Dijkstra Goes Random

Weakest-Precondition-Reasoning on Probabilistic Programs

Joost-Pieter Katoen

The 20th KeY Symposium, July 2024

Probabilistic programs

Programs with random assignments and conditioning

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

probabilistic branching

$\Pr\{w=0\} = \frac{5}{7}$ $\Pr\{w=1\} = \frac{2}{7}$

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

Probabilistic programs

Programs with random assignments and conditioning

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

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

Probabilistic programs

Programs with random assignments and conditioning

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

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

Probabilistic programs

Programs with random assignments and conditioning

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

They encode:

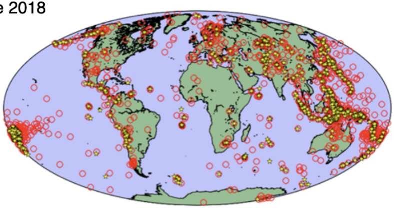
- randomised algorithms
- probabilistic graphical models beyond Bayes' networks
- controllers for autonomous systems
- security mechanisms
- ......

"Probabilistic programming aims to make probabilistic modeling and machine learning accessible to the programmer."[1]

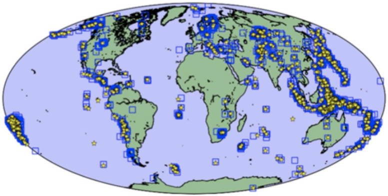
[1] [Gordon, Henzinger, Nori and Rajamani, FOSE 2014]

"Real" Examples

before 2018



since 2018

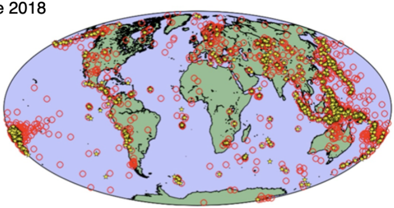


the UN diagnoses seismic events using probabilistic programs

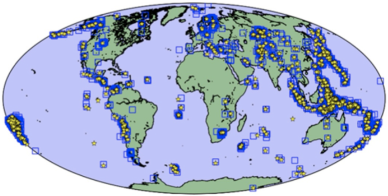
[Arora *et al*, Bull. Seism. 2017]

"Real" Examples

before 2018

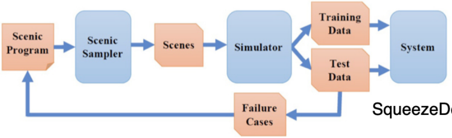


since 2018



the UN diagnoses seismic events using probabilistic programs

[Arora *et al*, Bull. Seism. 2017]



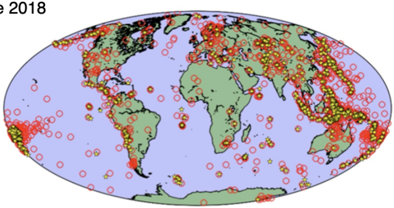
SqueezeDet

SCENIC generates more effective training sets

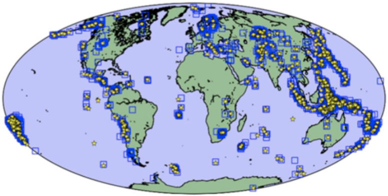
[Fremont *et al*, Mach. Learn. 2023]

“Real” Examples

before 2018

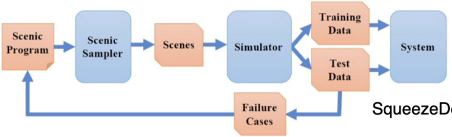


since 2018



the UN diagnoses seismic events using probabilistic programs

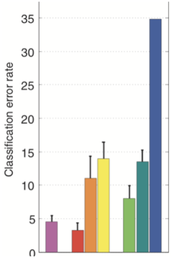
[Arora *et al*, Bull. Seism. 2017]



SqueezeDet

SCENIC generates more effective training sets

[Fremont *et al*, Mach. Learn. 2023]



STAN, PyMC, Edward, Pyro, ProbLog, WebPPL

humans

programs

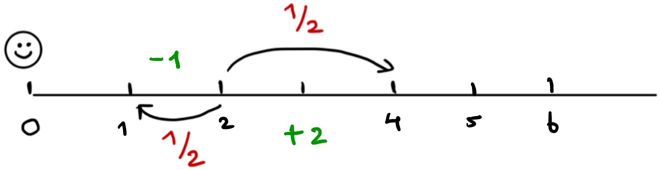
neural networks

[Lake *et al*, Science 2015]

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?

Even if all loops are bounded **[Flajolet *et al*, 2009]**

```
x := geometric(1/4);
y := geometric(1/4);
t := x+y+1 [5/9] t := x+y;
r := 1;
for i in 1..3 {
   s := 0
   for j in 1..2t {
       s := s+1 [1/2] skip
   }
   r := (s == t)
}
```

Even if all loops are bounded [Flajolet *et al*, 2009]

```
x := geometric(1/4);
y := geometric(1/4);
t := x+y+1 [5/9] t := x+y;
r := 1;
for i in 1..3 {
   s := 0
   for j in 1..2t {
       s := s+1 [1/2] skip
   }
   r := (s == t)
}
```

What is the probability that r equals one on termination?

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

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

;

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

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

Finite termination time!
PAST.

Expected runtime of these programs in sequence?

Positive AST

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

;

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

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

Finite termination time!
PAST.

Expected runtime of these programs in sequence?

∞

$\mathrm{PAST}(P) \wedge \mathrm{PAST}(Q) \not\Rightarrow \mathrm{PAST}(P\,;Q)$

Our objective

A powerful, simple proof calculus for probabilistic programs.

At the source code level.

No “descend” into the underlying probabilistic model.

Our objective

A powerful, simple proof calculus for probabilistic programs.

At the source code level.

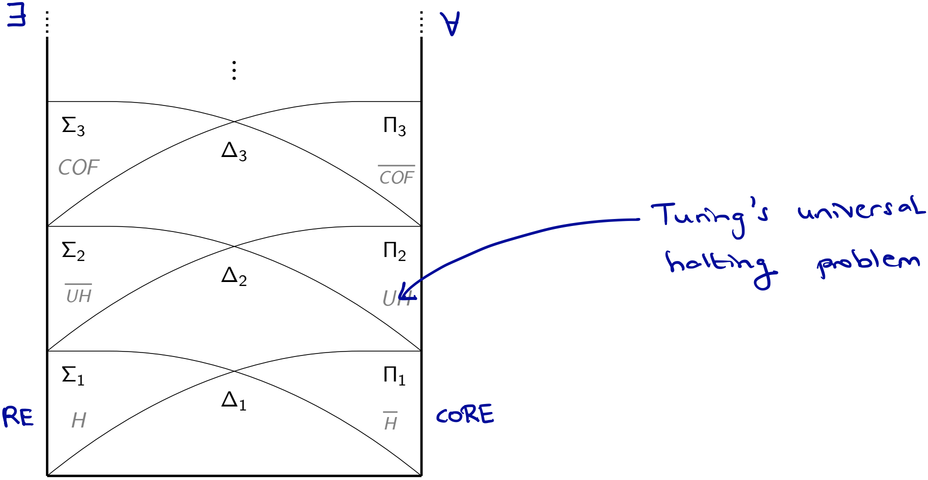
No “descend” into the underlying probabilistic model.

Push automation as much as we can.

This is a true challenge: undecidability!

