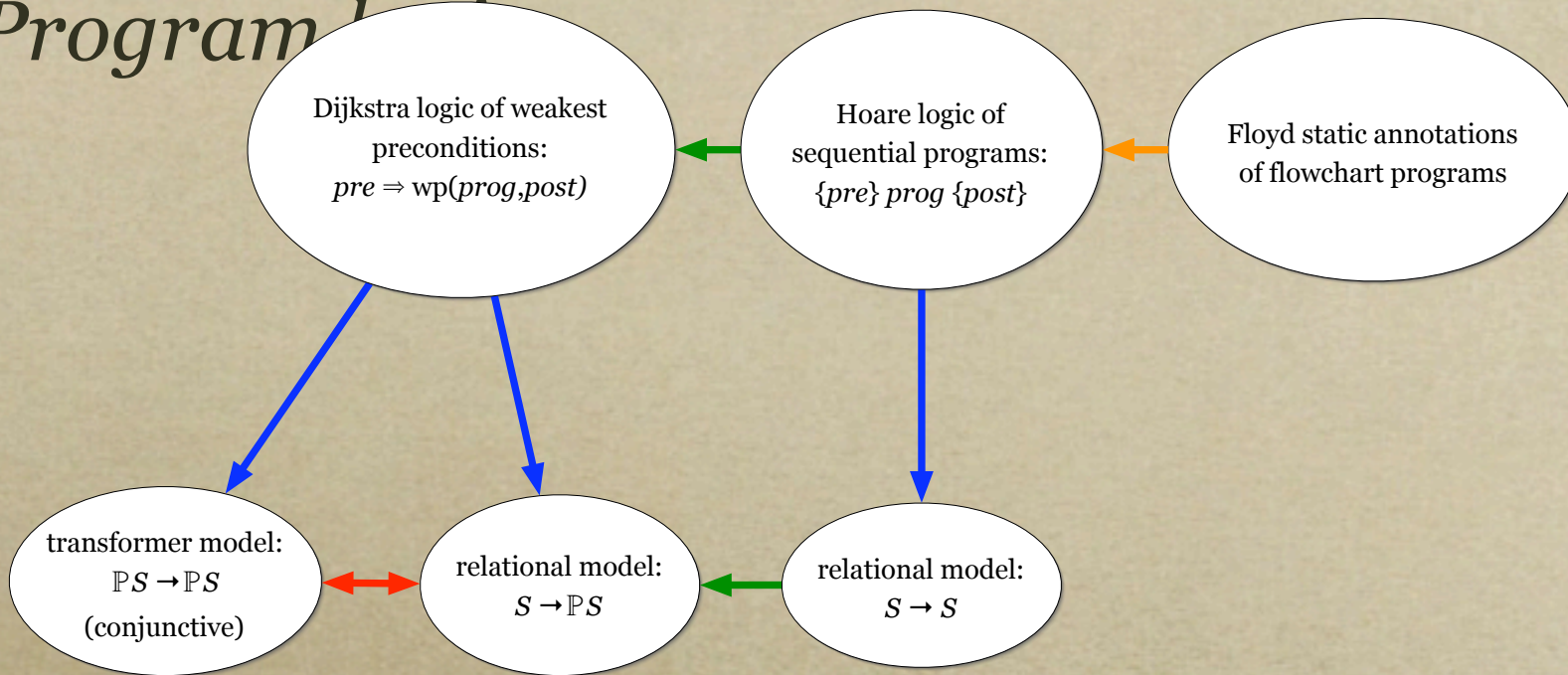
Inspired...  
Is modelled by...  
Generalises to...  
Has a Galois connection between...

transformer model:  
 $\mathbb{P}S \rightarrow \mathbb{P}S$   
(conjunctive)

relational model:  
 $S \rightarrow \mathbb{P}S$

add non-determinism

# Program Logic

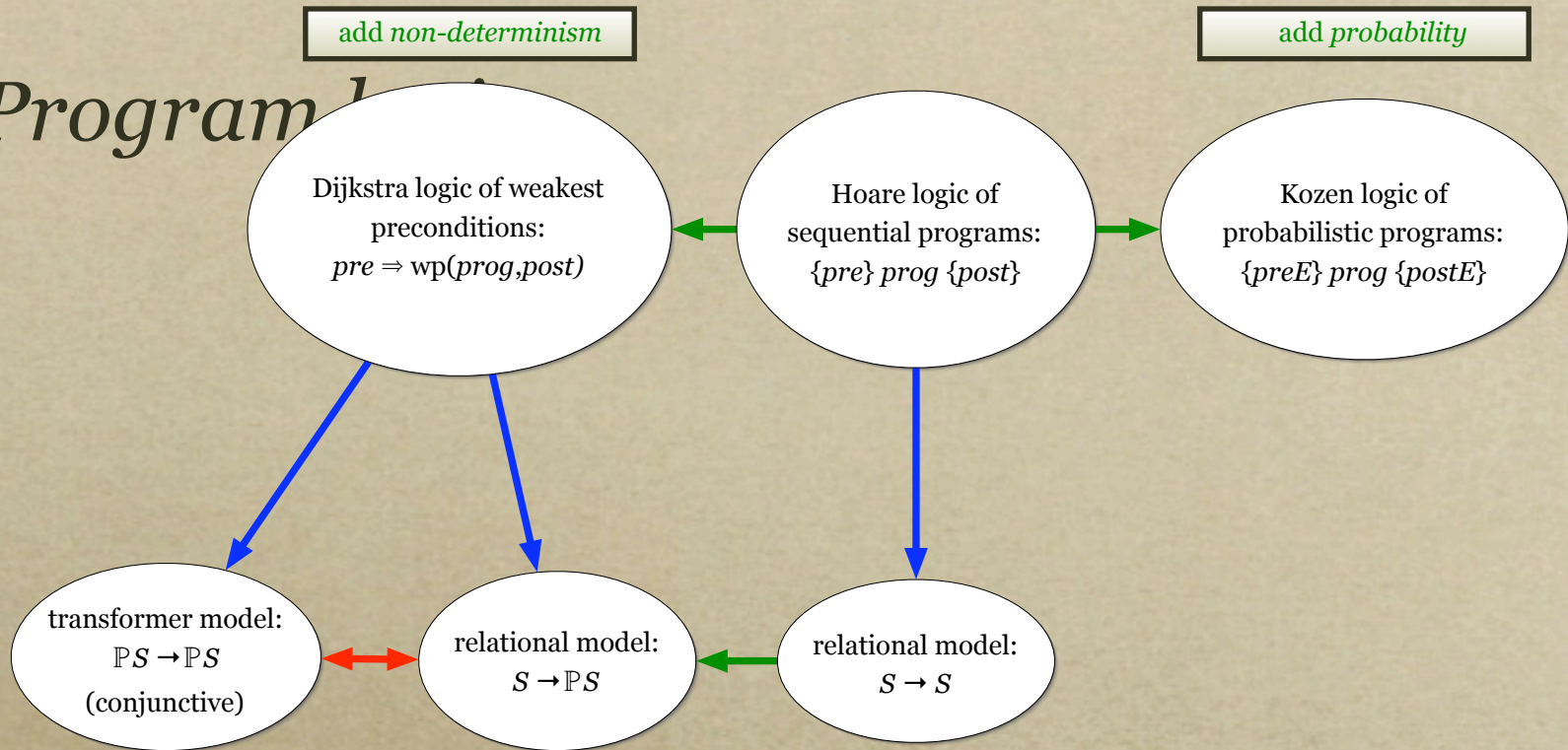


Inspired...  
Is modelled by...  
Generalises to...  
Has a Galois connection between...

transformer model:  
 $\mathbb{P}S \rightarrow \mathbb{P}S$   
(conjunctive)

relational model:  
 $S \rightarrow \mathbb{P}S$

# Program Logic



Is modelled by...  
Generalises to...  
Has a Galois connection between...

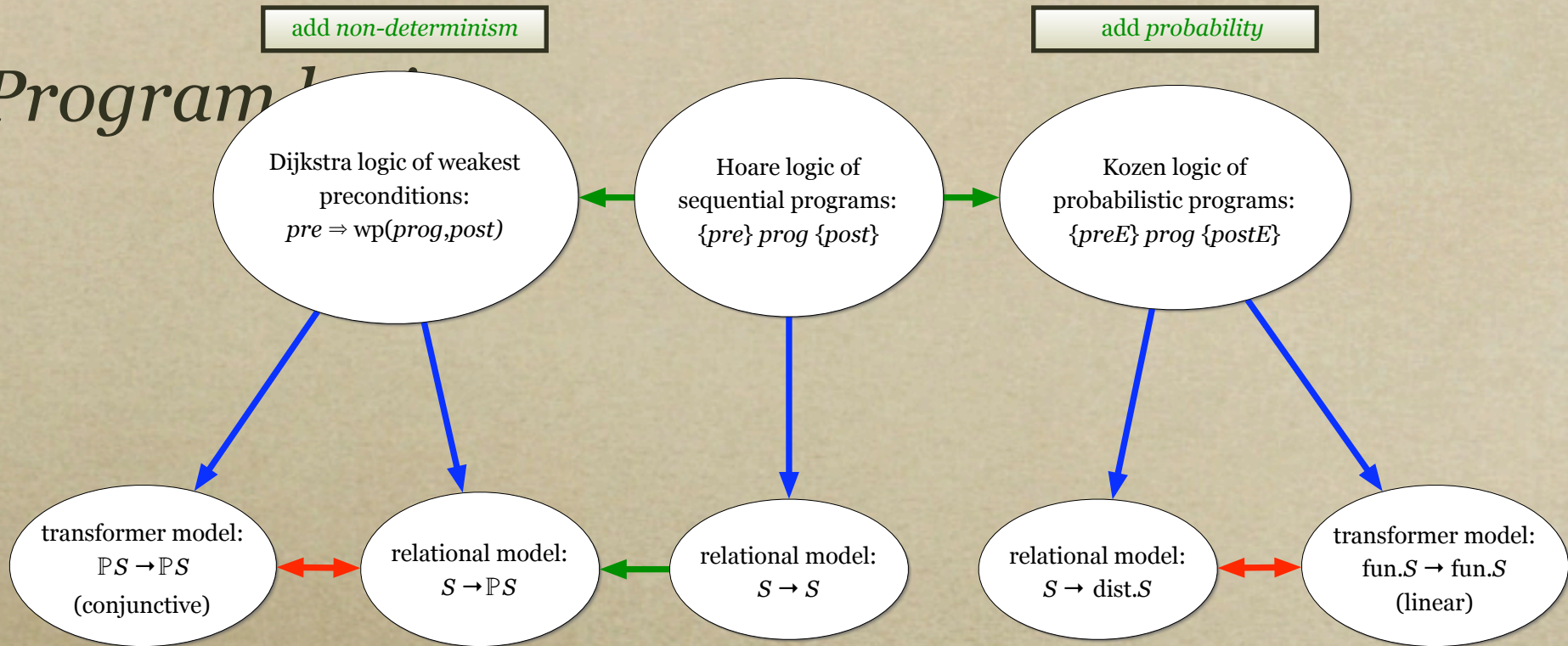
add probability

Kozen logic of probabilistic programs:  
 $\{preE\} prog \{postE\}$

D. Kozen.  
Semantics of probabilistic programs.  
*Journal of Computer and System Sciences*,  
1981

A probabilistic PDL. *Proc. 15th STOC*, ACM, 1983

# Program Logics

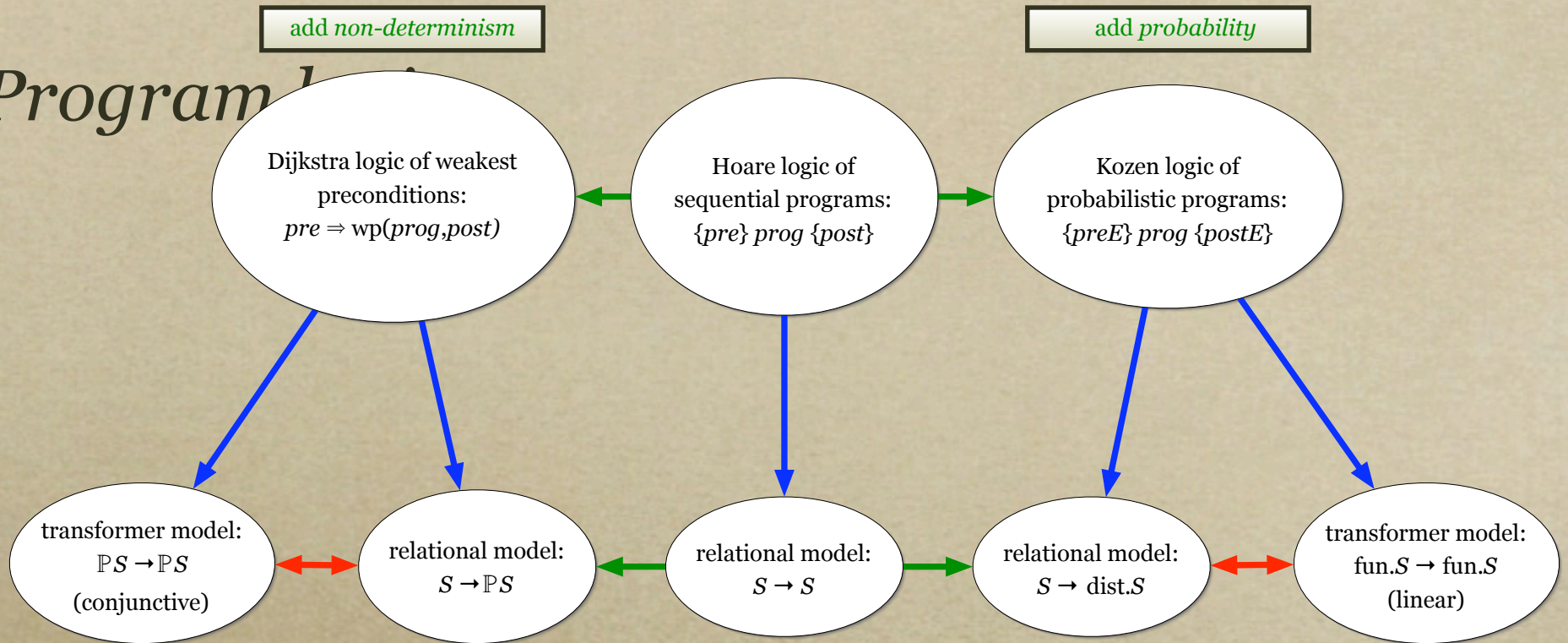


Is modelled by...  
Generalises to...  
Has a Galois connection between...

relational model:  
 $S \rightarrow \text{dist}.S$

transformer model:  
 $\text{fun}.S \rightarrow \text{fun}.S$   
(linear)

# Program Logics



Is modelled by...  
Generalises to...  
Has a Galois connection between...

relational model:  
 $S \rightarrow \text{dist}.S$

transformer model:  
 $\text{fun}.S \rightarrow \text{fun}.S$   
(linear)

---

1983–96...

