Deductive Verification of Probabilistic Programs

Joost-Pieter Katoen

PROBABILISTIC PROGRAMMING

Every programming language has a probabilistic variant

The piranha problem

[Tijms, 2004]

One fish is contained within the confines of an opaque fishbowl. The fish is equally likely to be a piranha or a goldfish. A sushi lover throws a piranha into the fish bowl alongside the other fish. Then, immediately, before either fish can devour the other, one of the fish is blindly removed from the fishbowl. The fish that has been removed from the bowl turns out to be a piranha. What is the probability that the fish that was originally in the bowl by itself was a piranha?

Probabilistic programs

Programs with random assignments and conditioning

```
f1 := gf [0.5] f1 := pir;
f2 := pir;
s := f1 [0.5] s := f2;
observe (s = pir)
```

$Pr\{f_1 = gf\} = \frac{1}{2}$ $Pr\{f_1 = pir\} = \frac{1}{2}$

Probabilistic programs

Programs with random assignments and conditioning

```
f1 := gf [0.5] f1 := pir;
f2 := pir;
s := f1 [0.5] s := f2;
observe (s = pir)
```

Probabilistic programs

Programs with random assignments and conditioning

```
f1 := gf [0.5] f1 := pir;
f2 := pir;
s := f1 [0.5] s := f2;
observe (s = pir)
```

What is the probability that the original fish in the bowl was a piranha?

Probabilistic programs

Programs with random assignments and conditioning

```
f1 := gf [0.5] f1 := pir;
f2 := pir;
s := f1 [0.5] s := f2;
observe (s = pir)
```

They encode:

- randomised algorithms
- probabilistic graphical models beyond Bayes' networks
- controllers for autonomous systems
- security mechanisms
- .....

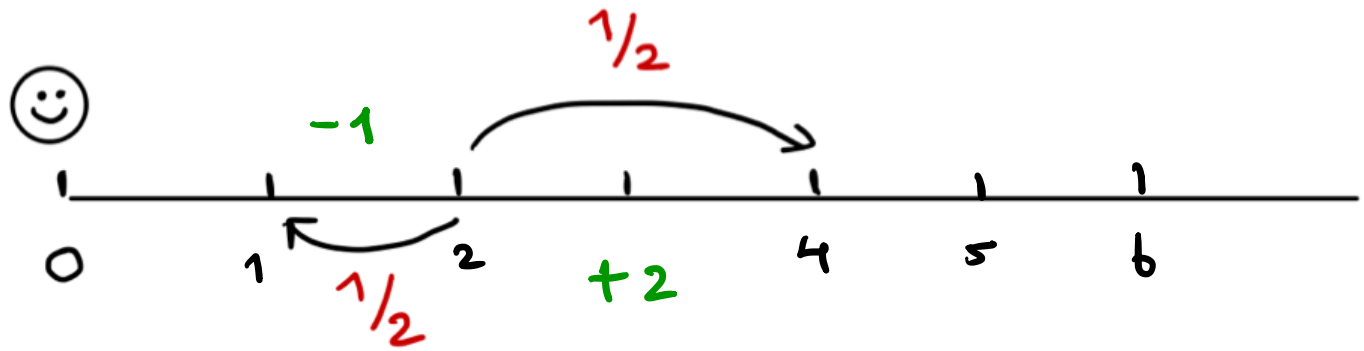
"Probabilistic programming aims to make probabilistic modeling and machine learning accessible to the programmer."

[Gordon, Henzinger, Nori and Rajamani, FOSE 2014]

Probabilistic programs are hard to grasp

Does this program almost surely terminate? That is, is it AST?

```
x := 1;
while (x > 0) {
    x := x+2 [1/2] x := x-1
}
```



Probabilistic programs are hard to grasp

Does this program almost surely terminate? That is, is it AST?

```
x := 1;
while (x > 0) {
    x := x+2 [1/2] x := x-1
}
```

If not, what is its probability to diverge?

Positive AST

```
int x := 1;
bool c := true;
while (c) {
    c := false [0.5] c := true;
    x := 2*x
}
```

Finite expected termination time?
aka: is this program positive AST?

Positive AST

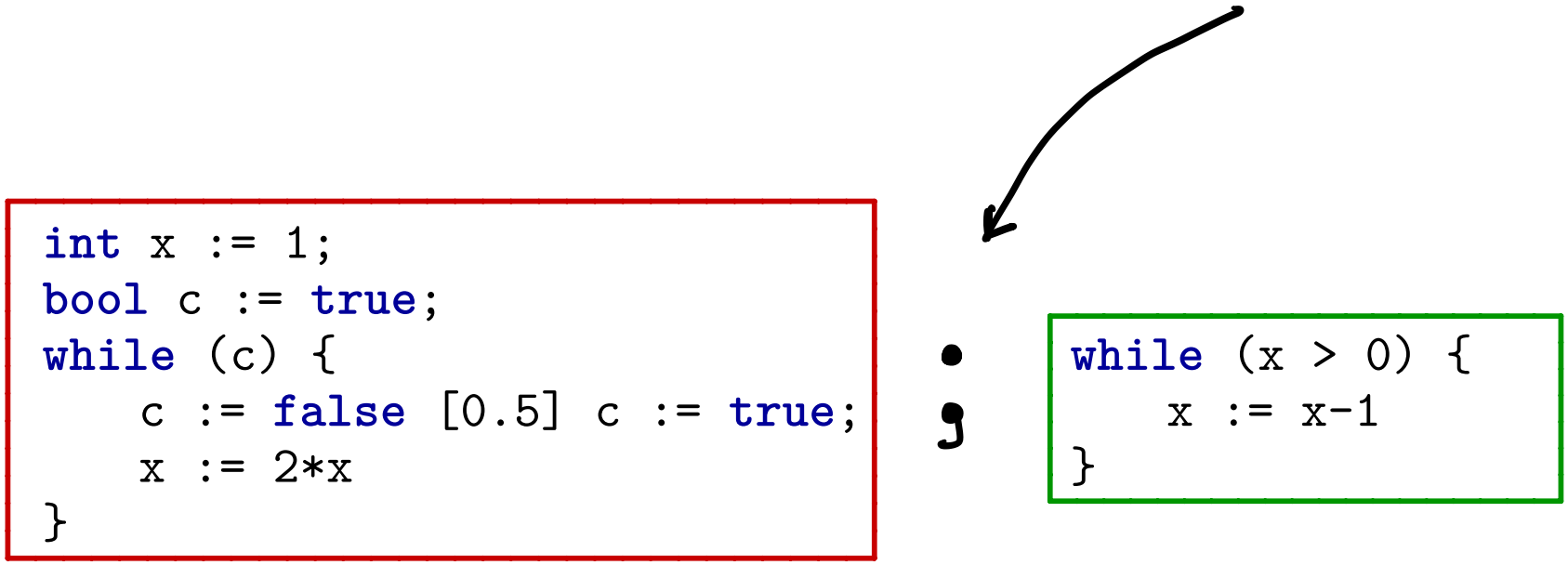
```
int x := 1;
bool c := true;
while (c) {
    c := false [0.5] c := true;
    x := 2*x
}
```

Finite expected termination time?
aka: is this program positive AST?

```
while (x > 0) {
    x := x-1
}
```

Finite termination time!
PAST.

Positive AST



Finite expected termination time?
aka: is this program positive AST?

Finite termination time!
PAST.

Expected runtime of these programs in sequence?

Our objective

A powerful, simple proof calculus for probabilistic programs.

At the source code level.

