

# Automatic Verification of Probabilistic Programs

Joost-Pieter Katoen & **Kevin Batz**



Lecture on Probabilistic Programming

### Automatic Verification of Probabilistic Loops

**How and in which cases** can we automate the wp-calculus?

What about the ert-calculus for expected runtimes?

**How:** Quantitative Loop Invariants.

**In which cases:** Restrict to a certain class of probabilistic loops and expectations.

**Motivation:**

Verification of **Randomized Algorithms** and **Stochastic Processes**

### The Bounded Retransmission Protocol [Helmink *et al.* '93, D'Argenio *et al.* '97]

Goal: Sent file of  $N$  packets via lossy channel.

Transmitting a single packet fails with some probability, say 0.01.

Allow for at most  $F$  retransmission tries per packet; otherwise fail.

```
sent := 0; fail := 0; totalFail := 0;
```

```
while (sent < N  $\wedge$  fail < F) {
```

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

**Is the probability of failing to transmit file (reaching  $fail = F$ ) at most 0.0001?**

**Is the expected number of retransmission tries ( $totalFail$ ) at most 10?**

### Optimal Discrete Uniform Generation from Coin Flips [Lumbroso '13]

Sample uniformly from  $\{0, \dots, n - 1\}$  using **fair coin flips only**.

```
 $v := 1; c := 0; term := 0;$   
while ( $term = 0$ ) {  
   $v := 2 \cdot v;$   
   $\{c := 2 \cdot c\} [1/2] \{c := 2 \cdot c + 1\};$   
  if ( $v \geq n$ ) {  
    if ( $c < n$ ) {  $term := 1$  }  
    else {  $v := v - n; c := c - n$  }  
  }  
}
```

Is  $\Pr(c = i) = 1/n$  for all  $i \in \{0, \dots, n - 1\}$ ?

Probabilistic Programs

Recap: Weakest Preexpectations & Loop Invariants

Automatic Verification of Probabilistic Loops

# Weakest Preexpectation Reasoning

What is the expected value of program variable  $x$  after termination?

What is the probability of reaching a state  $fail = F$ ?

What is the termination probability?

## Recap: Weakest Preexpectations & Loop Invariants

---

Consider the complete lattice  $(\mathbb{E}, \sqsubseteq)$  of *expectations*:

$$\mathbb{E} = \{f \mid f: \text{States} \rightarrow \mathbb{R}_{\geq 0}^{\infty}\} \quad \text{with} \quad f \sqsubseteq g \quad \text{iff} \quad \forall \sigma \in \text{States}: f(\sigma) \leq g(\sigma)$$

*Weakest preexpectation* transformer:

$$\text{wp}[[C]]: \mathbb{E} \rightarrow \mathbb{E} \quad \text{wp}[[C]](f)(\sigma) = \text{expected value of } f \text{ w.r.t. distribution of final states reached after executing } C \text{ on } \sigma$$

In this lecture, we restrict to **computable** postexpectations  $f: \text{States} \rightarrow \mathbb{Q}_{\geq 0}^{\infty}$ .

$$\text{wp}[[\{\text{skip}\} [1/2] \{x := x + 2\}]](x) = 1/2 \cdot x + 1/2 \cdot (x + 2) = x + 1$$

$$\text{wp}[[\{\text{skip}\} [1/2] \{x := x + 2\}]]([x = 4]) = 1/2 \cdot [x = 4] + 1/2 \cdot [x = 2]$$

$$\text{wp}[[\text{while}(c = 1) \{c := 0 [1/2] x := x + 1\}]](x) = [c = 1] \cdot (x + 1) + [c \neq 1] \cdot x$$

## Recap: Weakest Preexpectations & Loop Invariants

---

$C$	$\text{wp} \llbracket C \rrbracket (f)$
skip	$f$
$x := a$	$f[x/a]$
$C_1 ; C_2$	$\text{wp} \llbracket C_1 \rrbracket (\text{wp} \llbracket C_2 \rrbracket (f))$
$\{ C_1 \} [p] \{ C_2 \}$	$p \cdot \text{wp} \llbracket C_1 \rrbracket (f) + (1 - p) \cdot \text{wp} \llbracket C_2 \rrbracket (f)$
if ( $\varphi$ ) { $C_1$ } else { $C_2$ }	$[\varphi] \cdot \text{wp} \llbracket C_1 \rrbracket (f) + [\neg\varphi] \cdot \text{wp} \llbracket C_2 \rrbracket (f)$
while ( $\varphi$ ) { <i>body</i> }	$\text{lfp } h. [\neg\varphi] \cdot f + [\varphi] \cdot \text{wp} \llbracket \textit{body} \rrbracket (h)$

The lfp is taken w.r.t.  $\sqsubseteq$  on expectations.



