Automating Proof Rules for Probabilistic Programs

Christoph Matheja

Carl von Ossietzky Universität Oldenburg

DTU

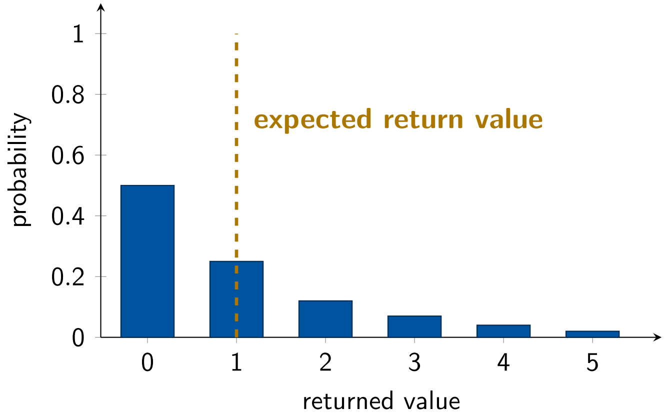
joint work with Kevin Batz, Benjamin Kaminski, Joost-Pieter Katoen, Philipp Schröer, Oliver Bøving

Aarhus, QEST+FORMATS 2025

What are probabilistic programs (PPs)?

Probabilistic program = ordinary program + sampling from probability distributions

```
fn geo() -> int {
    coin := flip();
    if (coin = heads) {
        return 0
    } else {
        return 1 + geo()
    }
}
```



What are probabilistic programs good for?

Universal modeling formalism

- Randomized algorithms
- Various kinds of (infinite-state) Markov models
- Communication and security protocols
- Bayesian networks, statistical models, ...

Typical analysis problems

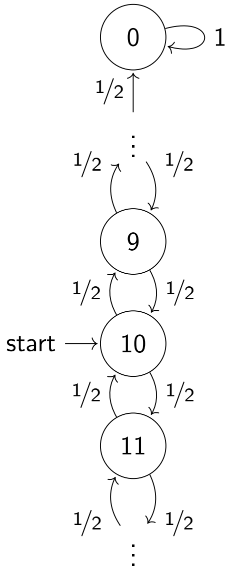
- Bounding probabilities of temporal properties
- Expected resource usage
- Sensitivity analysis, higher moments, ...,

Example: Random Walk

```
$x := 10$;
while $(x \neq 0)$ {
   if $(\mathit{flip}())$ {
      $x := x - 1$
   } else {
      $x := x + 1$
   }
}
```

Termination probability: 1

Expected runtime: ∞



Example: Probabilistic Termination Phenomena

```
fn foo() -> int {
  if (flip() = heads) {
    return 0
  } else {
    return 1 + foo()
             + foo()
             + foo()
  }
}
```

What is the probability that *foo* terminates?

1 (almost-sure)

1

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

Proving almost-sure termination on *one* input is as hard as proving that an ordinary program terminates on *all* inputs [Acta Inf. 2019]

Proof rules for reasoning about PPs (highly incomplete)

- **Expectation transformers**

  [Kozen 1983] [McIver & Morgan 2005] [JACM 2018] [POPL 2019-2023] [CAV 2021]

- **Supermartingales**

  [Chakarov et al. 2013] [Chatterjee et al. 2017-2025] [McIver et al. 2017]
  [Abate et al. 2024, 2025]

- **Probabilistic Hoare logics**

  [den Hartog 2002] [Barthe et al. 2016-2025] [Li et al. 2023] [Bao et al. 2025]
- **Exact inference techniques** [Gehr et al. 2016] [Saad et al. 2021]
- **Algebraic techniques** [Moosbrugger et al. 2020-2024]

Goal

Develop an **intermediate language** for probabilistic program verification techniques