Is modelled by...

Generalises to...

Has a Galois connection between...

C Jones. *Probabilistic nondeterminism*.  
Monograph *ECS-LFCS-90-105* (PhD Thesis),  
University of Edinburgh, 1989.

C Jones and G Plotkin. *A probabilistic powerdomain of evaluations*.  
Proc. 4th IEEE LICS Symp., 168-195, 1989.

Combined logic of weakest pre-expectations

$$preE \Rightarrow wp(prog, postE)$$

add non-determinism and probability

is modelled by...

Generalises to...

Has a Galois connection between...

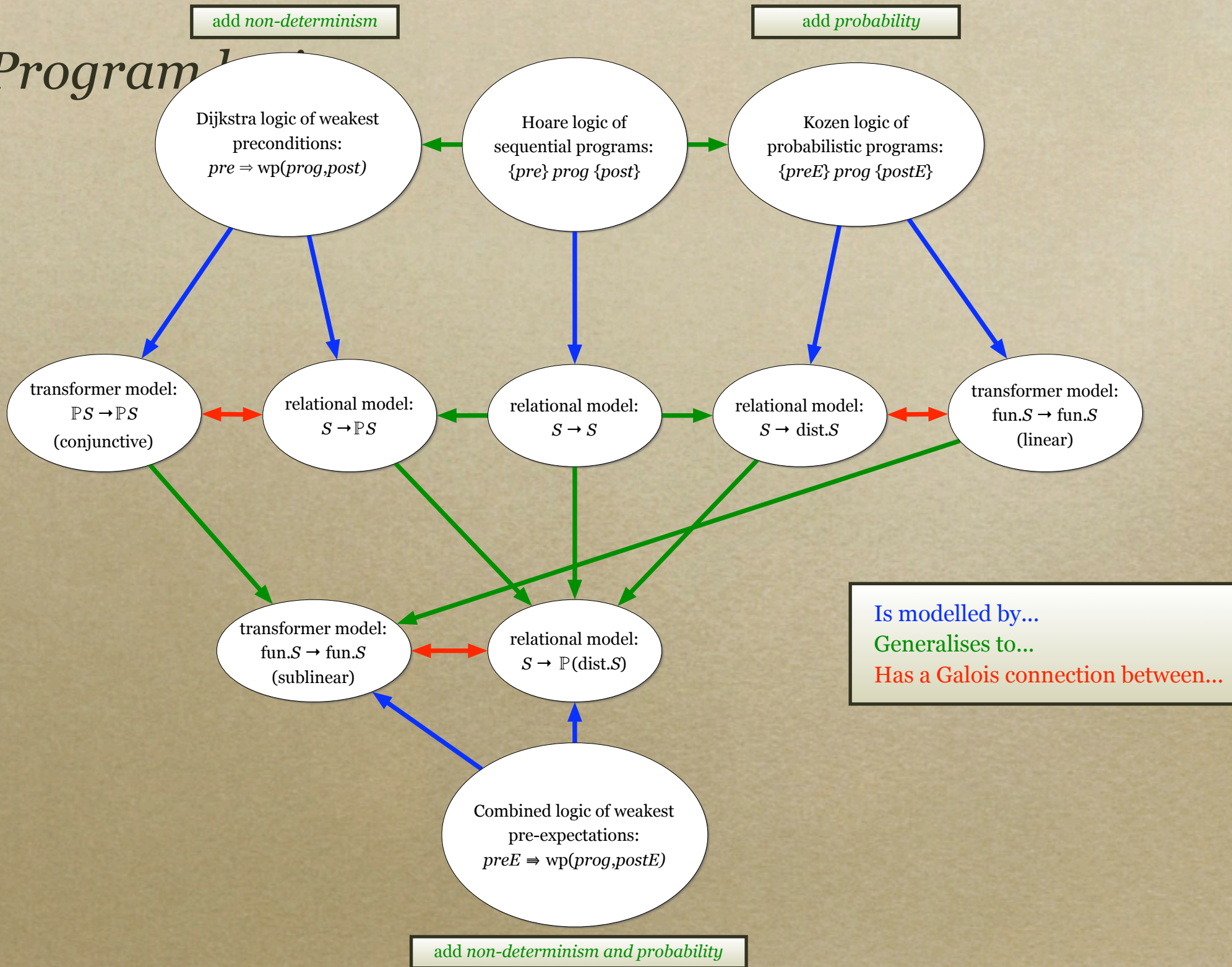
J. He, A. McIver. K. Seidel  
Probabilistic models for the  
guarded command language  
*Science of Computer Programming* 28,  
1997

Combined logic of weakest  
pre-expectations:  
 $preE \Rightarrow wp(prog, postE)$

add non-determinism and probability

C.C. Morgan, A. McIver. K. Seidel  
Probabilistic predicate transformers  
*ACM TOPLAS* 18(3)  
1996

# Program Logic

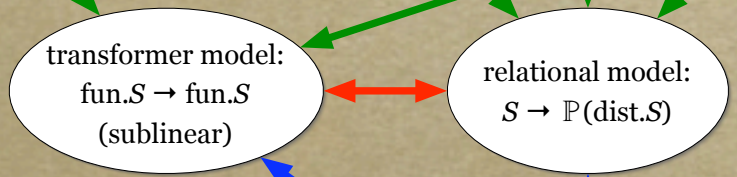
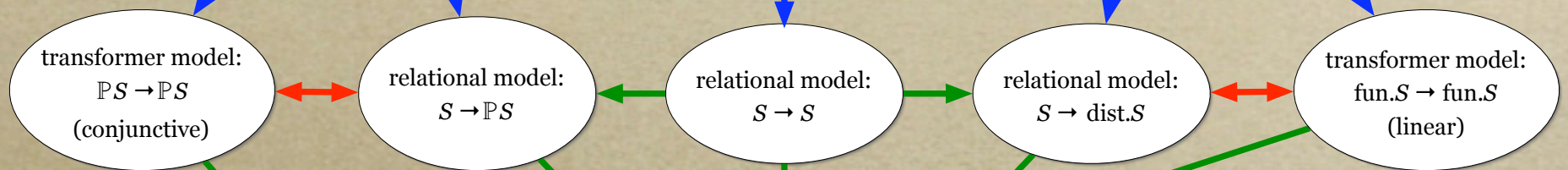


Probability,  
abstraction and  
refinement

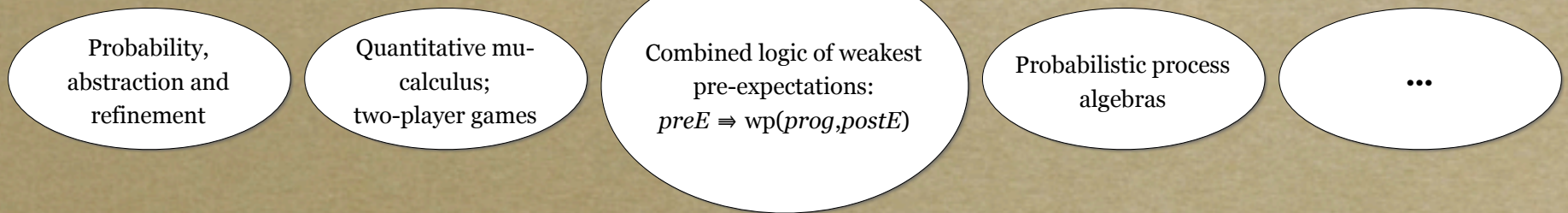
Quantitative mu-  
calculus; two-  
player games

Probabilistic process  
algebras

probabilistic programs:  
 $\{preE\} prog \{postE\}$



Is modelled by...  
Generalises to...  
Has a Galois connection between...



add non-determinism and probability

# Probabilistic-program logic: introduction

What is the **probability** that the **program**

$coin := heads \boxed{\frac{1}{2} \oplus} tails$

establishes the **postcondition**  $coin = heads$  ?

• *Probabilistic choice:  $\frac{1}{2}$  left;  $(1-\frac{1}{2})$  right.*

---

We can abbreviate “ $coin := heads \frac{1}{2} \oplus coin := tails$ ” as just

$coin := heads \frac{1}{2} \oplus tails$

because the left-hand sides “ $coin :=$ ” are the same.

# Probabilistic-program logic: introduction

What is the **probability** that the **program**

$coin := heads_{1/2} \oplus tails$

establishes the **postcondition**  $coin = heads$  ?

In the program logic we write

$$wp.(coin := heads_{1/2} \oplus tails).[coin = heads] \equiv 1/2$$

to say that the probability is just 1/2.

CC Morgan, AK McIver and K Seidel. *Probabilistic predicate transformers*.  
TOPLAS 18(3):325-353, 1996.

D Kozen. *A probabilistic PDL*. J. Comp. & Sys. Sci. 30(2):162-178, 1985.

# Probabilistic-program logic: introduction

program (fragment)

non-negative real-valued  
expression over program  
variables

$$wp.(coin := heads_{1/2} \oplus tails).[coin = heads] \equiv 1/2$$

# Interpretation

1. Assignment statements;
2. Probabilistic choice;
3. Conditionals;
4. Sequential composition;
5. Demonic choice.

— written in *pGCL*.

We will look at these in turn: what we need to know for each type of program fragment *prog* is

What is  $wp.prog.B$  for arbitrary postcondition  $B$  ?

The usual technique for setting this out is *structurally* over the syntax of the programming language.

# Interpretation: assignments

$x := E$

Assign the value  
of expression  $E$   
to the variable  $x$ .

Informal  
description.

$wp.(x := E).B$

$\equiv$

$B \langle x := E \rangle$

Definition.

*Syntactic substitution.*

$wp.(x := x+1).[x=3]$

$\equiv [x=3] \langle x := x+1 \rangle$

*definition*

$\equiv [(x+1)=3]$

*substitution*

$\equiv [x=2]$

*arithmetic*

Example.

*Why are these here?*

# Interpretation: embedding Booleans

$$\begin{aligned} & wp.(x := x+1).[x=3] \\ \equiv & [x=3] \langle x := x+1 \rangle && \text{definition} \\ \equiv & [(x+1)=3] && \text{substitution} \\ \equiv & [x=2] && \text{arithmetic} \end{aligned}$$

The *probability* that  $x := x+1$  achieves  $x=3$  is *one* if  $x=2$  initially, and *zero* otherwise.

Thus “[•]” must be an *embedding function* that takes *true* to one and *false* to zero.

# Interpretation: probabilistic choice

$prog_1 \text{ } p \oplus \text{ } prog_2$

Execute the left-hand side with probability  $p$ , otherwise execute the right-hand side (probability  $1-p$ ).

---

$$wp.(prog_1 \text{ } p \oplus \text{ } prog_2).B \equiv p \times wp.prog_1.B + (1-p) \times wp.prog_2.B$$

---

$$\begin{aligned} & wp.(c := H \text{ }_{1/2} \oplus \text{ } T).[c=H] \\ \equiv & \quad 1/2 \times wp.(c := H).[c=H] \\ & + (1-1/2) \times wp.(c := T).[c=H] \\ \equiv & \quad 1/2 \times [H=H] + 1/2 \times [T=H] \\ \equiv & \quad 1/2 \times 1 + 1/2 \times 0 \\ \equiv & \quad 1/2 . \end{aligned}$$

*definition*

*assignment*

*embedding*

*arithmetic*

# Interpretation: deterministic choice

**if**  $G$  **then**  $prog$  **fi**

If guard  $G$  holds, then execute the body  $prog$ ; otherwise do nothing.

**if**  $G$  **then**  $prog$  **else** skip **fi**

**skip**

Do nothing.

$x := x$

**if**  $G$   
**then**  $prog_1$   
**else**  $prog_2$   
**fi**

If guard  $G$  holds, then execute  $prog_1$ ; otherwise execute  $prog_2$ .

If  $G$  holds, then go left  
with probability 1,  
and vice versa.