## Recap: Weakest Preexpectations & Loop Invariants

---

$\lll [y = 5] \cdot (x + 1) + [y \neq 5] \cdot 5$

this is  $\text{wp}[C](x)$

if ( $y = 5$ ) {

$\lll x + 1$

{ skip } [1/2] {  $x := x + 2$  }

$\lll x$

} else {

$\lll 5$

$x := 5$

$\lll x$

}

$\lll x$

$x$  is the postexpectation

**Observation: Determining  $\text{wp}$  for loop-free programs is automatable.**

## Recap: Weakest Preexpectations & Loop Invariants

---

$C$	$\text{wp} \llbracket C \rrbracket (f)$
skip	$f$
$x := a$	$f[x/a]$
$C_1 ; C_2$	$\text{wp} \llbracket C_1 \rrbracket (\text{wp} \llbracket C_2 \rrbracket (f))$
$\{ C_1 \} [p] \{ C_2 \}$	$p \cdot \text{wp} \llbracket C_1 \rrbracket (f) + (1 - p) \cdot \text{wp} \llbracket C_2 \rrbracket (f)$
if ( $\varphi$ ) { $C_1$ } else { $C_2$ }	$[\varphi] \cdot \text{wp} \llbracket C_1 \rrbracket (f) + [\neg\varphi] \cdot \text{wp} \llbracket C_2 \rrbracket (f)$
while ( $\varphi$ ) { <i>body</i> }	lfp $h$ . $[\neg\varphi] \cdot f + [\varphi] \cdot \text{wp} \llbracket \textit{body} \rrbracket (h)$

**Observation:** Determining wp for loop-free programs is automatable.

**This does not hold for loops (Lec. 13). How to tackle the verification of loops?**

# Verifying Probabilistic Loops

... by means of quantitative loop invariants.

Goal: Given computable expectations  $f, g$ , verify that  $\text{wp}[\text{while}(\varphi)\{body\}](f) \sqsubseteq g$ .

Example:  $\text{wp}[\text{bounded retransmission protocol}](\text{fail} = F) \sqsubseteq 0.0001$

Problem: for  $\text{wp}[\text{while}(\varphi)\{body\}](f) = \text{lfp } \Phi_f$ , where

$$\Phi_f: \mathbb{E} \rightarrow \mathbb{E}, \quad \Phi_f(h) = [\neg\varphi] \cdot f + [\varphi] \cdot \text{wp}[body](h),$$

$\text{lfp } \Phi_f$  is uncomputable in general (Lec. 13).

**Solution: Quantitative Loop Invariants**

## Recap: Weakest Preexpectations & Loop Invariants

---

$\text{wp}[\text{while}(\varphi)\{body\}](f) = \text{lfp } \Phi_f$ , where

$$\Phi_f: \mathbb{E} \rightarrow \mathbb{E}, \quad \Phi_f(h) = [\neg\varphi] \cdot f + [\varphi] \cdot \text{wp}[\text{body}](h)$$

### Theorem (Superinvariants for Loops)

Let  $I \in \mathbb{E}$ . We have

$$\underbrace{\Phi_f(I) \sqsubseteq I}_{I \text{ is wp-superinvariant}} \quad \text{implies} \quad \text{wp}[\text{while}(\varphi)\{body\}](f) \sqsubseteq I.$$

Loop invariants reduce reasoning about loops to **reasoning about one loop iteration.**

## Recap: Weakest Preexpectations & Loop Invariants

Goal:  $\text{wp}[\text{while } (c = 1) \{ c := 0 [1/2] x := x + 1 \}](x) \sqsubseteq \underbrace{[c \neq 1] \cdot x + [c = 1] \cdot (x + 1)}_{\text{choose as superinvariant } I}$

```
while (c = 1) {  
    // 1/2 · (x + [c = 1] · (x + 2) + [c ≠ 1] · (x + 1))  
    { c := 0 } [1/2] { x := x + 1 }  
    // [c = 1] · (x + 1) + [c ≠ 1] · x  
}
```

$$\begin{aligned}\Phi_x(I) &= [c \neq 1] \cdot x + [c = 1] \cdot \text{wp}[\text{body}](I) \\ &= \underbrace{[c \neq 1] \cdot x + [c = 1] \cdot (1/2 \cdot (x + [c = 1] \cdot (x + 2) + [c \neq 1] \cdot (x + 1)))}_{\text{When and how is this decidable?}} \sqsubseteq I \quad \checkmark\end{aligned}$$

Being able to decide  $\Phi_f(I) \sqsubseteq I$  enables automatic verification of loops.