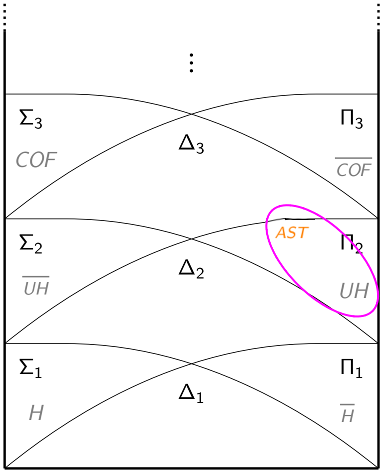
Typically “more undecidable” than deterministic programs

Probabilistic programs are “more undecidable”



arithmetic hierarchy

Probabilistic programs are “more undecidable”



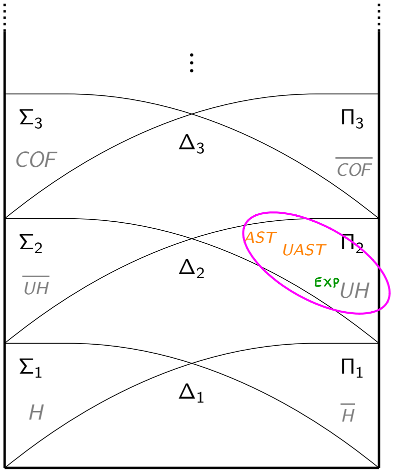
arithmetic hierarchy

AST for one input

is as hard as

the halting problem for all inputs

Probabilistic programs are "more undecidable"



arithmetic hierarchy

AST for one input

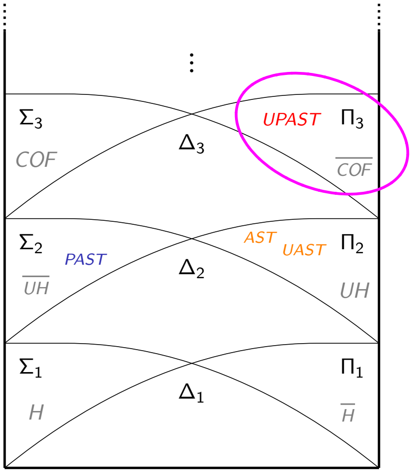
is as hard as

the halting problem for all inputs

is as hard as

computing expected outcomes

Probabilistic programs are "more undecidable"



arithmetic hierarchy

AST for one input

is as hard as

the halting problem for all inputs

is as hard as

computing expected outcomes

but

deciding finite expected runtime?

is "even more undecidable"

[Kaminski, K., MFCS 2015]

Roadmap of this talk

Part 1

- Probabilistic weakest preconditions

Part 2

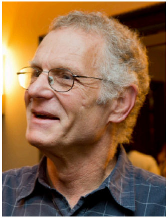
- Proof rules for probabilistic loops

Part 3

- Relative completeness and automation

"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):

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

[2] ≠ expectations in probability theory.

Expectations

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

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

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

[2] ≠ expectations in probability theory.

Expectations

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

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

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

$(\mathbb{E}, \sqsubseteq)$ 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 $wp[\![P]\!] : \mathbb{E} \rightarrow \mathbb{E}$ 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 $wp[\![P]\!] : \mathbb{E} \rightarrow \mathbb{E}$ 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

Kozen's duality theorem

$wp[\![P]\!](f)(s)$ is the expected value of f after running P on input s

Let μ_P^s be the distribution over P's final states when P starts in s.

Kozen's duality theorem

$wp[\![P]\!](f)(s)$ is the expected value of f after running P on input s

Let μ_P^s be the distribution over P's final states when P starts in s.

Then for post-expectation f: $\underbrace{wp[\![P]\!](f)(s)}_{\text{"backward"}} = \underbrace{\sum_{t \in \mathbb{S}} \mu_P^s(t) \cdot f(t)}_{\text{"forward"}}$

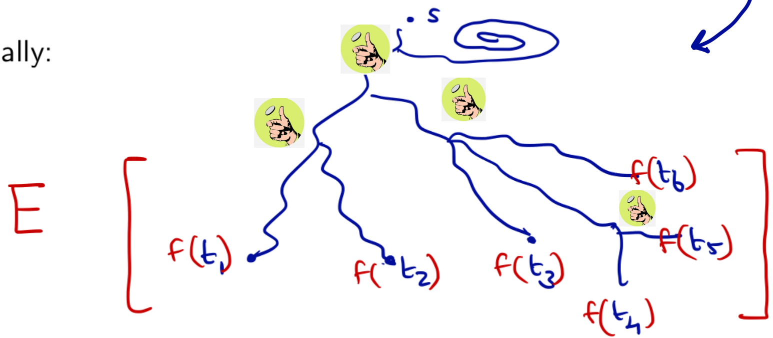
Kozen's duality theorem

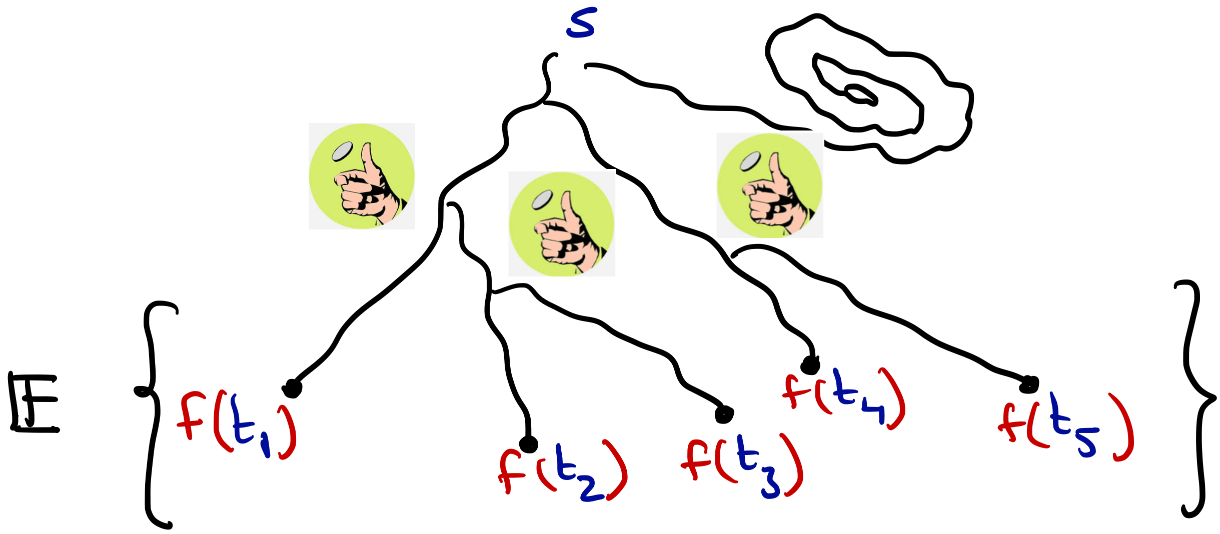
$wp[\![P]\!](f)(s)$ is the expected value of f after running P on input s

Let μ_P^s be the distribution over P's final states when P starts in s.

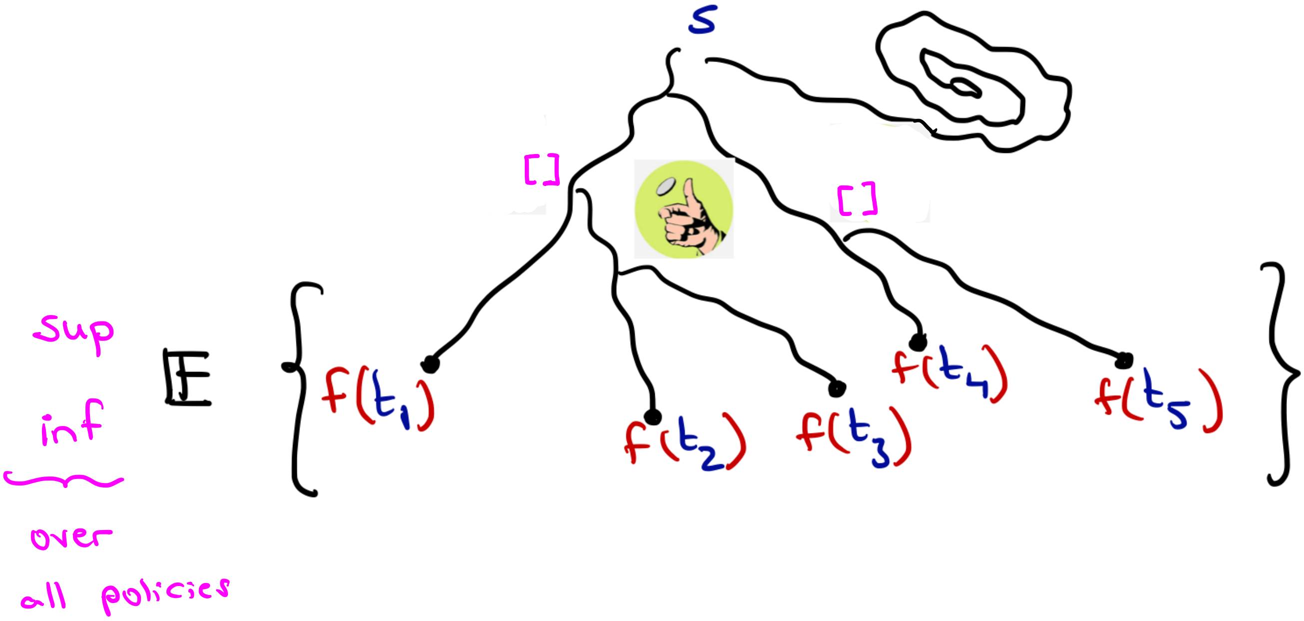
Then for post-expectation f: $\underbrace{wp[\![P]\!](f)(s)}_{\text{"backward"}} = \boxed{\underbrace{\sum_{t \in \mathbb{S}} \mu_P^s(t) \cdot f(t)}_{\text{"forward"}}}$

Pictorially:





s
E
f(t1)
f(t2)
f(t3)
f(t4)
f(t5)



s
sup
inf
over
all policies
E
f(t1)
f(t2)
f(t3)
f(t4)
f(t5)

How to obtain wp for a program?