$prog_1 [G]^\oplus prog_2$

# Interpretation: deterministic choice

$$wp.(\text{if } x \geq 1 \text{ then } x := x - 1 \text{ else } x := x + 2 \text{ fi}).[x \geq 2]$$

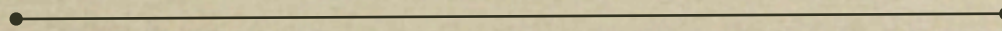
$$\begin{aligned} &\equiv wp.(x := x - 1 \ [x \geq 1] \oplus \ x := x + 2).[x \geq 2] && \text{“sugar”} \\ &\equiv [x \geq 1] \times wp.(x := x - 1).[x \geq 2] && \text{prob. choice} \\ &\quad + \ [1 - [x \geq 1]] \times wp.(x := x + 2).[x \geq 2] \\ &\equiv [x \geq 1] \times [(x - 1) \geq 2] \ + \ [x < 1] \times [(x + 2) \geq 2] && \text{assignment} \\ &\equiv [x \geq 1] \times [x \geq 3] \ + \ [x < 1] \times [x \geq 0] && \text{arithmetic} \\ &\equiv [x \geq 1] \wedge x \geq 3 \ \vee \ x < 1 \wedge x \geq 0 && \text{embedding} \\ &\equiv [x \geq 3 \ \vee \ 0 \leq x < 1]. && \text{logic} \end{aligned}$$

For a *standard* conditional, the reasoning is just “as usual”.

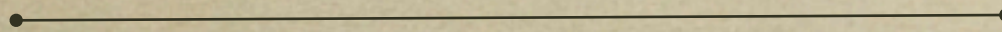
# Interpretation: sequential composition

$prog_1 ; prog_2$

Execute the first program;  
then execute the second.



$wp.(prog_1 ; prog_2).B \equiv wp.prog_1.(wp.prog_2.B)$



$wp.(c := H_{1/2} \oplus T ; d := H_{1/2} \oplus T).[c=d]$

$\equiv wp.(c := H_{1/2} \oplus T). (wp.(d := H_{1/2} \oplus T).[c=d])$  *definition*

$\equiv wp.(c := H_{1/2} \oplus T). ($  *prob. choice; assignment*

$1/2 \times [c=H] + 1/2 \times [c=T]$

)

$\equiv 1/2 \times (1/2 \times [H=H] + 1/2 \times [H=T])$  *prob. choice; assignment*

$+ 1/2 \times (1/2 \times [T=H] + 1/2 \times [T=T])$

$\equiv 1/4 + 1/4$

*embedding*

$\equiv 1/2 .$

*arithmetic*

# Interpretation: sequential composition

$prog_1 ; prog_2$

Execute the first program;  
then execute the second.

---

$$wp.(prog_1 ; prog_2).B \equiv wp.prog_1.(wp.prog_2.B)$$

---

$$wp.(c := H_{1/2} \oplus T ; d := H_{1/2} \oplus T).[c=d]$$

$$\equiv wp.(c := H_{1/2} \oplus T).(wp.(d := H_{1/2} \oplus T).[c=d])$$

*definition*

$$\equiv wp.(c := H_{1/2} \oplus T).($$

*prob. choice; assignment*

$$1/2 \times [c=H] + 1/2 \times [c=T]$$

)

$$\equiv 1/2 \times (1/2 \times [H=H] + 1/2 \times [H=T])$$

*prob. choice; assignment*

$$+ 1/2 \times (1/2 \times [T=H] + 1/2 \times [T=T])$$

$$\equiv 1/4 + 1/4$$

*embedding*

$$\equiv 1/2 .$$

*arithmetic*

## *Interpretation: a proper extension*

$$wp.(c := H \oplus T).(1/2 \times [c=H] + 1/2 \times [c=T])$$

$$\equiv \begin{aligned} & 1/2 \times (1/2 \times [H=H] + 1/2 \times [H=T]) \\ + & 1/2 \times (1/2 \times [T=H] + 1/2 \times [T=T]) \end{aligned}$$

$$\equiv 1/2 .$$

The *expected value* of the function  $1/2 \times [c=H] + 1/2 \times [c=T]$  over the distribution of states produced by the program is  $1/2$  .

As a special case (from elementary probability theory) we know that the expected value of the function  $[pred]$ , for some Boolean  $pred$ , is just the probability that  $pred$  holds.

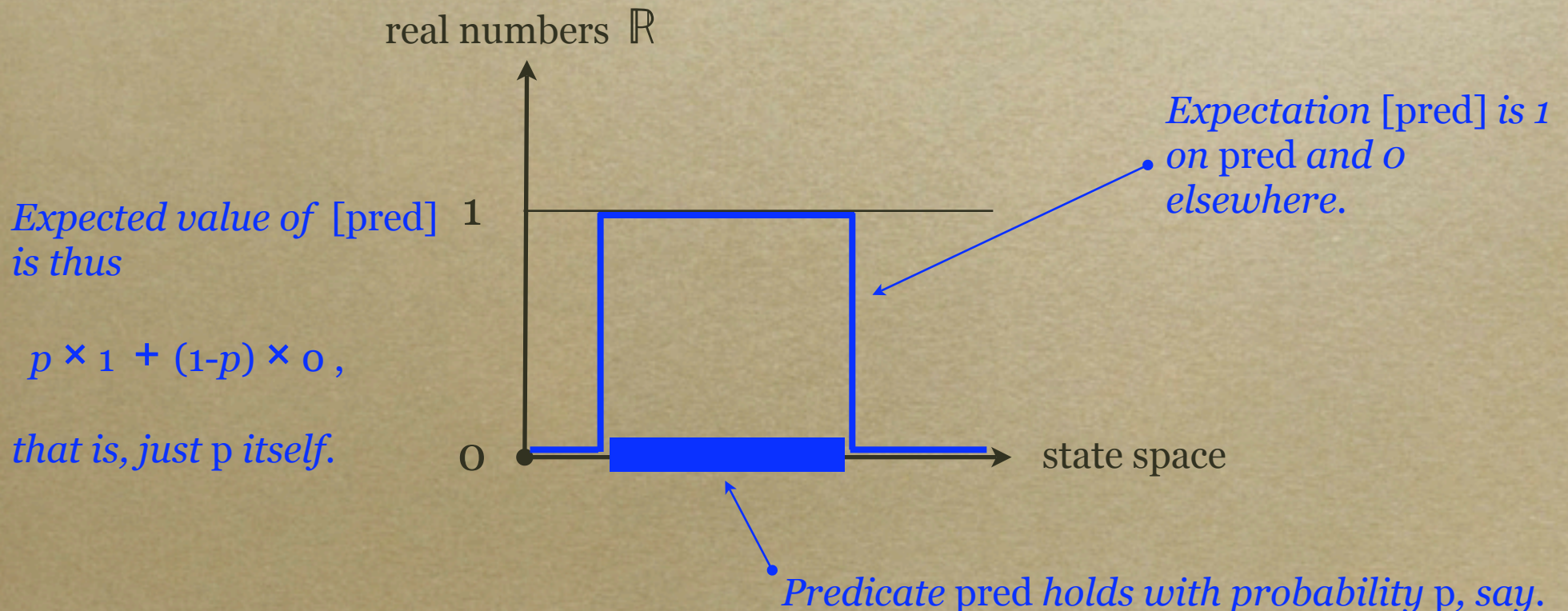
That's why  $wp.prog.[pred]$  gives the probability that  $pred$  is achieved by  $prog$ . But, as we see above, we can be much more general if we wish.

# Interpretation: a proper extension

The expression  $wp.prog.B$  gives, as a function of the initial state, the *expected value* of the “post-expectation”  $B$  over the distribution of final states that  $prog$  will produce from there.

We call it the *greatest pre-expectation* of  $prog$  with respect to  $B$ . When  $prog$  and  $B$  are standard (i.e. non-probabilistic), it is the same as the *weakest precondition*... except that it is 0/1-valued rather than Boolean.

As a “hybrid”, we have that  $wp.prog.[pred]$  is the probability that  $pred$  will be achieved.



# *Interpretation: a conservative extension*

We note that the standard logic can be embedded in the probabilistic logic simply by converting all Booleans *false, true* to the integers 0,1 (a technique familiar to C programmers). The probabilistic *wp*-logic (greatest pre-expectations) extends the standard *wp*-logic (weakest preconditions) *conservatively* in this sense:

If we restrict ourselves to standard programs (*i.e.* do not use the probabilistic choice operator), then the theorems for those programs are exactly the same as before.

Mathematically this is expressed as follows:

For all standard programs *prog*, and Boolean postconditions *post*, we have

$$[wp.prog.post] \equiv wp.prog.[post] ,$$

where on the left the *wp* is weakest precondition, and on the right it is greatest pre-expectation.

$$\begin{aligned}
wp.\mathbf{abort}.postE &:= 0 \\
wp.\mathbf{skip}.postE &:= postE \\
wp.(x := expr).postE &:= postE \langle x \mapsto expr \rangle \\
wp.(prog; prog').postE &:= wp.prog.(wp.prog'.postE) \\
wp.(prog \sqcap prog').postE &:= wp.prog.postE \min wp.prog'.postE \\
wp.(prog_p \oplus prog').postE &:= p * wp.prog.postE + \bar{p} * wp.prog'.postE
\end{aligned}$$

Recall that  $\bar{p}$  is the complement of  $p$ .

The expression on the right gives the *greatest pre-expectation* of  $postE$  with respect to each  $pGCL$  construct, where  $postE$  is an expression of type  $\mathbb{E}S$  over the variables in state space  $S$ . (For historical reasons we continue to write  $wp$  instead of  $gp$ .)

In the case of recursion, however, we cannot give a purely syntactic definition. Instead we say that

$$(\mathbf{mu} \ xxx \bullet \mathcal{C}) \quad := \quad \text{least fixed-point of the function } cntx: \mathbb{T}S \rightarrow \mathbb{T}S \\
\text{defined so that } cntx.(wp.xxx) = wp.\mathcal{C}. \quad ^{40}$$

Figure 1.5.3. PROBABILISTIC  $wp$ -SEMANTICS OF  $pGCL$

# Interpretation: demonic choice

$prog_1 \sqcap prog_2$

Execute the left-hand side — or  
maybe execute the right-hand side.  
*Whatever...*

---

$$wp.(prog_1 \sqcap prog_2).B \equiv wp.prog_1.B \mathbf{min} wp.prog_2.B$$

---

$$\begin{aligned} & wp.(c := H \sqcap c := T).[c=H] \\ \equiv & wp.(c := H).[c=H] \mathbf{min} wp.(c := T).[c=H] \\ \equiv & [H=H] \mathbf{min} [T=H] \\ \equiv & 1 \mathbf{min} 0 \\ \equiv & 0 . \end{aligned}$$