# Automatic Verification of Loops

... by restricting expectations  $f, I$  and loops  $\text{while}(\varphi) \{ \text{body} \}$  such that the wp-superinvariant condition  $\Phi_f(I) \sqsubseteq I$  is **decidable**.

### Definition

The set  $\text{LinExp}$  of *piecewise linear expectations* consists of all expectations  $f$  of the form

$$f = [\varphi_1] \cdot e_1 + \dots + [\varphi_n] \cdot e_n \quad , \text{ where}$$

1. each  $\varphi_i$  is a Boolean combination of linear (in)equalities over  $\text{Vars}$ ,
2. each  $e_i$  is a linear expression over  $\text{Vars}$  with  $\forall \sigma \in \text{States}: \sigma \models \varphi_i \implies e_i(\sigma) \geq 0$ ,
3. the  $\varphi_i$  partition the state space: for all  $\sigma \in \text{States}$  we have  $\sigma \models \varphi_j$  for exactly one  $j$ .

▶  $[c \neq 1] \cdot x + [c = 1] \cdot (x + 1) \in \text{LinExp}$

▶  $[c \neq 1] \cdot (x \cdot y) + [c = 1] \cdot (x + 1) \notin \text{LinExp}$

▶  $[x \geq 1] \cdot x + [x \geq 2] \cdot y \notin \text{LinExp}$  but

$$[x \geq 1 \wedge x \geq 2] \cdot (x + y) + [x \geq 1 \wedge x \not\geq 2] \cdot x + [x \not\geq 1 \wedge x \not\geq 2] \cdot 0 \in \text{LinExp}$$



### Definition

A loop  $\text{while}(\varphi) \{ \text{body} \}$  is called *linear* if *body* is loop-free and all expressions occurring in  $\varphi$  and *body* are linear.

The following loop is linear:

```
while ( term = 0 ) {  
    v := 2 · v;  
    { c := 2 · c } [1/2] { c := 2 · c + 1 };  
    if ( v ≥ n ) {  
        if ( c < n ) { term := 1 }  
        else { v := v - n ; c := c - n }  
    }  
}
```

## Automatic Verification of Probabilistic Loops

---

### Definition

A loop  $\text{while}(\varphi)\{ \text{body} \}$  is called *linear* if *body* is loop-free and all expressions occurring in  $\varphi$  and *body* are linear.

### Lemma

Let  $f, I \in \text{LinExp}$  and let  $\text{while}(\varphi)\{ \text{body} \}$  be linear. Then  $\Phi_f(I) \in \text{LinExp}$ .

### Lemma

For  $f, g \in \text{LinExp}$ , it is decidable whether  $f \sqsubseteq g$  holds.

### Theorem

Let  $f, I \in \text{LinExp}$  and let  $\text{while}(\varphi)\{ \text{body} \}$  be linear. Then  $\Phi_f(I) \sqsubseteq I$  is decidable.

## Automatic Verification of Probabilistic Loops

---

Given:  $f, g \in \text{LinExp}$  with

$$f = [\varphi_1] \cdot e_1 + \dots + [\varphi_n] \cdot e_n \quad \text{and} \quad g = [\psi_1] \cdot a_1 + \dots + [\psi_m] \cdot a_m$$

Goal: Decide whether  $f \sqsubseteq g$  holds.

Idea: Compute logical formula  $\Theta$  over Vars such that

$$f \sqsubseteq g \quad \text{iff} \quad \underbrace{\forall \sigma \in \text{States}: \sigma \models \Theta}_{\text{decidable using existing tools}} .$$

## Automatic Verification of Probabilistic Loops

---

Given:  $f, g \in \text{LinExp}$  with

$$f = [\varphi_1] \cdot e_1 + \dots + [\varphi_n] \cdot e_n \quad \text{and} \quad g = [\psi_1] \cdot a_1 + \dots + [\psi_m] \cdot a_m$$

Goal: Decide whether  $f \sqsubseteq g$  holds.