Syntax probabilistic program P	Semantics $wp[\![P]\!](f)$
skip	f
$x := E$	$f[x := E]$
$x :\approx \mu$	$\lambda s. \int_{\mathbb{Q}} (\lambda v. f(s[x := v]))\, d\mu_s$
$P; Q$	$wp[\![P]\!](wp[\![Q]\!](f))$

substitution

backwards!

How to obtain wp for a program?

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

How to obtain wp for a program?

Syntax probabilistic program P	Semantics $wp[\![P]\!](f)$
`skip`	f
$x := E$	$f[x := E]$
$x :\approx \mu$	$\lambda s. \int_{\mathbb{Q}} (\lambda v. f(s[x := v]))\, d\mu_s$
$P; Q$	$wp[\![P]\!]\,(wp[\![Q]\!](f))$
`if` (φ) P `else` Q	$[\varphi] \cdot wp[\![P]\!](f) + [\neg\varphi] \cdot wp[\![Q]\!](f)$
$P\,[p]\,Q$	$p \cdot wp[\![P]\!](f) + (1-p) \cdot wp[\![Q]\!](f)$
`while` (φ) $\{P\}$	$\mathrm{lfp}\, X.\, \underbrace{(([\varphi] \cdot wp[\![P]\!](X)) + [\neg\varphi] \cdot f)}_{\text{loop characteristic function } \Phi_f(X)}$

where lfp is the least fixed point wrt. the ordering $\sqsubseteq$ on $\mathbb{E}$.

Examples

1. Consider again program P:

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

For $f = x$, we have:

x+5

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

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

Examples

1. Consider again program P:

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

For $f = x$, we have:

x+5

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

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

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

$$\begin{aligned} wp[\![P]\!]([x{=}10]) &= \frac{4}{5}\cdot wp[\![x := x{+}5]\!]([x{=}10]) + \frac{1}{5}\cdot wp[\![x := 10]\!]([x{=}10]) \\ &= \frac{4}{5}\cdot[x{+}5 = 10] + \frac{1}{5}\cdot[10 = 10] \\ &= \boxed{\frac{4\cdot[x = 5] + 1}{5}} \end{aligned}$$

Loops

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

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: $\text{lfp}\;\Phi_f \;=\; \sup_{n\in\mathbb{N}} \Phi_f^n(\mathbf{0})$
- $\Phi_f^n(\mathbf{0})$ is f's expected value after n times running P, starting in $\mathbf{0}$

Examples

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

post-expectation: **1**

```
x := geometric(1/4);
y := geometric(1/4);
t := x+y+1 [5/9] t := x+y;
r := 1;
for i in 1..3 {
   s := 0
   for j in 1..2t {
       s := s+1 [1/2] skip
   }
   r := (s == t)
}
```

Examples

weakest pre-expectation: $\frac{\sqrt{5}-1}{2}$

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

post-expectation: **1**

```
x := geometric(1/4);
y := geometric(1/4);
t := x+y+1 [5/9] t := x+y;
r := 1;
for i in 1..3 {
   s := 0
   for j in 1..2t {
       s := s+1 [1/2] skip
   }
   r := (s == t)
}
```

Examples

weakest pre-expectation: $\frac{\sqrt{5}-1}{2}$

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

post-expectation: **1**

```
x := geometric(1/4);
y := geometric(1/4);
t := x+y+1 [5/9] t := x+y;
r := 1;
for i in 1..3 {
   s := 0
   for j in 1..2t {
       s := s+1 [1/2] skip
   }
   r := (s == t)
}
```

post-expectation: $[\mathbf{r} = \mathbf{1}]$

Examples

weakest pre-expectation: $\frac{\sqrt{5}-1}{2}$

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

post-expectation: **1**

weakest pre-expectation: $\frac{1}{\pi}$

```
x := geometric(1/4);
y := geometric(1/4);
t := x+y+1 [5/9] t := x+y;
r := 1;
for i in 1..3 {
   s := 0
   for j in 1..2t {
       s := s+1 [1/2] skip
   }
   r := (s == t)
}
```

post-expectation: $[\mathbf{r} = \mathbf{1}]$

Probabilistic weakest liberal preconditions

Restrict f to denote probabilities, i.e., $f(s) \leq 1$ for each state s

Probabilistic weakest liberal preconditions

Restrict f to denote probabilities, i.e., $f(s) \leq 1$ for each state s

Loops under wlp:

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

Probabilistic weakest liberal preconditions

Restrict f to denote probabilities, i.e., $f(s) \leq 1$ for each state s

Loops under wlp:

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

Relating weakest liberal preconditions to wp:

$$wlp[\![P]\!](f) \;=\; wp[\![P]\!](f) \;+\; \underbrace{(1 - wp[\![P]\!](\mathbf{1}))}_{\text{probability that } P \text{ diverges}}$$

partial correctness

total correctness

Probabilistic weakest liberal preconditions

Restrict f to denote probabilities, i.e., $f(s) \leq 1$ for each state s

Loops under wlp:

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

Relating weakest liberal preconditions to wp:

$$wlp[\![P]\!](f) \;=\; wp[\![P]\!](f) + \underbrace{(1 - wp[\![P]\!](\mathbf{1}))}_{\text{probability that } P \text{ diverges}}$$

If program P is AST:

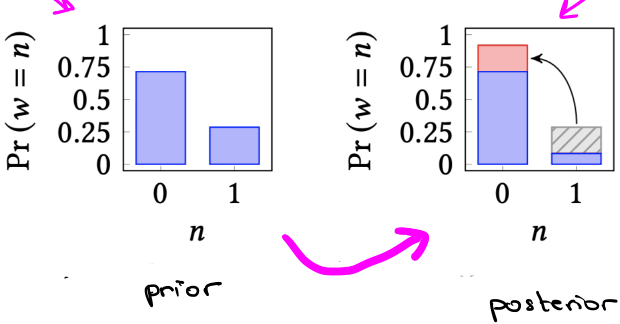
$$wlp[\![P]\!](f) \;=\; wp[\![P]\!](f) + \Big(1 - \underbrace{wp[\![P]\!](\mathbf{1})}_{=\mathbf{1}}\Big) \;=\; wp[\![P]\!](f)$$

Bayesian learning by example

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

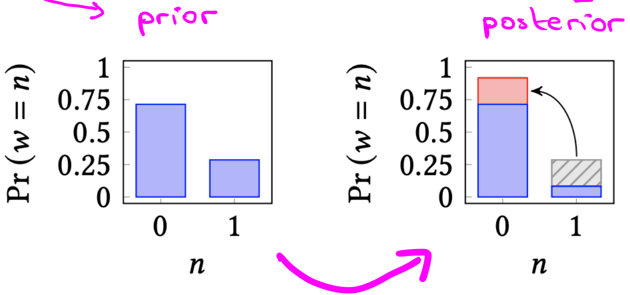
Bayesian learning by example

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



Bayesian learning by example

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



Probability mass is normalised by the probability of feasible runs

Learning [Nori *et al*, AAAI 2014; Olmedo, K., *et al*, TOPLAS 2018]

The probability of feasible program runs:

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

Learning [Nori *et al*, AAAI 2014; Olmedo, K., *et al*, TOPLAS 2018]

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 [Nori *et al*, AAAI 2014; Olmedo, K., *et al*, TOPLAS 2018]

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 [Nori *et al*, AAAI 2014; Olmedo, K., *et al*, TOPLAS 2018]

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: $\dfrac{wp[\![P]\!](f)}{wlp[\![P]\!](\mathbf{1})}$

Extensions of probabilistic wp

- ...... for recursion [LICS 2016]
- ...... for exact inference [TOPLAS 2018]
- ...... for continuous distributions [SETTS 2019]
- ...... for probabilistic separation logic [POPL 2019]
- ...... for weighted programs [OOPSLA 2022]
- ...... for expected runtime analysis [JACM 2018]
- ...... for amortised runtime analysis [POPL 2023]

HOW TO TREAT LOOPS?