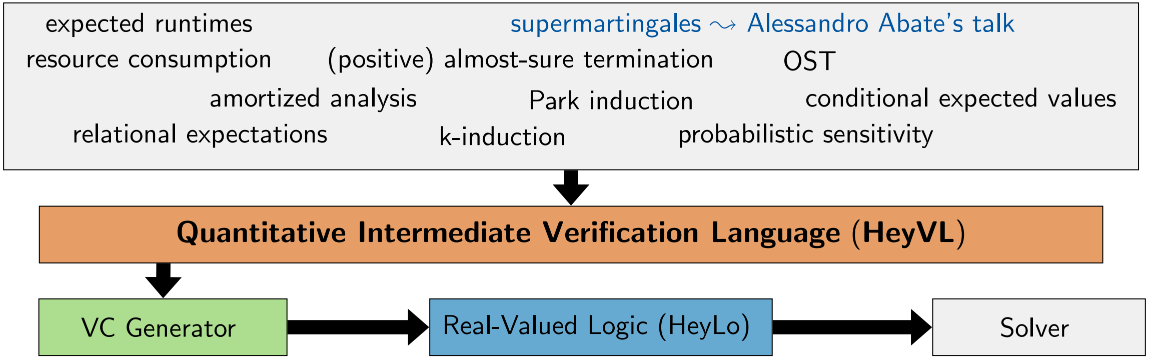
$\rightsquigarrow$ Support feature-rich probabilistic programs

$\rightsquigarrow$ Building efficient automated verifiers

Who is such a language for?

- Developers of proof rules $\rightsquigarrow$ rapid prototyping
- Developers of verification tools $\rightsquigarrow$ provide new verification backends?
- Practitioners $\rightsquigarrow$ combine and adapt verification techniques

Plan: A Verification Infrastructure for Probabilistic Programs

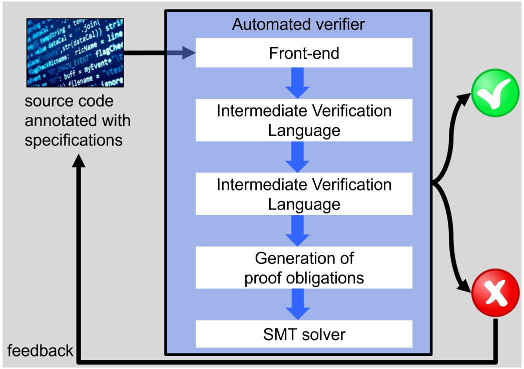

expected runtimes
supermartingales $\rightsquigarrow$ Alessandro Abate's talk
resource consumption
(positive) almost-sure termination
OST
amortized analysis
Park induction
conditional expected values
relational expectations
k-induction
probabilistic sensitivity
Quantitative Intermediate Verification Language (HeyVL)
VC Generator
Real-Valued Logic (HeyLo)
Solver

Inspiration: Classical Intermediate Languages à la Boogie

Idea: Build verifiers like compilers using a language for verification problems

- **Assertions** φ, ψ**:** first-order logic
- **Commands** C in intermediate language
- **Verification condition:** $\text{wp}[C](\texttt{true})$ valid

C	$\text{wp}[C](\varphi)$
`assert` ψ	$\psi \wedge \varphi$
`assume` ψ	$\psi \Rightarrow \varphi$
`havoc` x	$\forall x\colon \varphi$
C_1 `;` C_2	$\text{wp}[C_1](\text{wp}[C_2](\varphi))$
C_1 `[]` C_2	$\text{wp}[C_1](\varphi) \wedge \text{wp}[C_2](\varphi)$


source code
annotated with
specifications
Automated verifier
Front-end
Intermediate Verification Language
Intermediate Verification Language
Generation of proof obligations
SMT solver
feedback

Starting point: Weakest Preexpectations

[Kozen, 1983] [McIver & Morgan, 2005]

Why?

- All previous examples have been verified with expectation-based calculi
- Covers many supermartingales [McIver et al., 2017] [Takisaka et al., 2021]

Expectations

Program states: $\text{States} = \{\sigma \mid \sigma \colon \text{Vars} \to \mathbb{Q}\}$

Expectations: $\mathbb{E} = \{f \mid f \colon \text{States} \to \mathbb{R}_{\geq 0}^{\infty}\}$ think: **random variable**

$$f \preceq g \quad \text{iff} \quad \forall \sigma \in \text{States} \colon \quad f(\sigma) \leq g(\sigma)$$

Examples:

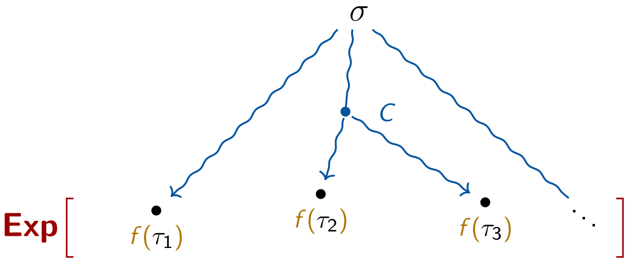
$$1 = \lambda\sigma.\ 1$$

$$[x < 10] = \lambda\sigma. \begin{cases} 1, & \text{if } \sigma \models x < 10 \\ 0, & \text{otherwise} \end{cases}$$

$$x^2 = \lambda\sigma.\ \sigma(x)^2$$

The Weakest Preexpectation

Given: probabilistic program C and postexpectation $f\colon \text{States} \to \mathbb{R}_{\geq 0}^{\infty}$
Running C on initial state σ yields a (sub-)distribution $[\![C]\!](s)$ over final states
Question: What is the **expected value** of f after termination of C?



$$\mathsf{wp}[C](f) \;=\; \lambda\sigma.\ \int_{[\![C]\!](\sigma)} f \;\in\; \mathbb{E} \quad = \{f \mid f\colon \text{States} \to \mathbb{R}_{\geq 0}^{\infty}\}$$

Examples

postexpectation f	**weakest preexpectation** $\mathrm{wp}[C](f)$
1	probability that C terminates
$[x < 10]$	probability that $x < 10$ holds upon termination
x^2	expected value of x^2 after termination of C

The weakest preexpectation calculus for pGCL

$\mathrm{wp}[C](f)$: expected value of f after termination of C evaluated in initial states

C	$\mathrm{wp}[C](f)$
skip	f
$x := \mu$	$\lambda\sigma.\sum_{v\in\mathbb{Q}} \mu(\sigma)(v)\cdot f[x\mapsto v](\sigma)$
$C_1 ; C_2$	$\mathrm{wp}[C_1](\,\mathrm{wp}[C_2](f)\,)$
if (b) $\{\ C_1\ \}$ **else** $\{\ C_2\ \}$	$[b]\cdot\mathrm{wp}[C_1](f) \ +\ [\neg b]\cdot\mathrm{wp}[C_2](f)$
$\{C_1\}\ [p]\ \{C_2\}$	$p\cdot\mathrm{wp}[C_1](f)\ +\ (1-p)\cdot\mathrm{wp}[C_2](f)$
while (b) $\{\ C\ \}$	$\mathit{lfp}(\Phi_f)$, where $\underbrace{\Phi_f(X) \overset{\text{def}}{=} [b]\cdot\mathrm{wp}[C](X)\ +\ [\neg b]\cdot f}_{\text{characteristic function of the loop}}$

Example: Loop-free programs

```
/// $1/2 \cdot 0 + 1/2 \cdot 1$
{
    /// $0$
    $x := 0$
    /// $x$
} [1/2] {
    /// $1$
    $x := 1$
    /// $x$
}
/// $x$
```

Proving upper bounds on expected values of loops

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

Lemma (Loop invariants from Park induction)

If $\Phi_f(I) \preceq I$ then $\mathrm{wp}[\mathbf{while}\ (b)\ \{\ C\ \}](f) = \mathit{lfp}(\Phi_f) \preceq I$

Invariant: $I \overset{\text{def}}{=} x + [c = 1]$

$$\Phi_x(I) = [c \neq 1] \cdot x + [c = 1] \cdot 1/2 \cdot x + [c = 1] \cdot 1/2 \cdot (x + 2)$$
$$= x + [c = 1] \preceq I \quad \checkmark$$

Towards a verification infrastructure for probabilistic programs

1. **What are quantitative assertions?**

2. **What is an intermediate language for probabilistic program verification?**

3. **What can be encoded in such a language?**

4. **What automation is available?**

Syntactic Expectations

Classical verification:

$$Pre \;\models\; \mathrm{wp}[C](Post)$$

Probabilistic verification:

$$g \;\preceq/\succeq\; \mathrm{wp}[C](f)$$

Theorem (Cook, 1978)

If $C \in$ GCL and $Post \in$ FO-arithmetic then

$$\mathrm{wp}[C](Post) \;\in\; \text{FO-arithmetic.}$$

Expressiveness for expectations?