*definition*

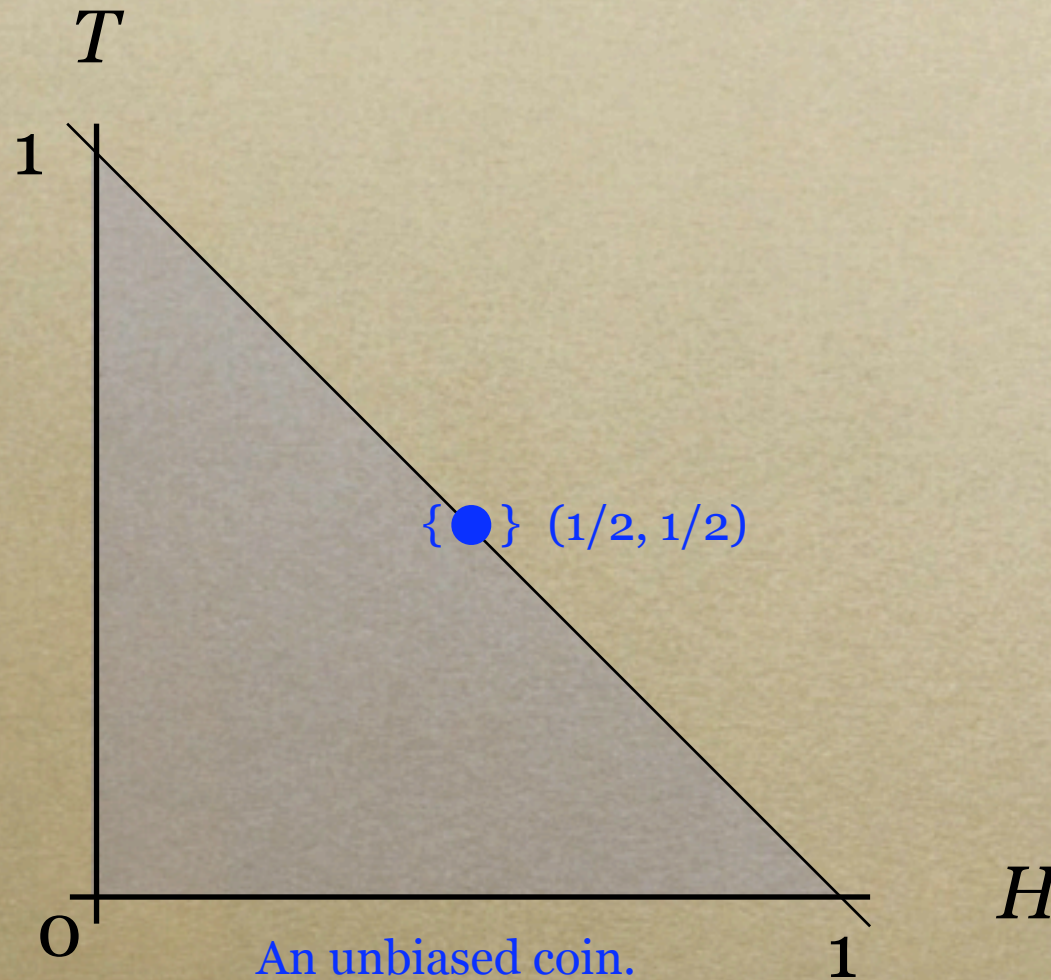
*assignment*

*embedding*

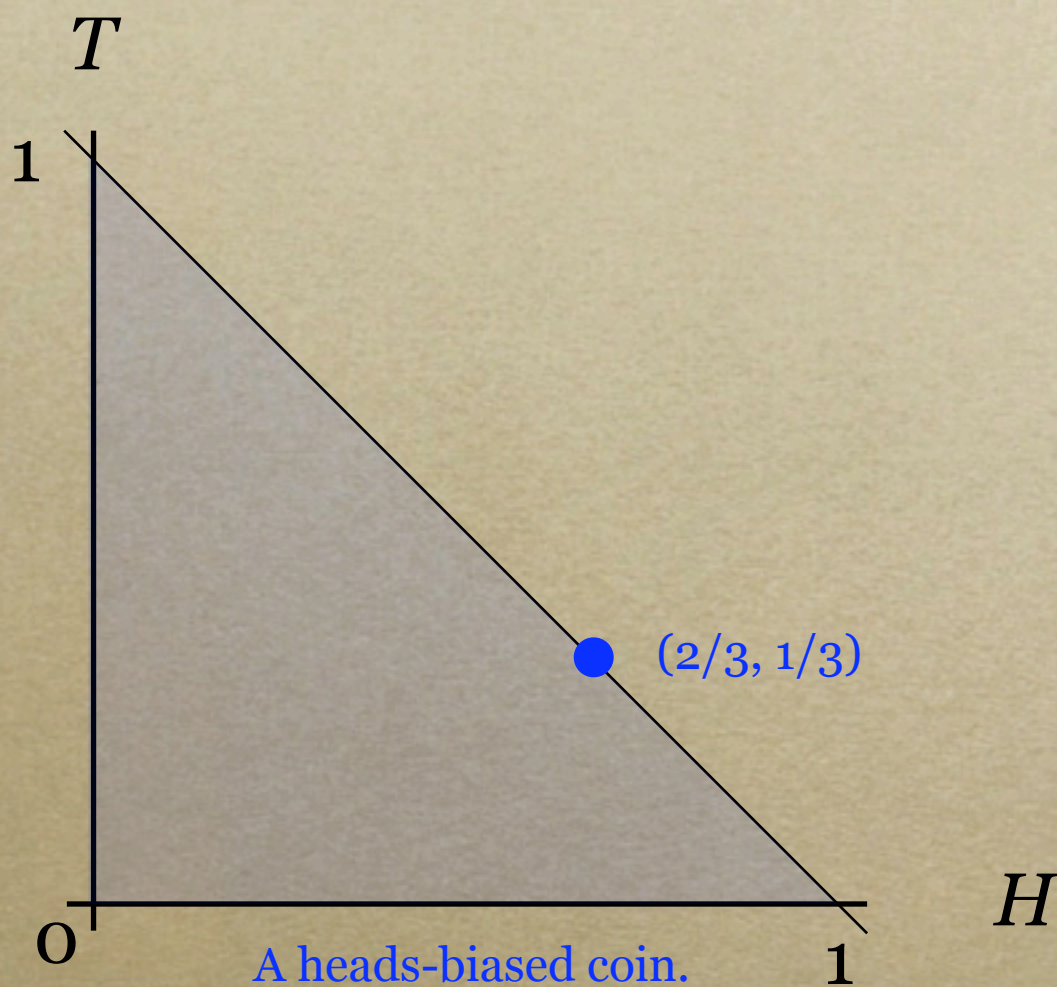
*arithmetic*

Although the program *might*  
achieve  $c=H$ , the largest  
probability of that which can be  
*guaranteed...* is zero.