$f \sqsubseteq g$  iff for all  $\sigma \in \text{States}$ :  $f(\sigma) \leq g(\sigma)$

iff for all  $\sigma \in \text{States}$  and all  $\varphi_i, \psi_j$ :  $( \underbrace{\sigma \models \varphi_i}_{f(\sigma)=e_i(\sigma)} \text{ and } \underbrace{\sigma \models \psi_j}_{g(\sigma)=a_j(\sigma)} )$  implies  $e_i(\sigma) \leq a_j(\sigma)$

iff for all  $\sigma \in \text{States}$ :  $\sigma \models \underbrace{\bigwedge_{i=1}^n \bigwedge_{j=1}^m (\varphi_i \wedge \psi_j)}_{\text{formula of linear inequalities}} \longrightarrow e_i \leq a_j$

The latter problem is decidable using **Satisfiability Modulo Theories** techniques.<sup>a</sup>

---

<sup>a</sup><https://www.microsoft.com/en-us/research/project/z3-3/>

Hence, our example

$$\text{wp}[\text{while } (c = 1) \{ c := 0 [1/2] \ x := x + 1 \}] (x) \sqsubseteq [c \neq 1] \cdot x + [c = 1] \cdot (x + 1)$$

can be verified **automatically**.

Given  $f, I \in \text{LinExp}$ , and linear  $\text{while } (\varphi) \{ \text{body} \}$ :

1. Compute  $\Phi_f(I) \in \text{LinExp}$ .
2. If  $\Phi_f(I) \sqsubseteq I$ , then return **verify**; otherwise return **inconclusive**.

**Question:** What to do if  $\Phi_f(I) \sqsubseteq I$  does **not** hold?

Hence, our example

$$\text{wp}[\text{while } (c = 1) \{ c := 0 [1/2] \ x := x + 1 \}] (x) \sqsubseteq [c \neq 1] \cdot x + [c = 1] \cdot (x + 1)$$

can be verified **automatically**. **Problem: Not every upper bound is a superinvariant:**

$$\text{wp}[\text{while } (c = 1) \{ c := 0 [1/2] \ x := x + 1 \}] (x) \sqsubseteq x + 1$$

We have

$$\Phi_x(x + 1) = [c \neq 1] \cdot x + [c = 1] \cdot (x + 1.5) \not\sqsubseteq x + 1.$$

Even though  $\text{lfp } \Phi_x \sqsubseteq x + 1$  we have  $\Phi_x(x + 1) \not\sqsubseteq x + 1!$

**Mitigate this problem using a more powerful proof rule:  $k$ -induction.**

# Improved Automatic Verification of Loops

... by means of  $k$ -induction.

## Improved Automatic Verification of Probabilistic Loops

---

Given  $h, h' \in \mathbb{E}$ , define their *pointwise minimum* by  $h \sqcap h' = \lambda\sigma. \min\{h(\sigma), h'(\sigma)\}$ .

Example:

$$[c \neq 1] \cdot x + [c = 1] \cdot (x + 1.5) \sqcap x + 1 = [c \neq 1] \cdot x + [c = 1] \cdot (x + 1)$$



## Improved Automatic Verification of Probabilistic Loops

Given  $h, h' \in \mathbb{E}$ , define their *pointwise minimum* by  $h \sqcap h' = \lambda \sigma. \min\{h(\sigma), h'(\sigma)\}$ .

### Theorem

For  $f, I \in \mathbb{E}$  and loop  $C = \text{while}(\varphi)\{ \text{body} \}$ , it holds:

- ▶ 1-induction:  $\Phi_f(I) \sqsubseteq I$  implies  $\text{wp}\llbracket \text{while}(\varphi)\{ \text{body} \} \rrbracket(f) \sqsubseteq I$
- ▶ 2-induction:  $\Phi_f(\Phi_f(I) \sqcap I) \sqsubseteq I$  implies  $\text{wp}\llbracket \text{while}(\varphi)\{ \text{body} \} \rrbracket(f) \sqsubseteq I$
- ▶ 3-induction:  $\Phi_f(\Phi_f(\Phi_f(I) \sqcap I) \sqcap I) \sqsubseteq I$  implies  $\text{wp}\llbracket \text{while}(\varphi)\{ \text{body} \} \rrbracket(f) \sqsubseteq I$
- ▶ ...