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(φ){P} and expectations f and I:

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

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(φ)$\{P\}$ and expectations f and I:

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

Example:

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

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

Lower bounds

[Hark, K. *et al*, POPL 2020]

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

Lower bounds

[Hark, K. *et al*, POPL 2020]

$$\Big(I \sqsubseteq \Phi_f(I) \;\wedge\; \underbrace{\text{side conditions}}\Big) \quad \textit{implies} \quad I \sqsubseteq \operatorname{lfp} \Phi_f$$

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

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

Lower bounds

[Hark, K. *et al*, POPL 2020]

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

where the side conditions for the loop while(φ){P} are:

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

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

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

A proof rule for AST

[McIver, K. *et al*, POPL 2018]

Consider the loop `while`(φ){ *body* } and let:

- $V : \mathbb{S} \to \mathbb{R}_{\geq 0}$ with $[\neg V] = [\neg \varphi]$ — V indicates termination
- $p : \mathbb{R}_{\geq 0} \to (0, 1]$ antitone — probability
- $d : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ antitone — decrease

$$x \leq y \longrightarrow d(y) \leq d(x)$$

A proof rule for AST

[McIver, K. *et al*, POPL 2018]

Consider the loop `while`$(\varphi)\{$ *body* $\}$ and let:

- $V : \mathbb{S} \to \mathbb{R}_{\geq 0}$ with $[\neg V] = [\neg \varphi]$ — V indicates termination
- $p : \mathbb{R}_{\geq 0} \to (0, 1]$ antitone — probability
- $d : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ antitone — decrease

If:

$$\underbrace{[\varphi] \cdot wp[\![body]\!](V) \;\leq\; V}_{\text{expected value of } V \text{ does not decrease by an iteration}}$$

expected value of V does not ~~d~~ecrease by an iteration
in

and

$$\underbrace{[\varphi] \cdot (p \circ V) \;\leq\; \lambda s.\, wp[\![body]\!] \left([\![V \leq V(s) - d(V(s))]\!] \right)(s)}_{\text{with at least prob. } p,\ V \text{ decreases at least by } d}$$

with at least prob. p, V decreases at least by d

A proof rule for AST

[McIver, K. *et al*, POPL 2018]

Consider the loop `while`(φ){ *body* } and let:

- $V : \mathbb{S} \to \mathbb{R}_{\geq 0}$ with $[\neg V] = [\neg \varphi]$ — V indicates termination
- $p : \mathbb{R}_{\geq 0} \to (0, 1]$ antitone — probability
- $d : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ antitone — decrease

If:

$$\underbrace{[\varphi] \cdot wp[\![body]\!](V) \;\leq\; V}_{\text{expected value of } V \text{ does not decrease by an iteration}}$$

in

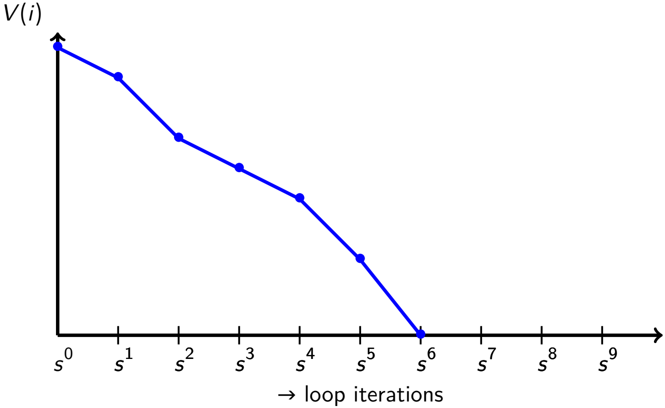
and

$$\underbrace{[\varphi] \cdot (p \circ V) \;\leq\; \lambda s.\, wp[\![body]\!]\, (|V \leq V(s) - d(V(s))|)\, (s)}_{\text{with at least prob. } p,\ V \text{ decreases at least by } d}$$

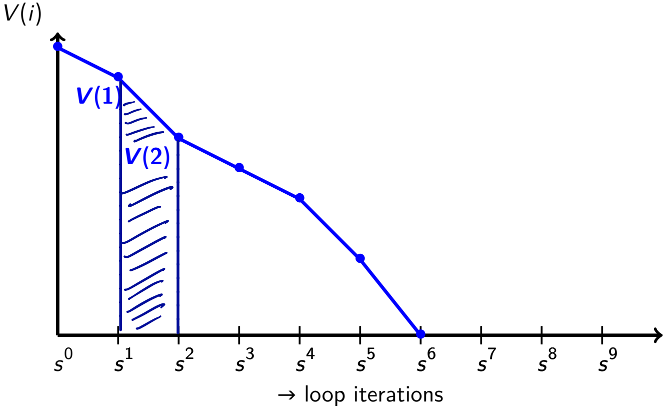
Then:

$$wp[\![\text{loop}]\!](\mathbf{1}) \;=\; \mathbf{1} \quad \text{i.e., loop is AST}$$

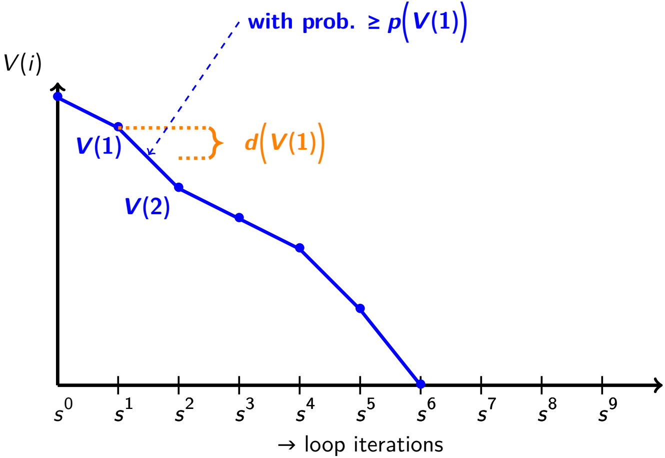
Intuition



Intuition

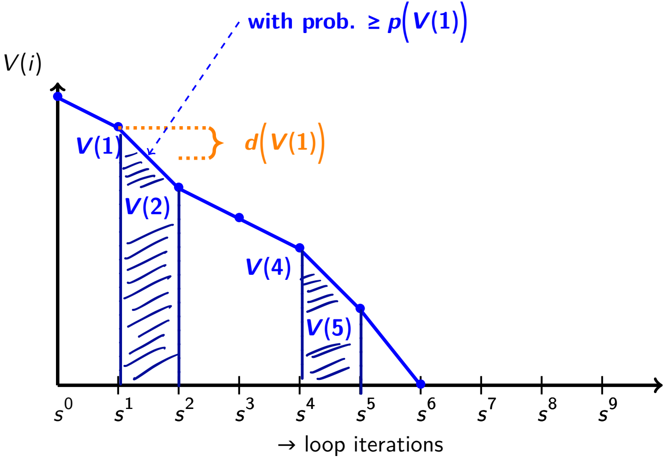


Intuition



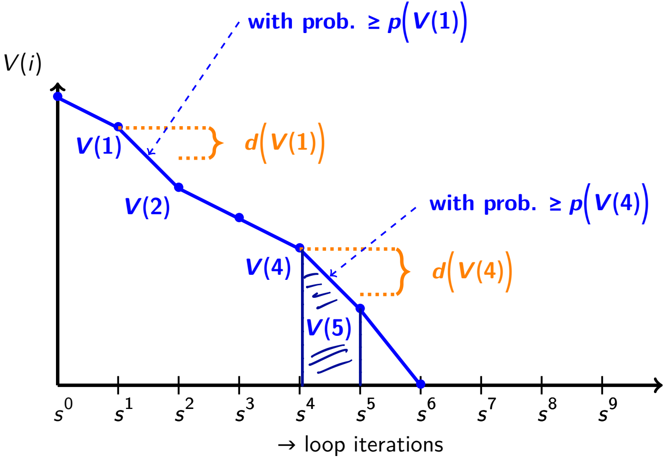
with prob. ≥ p(V(1))
V(i)
V(1)
d(V(1))
V(2)
s0 s1 s2 s3 s4 s5 s6 s7 s8 s9
→ loop iterations