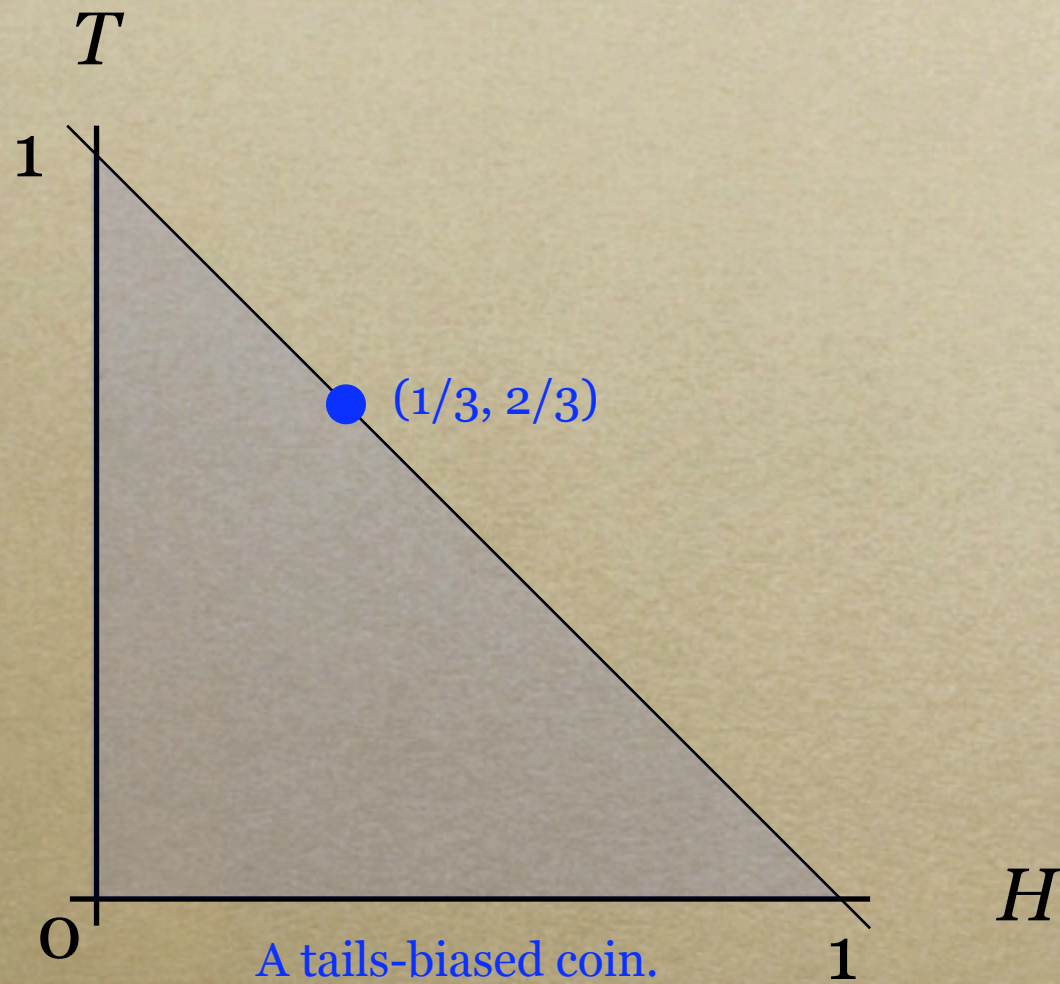
# *Interpretation: a geometric view*



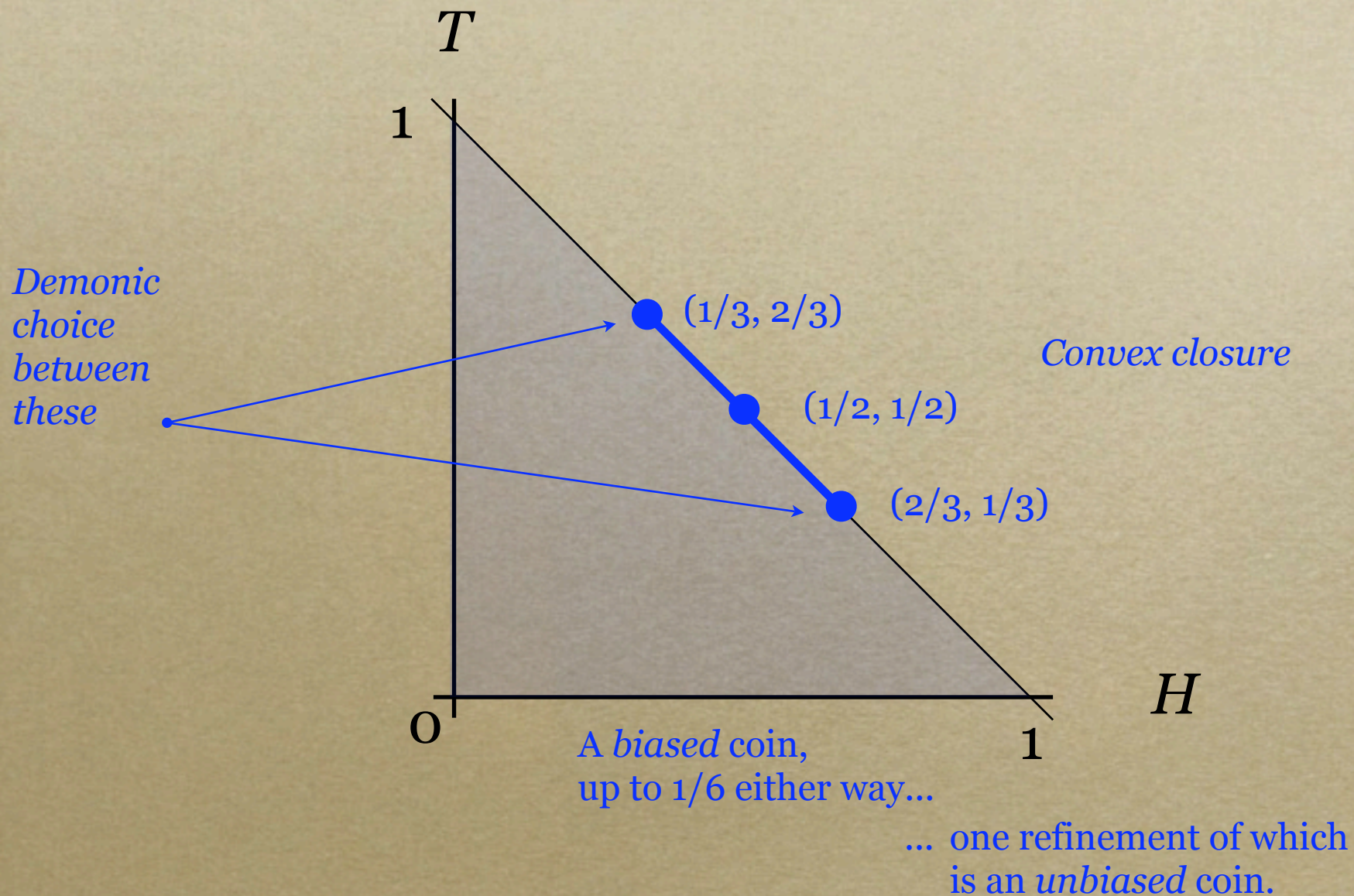
# *Interpretation: a geometric view*



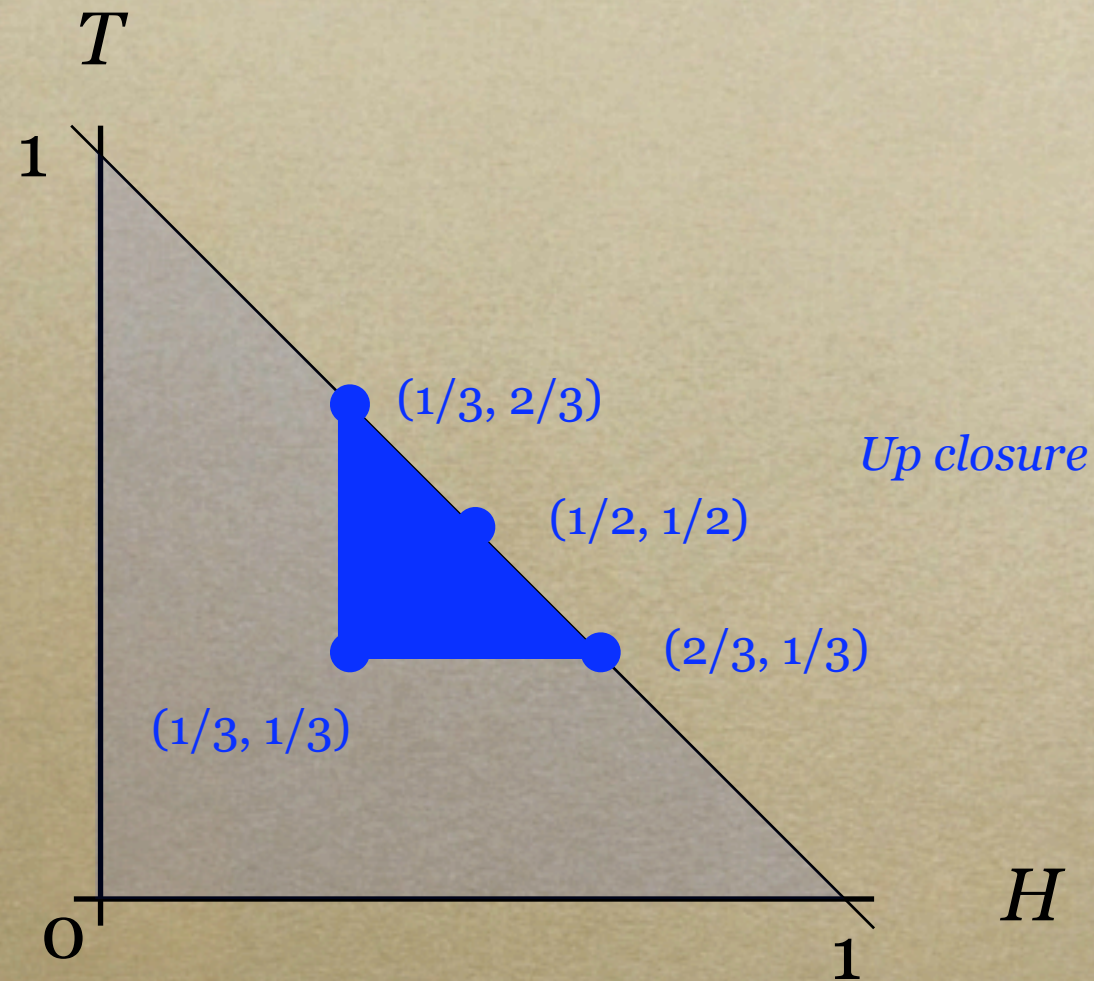
# *Interpretation: a geometric view*



# Interpretation: a geometric view

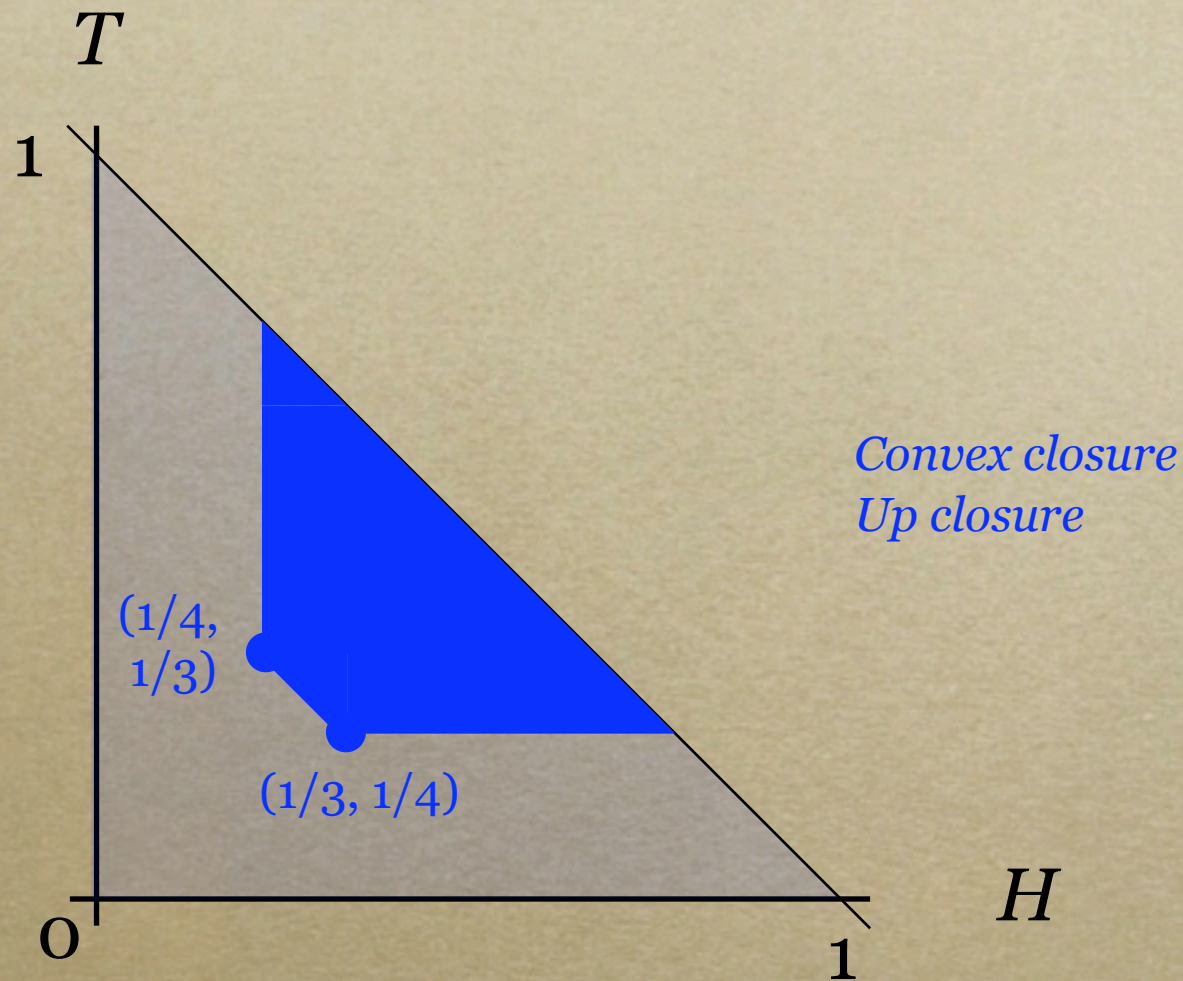


# *Interpretation: a geometric view*



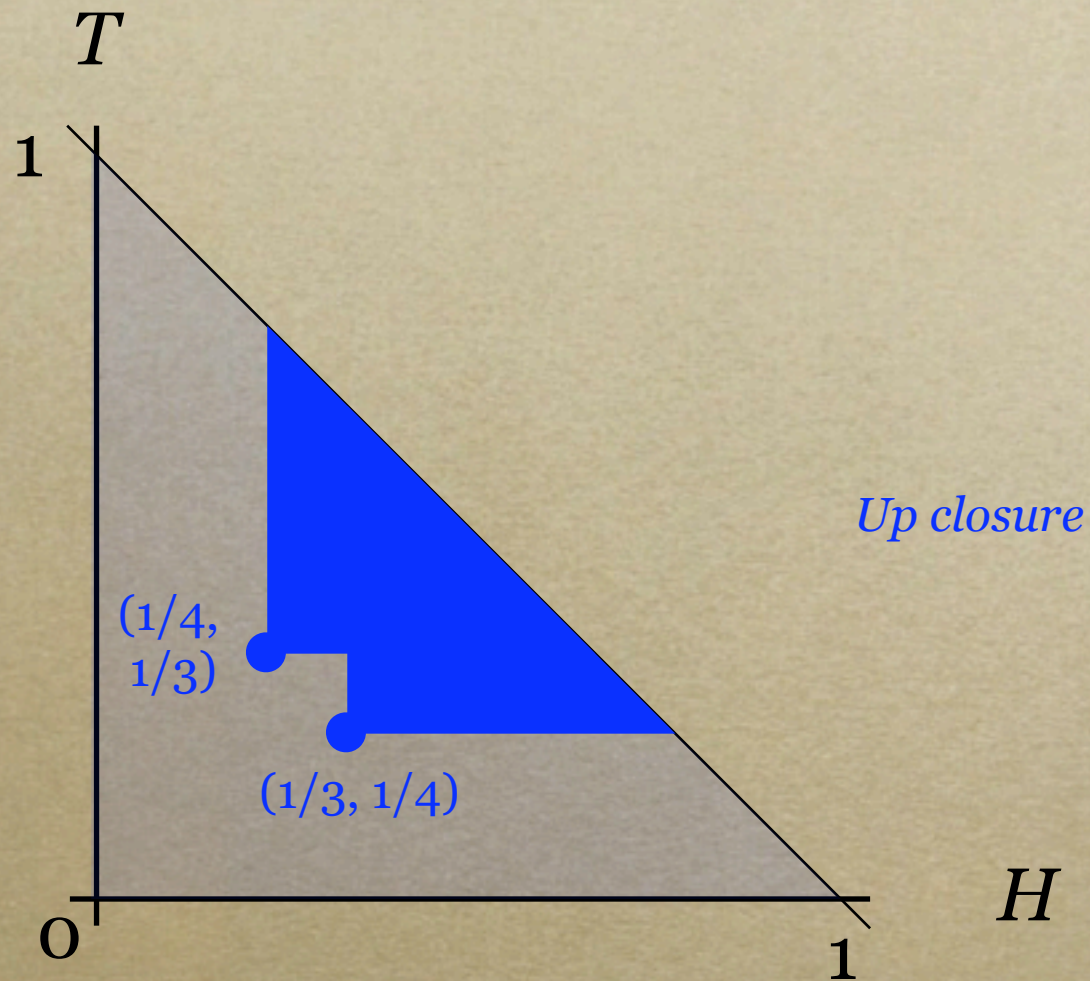
*A possibly nonterminating coin... whose refinements include all three coins before.*

# *Interpretation: a geometric view*



*Demonically, either of two possibly nonterminating coins.*

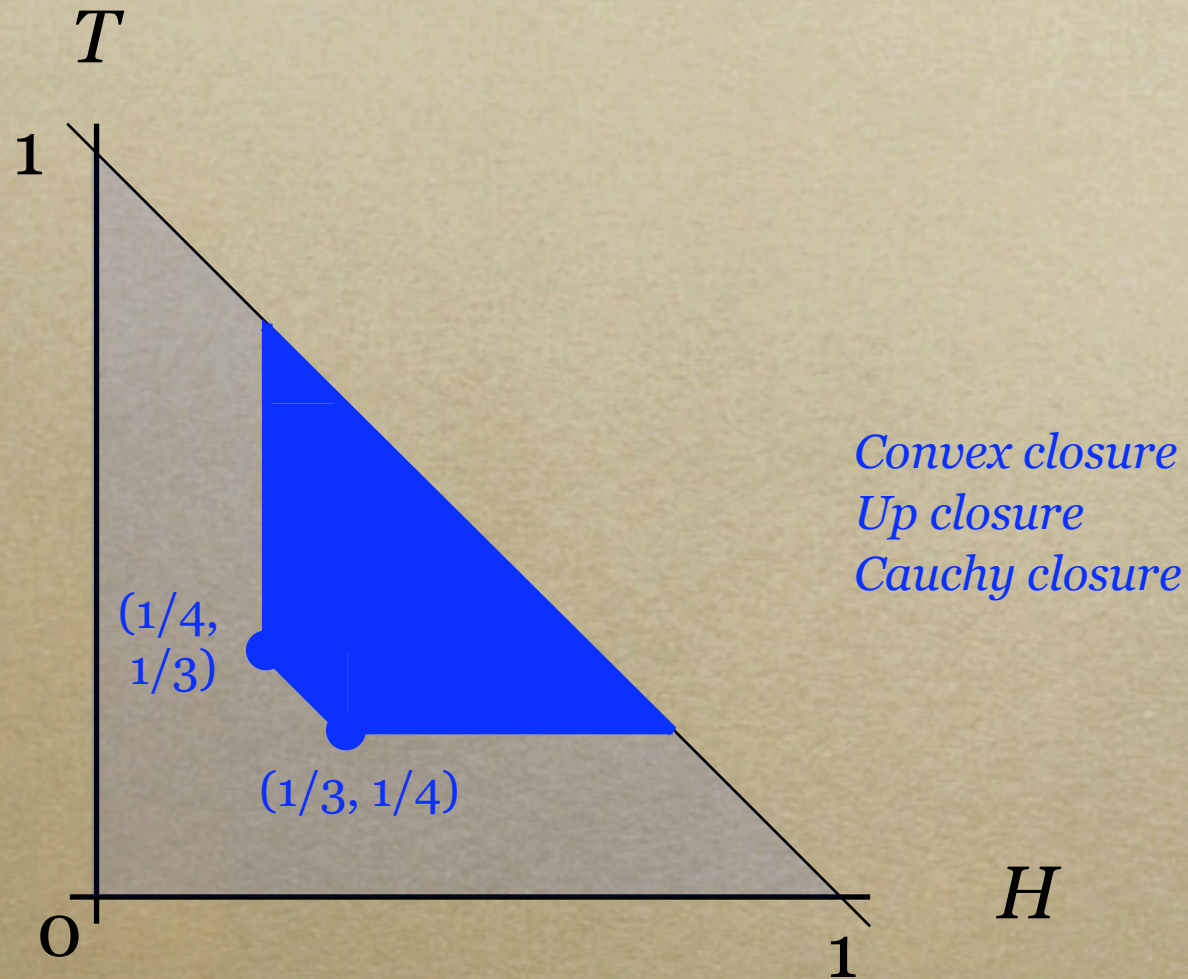
# *Interpretation: a geometric view*



*Demonically, either of two possibly nonterminating coins.*

# Interpretation: a geometric view

He, McIver and Seidel.  
*Probabilistic models for  
the guarded command  
language*. *Sci. Comp.*  
Prog. 28:171-192, 1997.