### Theorem

For  $f, I \in \text{LinExp}$ , linear  $C$ , and  $k \geq 1$ , the condition for  $k$ -induction is decidable.

Follows from the fact that for  $h, h' \in \text{LinExp}$ , one can compute  $h \sqcap h' \in \text{LinExp}$ .

Back to our example

$$\text{wp}[\text{while } (c = 1) \{ c := 0 [1/2] \ x := x + 1 \}](x) \sqsubseteq x + 1,$$

1-induction fails:

$$\Phi_x(x + 1) = [c \neq 1] \cdot x + [c = 1] \cdot (x + 1.5) \not\sqsubseteq x + 1$$

Now consider 2-induction:

$$\begin{aligned} \Phi_x(\Phi_x(x + 1) \sqcap x + 1) &= \Phi_x([c \neq 1] \cdot x + [c = 1] \cdot (x + 1.5) \sqcap x + 1) \\ &= \Phi_x([c \neq 1] \cdot x + [c = 1] \cdot (x + 1)) \\ &= [c \neq 1] \cdot x + [c = 1] \cdot (x + 1) \sqsubseteq x + 1 \checkmark \end{aligned}$$

***k*-induction is more powerful and decidable in the linear setting.**

We have implemented  $k$ -induction for probabilistic loops:

### **KIPRO2: $k$ -Induction for PRObabilistic PROgrams<sup>1</sup>**

Given piecewise linear  $f, I$  and linear  $\text{while}(\varphi)\{body\}$ , `kipro2` aims to verify

$$\text{wp}[\text{while}(\varphi)\{body\}](f) \sqsubseteq I$$

by checking:

1-induction: If  $\Phi_f(I) \sqsubseteq I$ , then return **verify**; otherwise

2-induction: If  $\Phi_f(\Phi_f(I) \sqcap I) \sqsubseteq I$ , then return **verify**; otherwise

3-induction: If  $\Phi_f(\Phi_f(\Phi_f(I) \sqcap I) \sqcap I) \sqsubseteq I$ , then return **verify**; otherwise

...

---

<sup>1</sup><https://github.com/moves-rwth/kipro2>

For  $C_{\text{brp}}$  given by

$$\text{while } (sent < N \wedge fail < F) \{$$
$$\underbrace{\{ fail := fail + 1; totalFail := totalFail + 1 \}}_{\text{failed transmission}} [0.1] \underbrace{\{ fail := 0; sent := sent + 1 \}}_{\text{successful transmission}} \}$$

We fully automatically **verify** that

$$\text{wp}[\![C_{\text{brp}}]\!] (totalFail) \sqsubseteq [N \leq 3] \cdot (totalFail + 1) + [N > 3] \cdot \infty$$

by means of **is 4-induction**.

### Optimal Discrete Uniform Generation from Coin Flips [Lumbroso '13]

Sample uniformly from  $\{0, \dots, n - 1\}$  using **fair coin flips only**.

```
 $v := 1; c := 0; term := 0;$   
while ( $term = 0$ ) {  
   $v := 2 \cdot v;$   
   $\{c := 2 \cdot c\} [1/2] \{c := 2 \cdot c + 1\};$   
  if ( $v \geq n$ ) {  
    if ( $c < n$ ) {  $term := 1$  }  
    else {  $v := v - n; c := c - n$  }  
  }  
}
```

For  $n \in \{2, 3, 4, 5\}$ , we automatically **verify**  
 **$\Pr(c = i) \leq 1/n$  for all  $i \in \{0, \dots, n - 1\}$**   
by means of **2-, 3-, and 5-induction**.

***k*-induction is more powerful and decidable in the linear setting.**

The property

$$\text{wp}[\text{while}(c = 1) \{ c := 0 [1/2] \ x := x + 1 \}] (x) \sqsubseteq x + 1,$$

is provable by 2-induction. **But: Not every upper bound is provable by *k*-induction:**

$$\text{wp}[\text{while}(c = 1) \{ c := 0 [1/2] \ x := x + 1 \}] (x) \sqsubseteq 2x + 1$$

We have  $\Phi_x(2x + 1) \not\sqsubseteq 2x + 1$ ,  $\Phi_x((\Phi_x(2x + 1) \sqcap 2x + 1)) \not\sqsubseteq 2x + 1, \dots$

**Solution: Invariant Synthesis**

# Invariant Synthesis

... i.e., computing superinvariants.