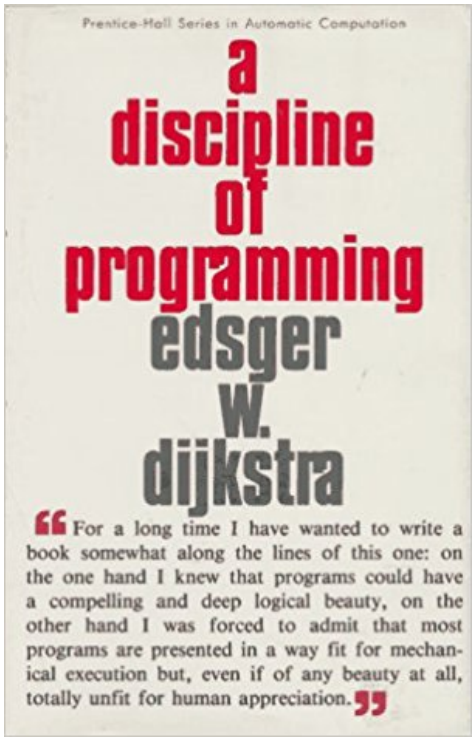
No "descend" into the underlying probabilistic model.

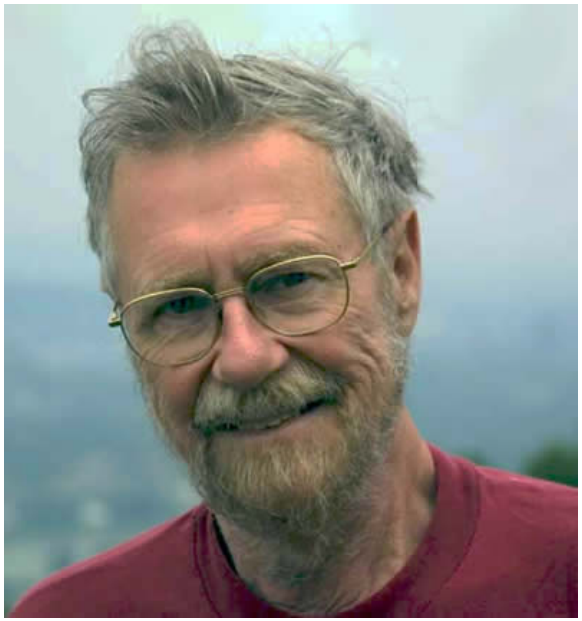
Push automation as much as we can.

This is a true challenge: undecidability!

Typically "more undecidable" than deterministic programs

WEAKEST PRECONDITIONS





Edsger Wybe Dijkstra

Predicate transformers

Let the set of states be:

$$\mathbb{S} = \{ s \mid s : Vars \to \mathbb{Q}_{\geq 0} \}$$

e.g. $s(x) = 1$

$s(y) = 3$

$s(z) = 0$

Predicate transformers

Let the set of states be:

$$\mathbb{S} = \{ s \mid s : Vars \to \mathbb{Q}_{\geq 0} \}$$

Let the set of predicates be:

$$\mathbb{P} = \left\{ F \mid F : \underbrace{\mathbb{S}}_{\text{states}} \to \underbrace{\{0,1\}}_{\mathbb{B}\ (\text{booleans})} \right\}$$

example: $F = x > 0 \wedge 0 \leq y < 10$

$s(x) = 4,\ s(y) = 3$ then $s \models F$

$s'(x) = 4,\ s'(y) = 10$ then $s' \not\models F$

Predicate transformers

Let the set of states be:

$$\mathbb{S} = \{ s \mid s : Vars \to \mathbb{Q} \}$$

Let the set of predicates be:

$$\mathbb{P} = \left\{ F \mid F : \underbrace{\mathbb{S}}_{\text{states}} \to \{0,1\} \right\}$$

Predicate F is typically a first-order logic formula. It equals $\{ s \in \mathbb{S} \mid s \models F \}$. Thus $\mathbb{P} = 2^{\mathbb{S}}$.

$(\mathbb{P}, \sqsubseteq)$ is a complete lattice where $F \sqsubseteq G$ if and only if $F \Rightarrow G$

Predicate transformers

Let the set of states be:

$$\mathbb{S} = \{ s \mid s : Vars \to \mathbb{Q} \}$$

Let the set of predicates be:

$$\mathbb{P} = \left\{ F \mid F : \underbrace{\mathbb{S}}_{\text{states}} \to \{0,1\} \right\}$$

Predicate F is typically a first-order logic formula. It equals $\{ s \in \mathbb{S} \mid s \models F \}$. Thus $\mathbb{P} = 2^{\mathbb{S}}$. Let partial order $\sqsubseteq$ equal $\subseteq$. Ergo:

$(\mathbb{P}, \sqsubseteq)$ is a complete lattice where $F \sqsubseteq G$ if and only if $F \Rightarrow G$

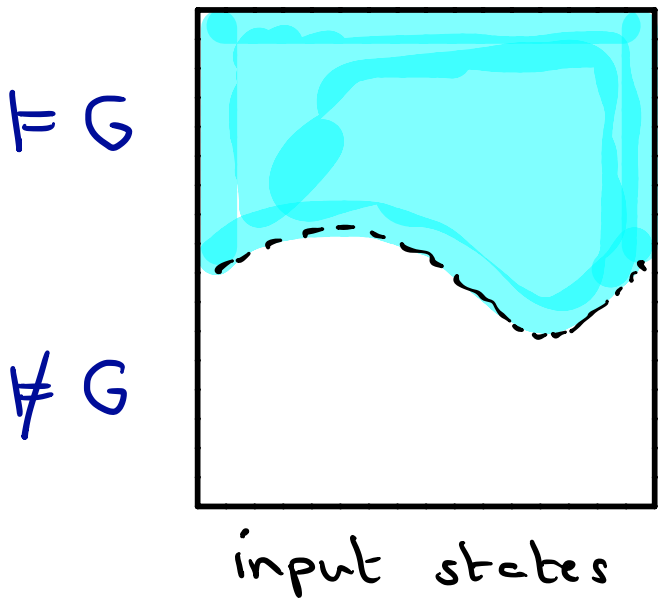
Function $\Phi : \mathbb{P} \to \mathbb{P}$ is called a predicate transformer

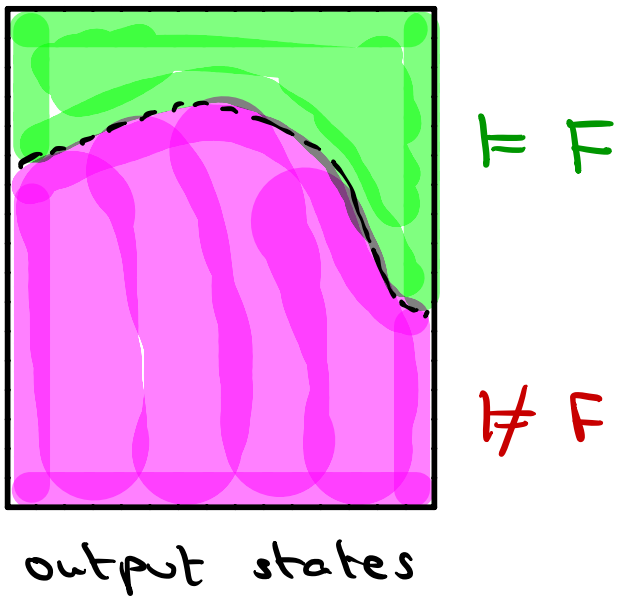
Weakest preconditions

For program P, let $wp[\![P]\!] : \mathbb{P} \to \mathbb{P}$ a predicate transformer.