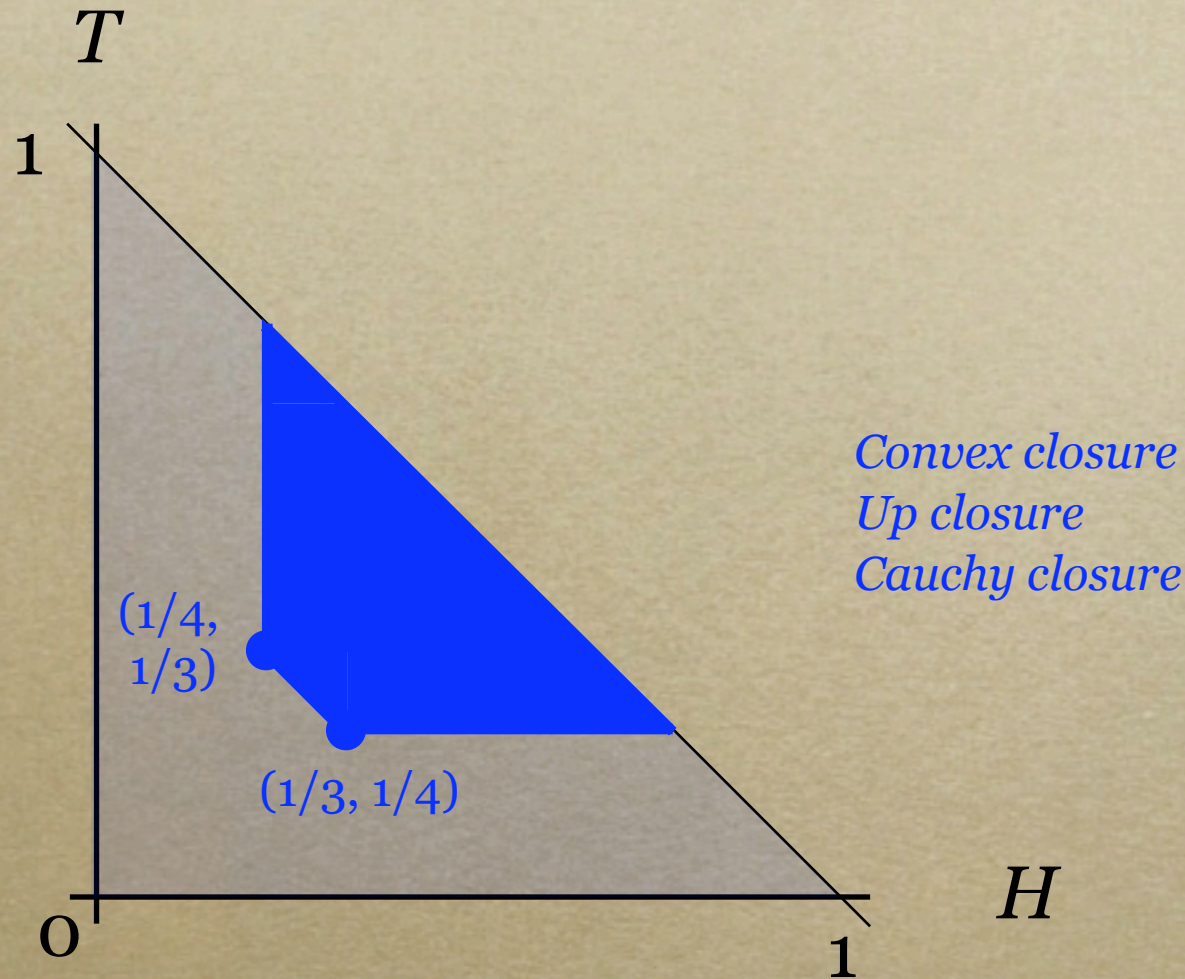
Demonically, either of two  
possibly *nonterminating* coins.

# Interpretation: a geometric view

... but what's the connection with the programming logic?

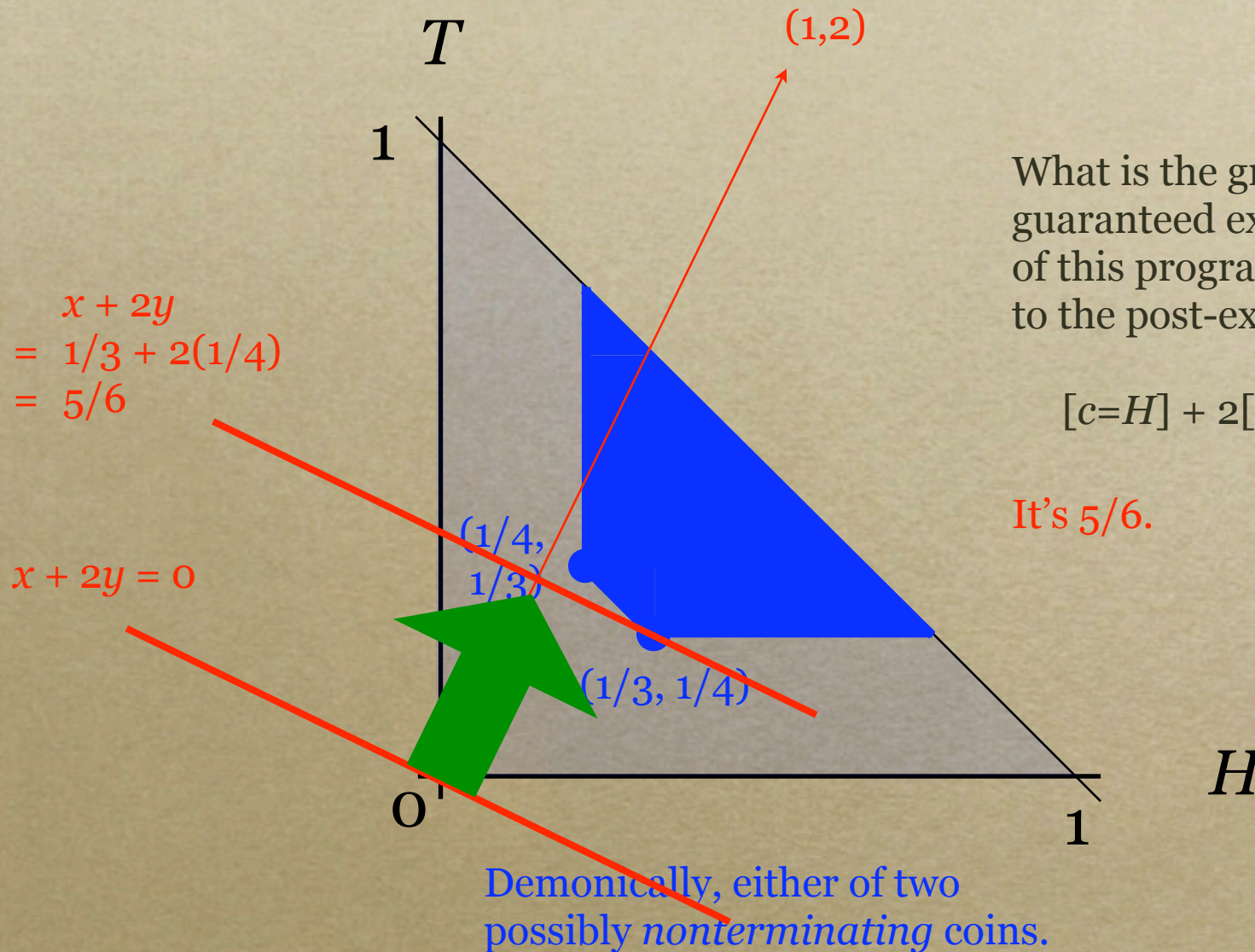
He, McIver and Seidel.  
*Probabilistic models for the guarded command language*. *Sci. Comp. Prog.* 28:171-192, 1997.

Morgan, McIver and Seidel. *Probabilistic predicate transformers*. *ACM TOPLAS* 18(3): 325-353, 1996.



# Interpretation: a geometric view

... but what's the connection with the programming logic?



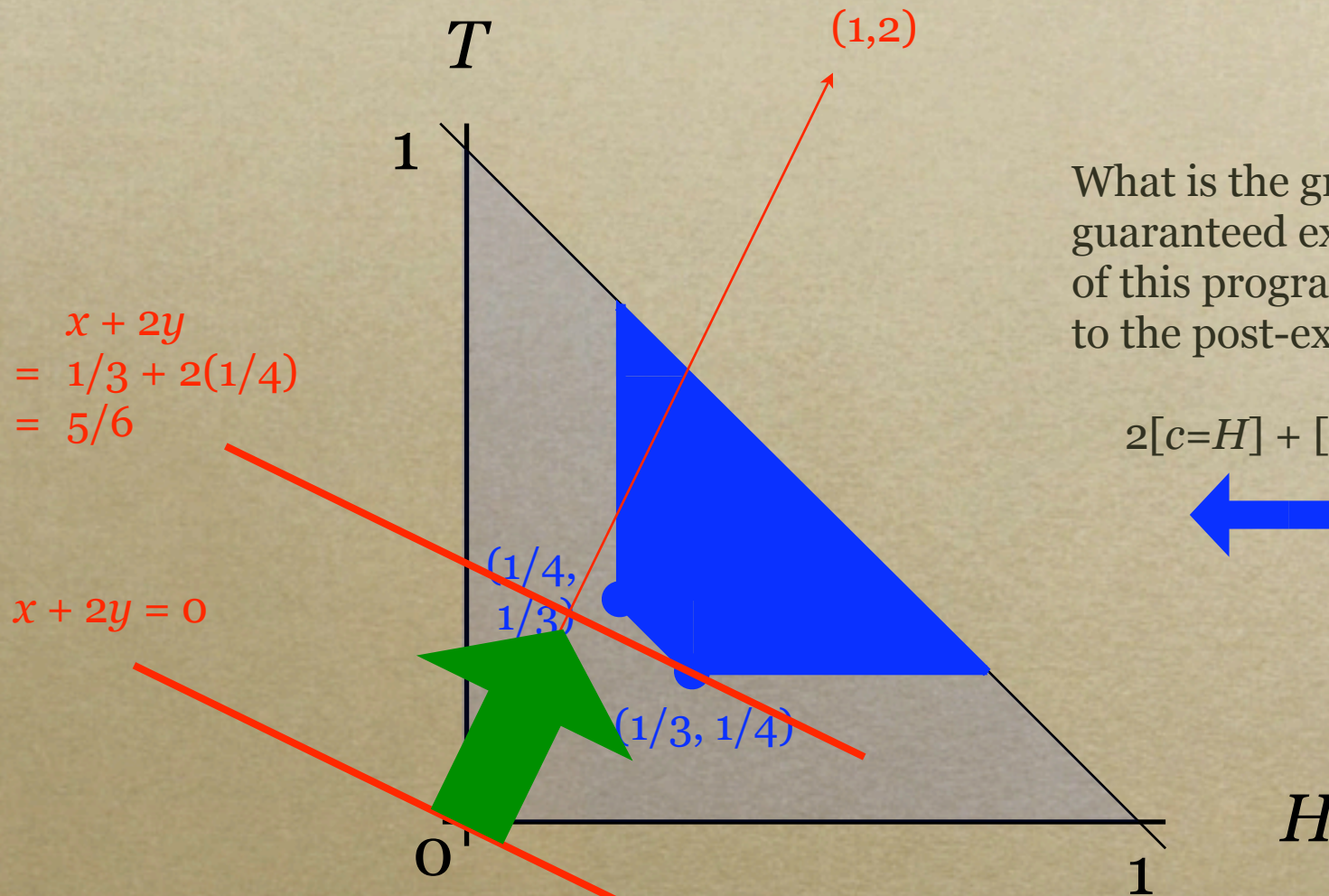
What is the greatest guaranteed expected value of this program with respect to the post-expectation

$$[c=H] + 2[c=T] \quad ?$$

It's 5/6.

# Interpretation: a geometric view

... but what's the connection with the programming logic?

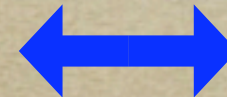


$$\begin{aligned}x + 2y &= 1/3 + 2(1/4) \\ &= 5/6\end{aligned}$$

$$x + 2y = 0$$

What is the greatest guaranteed expected value of this program with respect to the post-expectation

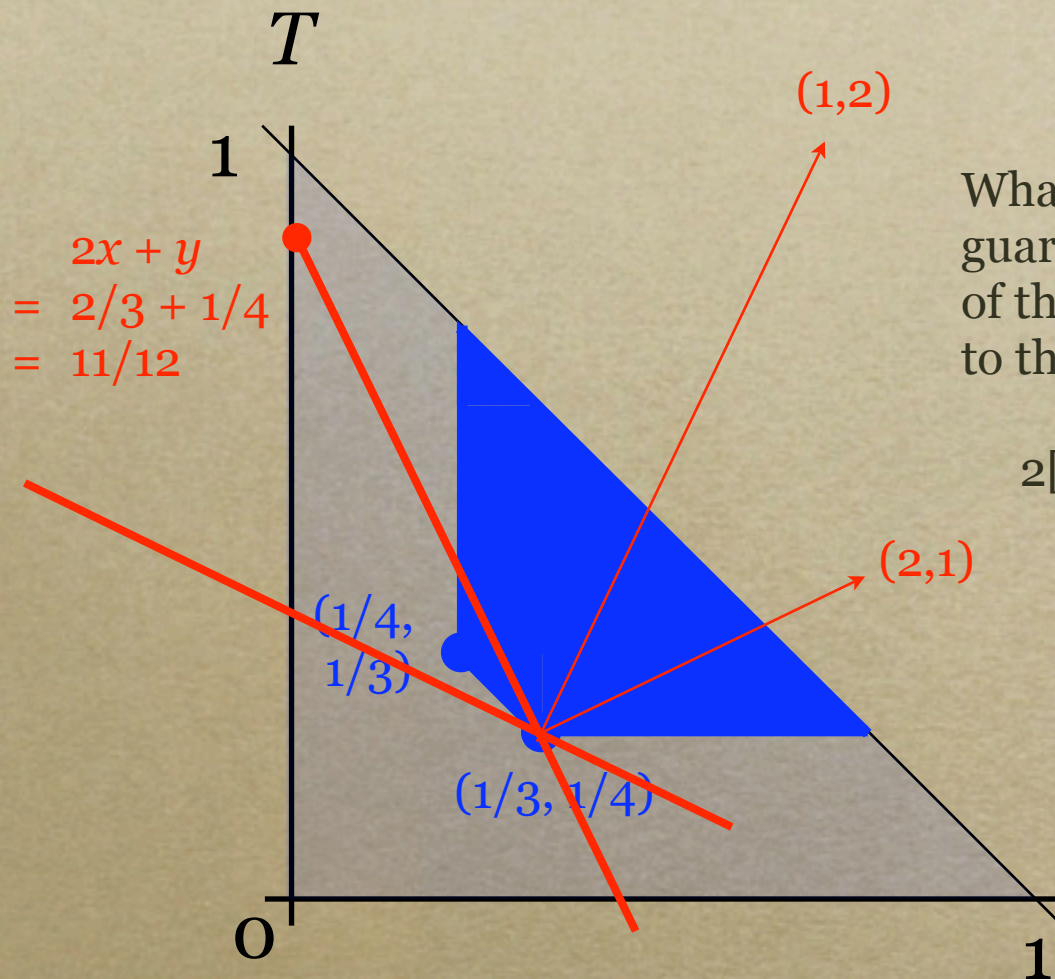
$$2[c=H] + [c=T] \quad ?$$



Demonically, either of two possibly nonterminating coins.