Intuition

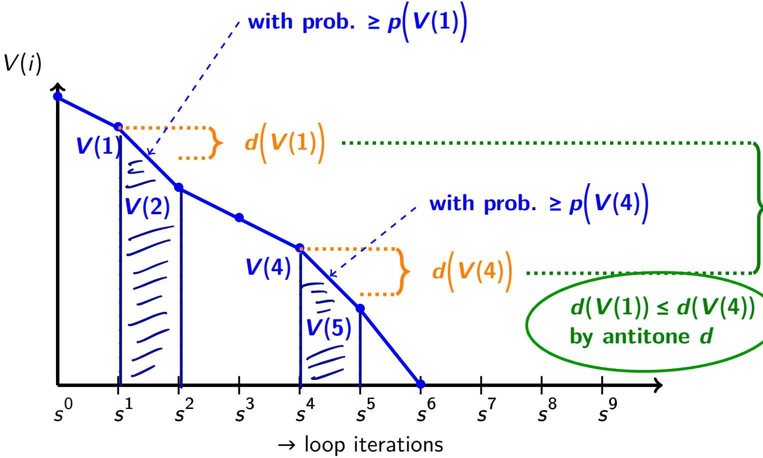


with prob. ≥ p(V(1))
V(i)
d(V(1))
V(1)
V(2)
V(4)
V(5)
s^0 s^1 s^2 s^3 s^4 s^5 s^6 s^7 s^8 s^9
→ loop iterations

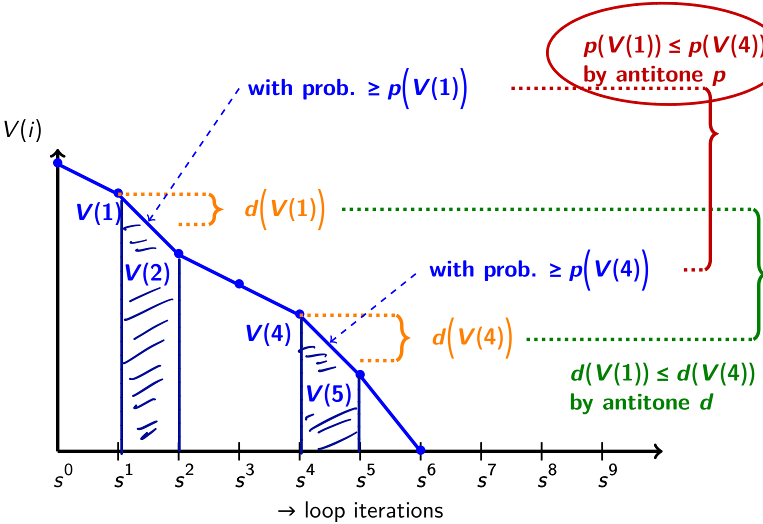
Intuition


with prob. ≥ p(V(1))
V(i)
V(1)
d(V(1))
V(2)
with prob. ≥ p(V(4))
V(4)
d(V(4))
V(5)
s^0 s^1 s^2 s^3 s^4 s^5 s^6 s^7 s^8 s^9
→ loop iterations

Intuition

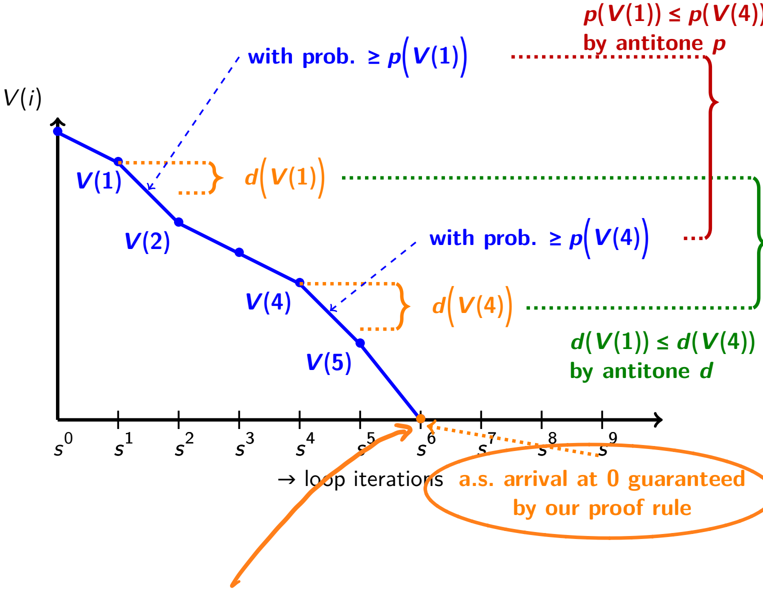

with prob. ≥ p(V(1))
V(i)
V(1)
d(V(1))
V(2)
with prob. ≥ p(V(4))
V(4)
d(V(4))
V(5)
d(V(1)) ≤ d(V(4))
by antitone d
s0 s1 s2 s3 s4 s5 s6 s7 s8 s9
→ loop iterations

Intuition



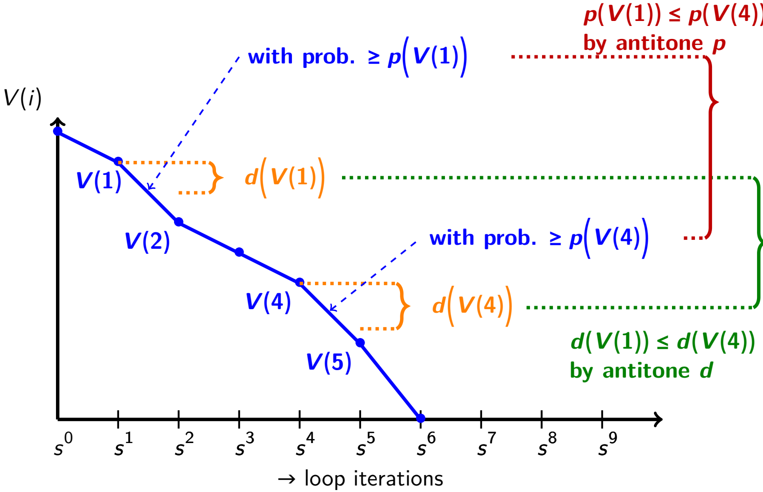
$p(V(1)) \leq p(V(4))$
by antitone p
with prob. $\geq p(V(1))$
$V(i)$
$V(1)$
$d(V(1))$
$V(2)$
with prob. $\geq p(V(4))$
$V(4)$
$d(V(4))$
$V(5)$
$d(V(1)) \leq d(V(4))$
by antitone d
s^0 s^1 s^2 s^3 s^4 s^5 s^6 s^7 s^8 s^9
→ loop iterations

Intuition



p(V(1)) ≤ p(V(4))
by antitone p
with prob. ≥ p(V(1))
V(i)
V(1)
d(V(1))
V(2)
with prob. ≥ p(V(4))
V(4)
d(V(4))
V(5)
d(V(1)) ≤ d(V(4))
by antitone d
s0 s1 s2 s3 s4 s5 s6 s7 s8 s9
→ loop iterations
a.s. arrival at 0 guaranteed
by our proof rule

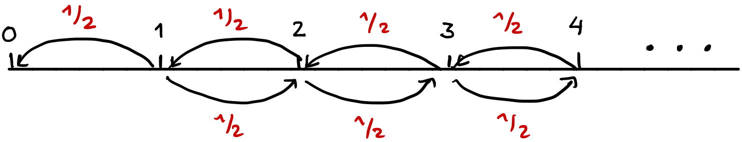
Intuition



The closer to termination, the more V decreases and this becomes more likely

Example: symmetric 1D random walk

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



Example: symmetric 1D random walk

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

- Terminates almost surely

- Witness of almost-sure termination:
  - $V = x$
  - $p = 1/2$ and
  - $d = 1$

Example: symmetric 1D random walk

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

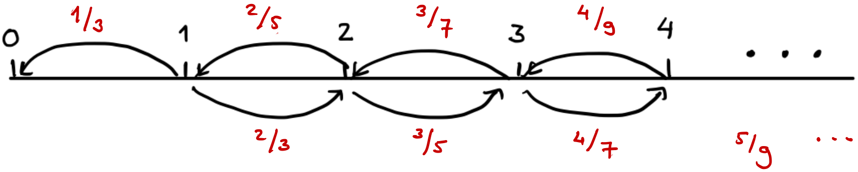
1/2 − ε is not AST

- Terminates almost surely
- Witness of almost-sure termination:
  - $V = x$
  - $p = 1/2$ and
  - $d = 1$

That's all you need to prove almost-sure termination!