$G = wp[\![P]\!](F)$ is P's weakest precondition w.r.t. postcondition F iff

- If P starts in a state $s \models G$, it terminates in a state $t \models F$.
- Otherwise, P either terminates in a state $t \not\models F$ or diverges





Weakest preconditions versus Hoare triples

Weakest preconditions are functional

- For each $F \in \mathbb{P}$ there is a unique $G \in \mathbb{P}$ such that

$$wp[\![P]\!](F) \;=\; G$$

- Weakest preconditions respect Hoare triples:

$\{\, wp[\![P]\!](F) \,\}\ P\ \{\, F \,\}$ is a valid statement

- For terminating[1] P:

$\{\, G \,\}\ P\ \{\, F \,\}$ is a valid statement, then $\{\, G \,\} \Rightarrow wp[\![P]\!](F)$

[1] For diverging P, the statement $\{\text{true}\}\, P\, \{F\}$ is trivially true, but $wp[\![P]\!](F) = \text{false}$.

Weakest preconditions for GCL

Syntax program P	Weakest precondition $wp[\![P]\!](F)$
`skip`	F
$x := E$	$F[x := E]$
$P; Q$	$wp[\![P]\!](wp[\![Q]\!](F))$
`if` (φ) P `else` Q	$(\varphi \wedge wp[\![P]\!](F)) \vee (\neg\varphi \wedge wp[\![Q]\!](F))$

Example

$P = a{+}{+} \,;\, b{-}{-}$

$F = a \cdot b = 0$

What is $wp\,[\![P]\!]\,(F)$?

Example

wp ⟦P⟧ ($y^2 > 0$) where program P is:

F

```
if (y>0) { x:=5 } else { x:=2 ; y++ } ;
y := x-3
```

read this from bottom to top:

// $(y>0 \wedge true) \vee (\neg(y<0) \wedge false) \equiv y>0$

wp ⟦P⟧ (F)

```
if (y>0) {
// (5-3)^2 > 2 ≡ true
    x:=5
// (x-3)^2 > 2
} else {
// (2-3)^2 > 2 ≡ false
    x:=2;
// (x-3)^2 > 2
    y++
// (x-3)^2 > 2
};
// (x-3)^2 > 2
y := x-3
```

// $y^2 > 2$ ← post condition

Loops

while (e) {p}

=

Possibly unbounded loops

$$wp[\![\text{while } (\varphi)\,\{\,P\,\},F]\!] \;=\; \text{lfp}\,X.\;\underbrace{((\varphi \wedge wp[\![P]\!](X)) \vee (\neg\varphi \wedge F))}_{\text{loop characteristic function } \Phi_F(X)}$$

does exist!

- The function $\Phi_F : \mathbb{P} \to \mathbb{P}$ is Scott continuous on $(\mathbb{P}, \sqsubseteq)$.
- Kleene's fixed point theorem yields: $\text{lfp}\,\Phi_F = \sup_{n\in\mathbb{N}} \Phi_F^n(\text{false})$.
- $\Phi_F^n(\text{false})$ denotes the wp of running the loop n times starting from $\varnothing$, the empty set of states.

“Dijkstra’s weakest preconditions go random”

WEAKEST PRE-EXPECTATIONS



Dexter Kozen, Annabelle McIver, and Carroll Morgan

From predicates to quantities

- Let program P be:

```
x := x+5 [4/5] x := 10
```

The expected value of `x` on P's termination is:

initial value of x

$$\frac{4}{5} \cdot (x + 5) + \frac{1}{5} \cdot 10 \;=\; \frac{4x}{5} + 6$$

From predicates to quantities

- Let program P be:

```
x := x+5 [4/5] x := 10
```

The expected value of x on P's termination is:

$$\frac{4}{5} \cdot (x+5) + \frac{1}{5} \cdot 10 \;=\; \frac{4x}{5} + 6$$

- The probability that x = 10 on P's termination is:

$$\frac{4}{5} \cdot \underbrace{[x{+}5 = 10]}_{\text{Iverson brackets}} + \frac{1}{5} \cdot 1 \;=\; \frac{4 \cdot [x = 5] + 1}{5}$$

Expectations

The set of expectations[2] (read: random variables):

$$\mathbb{E} = \left\{ f \mid f : \underbrace{\mathbb{S}}_{\text{states}} \to \mathbb{R}_{\geq 0} \cup \{\infty\} \right\}$$

[2] $\neq$ expectations in probability theory.

Expectations

The set of expectations[2] (read: random variables):

$$\mathbb{E} = \left\{ f \mid f : \underbrace{\mathbb{S}}_{\text{states}} \to \mathbb{R}_{\geq 0} \cup \{\infty\} \right\}$$

Examples: $[x = 5]$ $\frac{4x}{5} + 6$ $\frac{4 \cdot [x=5]+1}{5}$ $\mathbf{1}$ $x^2 + \sqrt{(y+1)} \ldots$

[2] $\neq$ expectations in probability theory.

Expectations

The set of expectations[2] (read: random variables):

$$\mathbb{E} = \left\{ f \mid f : \underbrace{\mathbb{S}}_{\text{states}} \to \mathbb{R}_{\geq 0} \cup \{\infty\} \right\}$$

Examples: $[x = 5]$ $\frac{4x}{5} + 6$ $\frac{4 \cdot [x=5]+1}{5}$ $\mathbf{1}$ $x^2 + \sqrt{(y+1)} \ldots$

$(\mathbb{E}, \sqsubseteq)$ is a complete lattice where $f \sqsubseteq g$ if and only if $\forall s \in \mathbb{S}.\, f(s) \leq g(s)$

expectations are the quantitative analogue of predicates

[2] $\neq$ expectations in probability theory.

Weakest pre-expectations

For program P, let $wp[\![P]\!] : \mathbb{E} \to \mathbb{E}$ an expectation transformer

$g = wp[\![P]\!](f)$ is P's weakest pre-expectation w.r.t. post-expectation f iff

the expected value of f after executing P on input s equals $g(s)$

Weakest pre-expectations

For program P, let $wp[\![P]\!] : \mathbb{E} \to \mathbb{E}$ an expectation transformer

$g = wp[\![P]\!](f)$ is P's weakest pre-expectation w.r.t. post-expectation f iff

the expected value of f after executing P on input s equals $g(s)$

Examples:

For P:: `x := x+5 [4/5] x := 10` we have:

$$wp[\![P]\!](x) = \frac{4x}{5} + 6 \quad \text{and} \quad wp[\![P]\!]([x = 10]) = \frac{4 \cdot [x = 5] + 1}{5}$$

expected value of x

prob. x = 10

Weakest pre-expectations

For program P, let $wp[\![P]\!] : \mathbb{E} \to \mathbb{E}$ an expectation transformer

$g = wp[\![P]\!](f)$ is P's weakest pre-expectation w.r.t. post-expectation f iff

the expected value of f after executing P on input s equals $g(s)$

Examples:

For P:: `x := x+5 [4/5] x := 10`, we have:

$$wp[\![P]\!](x) = \frac{4x}{5} + 6 \quad \text{and} \quad wp[\![P]\!]([x = 10]) = \frac{4 \cdot [x = 5] + 1}{5}$$

$wp[\![P]\!]([\varphi])$ is the probability of predicate φ on P's termination

$wp[\![P]\!](\mathbf{1})$ is P's termination probability

How to obtain wp for a program?

Syntax probabilistic program P	Semantics $wp[\![P]\!](f)$
`skip`	f
$x := E$	$f[x := E]$
$x :\approx \mu$	$\sum_{v} f(s[x := v]) \cdot \mu_s(v)$
$P; Q$	$wp[\![P]\!](wp[\![Q]\!](f))$

substitution

backwards!