Idea: To verify

$$\text{wp}[\text{while}(c = 1) \{ c := 0 \ [1/2] \ x := x + 1 \}] (x) \sqsubseteq 2x + 1$$

we **compute** an expectation  $I$  such that

1.  $I$  is a superinvariant:  $\Phi_x(I) \sqsubseteq I$
2.  $I$  is not too large:  $I \sqsubseteq 2x + 1$ .

We then have

$$\text{wp}[\text{while}(c = 1) \{ c := 0 \ [1/2] \ x := x + 1 \}] (x) \underbrace{\sqsubseteq}_I I \underbrace{\sqsubseteq}_{I \text{ is not too large}} 2x + 1 \checkmark$$

$I$  is superinvariant       $I$  is not too large

**Question: How to compute  $I$ ?**



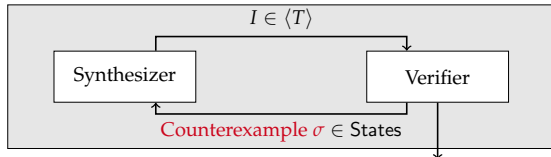
$$\text{wp}[\text{while } (c = 1) \{ c := 0 \ [1/2] \ x := x + 1 \}] (x) \sqsubseteq 2x + 1$$

Goal: Compute  $I \in \text{LinExp}$  such that  $\Phi_x(I) \sqsubseteq I$  and  $I \sqsubseteq 2x + 1$ .

Idea: Set up a *template*

$$T = [c \neq 1] \cdot x + [c = 1] \cdot (\alpha \cdot c + \beta \cdot x + \gamma)$$

New goal: Find appropriate  $\alpha, \beta, \gamma \in \mathbb{Q}$ .



$I$  is superinvariant and not too large ✓

Remember:

$$\Phi_x(I) \sqsubseteq I \text{ iff } \forall \sigma: \Phi_x(I)(\sigma) \leq I(\sigma)$$

$$I \sqsubseteq 2x + 1 \text{ iff } \forall \sigma: I(\sigma) \leq 2 \cdot \sigma(x) + 1$$

$$\text{wp}[\text{while } (c = 1) \{ c := 0 [1/2] \ x := x + 1 \}] (x) \sqsubseteq 2x + 1$$

$$T = [c \neq 1] \cdot x + [c = 1] \cdot (\alpha \cdot c + \beta \cdot x + \gamma)$$

Goal: Find  $I \in \langle T \rangle$  such that  $\Phi_x(I) \sqsubseteq I$  and  $I \sqsubseteq 2x + 1$ .

Synthesizer:  $I = [c = 1] \cdot (0 \cdot c + 0 \cdot x + 0) + [c \neq 1] \cdot x$

Verifier: **cex.**  $\sigma(c) = 1, \sigma(x) = 1$  with  $\Phi_x(I)(\sigma) > I(\sigma)$

Rule out the **cex.**  $\sigma$  in all future iterations.

$$\Phi_x(T) = [c \neq 1] \cdot x + [c = 1] \cdot \frac{1}{2} \cdot (\alpha \cdot c + \beta \cdot (x + 1) + \gamma + x)$$

$$T = [c \neq 1] \cdot x + [c = 1] \cdot (\alpha \cdot c + \beta \cdot x + \gamma)$$

$$\text{wp}[\text{while } (c = 1) \{ c := 0 \ [1/2] \ x := x + 1 \}] (x) \sqsubseteq 2x + 1$$

$$T = [c \neq 1] \cdot x + [c = 1] \cdot (\alpha \cdot c + \beta \cdot x + \gamma)$$

Goal: Find  $I \in \langle T \rangle$  such that  $\Phi_x(I) \sqsubseteq I$  and  $I \sqsubseteq 2x + 1$ .

Synthesizer: $I = [c = 1] \cdot (0 \cdot c + 0 \cdot x + 1) + [c \neq 1] \cdot x$	Learned constraints: $\frac{1}{2}(\alpha + 2\beta + \gamma + 1) \leq \alpha + \beta + \gamma$
Verifier: <b>cex.</b> $\sigma(c) = 1, \sigma(x) = 2$ with $\Phi_x(I)(\sigma) > I(\sigma)$	

$$\Phi_x(T) = [c \neq 1] \cdot x + [c = 1] \cdot \frac{1}{2} \cdot (\alpha \cdot c + \beta \cdot (x + 1) + \gamma + x)$$

$$T = [c \neq 1] \cdot x + [c = 1] \cdot (\alpha \cdot c + \beta \cdot x + \gamma)$$

$$\text{wp}[\text{while } (c = 1) \{ c := 0 \ [1/2] \ x := x + 1 \}] (x) \sqsubseteq 2x + 1$$

$$T = [c \neq 1] \cdot x + [c = 1] \cdot (\alpha \cdot c + \beta \cdot x + \gamma)$$

Goal: Find  $I \in \langle T \rangle$  such that  $\Phi_x(I) \sqsubseteq I$  and  $I \sqsubseteq 2x + 1$ .