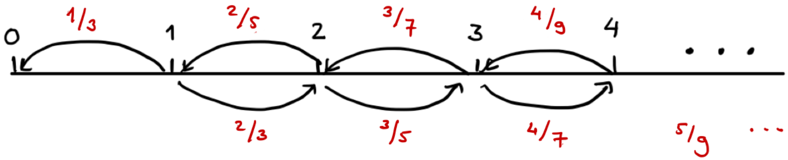
Example: fair-in-the-limit 1D random walk

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



The closer to 0, the more unfair — drifting away from 0 — it gets

Example: fair-in-the-limit 1D random walk



- The closer to 0, the more unfair — drifting away from 0 — it gets
- Witness of almost-sure termination:
  - $V = H_x$, the x-th Harmonic number $1 + 1/2 + \ldots + 1/x$
  - $d(v) = \begin{cases} 1/n & \text{if } H_{n-1} < v \leq H_n \\ 1 & \text{if } v = 0 \end{cases}$, and
  - $p = 1/3$

So far we studied several proof rules to verify whether

a given expectation (or triple V, p, d) meets

constraints that imply upper/lower bounds on weakest pre-expectations

So far we studied several proof rules to verify whether

a given expectation (or triple V, p, d) meets

constraints that imply upper/lower bounds on weakest pre-expectations

This can be extended to expected runtimes too

e.g. proving PAST

So far we studied several proof rules to verify whether

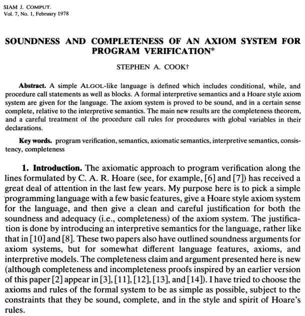
a given expectation (or triple V, p, d) meets

constraints that imply upper/lower bounds on weakest pre-expectations

This can be extended to expected runtimes too

To automate, we need a concrete syntax for expectations!

RELATIVE COMPLETENESS



SIAM J. COMPUT.
Vol. 7, No. 1, February 1978

SOUNDNESS AND COMPLETENESS OF AN AXIOM SYSTEM FOR PROGRAM VERIFICATION*

STEPHEN A. COOK†

Abstract. A simple ALGOL-like language is defined which includes conditional, while, and procedure call statements as well as blocks. A formal interpretive semantics and a Hoare style axiom system are given for the language. The axiom system is proved to be sound, and in a certain sense complete, relative to the interpretive semantics. The main new results are the completeness theorem, and a careful treatment of the procedure call rules for procedures with global variables in their declarations.

Key words. program verification, semantics, axiomatic semantics, interpretive semantics, consistency, completeness

1. Introduction. The axiomatic approach to program verification along the lines formulated by C. A. R. Hoare (see, for example, [6] and [7]) has received a great deal of attention in the last few years. My purpose here is to pick a simple programming language with a few basic features, give a Hoare style axiom system for the language, and then give a clean and careful justification for both the soundness and adequacy (i.e., completeness) of the axiom system. The justification is done by introducing an interpretive semantics for the language, rather like that in [10] and [8]. These two papers also have outlined soundness arguments for axiom systems, but for somewhat different language features, axioms, and interpretive models. The completeness claim and argument presented here is new (although completeness and incompleteness proofs inspired by an earlier version of this paper [2] appear in [3], [11], [12], [13], and [14]). I have tried to choose the axioms and rules of the formal system to be as simple as possible, subject to the constraints that they be sound, complete, and in the style and spirit of Hoare's rules.

SIAM J. on Computing, 1978

Stephen Cook

Relative complete verification

Ordinary Programs

$F \in$ FO-Arithmetic

implies

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

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 $\Rightarrow$

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

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

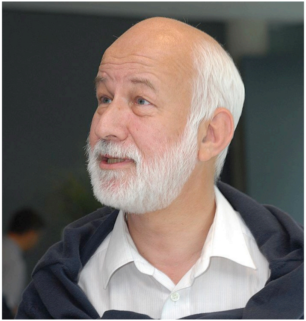
modulo an oracle for deciding $\sqsubseteq$

between two syntactic expectations.

Q: How does the SomeSyntax look like?

50 years of Hoare logic

"Completeness is a subtle manner and requires a careful analysis"



Krzysztof R. Apt

Ernst-Rüdiger Olderog

Requirements on a syntax

$\frac{1}{\pi}$

$\frac{\sqrt{5}-1}{2}$

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

1

```
x := geometric(1/4);
y := geometric(1/4);
t := x+y+1 [5/9] t := x+y;
r := 1;
for i in 1..3 {
   s := 0
   for j in 1..2t {
       s := s+1 [1/2] skip
   }
   r := (s == t)
}
```

$[r = 1]$

rational numbers, algebraic numbers, transcendental numbers, etc.

Syntax of expectations

- The set Exp of syntactic expectations

$f \longrightarrow$	a	arithmetic expressions
	$\mid [\varphi] \cdot f$	guarding
	$\mid f + f$	addition
	$\mid f \cdot f$	multiplication
	$\mid \mathsf{S}\, x\colon f$	supremum over variable x
	$\mid \mathsf{J}\, x\colon f$	infimum over variable x

Syntax of expectations

- The set Exp of syntactic expectations

f	$\longrightarrow$	a	arithmetic expressions
	$\mid$	$[\varphi] \cdot f$	guarding
	$\mid$	$f + f$	addition
	$\mid$	$f \cdot f$	multiplication
	$\mid$	$\mathsf{S}\, x \colon f$	supremum over variable x
	$\mid$	$\mathsf{J}\, x \colon f$	infimum over variable x

- Examples:

$$\mathsf{S}\, x \colon [x \cdot x < y] \cdot x$$

Syntax of expectations

- The set Exp of syntactic expectations

$$
\begin{aligned}
f \quad \longrightarrow \quad & a && \text{arithmetic expressions} \\
\mid \; & [\varphi] \cdot f && \text{guarding} \\
\mid \; & f + f && \text{addition} \\
\mid \; & f \cdot f && \text{multiplication} \\
\mid \; & \mathsf{S}\, x \colon f && \text{supremum over variable } x \\
\mid \; & \mathsf{J}\, x \colon f && \text{infimum over variable } x
\end{aligned}
$$

- Examples:

$$\mathsf{S}\, x \colon [x \cdot x < y] \cdot x \;\equiv\; \sqrt{y}$$

Syntax of expectations

- The set Exp of syntactic expectations

$$
\begin{array}{lllr}
f & \longrightarrow & a & \text{arithmetic expressions} \\
 & | & [\varphi] \cdot f & \text{guarding} \\
 & | & f + f & \text{addition} \\
 & | & f \cdot f & \text{multiplication} \\
 & | & \mathsf{S}\, x\colon f & \text{supremum over variable } x \\
 & | & \mathsf{l}\, x\colon f & \text{infimum over variable } x
\end{array}
$$

- Examples:

$$\mathsf{S}\, x\colon [x \cdot x < y] \cdot x \;\equiv\; \sqrt{y}$$

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

Examples

- polynomials $y + x^3 + 2x^2 + x - 7$ widely used as templates
- rational functions $\dfrac{x^2 - 3x + 4}{y^2 \cdot x - 3y + 1}$
- 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

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

[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:

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

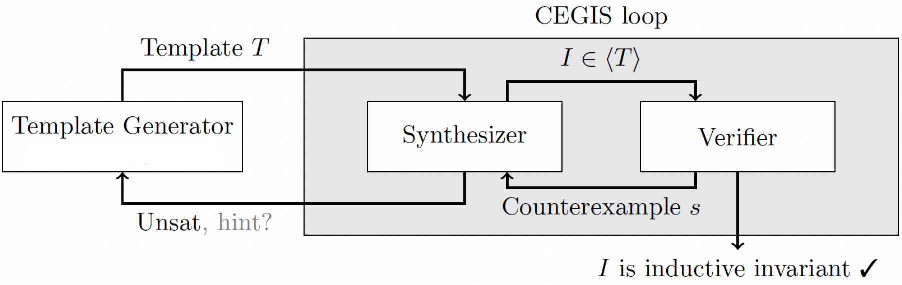
for some syntactic expectation $g \in \text{Exp}$.