Example program P : $x :\approx \text{unif}[y+x, 2\cdot y]$

post-expectation f : $[x > b]$

initial state s : $s(x) = 0$, $s(y) = 4$

What is $\text{wp}[\![P]\!](f)(s)$?

$$\text{wp}[\![P]\!](f)(s) = \sum_{v \in \mathbb{Q}} \mu_s(v) \cdot f(s[x := v])$$

$$\mu_s(v) =$$

How to obtain wp for a program?

Syntax probabilistic program P	Semantics $wp[\![P]\!](f)$
`skip`	f
$x := E$	$f[x := E]$
$x :\approx \mu$	$\sum_{v \in \mathbb{Q}} f(s[x := v]) \cdot \mu_s(v)$
$P; Q$	$wp[\![P]\!](wp[\![Q]\!](f))$
`if` (φ) P `else` Q	$[\varphi] \cdot wp[\![P]\!](f) + [\neg\varphi] \cdot wp[\![Q]\!](f)$
$P\,[p]\,Q$	$p \cdot wp[\![P]\!](f) + (1-p) \cdot wp[\![Q]\!](f)$

For $f \in \mathbb{E}$ and $c \in \mathbb{R}_{\geq 0}$, $(c \cdot f)(s) = c \cdot f(s)$

For $f, g \in \mathbb{E}$, let $(f + g)(s) = f(s) + g(s)$

Examples

1. Consider again program P:

```
x := x+5 [4/5] x := 10
```

For $f = x$, we have:

$$wp[\![P]\!](x) = \tfrac{4}{5} \cdot wp[\![x := 5]\!](x) + \tfrac{1}{5} \cdot wp[\![x := 10]\!](x)$$

$$= \tfrac{4}{5} \cdot (x+5) + \tfrac{1}{5} \cdot 10 \;=\; \boxed{\frac{4x}{5} + 6}$$

Examples

1. Consider again program P:

```
x := x+5 [4/5] x := 10
```

For $f = x$, we have:

$$wp[\![P]\!](x) = \tfrac{4}{5} \cdot wp[\![x := 5]\!](x) + \tfrac{1}{5} \cdot wp[\![x := 10]\!](x)$$

$$= \tfrac{4}{5} \cdot (x+5) + \tfrac{1}{5} \cdot 10 \;=\; \boxed{\frac{4x}{5} + 6}$$

2. For program P (again) and $f = [x = 10]$, we have:

$$wp[\![P]\!]([x=10]) \;=\; \tfrac{4}{5} \cdot wp[\![x := x+5]\!]([x=10]) + \tfrac{1}{5} \cdot wp[\![x := 10]\!]([x=10])$$

$$= \tfrac{4}{5} \cdot [x+5 = 10] + \tfrac{1}{5} \cdot [10 = 10]$$

$$= \boxed{\frac{4 \cdot [x = 5] + 1}{5}}$$

Loops

$$wp[\![\text{while } (\varphi)\,\{\,P\,\}]\!](f) \;=\; \text{lfp}\, X.\; \underbrace{([\varphi]\cdot wp[\![P]\!](X) \;+\; [\neg\varphi]\cdot f)}_{\text{loop characteristic function } \Phi_f(X)}$$

Loops

$$wp[\![\text{while } (\varphi)\,\{\,P\,\}]\!](f) \;=\; \text{lfp}\, X.\; \underbrace{([\varphi]\cdot wp[\![P]\!](X) \;+\; [\neg\varphi]\cdot f)}_{\text{loop characteristic function } \Phi_f(X)}$$

- Function $\Phi_f : \mathbb{E} \to \mathbb{E}$ is Scott continuous on $(\mathbb{E}, \sqsubseteq)$
- By Kleene's fixed point theorem: $\boxed{\text{lfp}\ \Phi_f \;=\; \sup_{n\in\mathbb{N}} \Phi_f^n(\mathbf{0})}$
- $\Phi_f^n(\mathbf{0})$ is f's expected value after n times running P, starting in $\mathbf{0}$

The good, the bad, and the ugly

A B

Duelling cowboys

a, b ∈ [0,1]

```
int cowboyDuel(float a, b) {
  int t := A
  bool c := true;
  while (c) {
    if (t = A) {
      (c := false [a] t := B);
    } else {
      (c := false [b] t := A);
    }
  }
  return t;
}
```

Duelling cowboys

```
int cowboyDuel(float a, b) { // 0 < a < 1, 0 < b < 1
   int t := A
   bool c := true;
   while (c) {
     if (t = A) {
         (c := false [a] t := B); // A shoots B with prob. a
     } else {
         (c := false [b] t := A); // B shoots A with prob. b
     }
   }
return t; // the survivor
}
```

Cowboy A wins the duel with probability ?

Computing survival probabilities

the probability that cowboy A wins the duel:

$$= \sum_{i=0}^{\infty} \left((1-a)\cdot(1-b)\right)^i \cdot a$$

$$= (* \text{ geometric series } *)$$

$$\frac{a}{a+b-ab}$$

HOW TO TREAT LOOPS?

Upper bounds by inductive invariants

Recall:

$$wp[\![\text{while } (\varphi)\{ P \}]\!](f) \;=\; \text{lfp}\, X.\, \underbrace{([\varphi] \cdot wp[\![P]\!](X) \;+\; [\neg\varphi] \cdot f)}_{\Phi_f(X)}$$

By Park's lemma: for while$(\varphi)\{P\}$ and expectations f and I:

$$\underbrace{\Phi_f(I) \sqsubseteq I}_{\text{“upper” invariant } I} \quad \text{implies} \quad \underbrace{wp[\![\text{while}(\varphi)\{P\}]\!](f)}_{\text{lfp}\,\Phi_f} \sqsubseteq I$$

→ such I is an inductive invariant

Upper bounds

Recall:

$$wp[\![\text{while } (\varphi)\,\{\,P\,\}]\!](f) \;=\; \text{lfp}\, X.\, \underbrace{([\varphi]\cdot wp[\![P]\!](X) \;+\; [\neg\varphi]\cdot f)}_{\Phi_f(X)}$$

By Park's lemma: for while$(\varphi)\{P\}$ and expectations f and I:

$$\underbrace{\Phi_f(I) \sqsubseteq I}_{\text{“upper” invariant } I} \quad \text{implies} \quad \underbrace{wp[\![\text{while}(\varphi)\{P\}]\!](f)}_{\text{lfp}\,\Phi_f} \sqsubseteq I$$

Example: `while(c = 0) { x++ [p] c := 1 }`

$I \;=\; x + [c = 0]\cdot \frac{p}{1-p}$ is an “upper”-invariant w.r.t. $f = x$

$P::$ $\quad \text{while}\,(c=0)\ \{\ x{+}{+}\ [p]\ c:=1\ \}$

claim $\quad \boxed{I = x + [c=0]\cdot \frac{p}{1-p}}$ is a wp-superinvariant wrt. $f = x$