If $C \in$ pGCL and $f \in$ Exp then

$$\mathrm{wp}[C](f) \;\in\; \mathrm{Exp}.$$

What is an expressive syntax Exp for expectations $\mathbb{E} = \{f \mid f\colon \text{States} \to \mathbb{R}_{\geq 0}^{\infty}\}$?

A trivial expressive syntax

$$\mathsf{Exp} \;=\; \{0\} \quad \text{since} \quad \mathsf{wp}[C](0) = 0 \text{ for all } C \in \mathsf{pGCL}$$

What is a sensible syntax Exp for expectations?

Towards a sensible syntax

Requirement: $[b] \in \mathsf{Exp}$ for every Boolean expression b

```
/// $\frac{\sqrt{5}-1}{2} \notin \mathbb{Q}_{\geq 0}$
$x := 1;$
while $(x > 0)$ {
   $\{x := x + 2\}$ $[1/2]$ $\{x := x - 1\}$
}
/// $[\mathtt{true}] = 1 \in \mathbb{Q}_{\geq 0}$
```

$\rightsquigarrow$ A sensible syntax must cover irrational and non-algebraic numbers

An Expressive Syntax for Expectations

$$\begin{array}{lcll}
\varphi & ::= & a & \text{(arithmetic expressions over rational variables)} \\
 & \mid & [b] \cdot \varphi & \text{(guarding with Boolean expressions)} \\
 & \mid & a \cdot \varphi & \text{(rescaling with arithmetic expressions)} \\
 & \mid & \varphi + \varphi & \text{(addition of expectations)} \\
 & \mid & \reflectbox{S} x.\ \varphi & \text{(supremum quantifier over variable } x\text{)} \\
 & \mid & \rotatebox{-30}{\reflectbox{J}} x.\ \varphi & \text{(infimum quantifier over variable } x\text{)}
\end{array}$$

Example: $\reflectbox{$\mathsf{S}$} x.\ 3 \cdot [x \cdot x < 2] \cdot x \quad = \quad 3 \cdot \sqrt{2}$

Examples of expressible expectations

Polynomials	$x^2 + 3 \cdot y + 4$	(appear in martingale-based reasoning)
Rational functions	$\frac{x^2 + 3 \cdot y + 4}{2 \cdot x + y}$	(appear in analysis of probabilistic models)
Harmonic numbers	$\sum_{k=1}^{x} {}^{1}/_{k}$	(appear in runtime analysis of randomized algorithms)

Expressing Weakest Preexpectations of Loops

$$\mathrm{wp}[\texttt{while}\ (b)\ \{\ C\ \}](\varphi)$$

$$= \quad \lambda\sigma_0.\ \sum_{\sigma_0\ldots\sigma_{k-1}} [\neg b]\,(\sigma_{k-1}) \ \cdot\ \varphi(\sigma_{k-1}) \ \cdot\ \prod_{i=0}^{k-2} \mathrm{wp}[\texttt{if}\ (b)\ \{C\}](\varphi_{\sigma_{i+1}})(\sigma_i)$$

Technical challenges:

- Encoding sequences of rationals and states via Gödelization
- Encoding variable-length sums and products
- Averaging over potentially irrational values via Dedekind cuts

Relative Completeness

Theorem (Expressiveness, POPL 2021)

If $C \in \mathsf{pGCL}$ and $\varphi \in \mathsf{Exp}$, one can construct a syntactic expectation $\psi \in \mathsf{Exp}$ such that

$$\psi \quad = \quad \mathsf{wp}[C](\varphi).$$

Idea: extend the syntax Exp to enable encoding proof rules for bounds on $\mathsf{wp}[C](\varphi)$

$$\begin{array}{lclcl}
\mathsf{wp}[\mathsf{havoc}\ x](\varphi) & = & \forall x.\varphi & \rightsquigarrow & \reflectbox{J}\, x.\ \varphi \\
\mathsf{wp}[\mathsf{assert}\ \psi](\varphi) & = & \psi \wedge \varphi & \rightsquigarrow & ? \\
\mathsf{wp}[\mathsf{assume}\ \psi](\varphi) & = & \psi \Rightarrow \varphi & \rightsquigarrow & ?
\end{array}$$

Our language should enable reasoning about *lower* and *upper* bounds

Expectations for Quantitative Conjunctions

Definition

$$\varphi \sqcap \psi \quad = \quad \lambda\sigma.\, \min\{\varphi(\sigma), \psi(\sigma)\}$$