Expressiveness does not mean decidability, e.g.,

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

Inductive invariant synthesis

Automated synthesis of inductive invariants



Bounded retransmission protocol [Helmink *et al*, 1993]

- Send file of $N \approx 10^{10}$ packets via lossy channel
- Packet loss probability $\frac{1}{100}$, say
- $\#$ packet retransmissions ≤ 10; otherwise file transmission fails

Bounded retransmission protocol [Helmink *et al*, 1993]

- Send file of $N \approx 10^{10}$ packets via lossy channel
- Packet loss probability $\frac{1}{100}$, say
- $\#$ packet retransmissions ≤ 10; otherwise file transmission fails

$$sent := 0;\ fail := 0;$$

$$\texttt{while}\,(\,sent < N \wedge fail < F\,)\,\{$$

$$\quad \{\underbrace{fail := fail + 1}_{\text{failed transmission}}\}[0.01]\{\underbrace{fail := 0\,;sent := sent + 1}_{\text{successful transmission}}\}\}$$

Bounded retransmission protocol [Helmink *et al*, 1993]

- Send file of $N \approx 10^{10}$ packets via lossy channel
- Packet loss probability $\frac{1}{100}$, say
- # packet retransmissions ≤ 10; otherwise file transmission fails

BRP

$$sent := 0;\ fail := 0;$$

$$\texttt{while}\,(\,sent < N \wedge fail < F\,)\,\{$$

$$\{\,\underbrace{fail := fail + 1}_{\text{failed transmission}}\,\}\,[0.01]\,\{\,\underbrace{fail := 0\,;\,sent := sent + 1}_{\text{successful transmission}}\,\}\}$$

We verify $wp[\![BRP]\!]([fail = 10]) \leq \frac{1}{1000}$ in 11 seconds. Fully automatically.

Impossible for probabilistic model checkers!

An upper bound

```
failed) <= 2.0) & ((sent) <= 2000000000.0) & ((sent) < 8000000000.0) & ((failed) < 10)] *
525957797341567695057760649001/6148435478882796002500000000000000000000000000000*sent
7069821936815201810718304251099/6148435478882796002500000000000000000000000000000*failed
2076717881671108079960003920597606490001/61484354788827960025000000000000000000000000000000) + [(((failed) <= 5.0) & ((sent) <= 2000000000.0) & ((failed
) & ((sent) < 8000000000.0) & (((failed) * -1.0) < -2)] * (-1552595779734156769505776064900 1/6148435478882796002500000000000000000000000000000*sent
7069821936815201810718304251099/3089666069790349750000000000000000000000000000*failed
1452720885396153003793305795939056 37/768554434860349500312500000000000000000000000) + [(((failed) <= 7.0) & (((failed) * -1.0) < -5.0) & ((sent) <=
000000.0) & ((sent) < 8000000000.0) & ((failed) < 10)] * (-1552595779734156769505776064900 1/6148435478882796002500000000000000000000000000000*sent
7069821936815201810718304251099/155259601497002500000000000000000000000000*failed
2181401479122274118080107750725340393/614843547888279600250000000000000000000000000) + [((((failed) * -1.0) < -7.0) & ((sent) <= 2000000000.0) & ((ser
000000.0) & ((failed) < 10)] * (-1552595779734156769505776064900 1/6148435478882796002500000000000000000000000000000*sent
7069821936815201810718304251099/1552596014970025000000000000000000000000*failed
5339930616708300621427955609394113371/61484354788827960025000000000000000000000) + [((((sent) * -1.0) < -2000000000.0) & ((sent) <= 4000000000.0) &
iled) <= 2.0) & ((sent) < 8000000000.0) & ((failed) < 10)] * (-7841195212980020198004 99/310519202994005000000000000000000000000000*sent
2783260850219996024940 1/310519202994005000000000000000000000000000*failed +627295640799000803960099801980049 9/31051920299400500000000000000000000000000
(sent) * -1.0) < -2000000000.0) & ((failed) <= 5.0) & ((sent) <= 4000000000.0) & ((failed) < 10.0) & ((sent) < 8000000000.0) & (((failed) * -1.0) < -
41195212980020198004 99/310519202994005000000000000000000000000000*sent +77627832608502199960249401/156039800499500000000000000000000000*failed
542203697204345900695392586 3/38814900374250625000000000000000000000000) + [(((failed) <= 7.0) & (((sent) * -1.0) < -2000000000.0) & ((sent) <= 400000000
(failed) * -1.0) < -5.0) & ((sent) < 8000000000.0) & ((failed) < 10)] * (-7841195212980020198004 99/310519202994005000000000000000000000000000*sent
2783260850219996024940 1/784119600500000000000000000000000*failed +-177864912368922626939386000131 07/31051920299400500000000000000000000000) + [((((fail
) < -7.0) & (((sent) * -1.0) < -2000000000.0) & ((sent) <= 4000000000.0) & ((sent) < 8000000000.0) & ((failed) < 10)] *
41195212980020198004 99/310519202994005000000000000000000000000000*sent +77627832608502199960249401/784119600500000000000000000000*failed
5620037463073707281353896411 49/31051920299400500000000000000000000) + [(((sent) <= 6000000000.0) & ((failed) <= 2.0) & (((sent) * -1.0) < -4000000000.0
nt) < 8000000000.0) & ((failed) < 10)] * (-39600998000000001/15682392010000000000000000000000*sent
0498802000000099/15682392010000000000000000000000*failed +3168079880000000040000000 01/15682392010000000000000000000000) + [(((failed) <= 5.0) &
nt) <= 6000000000.0) & (((sent) * -1.0) < -4000000000.0) & ((failed) < 10.0) & ((sent) < 8000000000.0) & (((failed) * -1.0) < -2)] *
5009980000000001/15682392010000000000000000000000*sent +39204988020000000 99/788059900000000000000000000*failed
7184028022247574926 37/19602990012500000000000000000000) + [(((failed) <= 7.0) & (((failed) * -1.0) < -5.0) & ((sent) <= 6000000000.0) & (((sent) * -1.
0000000.0) & ((sent) < 8000000000.0) & ((failed) < 10)] * (-39600998000000001/15682392010000000000000000000000*sent
0498802000000099/396010000000000000000000*failed +-89828500043017412288339 3/15682392010000000000000000000000) + [((((failed) * -1.0) < -7.0) & ((sent)
000000.0) & (((sent) * -1.0) < -4000000000.0) & ((sent) < 8000000000.0) & ((failed) < 10)] * (-39600998000000001/15682392010000000000000000000000*s
0498802000000099/3960100000000000000000*failed +-124047397726233842733433 1/1568239201000000000000000000) + [(((failed) <= 2.0) & (((sent) * -1.0) <
0000000.0) & ((sent) < 8000000000.0) & ((failed) < 10)] * (-1/39601000000000000*sent +99/39601000000000000*failed +1/4950125) + [((((sent) * -1.0) <
0000000.0) & ((failed) <= 5.0) & ((failed) < 10.0) & ((sent) < 8000000000.0) & (((failed) * -1.0) < -2)] * (-1/39601000000000000*sent
1990000000000*failed +7994109599/39601000000000000) + [((((sent) * -1.0) < -6000000000.0) & ((failed) <= 7.0) & (((failed) * -1.0) < -5.0) & ((sent)
000000.0) & ((failed) < 10)] * (-1/39601000000000000*sent +99/100000000*failed +-226833930001/39601000000000000) + [((((failed) * -1.0) < -7.0) & (((
.0) < -6000000000.0) & ((sent) < 8000000000.0) & ((failed) < 10)] * (-1/39601000000000000*sent +99/10000*failed +-3132431100000001/39601000000000000)
 (((sent) < 8000000000.0) & ((failed) < 10.0))) & (((failed) * -1.0) <= -10.0) & ((failed) <= 10)] * (1.0) + [((! (((sent) < 8000000000.0) & ((faile
)) & (! ((((failed) * -1.0) <= -10.0) & ((failed) <= 10)))] * (0)
```