$\Phi_f(I) \sqsubseteq I$

$$\Phi_f(I) = [c\neq 0]\cdot x + [c=0]\ \text{wp}(x{+}{+}\ [p]\ c:=1, I)$$

$$= [c\neq 0]\cdot x + [c=0]\cdot \Big(\underbrace{p\cdot\big((x+1) + [c=0]\tfrac{p}{1-p}\big)}_{I[x{+}{+}]} + \underbrace{(1-p)\cdot x}_{I[c:=1]} \Big)$$

$$= x + [c=0]\Big(\underbrace{p + \frac{p^2}{1-p}}\Big)$$

$$= \frac{p}{1-p}$$

$$\Phi_f(I) = \underbrace{x + [c=0]\frac{p}{1-p}}_{= I} \implies$$

only equalities used, so $\Phi_f(I) = I$

i.e. I is a fixed point of Φ_f

Lower bounds by loop unrolling

$$wp[\![\text{while } (\varphi)\{ P \}]\!](f) \;=\; \text{lfp}\, X.\, \Phi_f(I)$$

$$\text{lfp}\ \Phi_f \;=\; \sup_{n\in\mathbb{N}} \Phi_f^n(\mathbf{0})$$

so: $\Phi_f^0(0), \Phi_f^1(0), \Phi_f^2(0), \Phi_f^3(0) \ldots \sqsubseteq \text{lfp}\ \Phi_f$

loop unrollings

wp [[loop]]

$$\Phi_f^k(0) = wp[\![\text{while}^k(e)\{P\}]\!](f)$$

Lower bounds : OST Proof Rule

$$\big(I \sqsubseteq \Phi_f(I) \;\wedge\; \text{side conditions}\big) \quad \textit{implies} \quad I \sqsubseteq \mathsf{lfp}\,\Phi_f$$

Lower bounds : OST Proof Rule

$$\big(I \sqsubseteq \Phi_f(I) \;\wedge\; \text{side conditions}\big) \quad \textit{implies} \quad I \sqsubseteq \mathrm{lfp}\,\Phi_f$$

where the side conditions for the loop while$(\varphi)\{P\}$ are:

1. the loop is PAST, and
2. for any $s \vDash \varphi$, $\quad wp[\![P]\!](|I(s) - I|)(s) \;\leq\; c \quad$ for some $c \in \mathbb{R}_{\geq 0}$

Lower bounds : OST Proof Rule

$$\big(I \sqsubseteq \Phi_f(I) \;\wedge\; \text{side conditions}\big) \quad \textit{implies} \quad I \sqsubseteq \mathrm{lfp}\,\Phi_f$$

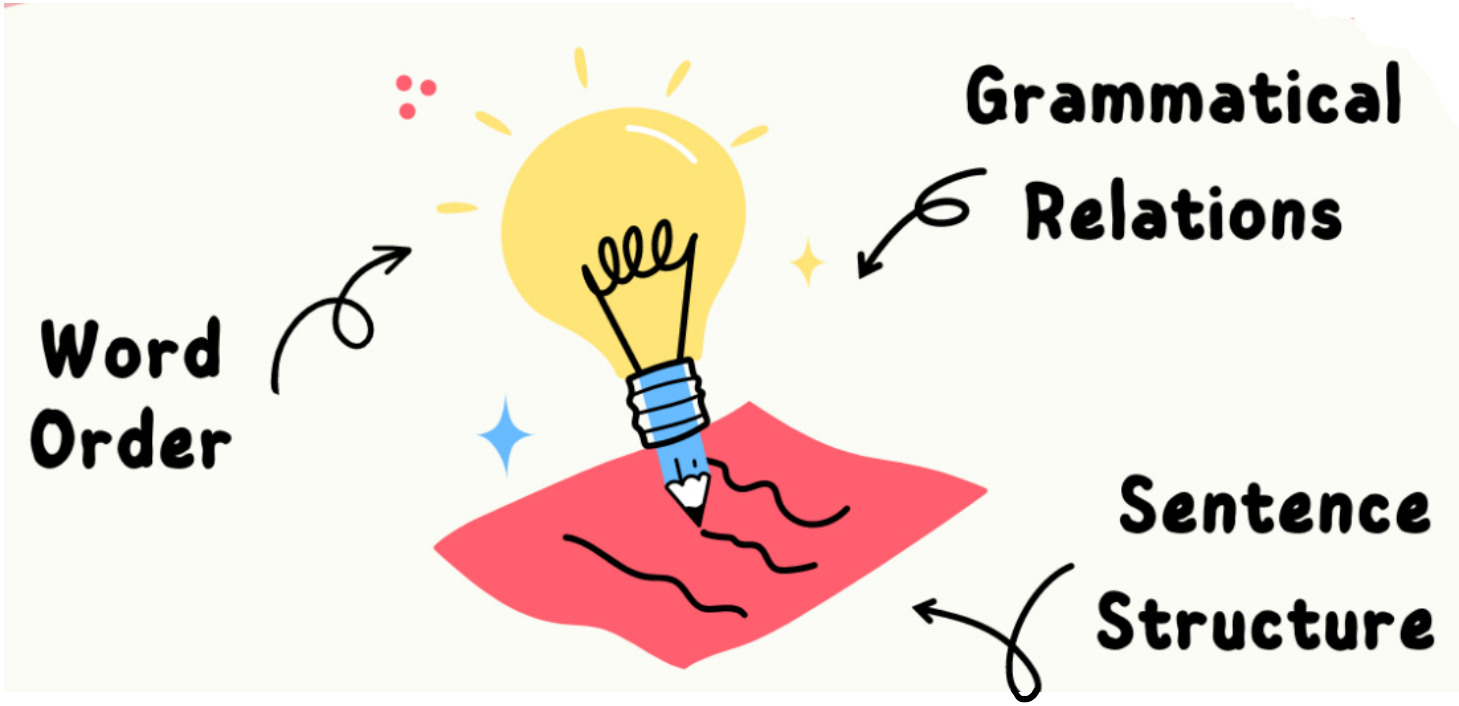
where the side conditions for the loop $\mathrm{while}(\varphi)\{P\}$ are:

1. the loop is PAST, and
2. for any $s \vDash \varphi$, $\underbrace{wp[\![P]\!](|I(s) - I|)(s) \;\leq\; c}_{\text{"conditional difference boundedness"}}$ for some $c \in \mathbb{R}_{\geq 0}$

Example. Program: $\texttt{while}(c = 0)\{\, x{+}{+} \,[p]\, c := 1 \,\}$ satisfies the conditions.

$I = x + [c = 0] \cdot \frac{p}{1-p}$ is a "lower"-invariant w.r.t. $f = x$

SYNTAX FOR EXPECTATIONS



Relative complete verification

Ordinary Programs

$F \in$ FO-Arithmetic

implies

$wp[\![P]\!](F) \in$ FO-Arithmetic

$G \Longrightarrow wp[\![P]\!](F)$

is effectively decidable

modulo an oracle for deciding $\Rightarrow$

Probabilistic Programs

$f \in$ SomeSyntax

implies

$wp[\![P]\!](f) \in$ SomeSyntax

$g \sqsubseteq wp[\![P]\!](f)$

is effectively decidable

modulo an oracle for deciding $\sqsubseteq$

between two syntactic expectations.

Relative complete verification

Ordinary Programs

$F \in$ FO-Arithmetic

implies

$wp[\![P]\!](F) \in$ FO-Arithmetic

$G \Longrightarrow wp[\![P]\!](F)$

is effectively decidable

modulo an oracle for deciding $\Rightarrow$

Probabilistic Programs

$f \in$ SomeSyntax

implies

$wp[\![P]\!](f) \in$ SomeSyntax

$g \sqsubseteq wp[\![P]\!](f)$

is effectively decidable

modulo an oracle for deciding $\sqsubseteq$

between two syntactic expectations.

Q: How does the SomeSyntax look like?