Synthesizer: $I = [c = 1] \cdot (0 \cdot c + 1 \cdot x + 1) + [c \neq 1] \cdot x$		Learned constraints:
Verifier: <b>verified!</b> ✓		$\frac{1}{2}(\alpha + 2\beta + \gamma + 1) \leq \alpha + \beta + \gamma$ $\frac{1}{2}(\alpha + 3\beta + \gamma + 1) \leq \alpha + 2\beta + \gamma$

We have implemented invariant synthesis for probabilistic loops:

### CEGISRO2: CounterExample Guided Synthesis for PRObabilistic PROgrams

Given piecewise linear  $f, g$  and linear  $\text{while}(\varphi)\{body\}$ , cegispro2 aims to verify

$$\text{wp}[\text{while}(\varphi)\{body\}](f) \sqsubseteq g$$

by **computing** a piecewise linear expectation  $I$  with  $\Phi_f(I) \sqsubseteq I$  and  $I \sqsubseteq g$ .

Using the same technique cegispro2 can also compute upper bounds on

$$\text{ert}[\text{while}(\varphi)\{body\}](0)$$

by computing an ert-superinvariant.

Invariant synthesis enables verifying programs with gigantic state spaces.

For  $C_{\text{brp}}$  given by

$$\text{while} \left( \text{sent} < 8 \cdot 10^9 \wedge \text{fail} < 10 \right) \{$$
$$\underbrace{\{ \text{fail} := \text{fail} + 1; \text{totalFail} := \text{totalFail} + 1 \}}_{\text{failed transmission}} [0.01] \underbrace{\{ \text{fail} := 0; \text{sent} := \text{sent} + 1 \}}_{\text{successful transmission}} \}$$

We fully automatically **verify** that

$$\text{wp}[\![C_{\text{brp}}]\!] ([\text{fail} = 10]) \sqsubseteq [\text{sent} = 0 \wedge \text{fail} = 0] \cdot 0.0001 + [\neg(N = 0 \wedge \text{fail} = 0)] \cdot \infty.$$

in 58 seconds using 106 counterexamples.



Invariant synthesis enables upper bounding **expected runtimes** of infinite-state programs.

Consider the program  $C$  modeling a race between a tortoise and a hare:

```
while ( $h \leq t$ ) {  
   $t := t + 1$ ;  
  { skip } [1/2] {  $h := 1/11 \cdot [h + 0] + \dots + 1/11 \cdot [h + 10]$  } }
```

We fully automatically **compute** an ert-superinvariant<sup>2</sup> and obtain

$$\text{ert}\llbracket C \rrbracket(0) \sqsubseteq [h \leq t] \cdot (2/3 \cdot t - 2/3 \cdot h + 13/3) + [h > t] \cdot 1$$

in 1.3 seconds using 32 counterexamples.

<sup>2</sup>counting only the number of loop guard evaluations



The concepts of this lecture series provide the foundations for the **automatic verification of probabilistic programs.**

**Are you interested in writing a Bachelor/Master Thesis? Contact us!**

- ▶ Lutz Klinkenberg (Algebraic Techniques): `lutz.klinkenberg@cs.rwth-aachen.de`
- ▶ Tobias Winkler (Automata Theoretic Techniques):  
`tobias.winkler@cs.rwth-aachen.de`
- ▶ Ira Fesefeldt (Concurrency): `fesefeldt@cs.rwth-aachen.de`
- ▶ Philipp Schröder (Deductive Verification): `phisch@cs.rwth-aachen.de`
- ▶ Bahar Salmani (Bayesian Networks): `salmani@cs.rwth-aachen.de`
- ▶ Kevin Batz (Automatic Verificaton): `kevin.batz@cs.rwth-aachen.de`

`thesis@i2.informatik.rwth-aachen.de`