# Interpretation: a geometric view

... but what's the connection with the programming logic?



What is the greatest guaranteed expected value of this program with respect to the post-expectation

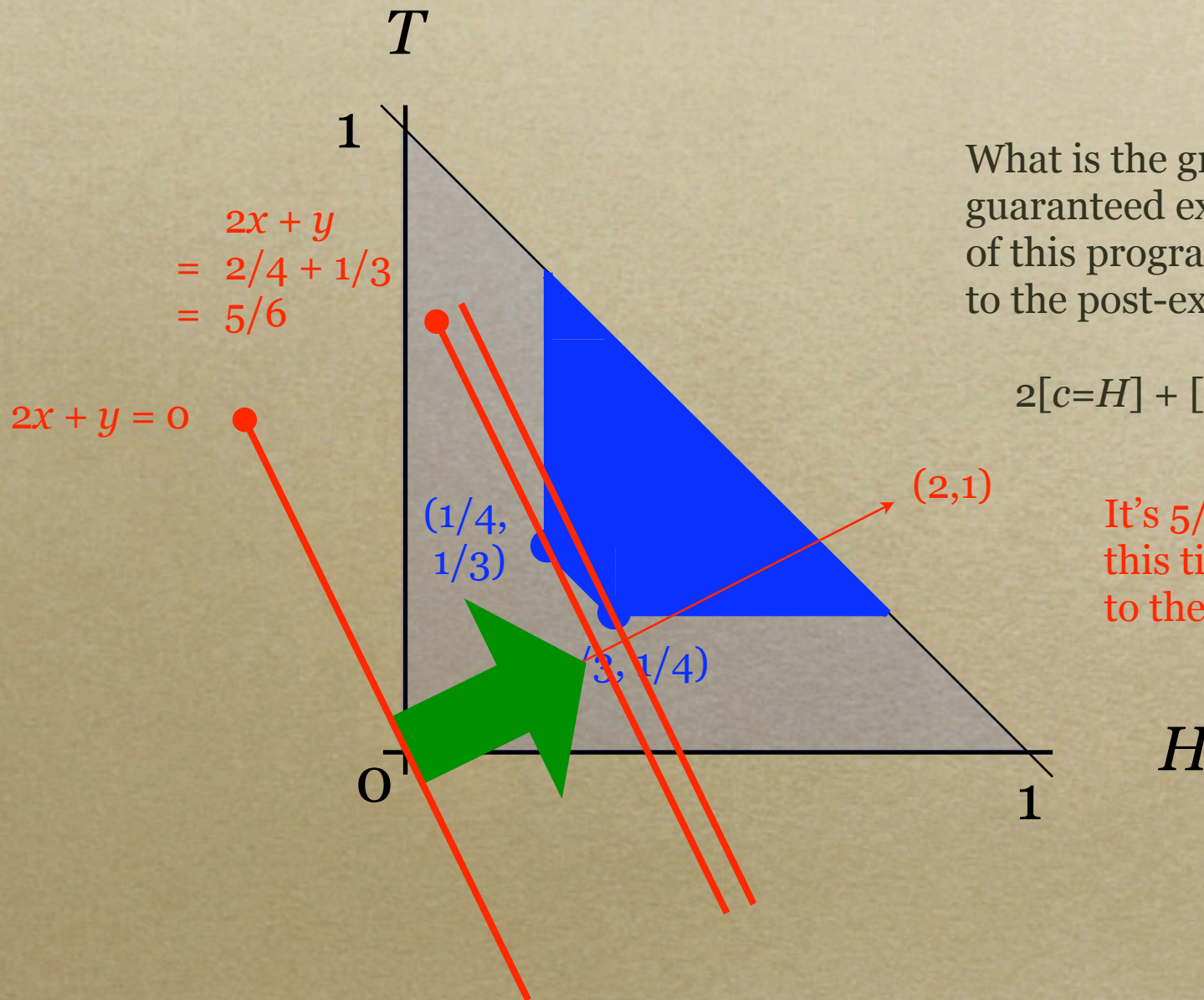
$$2[c=H] + [c=T] \quad ?$$

It's not 11/12.

Demonically, either of two possibly nonterminating coins.

Interpretation: a geometric view

... but what's the connection with the programming logic?



What is the greatest guaranteed expected value of this program with respect to the post-expectation

$$2[c=H] + [c=T] \quad ?$$

It's  $5/6$  again, because this time the demon goes to the other extreme.

# Metatheorems for iteration

// Implement  $p \oplus$  using unbiased random bits only.

$x := p;$

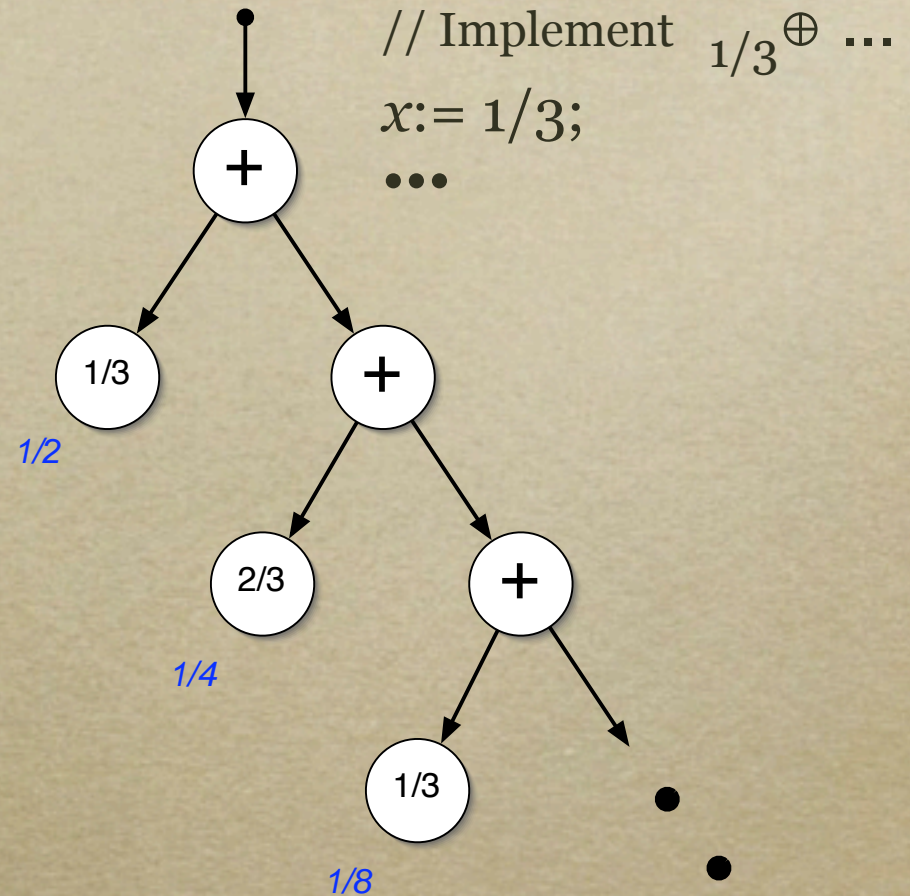
$b := \text{true} \oplus_{1/2} \text{false};$

**do**  $b \rightarrow$

$x := 2x - [x \geq 1/2];$

$b := \text{true} \oplus_{1/2} \text{false};$

**od;**



# Metatheorems for iteration

```
// Implement  $p \oplus$  using unbiased random bits only.
```

```
 $x := p$ ;
```

```
 $b := \text{true} \oplus_{1/2} \text{false}$ ;
```

```
do  $b \rightarrow$ 
```

```
   $x := 2x - [x \geq 1/2]$ ;
```

```
   $b := \text{true} \oplus_{1/2} \text{false}$ ;
```

```
od;
```

```
if  $x \geq 1/2$ 
```

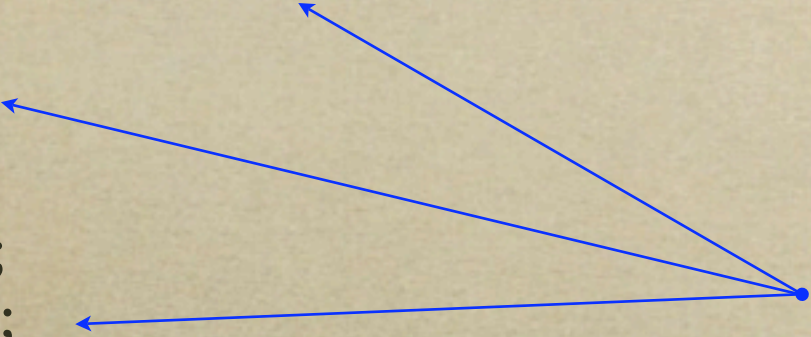
```
  then  $\text{prog}_1$ 
```

```
  else  $\text{prog}_2$ 
```

```
fi
```

*// Variable  $x$  at least  $1/2$  with probability exactly  $p$ .*

*e.g. /dev/random*



Example due to Joe Hurd (Cambridge, now Oxford).

CC Morgan. *Proof rules for probabilistic loops*. Proc. BCS-FACS 7th Refinement Workshop. Springer, 1996. [ewic.bcs.org/conferences/1996/refinement/papers/paper10.htm](http://ewic.bcs.org/conferences/1996/refinement/papers/paper10.htm)

# Metatheorems for iteration

$prog_1 \oplus_p prog_2$

*is  
implemented  
by*

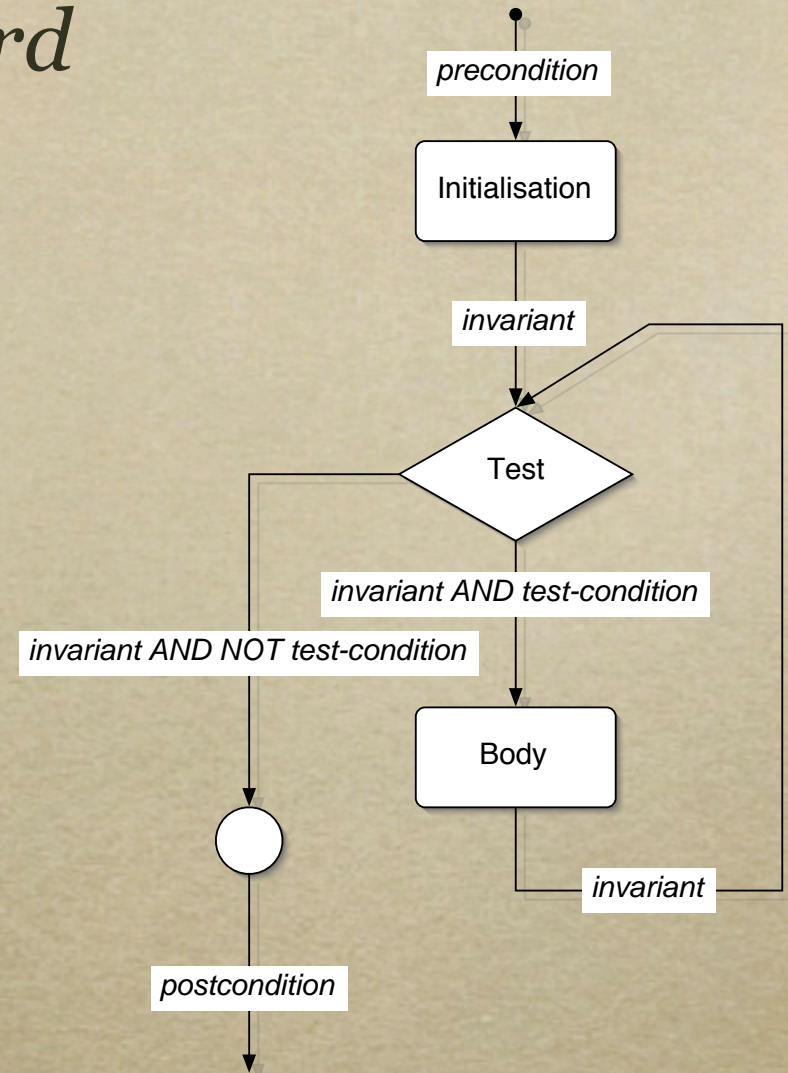
*and on the  
average uses  
only two  
random bits.*

```
begin  
  var  $x, b$ ;  
  
   $x := p$ ;  
   $b := \text{true}_{1/2} \oplus \text{false}$ ;  
  do  $b \rightarrow$   
     $x := 2x - [x \geq 1/2]$ ;  
     $b := \text{true}_{1/2} \oplus \text{false}$ ;  
  od;  
  
  if  $x \geq 1/2$   
    then  $prog_1$   
    else  $prog_2$   
  fi  
end
```

# Iteration: reminder of standard metatheorems

```
x,b,e:= 1,B,E;  
do e ≠ 0 →  
  if even e  
    then b,e := b2, e÷2  
    else e,x := e-1, x × b  
  fi  
od
```

Set  $x$  to  $B^E$  in logarithmic time.