Syntax: expressions

- Arithmetic expressions

$$
\begin{array}{lcll}
a & \longrightarrow & r \in \mathbb{Q}_{\geqslant 0} & \text{non-negative rational} \\
 & | & x \in \mathit{Vars} & \mathbb{Q}_{\geqslant 0}\text{-valued variable} \\
 & | & a + a & \text{addition} \\
 & | & a \cdot a & \text{multiplication} \\
 & | & a \dot{-} a & \text{subtraction truncated at zero}
\end{array}
$$

$$
a \dot{-} b = \begin{cases} a - b & \text{if } b \leq a \\ 0 & \text{else} \end{cases}
$$

- Boolean expressions

$$
\begin{array}{lcll}
\varphi & \longrightarrow & a < a & \text{comparing arithmetic expressions} \\
 & | & \varphi \wedge \varphi & \text{conjunction} \\
 & | & \neg \varphi & \text{negation}
\end{array}
$$

Syntax of expectations

- The set Exp of syntactic expectations

$$
\begin{array}{lcll}
f & \longrightarrow & a & \text{arithmetic expressions} \\
 & & \mid [\varphi] \cdot f & \text{guarding} \\
 & & \mid f + f & \text{addition} \\
 & & \mid f \cdot f & \text{multiplication} \\
 & & \mid \sqcup x{:}\, f & \text{supremum over variable } x \\
 & & \mid \sqcap x{:}\, f & \text{infimum over variable } x
\end{array}
$$

Syntax of expectations

- The set Exp of syntactic expectations

$$
\begin{array}{lcll}
f & \longrightarrow & a & \text{arithmetic expressions} \\
 & & \mid [\varphi] \cdot f & \text{guarding} \\
 & & \mid f + f & \text{addition} \\
 & & \mid f \cdot f & \text{multiplication} \\
 & & \mid \sqcup x{:}\, f & \text{supremum over variable } x \\
 & & \mid \sqcap x{:}\, f & \text{infimum over variable } x
\end{array}
$$

- Examples:

$$\boxed{\sqcup x{:}\, [x \cdot x < y] \cdot x}$$

Syntax of expectations

- The set Exp of syntactic expectations

$$
\begin{array}{lcll}
f & \longrightarrow & a & \text{arithmetic expressions} \\
 & & \mid [\varphi] \cdot f & \text{guarding} \\
 & & \mid f + f & \text{addition} \\
 & & \mid f \cdot f & \text{multiplication} \\
 & & \mid \sqsupset x{:}\, f & \text{supremum over variable } x \\
 & & \mid \sqsubset x{:}\, f & \text{infimum over variable } x
\end{array}
$$

- Examples:

$$\sqsupset x{:}\,[x \cdot x < y] \cdot x \;\equiv\; \sqrt{y}$$

Syntax of expectations

- The set Exp of syntactic expectations

$$
\begin{array}{lcll}
f & \longrightarrow & a & \text{arithmetic expressions} \\
 & & \mid [\varphi] \cdot f & \text{guarding} \\
 & & \mid f + f & \text{addition} \\
 & & \mid f \cdot f & \text{multiplication} \\
 & & \mid \sup x\colon f & \text{supremum over variable } x \\
 & & \mid \inf x\colon f & \text{infimum over variable } x
\end{array}
$$

- Examples:

$$\sup x\colon [x \cdot x < y] \cdot x \;\equiv\; \sqrt{y}$$

$$\boxed{\sup z\colon [z \cdot (x+1) = 1] \cdot z \;\equiv\; \frac{1}{x+1}}$$

Examples

- polynomials $y + x^3 + 2x^2 + x - 7$ widely used as templates
- rational functions $\frac{x^2 - 3x + 4}{y^2 \cdot x - 3y + 1}$
- square roots $\sqrt{x}$
- irrational, algebraic and transcendental numbers $\frac{\sqrt{5}-1}{2}$, π, e, ...
- Harmonic numbers $H_k = \sum_{k=1}^{x} \frac{1}{k}$ used in run-time/termination analysis

Semantics of syntactic expressions

Recall that state $s : \mathit{Vars} \to \mathbb{Q}_{\geqslant 0}$.

$$[\![a]\!]^s = [\![a]\!]^s$$

$$[\![[\varphi] \cdot f]\!]^s = \begin{cases} [\![f]\!]^s & \text{if } [\![\varphi]\!]^s = \mathit{true} \\ 0 & \text{otherwise} \end{cases}$$

$$[\![f + g]\!]^s = [\![f]\!]^s + [\![g]\!]^s$$

$$[\![\mathfrak{g} \cdot f]\!]^s = [\![\mathfrak{g}]\!]^s \cdot [\![f]\!]^s$$

$$[\![\mathfrak{S}\, x : f]\!]^s = \sup \left\{ [\![f]\!]^{s[x \mapsto r]} \mid r \in \mathbb{Q}_{\geqslant 0} \right\}$$

$$[\![\mathfrak{l}\, x : f]\!]^s = \inf \left\{ [\![f]\!]^{s[x \mapsto r]} \mid r \in \mathbb{Q}_{\geqslant 0} \right\}$$

Expressiveness

[Batz, K. *et al*, POPL 2021]

The set Exp of syntactic expectations is expressive.

For all pGCL programs P and $f \in \text{Exp}$ it holds:

$$wp[\![P]\!]([\![f]\!]) \quad = \quad [\![g]\!]$$

for some syntactic expectation $g \in \text{Exp}$.

Expressiveness

The set Exp of syntactic expectations is expressive.

For all pGCL programs P and $f \in \text{Exp}$ it holds:

$$wp[\![P]\!]([\![f]\!]) \quad = \quad [\![g]\!]$$

for some syntactic expectation $g \in \text{Exp}$.

Expressiveness does not mean decidability, e.g.,

for $f, g \in \text{Exp}$, does $[\![g]\!] \sqsubseteq wp[\![P]\!]([\![f]\!])$ is undecidable

“How long does your program take on average?”

EXPECTED RUNTIMES

The runtime of a probabilistic program

The runtime of a probabilistic program depends

on the input and

on the internal randomness of the program.

Expected runtimes

Expected run-time of program P on input s:

$$\sum_{i=1}^{\infty} i \cdot Pr\left(\begin{array}{c} \text{“}P \text{ terminates after} \\ i \text{ loops on input } s\text{”} \end{array} \right)$$

$ert[\![P]\!](t)(s)$ = expected runtime of P on s where t is runtime after P

Counting loops is incorrect

expected runtime ∞

```
while(true) { skip; x++ }
```

- Post: x, as seemingly x counts #loop iterations
- Characteristic function: $\Phi_x(Y) = Y(x \mapsto x+1)$
- Candidate upper bound: $I = \mathbf{0}$
- Induction: $\Phi_x(I) = \mathbf{0}(x \mapsto x+1) = \mathbf{0} = I \sqsubseteq I$
- By Park induction: $\Phi_x(I) \sqsubseteq I$ implies $wp[\![\text{loop}]\!](x) \sqsubseteq I$

We — wrongly — get runtime $\mathbf{0}$. wp is unsound for expected runtimes.

Expected run-time transformer

Solution: slightly adapt definition of wp

$$ert\,[\![\,\text{while}\,(\varphi)\,\{P\}\,]\!]\,(t) \;=\; lfp\;X.\,\Big(\,1 + [\varphi]\cdot ert\,[\![P]\!](X) + [\neg\varphi]\cdot t\,\Big)$$

↑ "counts" the # loop iterations

That's all!