New indicator function:

$$?(b) \quad = \quad [b] \cdot \infty \quad = \quad \lambda\sigma. \begin{cases} \infty, & \text{if } \sigma \models b \\ 0, & \text{otherwise} \end{cases}$$

Intuition: `true` and `false` are represented by ∞ and 0 in $\mathbb{R}_{\geq 0}^{\infty}$

Backward compatibility: $?(b_1 \wedge b_2) \quad = \quad ?(b_1) \sqcap ?(b_2)$

Expectations for Quantitative Implications

Definition

$$\varphi \;\Rightarrow\; \psi \quad = \quad \lambda\sigma. \begin{cases} \infty, & \text{if } \varphi(\sigma) \;\leq\; \psi(\sigma) \\ \psi(\sigma), & \text{otherwise} \end{cases}$$

Example

$$[\![?(b) \Rightarrow \varphi]\!](\sigma) \quad = \quad \begin{cases} [\![\varphi]\!](\sigma), & \text{if } \sigma \models b \\ \infty, & \text{otherwise} \end{cases}$$

Lemma (Adjointness of $\sqcap$ and $\Rightarrow$)

$$\rho \sqcap \varphi \;\preceq\; \psi \qquad \text{iff} \qquad \rho \;\preceq\; \varphi \Rightarrow \psi$$

The quantitative assertion language HeyLo

φ	$::=$	a	(arithmetic expressions over rational variables)
	$\mid$	$?(b)$	(embedding of Boolean expressions)
	$\mid$	$\varphi + \varphi$	(sums)
	$\mid$	$\varphi \cdot \varphi$	(products)
	$\mid$	$\varphi \sqcap \varphi$	(quantitative conjunction (minimum))
	$\mid$	$\varphi \Rightarrow \varphi$	(quantitative implication)
	$\mid$	$\reflectbox{$\mathsf{S}$} x.\ \varphi$	(supremum quantifier over variable x)
	$\mid$	$\reflectbox{$\mathsf{J}$} x.\ \varphi$	(infimum quantifier over variable x)
	$\mid$	$\ldots$	(dual versions for upper bound reasoning)

Algebraic Facts

Definition

φ is **valid** iff $\forall \sigma.\ \llbracket \varphi \rrbracket(\sigma) = \infty$

Theorem

$\varphi \preceq \psi$ iff $\varphi \Rightarrow \psi$ is valid

Definition

$$\neg \varphi \quad = \quad \varphi \Rightarrow 0 \quad = \quad \lambda \sigma. \begin{cases} \infty, & \text{if } \llbracket \varphi \rrbracket(\sigma) = 0 \\ 0, & \text{otherwise} \end{cases}$$

Example

$$\nabla(\varphi) \quad = \quad \neg\neg \varphi \quad = \quad \lambda \sigma. \begin{cases} 0, & \text{if } \llbracket \varphi \rrbracket(\sigma) = 0 \\ \infty, & \text{otherwise} \end{cases}$$

$(\mathsf{Exp}, \sqcap, \Rightarrow, \neg, 0, \infty)$ is a Heyting algebra (hence the name HeyLo)

Dual HeyLo Constructs

Main idea: construct dual Heyting algebra $(\mathsf{Exp}, \sqcup, \leftsquigarrow, \sim, \infty, 0)$ with analogous properties

$$0 \quad \rightsquigarrow \quad \texttt{true} \qquad \text{and} \qquad \infty \quad \rightsquigarrow \quad \texttt{false}$$

Co-conjunction

$$\varphi \sqcup \psi \quad = \quad \lambda\sigma.\, \max\{\varphi(\sigma), \psi(\sigma)\}$$

Coimplication

$$\varphi \leftsquigarrow \psi \quad = \quad \lambda\sigma. \begin{cases} 0, & \text{if } [\![\varphi]\!](\sigma) \geq [\![\psi]\!](\sigma) \\ [\![\psi]\!](\sigma), & \text{otherwise} \end{cases}$$

Co-negation

$$[\![\sim\varphi]\!] \;=\; \lambda\sigma. \begin{cases} 0, & \text{if } [\![\varphi]\!](\sigma) = \infty \\ \infty, & \text{otherwise}\ . \end{cases}$$

Double co-negation

$$[\![\triangle(\varphi)]\!] \;=\; [\![\sim\sim\varphi]\!] \;=\; \lambda\sigma. \begin{cases} \infty, & \text{if } [\![\varphi]\!](\sigma) = \infty \\ 0, & \text{otherwise} \end{cases}$$

What are those HeyLo formulae good for?