RW Floyd. *Assigning meanings to programs*. Proc. Symp. Appl. Math. Mathematical Aspects of Computer Science 19:19-32, JT Schwartz (ed.). American Math. Soc, 1967.

CAR Hoare, 1969.

EW Dijkstra, 1975.

Gries, Backhouse, Kaldewaij, Cohen...

# Standard metatheorems: invariants

$\{ B > 0 \text{ and } E \geq 0 \}$

$x, b, e := 1, B, E;$

$\{ b > 0 \text{ and } e \geq 0 \text{ and } B^E = x \times b^e \}$

**do**  $e \neq 0 \rightarrow$

$\{ \dots \text{ and } e > 0 \}$

**if** even  $e$

**then**  $\{ e \geq 2 \text{ and even } e \dots \quad b, e := b^2, e \div 2 \quad \{ B^E = x \times b^e \}$   
 $\dots \text{ and } B^E = x \times b^e \}$

**else**  $\{ B^E = x \times b^e \} \quad e, x := e-1, x \times b \quad \{ B^E = x \times b^e \}$

**fi**

$\{ B^E = x \times b^e \}$

**od**

$\{ x = B^E \}$

# *Standard metatheorems: invariants*

$\{ pre \}$

*init*;

$\{ inv \}$

**do**  $G \rightarrow$

$\{ G \wedge inv \}$

*body*

$\{ inv \}$

**od**

$\{ \neg G \wedge inv \}$

# Probabilistic metatheorems: invariants *again*

$\{pre\}$   
*init*;  
 $\{inv\}$   
**do**  $G \rightarrow$   
     $\{[G] \times inv\}$   
    *body*  
     $\{inv\}$   
**od**  
 $\{[\neg G] \times inv\}$

$\{pre\}$   
*init*;  
 $\{inv\}$   
**do**  $G \rightarrow$   
     $\{G \wedge inv\}$   
    *body*  
     $\{inv\}$   
**od**  
 $\{\neg G \wedge inv\}$

# Iteration: probabilistic example

{ ? }

$x := p;$

$b := \text{true} \oplus_{1/2} \text{false};$

**do**  $b \rightarrow$

$x := 2x - [x \geq 1/2];$

$b := \text{true} \oplus_{1/2} \text{false};$

**od**

{  $[x \geq 1/2]$  }

*What is the probability  
that  $x$  exceeds  $1/2$  on  
termination?*

# Example: iteration achieves its goal on *termination*

$$\begin{aligned}
 & [x \geq 1/2] \\
 \Leftrightarrow & [\neg b] \times \\
 & ( 2x - [x \geq 1/2] \boxed{\triangleleft} b \boxed{\triangleright} [x \geq 1/2] ) \\
 & \quad \quad \quad \begin{array}{c} | \quad | \\ \dots \text{ if } b \text{ else } \dots \end{array}
 \end{aligned}$$

```

x := p;
b := true 1/2 ⊕ false;
do b →
    x := 2x - [x ≥ 1/2];
    b := true 1/2 ⊕ false;
    { 2x - [x ≥ 1/2] < b >
      [x ≥ 1/2] }
od
{ [x ≥ 1/2] }
    
```

CAR Hoare. *A couple of novelties in the Propositional Calculus.*

Zeitschr. für Math. Logik und Grundlagen der Math. 31(2):173-178,

1985.

# Example: iteration *body* preserves invariant

$$\begin{aligned} \square &\equiv (2x - [x \geq 1/2] \triangleleft b \triangleright [x \geq 1/2]) \\ \square &\equiv (2x)/2 \\ &\equiv x \end{aligned}$$

$x := p;$

$b := \text{true}_{1/2} \oplus \text{false};$

**do**  $b \rightarrow$

$x := 2x - [x \geq 1/2];$   
 $\{x\}$

$b := \text{true}_{1/2} \oplus \text{false};$

$\{2x - [x \geq 1/2] \triangleleft b \triangleright$

**od**  $\{x \geq 1/2\}$   
 $\{[x \geq 1/2]\}$

# Example: iteration properly initialised

Assignment;  
loop initialisation;  
and then we repeat the earlier step.

```
x := p;  
{ x }  
b := true  $\frac{1}{2} \oplus$  false;  
{ 2x - [x ≥ 1/2] < b }  
do b → [x ≥ 1/2] }  
  { 2x - [x ≥ 1/2] }  
  x := 2x - [x ≥ 1/2];  
  { x }  
  b := true  $\frac{1}{2} \oplus$  false;  
  { 2x - [x ≥ 1/2] < b }  
od [x ≥ 1/2] }  
{ [x ≥ 1/2] }
```

# Example: correct overall

And finally we see that the pre-expectation overall...

is just  $p$ .

```
{ p }  
x := p;  
{ x }
```

```
b := true  $\frac{1}{2}$   $\oplus$  false;
```

```
{ 2x - [x  $\geq$  1/2]  $\triangleleft$  b  $\triangleright$ 
```

```
do b  $\rightarrow$  [x  $\geq$  1/2] }
```

```
{ 2x - [x  $\geq$  1/2] }
```

```
x := 2x - [x  $\geq$  1/2];
```

```
{ x }
```

```
b := true  $\frac{1}{2}$   $\oplus$  false;
```

```
{ 2x - [x  $\geq$  1/2]  $\triangleleft$  b  $\triangleright$ 
```

```
od [x  $\geq$  1/2] }
```

```
{ [x  $\geq$  1/2] }
```

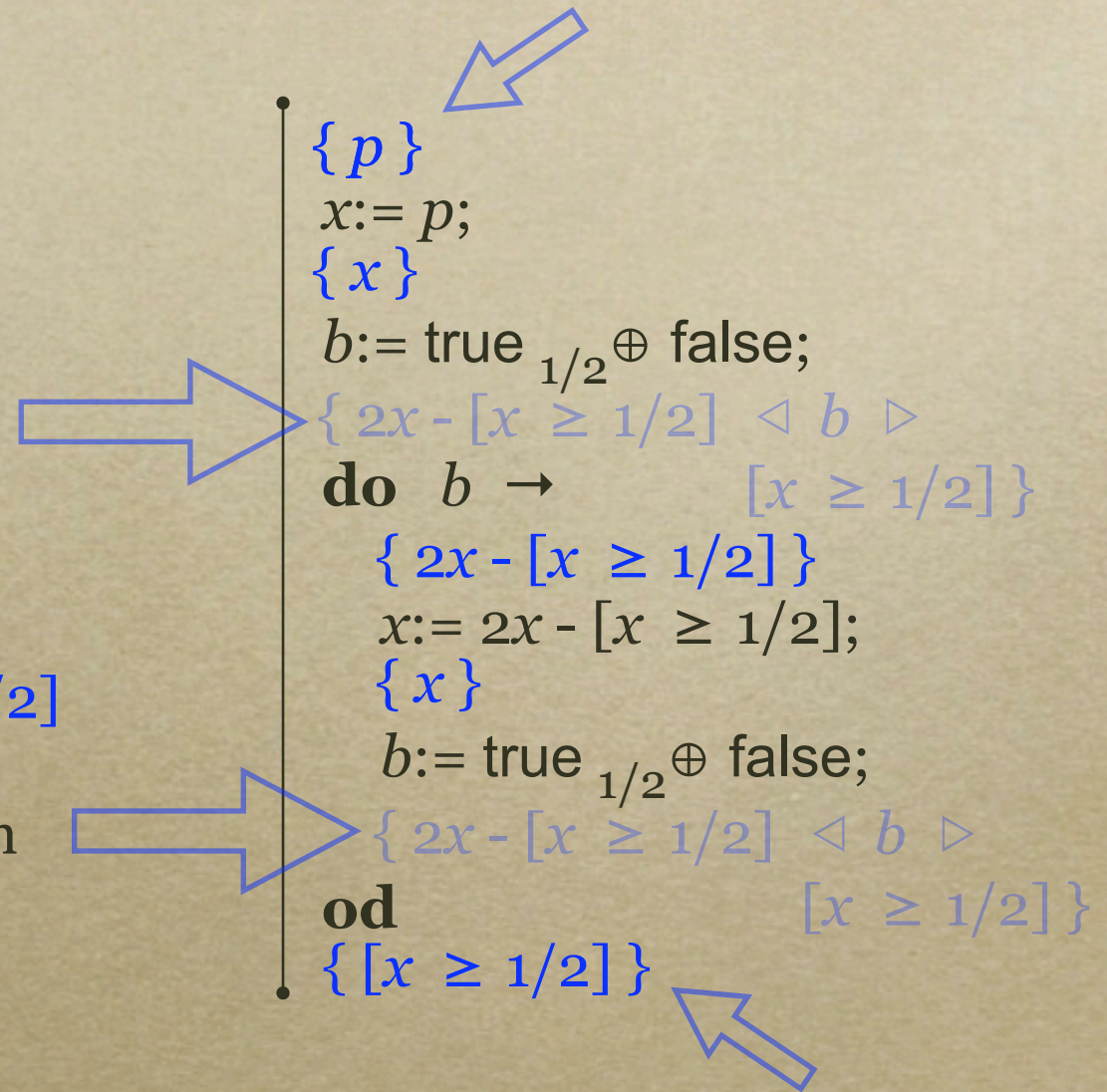
# Example: summary

The probability that the program establishes  $x \geq 1/2$  is just  $p$ .

The loop invariant was

$$2x - [x \geq 1/2] \triangleleft b \triangleright [x \geq 1/2]$$

“established” by the initialisation and “maintained” by the body.



# Termination of probabilistic iterations

$\{ inv \}$

**do**  $G \rightarrow$

$\{ [G] \times inv \}$

*body*

$\{ inv \}$

**od**

$\{ [\neg G] \times inv \}$

$\{ inv \}$

**do**  $G \rightarrow$

$\{ G \wedge inv \}$

*body*

$\{ inv \}$

**od**

$\{ \neg G \wedge inv \}$

In addition, show that  $inv \Rightarrow term$ , where  $term$  is the probability of termination ...

... in which case the conclusion  $\{ inv \} \mathbf{do} \cdots \mathbf{od} \{ [\neg G] \times inv \}$  expresses total — rather than just partial — correctness.

# *Exercises*

*Ex. 1: Probabilistic then demonic choice*

Calculate  $wp.(c := H_{1/2} \oplus T; d := H \sqcap T).[c=d]$ .

*Ex. 2: Demonic then probabilistic choice*

Calculate  $wp.(d := H \sqcap T; c := H_{1/2} \oplus T).[c=d]$ .

*Ex. 3: Explain the difference*

The answers you get to Ex. 1 and Ex. 2 should differ.  
Explain “in layman’s terms” why they do.

(Hint: Imagine an experiment with two people and two coins, in each case.)

## *Ex. 4: The nature of demonic choice*

It is sometimes suggested that *demonic* choice can be regarded as an arbitrary but unpredictable *probabilistic* choice; this would simplify matters because there would then only be one kind of choice to deal with.

Use our logic to investigate this suggestion; in particular, look at the behaviour of

$$c := H_{1/2} \oplus T; \quad d := H_p \oplus T \quad \text{for arbitrary } p,$$

and compare it with the program of Ex. 1. Explain your conclusions in layman's terms.

## Ex. 5: Compositionality

Consider the two programs

$A:$              $\text{coin} := \text{edge} \sqcap (\text{coin} := \text{heads} \text{ }_{1/2} \oplus \text{coin} := \text{tails})$

$B:$              $(\text{coin} := \text{edge} \sqcap \text{coin} := \text{heads})$   
 $\text{ }_{1/2} \oplus (\text{coin} := \text{edge} \sqcap \text{coin} := \text{tails}),$

which we will call  $A$  and  $B$ . Say that they are *similar* because from any initial state they have the same worst-case probability of achieving any given postcondition. (This can be shown by tabulation: there are only eight possible postconditions.)

Find a program  $C$  such that  $A;C$  and  $B;C$  are *not similar*, even though  $A$  and  $B$  are. (Use the *wp*-definition of “;” .) Why is this a problem?

More generally, let  $A$  and  $B$  be *any* two programs that are *not equal* in our *wp* logic. Show that there is *always* a program  $C$  as above, *i.e.* such that  $A;C$  and  $B;C$  are *not similar*. What does that tell you about our quantitative logic in terms of its possibly being a “minimal complication”?