$ert[\![P]\!](t)(s)$ = expected runtime of P on s where t is runtime after P

Positive almost-sure termination

For every pGCL program P and input state s:

$$\underbrace{ert[\![P]\!](\mathbf{0})(s) < \infty}_{\text{positive a.s-termination on } s} \quad \text{implies} \quad \underbrace{wp[\![P]\!](\mathbf{1})(s) = \mathbf{1}}_{\text{a.s.–termination on } s}$$

Moreover:

$$\underbrace{ert[\![P]\!](\mathbf{0}) < \infty}_{\text{universal positive a.s–termination}} \quad \text{implies} \quad \underbrace{wp[\![P]\!](\mathbf{1}) = \mathbf{1}}_{\text{universal a.s.–termination}}$$

Coupon collector's problem

ON A CLASSICAL PROBLEM OF PROBABILITY THEORY

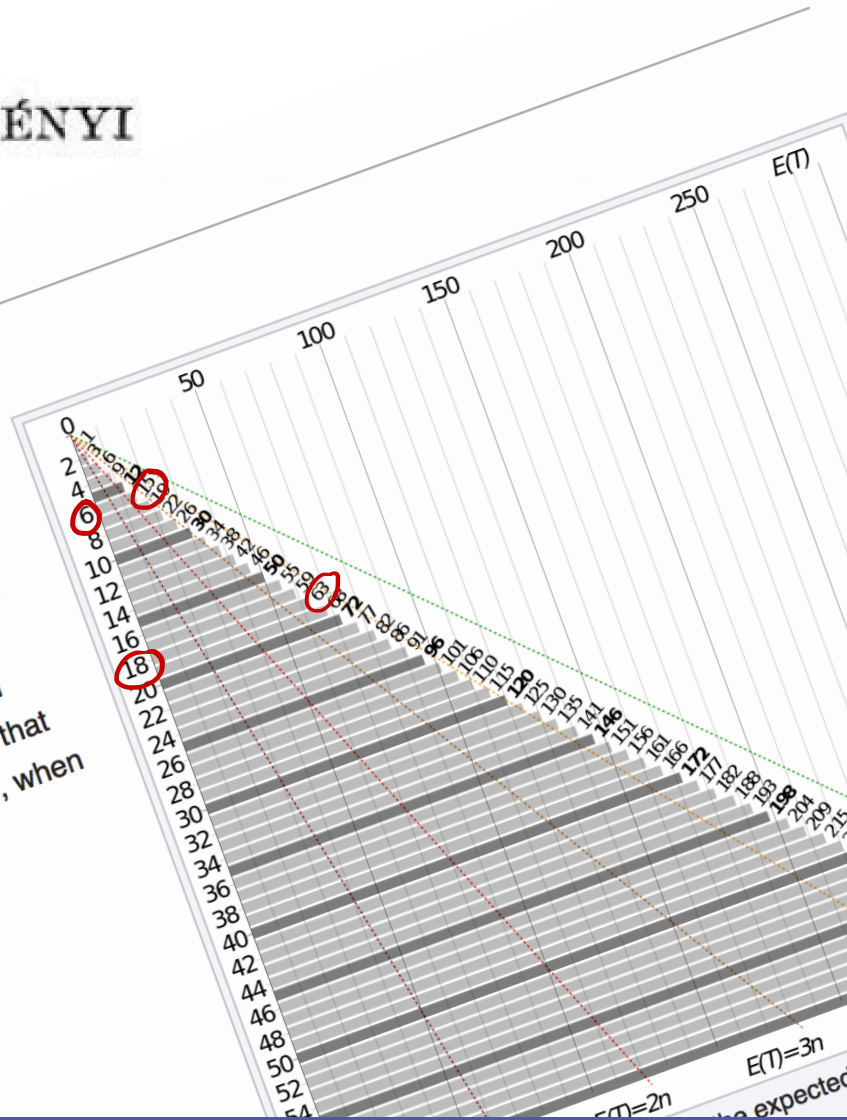
by

P. ERDŐS and A. RÉNYI

Coupon collector's problem

From Wikipedia, the free encyclopedia

In probability theory, the **coupon collector's problem** describes the "collect all coupons and win" contests. It asks the following question: Suppose that there is an urn of n different coupons, from which coupons are being collected, equally likely, with replacement. What is the probability that more than t sample trials are needed to collect all n coupons? An alternative statement is: Given n coupons, how many coupons do you expect you need to draw with replacement before having drawn each coupon at least once? The mathematical analysis of the problem reveals that the expected number of trials needed grows as $\Theta(n \log(n))$.[1] For example, when there are 50 coupons, about 225[2] trials to collect all 50 coupons.



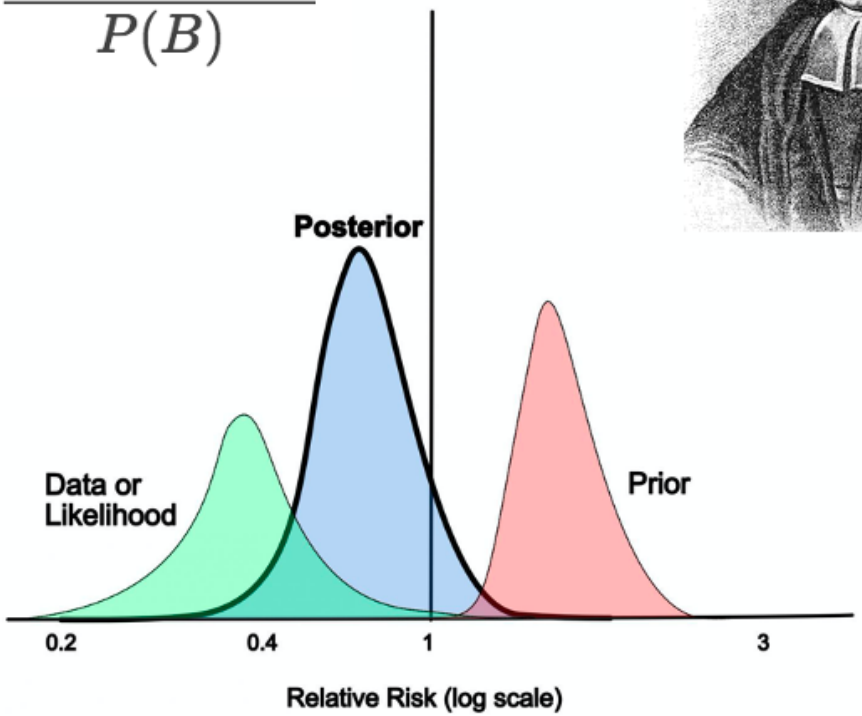
Coupon collector's problem

```
cp := [0,...,0]; // no coupons yet
i := 1; // coupon to be collected next
x := 0: // number of coupons collected
while (x < N) {
   while (cp[i] != 0) {
       i := uniform(1..N) // next coupon
   }
   cp[i] := 1;  // coupon i obtained
   x++;   // one coupon less to go
}
```

The expected runtime of this program is in $\Theta(N \cdot \log N)$.