Synthesising inductive invariants

Problem: find a piece-wise linear inductive invariant I s.t.

$$\underbrace{\Phi_f(I) \sqsubseteq I \text{ and } I \sqsubseteq g}_{I \text{ is inductive for } f \text{ and } g} \qquad \text{or determine there is no such } I$$

Synthesising inductive invariants

Problem: find a piece-wise linear inductive invariant I s.t.

$$\underbrace{\Phi_f(I) \sqsubseteq I \text{ and } I \sqsubseteq g}_{I \text{ is inductive for } f \text{ and } g} \qquad \text{or determine there is no such } I$$

Approach: use template-based invariants of the (simplified) form:

$$T \;=\; [b_1] \cdot a_1 + \cdots + [b_k] \cdot a_k$$

with

- b_i is a boolean combination of linear inequalities over program vars
- a_i a linear expression over the program variables with $[b_i] \cdot a_i \geq 0$
- the b_i's partition the state space, i.e., $s \models b_i$ for a unique i

Synthesising inductive invariants

Problem: find a piece-wise linear inductive invariant I s.t.

$$\underbrace{\Phi_f(I) \sqsubseteq I \text{ and } I \sqsubseteq g}_{I \text{ is inductive for } f \text{ and } g} \qquad \text{or determine there is no such } I$$

Approach: use template-based invariants of the (simplified) form:

$$T \;=\; [b_1] \cdot a_1 + \cdots + [b_k] \cdot a_k$$

with

- b_i is a boolean combination of linear inequalities over program vars
- a_i a linear expression over the program variables with $[b_i] \cdot a_i \geq 0$
- the b_i's partition the state space, i.e., $s \models b_i$ for a unique i

Example: $[c{=}1] \cdot (2{\cdot}x + 1) + [c{\neq}1] \cdot x$ is in the above form,
and $[x \geq 1] \cdot x + [x \geq 2] \cdot y$ can be rewritten into it.

Checking linear entailments **[K., McIver *et al*, SAS 2010]**

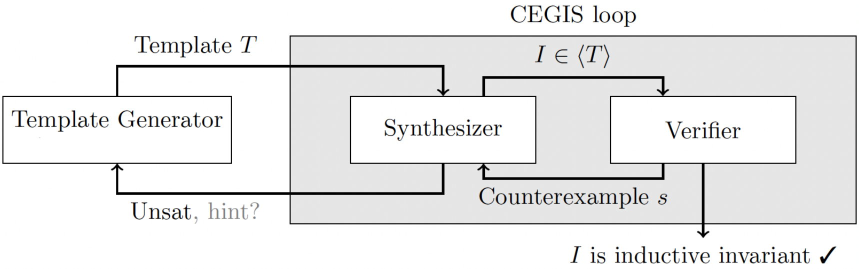
For piecewise linear expectations:

$$f = [b_1] \cdot a_1 + \cdots + [b_k] \cdot a_k \quad \text{and} \quad g = [c_1] \cdot e_1 + \cdots + [c_m] \cdot e_m$$

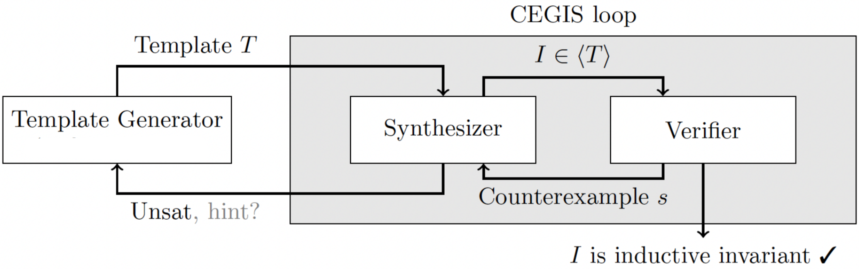
it is decidable whether the quantitative entailment $f \sqsubseteq g$ holds

$$f \sqsubseteq g \quad \text{if and only if} \quad \underbrace{\bigwedge_{i=1}^{k} \bigwedge_{j=1}^{m} (b_i \wedge c_j) \rightarrow a_i \sqsubseteq e_j}_{\text{formula in quantifier-free linear arithmetic}} \quad \text{is valid}$$

CEGIS for probabilistic invariants [Batz, K. *et al*, TACAS 2023]

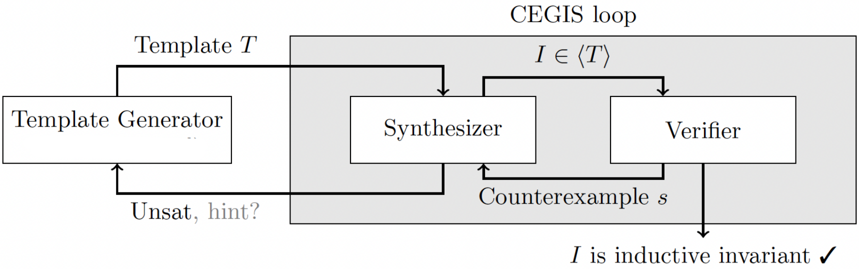


CEGIS for probabilistic invariants [Batz, K. *et al*, TACAS 2023]



- For finite-state programs, synthesis is sound and complete
- Applicable to lower bounds: UPAST and difference boundedness
- Uses SMT with QF-LRA (the synthesiser) and QF-LIRA (the verifier)

CEGIS for probabilistic invariants [Batz, K. *et al*, TACAS 2023]



- For finite-state programs, synthesis is sound and complete
- Applicable to lower bounds: UPAST and difference boundedness

- Uses SMT with QF-LRA (the synthesiser) and QF-LIRA (the verifier)

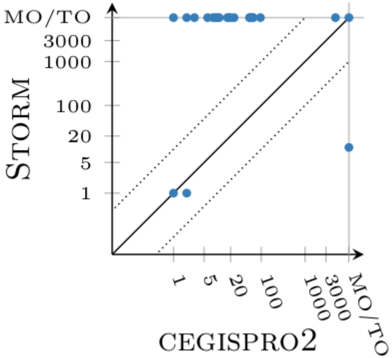
CEGISPRO2 tool: `https://github.com/moves-rwth/cegispro2`

check it out!

Experiments

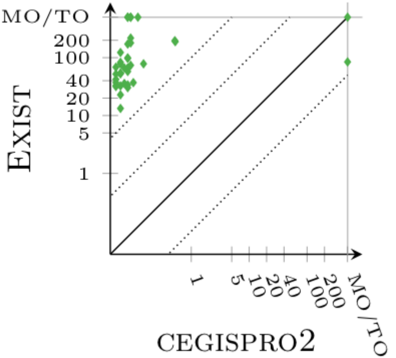
comparison to model checking

comparison to an ML technique



Synthesis of upper bounds for finite-state programs

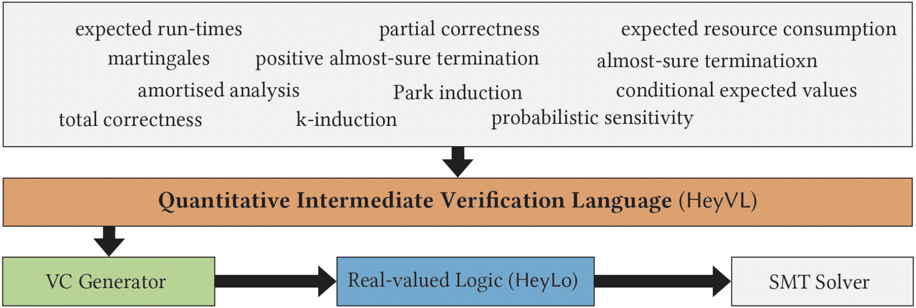
TO = 2h, MO = 8GB



Synthesis of lower bounds

TO = 5min

Outlook: a probabilistic Dafny? KeY?



Caesar: A verification infrastructure for probabilistic programs

`caesarverifier.org` ← check it out!

A big thanks to my co-workers!

Ezio Bartocci | Kevin Batz | Mingshuai Chen | Sebastian Junges | Benjamin Kaminski | Laura Kovacs | Lutz Klinkenberg

Christoph Matheja | Annabelle McIver | Marcel Moosbrugger | Carroll Morgan | Federico Olmedo | Philipp Schroer | Tobias Winkler