BAYESIAN INFERENCE

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$



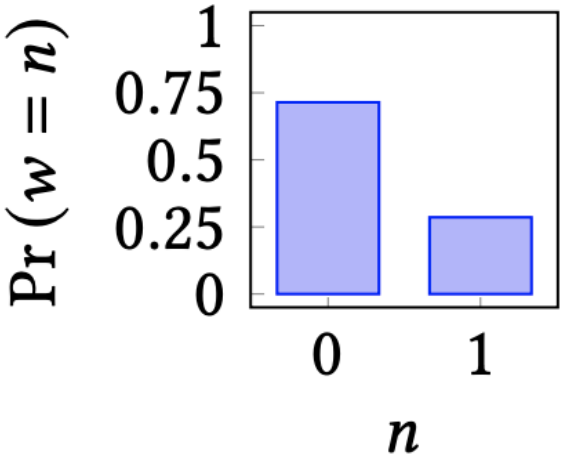
Bayesian learning by example

```
{ w := 0 } [5/7] { w := 1 };
if (w = 0) { c := poisson(6) }
else { c := poisson(2) };
observe (c = 5)
```

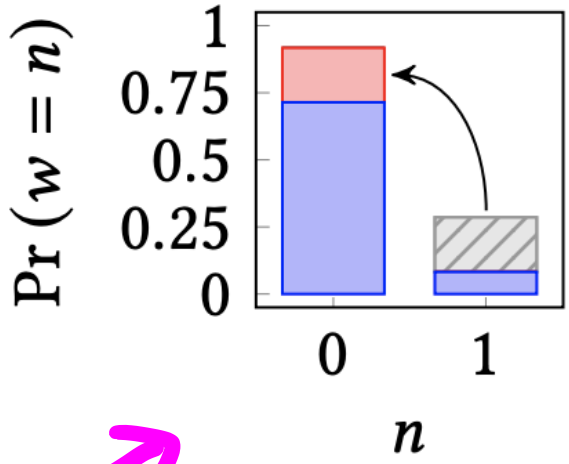
What is the posterior distribution of w (weekend or not)?

Bayesian learning by example

```
{ w := 0 } [5/7] { w := 1 };
if (w = 0) { c := poisson(6) }
else { c := poisson(2) };
observe (c = 5)
```



prior

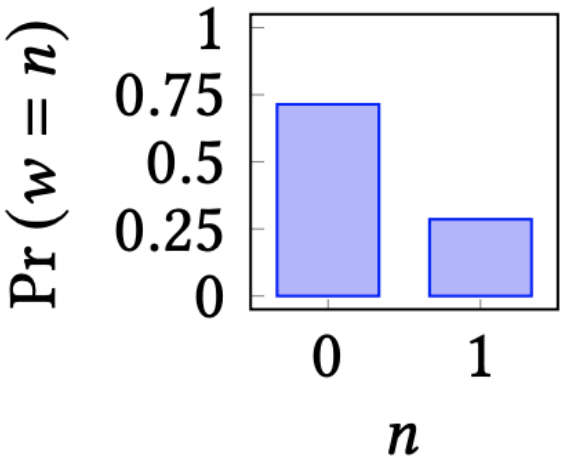


posterior

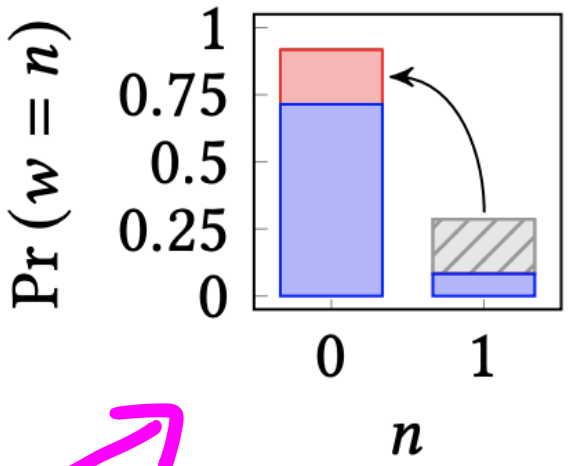
Bayesian learning by example

```
{ w := 0 } [5/7] { w := 1 };
if (w = 0) { c := poisson(6) }
else { c := poisson(2) };
observe (c = 5)
```

prior



posterior



Probability mass is normalised by the probability of feasible runs

Learning

The probability of feasible program runs:

$$wp[\![P]\!](\mathbf{1}) \;=\; 1 - Pr\{\, P \text{ violates an observation} \,\}$$

Learning

The probability of feasible program runs:

$$wp[\![P]\!](\mathbf{1}) \;=\; 1 - Pr\{\, P \text{ violates an observation} \,\}$$

Weakest pre-expectation of observations:

$$wp[\![\text{observe } \varphi]\!](f) \;=\; [\varphi] \cdot\; f$$

Learning

The probability of feasible program runs:

$$wp[\![P]\!](\mathbf{1}) \;=\; 1 - Pr\{\, P \text{ violates an observation} \,\}$$

Weakest pre-expectation of observations:

$$wp[\![\text{observe } \varphi]\!](f) \;=\; [\varphi] \cdot f$$

Normalisation:

$$\frac{wp[\![P]\!](f)}{wp[\![P]\!](\mathbf{1})}$$

Learning

The probability of feasible program runs:

$$wp[\![P]\!](\mathbf{1}) \;=\; 1 - Pr\{\, P \text{ violates an observation} \,\}$$

Weakest pre-expectation of observations:

$$wp[\![\text{observe } \varphi]\!](f) \;=\; [\varphi] \cdot\; f$$

Normalisation:

$$\frac{wp[\![P]\!](f)}{wp[\![P]\!](\mathbf{1})}$$

Fine point: under possible program divergence: $\dfrac{wp[\![P]\!](f)}{wlp[\![P]\!](\mathbf{1})}$

The piranha problem

[Tijms, 2004]

One fish is contained within the confines of an opaque fishbowl. The fish is equally likely to be a piranha or a goldfish. A sushi lover throws a piranha into the fish bowl alongside the other fish. Then, immediately, before either fish can devour the other, one of the fish is blindly removed from the fishbowl. The fish that has been removed from the bowl turns out to be a piranha. What is the probability that the fish that was originally in the bowl by itself was a piranha?

The piranha program

```
f1 := gf [0.5] f1 := pir;
f2 := pir;
s := f1 [0.5] s := f2;
observe (s = pir)
```

What is the probability that the original fish in the bowl was a piranha?

$$\frac{wp[\![P]\!](f)}{wp[\![P]\!](1)}$$

$f = [f_1 = pir]$

```
f1 := gf [0.5] f1 := pir;
f2 := pir;
s := f1 [0.5] s := f2;
observe (s = pir)
```

} P

$f = [f_1 = pir]$

$[f_1 = pir]$

observe $(s = pir)$

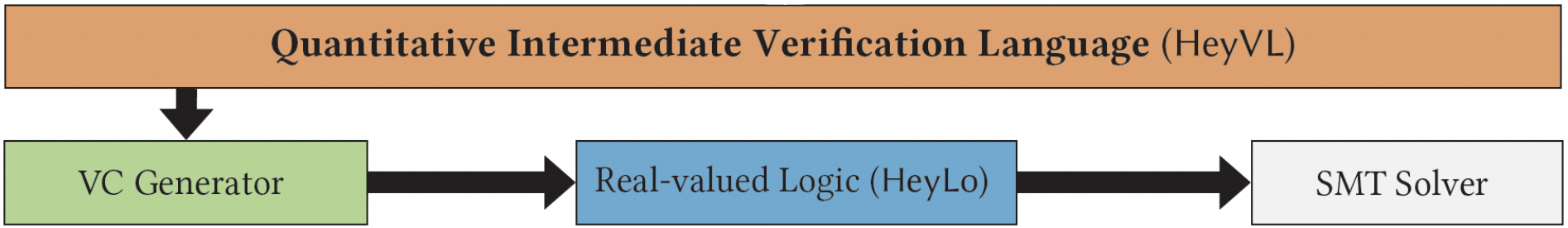
$[s = pir] \cdot [f_1 = pir]$

$s := f_1 \ [1/2] \ s := f_2$

$\frac{1}{2} [s = f_1][f_1 = pir] + \frac{1}{2} [s = f_2][f_1 = pir]$

Caesar

A Deductive Verifier for Probabilistic Programs



Caesar: A verification infrastructure for probabilistic programs

`caesarverifier.org`