Reminder: If $\Phi_{\varphi}(I) \preceq I$ then $\mathrm{wp}[\mathtt{while}\ (b)\ \{\ C\ \}](\varphi) \ = \ \mathit{lfp}(\Phi_{\varphi}) \ \preceq \ I$

Verification condition: [Navarro & Olmedo, 2022]

$$\mathrm{vc}[\mathtt{while}\ (b)\ \ \mathtt{invariant}\ I\ \{\ C\ \}](\varphi) \quad = \quad \begin{cases} I, & \text{if } \Phi_{\varphi}(I) \preceq I \\ 0, & \text{otherwise} \end{cases}$$

Corresponding HeyLo formula:

$$\underbrace{\mathcal{L}\, x_1.\ \ldots\ \mathcal{L}\, x_n.\ \triangle(\Phi_{\varphi}(I) \Rightarrow I)}_{\text{evaluate to 0 if } \Phi_{\varphi}(I) \not\preceq I} \qquad \underbrace{\sqcap}_{\text{and}} \qquad \underbrace{I}_{\text{evaluate to invariant otherwise}}$$

Towards a verification infrastructure for probabilistic programs

1. **What are quantitative assertions?**

   $\rightsquigarrow$ **HeyLo formulae**, e.g. $(\mathcal{L}\, x_1.\ \ldots\ \mathcal{L}\, x_n.\ \triangle(\Phi_{\varphi}(I) \Rightarrow I)) \sqcap I$

2. **What is an intermediate language for probabilistic program verification?**

3. **What can be encoded in such a language?**

4. **What automation is available?**

The Intermediate Verification Language HeyVL

Ingredients of HeyVL: Loop-free pGCL

- + Boogie-like verification-specific commands
- + `validate` for enforcing conditions of proof rules
- + Dual versions, e.g. for upper-bound reasoning
- + Rewards for reasoning about resource consumption

Semantics: wp-style verification condition generator

$$\mathrm{vc}[C] \colon \mathrm{HeyLo} \to \mathrm{HeyLo}$$

The Intermediate Verification Language HeyVL

C	$\mathrm{vc}[C](\varphi)$	dual $\mathrm{vc}[co\ldots](\varphi)$
$x \mathrel{:=} \mu$	$\mathrm{wp}[x \mathrel{:=} \mu](\varphi)$	
$C_1 ; C_2$	$\mathrm{vc}[C_1](\,\mathrm{vc}[C_2](\varphi)\,)$	
C_1 [] C_2	$\mathrm{vc}[C_1](\varphi) \sqcap \mathrm{vc}[C_2](\varphi)$	$\mathrm{vc}[C_1](\varphi) \sqcup \mathrm{vc}[C_2](\varphi)$
`assert` ψ	$\psi \sqcap \varphi$	$\psi \sqcup \varphi$
`assume` ψ	$\psi \Rightarrow \varphi$	$\psi \leftsquigarrow \varphi$
`havoc` x	$\ell x.\ \varphi$	$\text{Ƨ} x.\ \varphi$
`validate`	$\triangle(\varphi)$	$\triangledown(\varphi)$
`reward` a	$a + \varphi$	

Example: lower bound reasoning for weakest preexpectations

Given: pGCL program $C \in \text{HeyVL}$ expectations $\varphi, \psi \in \text{HeyLo}$

$$
\begin{array}{ll}
 & \psi \preceq \mathsf{wp}[C](\varphi) \\
\text{iff} & \psi \preceq \mathsf{vc}[C](\varphi) \\
\text{iff} & \psi \preceq \mathsf{vc}[C](\varphi \sqcap \infty) \\
\text{iff} & \psi \Rightarrow \mathsf{vc}[C](\varphi \sqcap \infty) \ \text{valid} \\
\text{iff} & \mathsf{vc}[\texttt{assume}\ \psi;\ C;\ \texttt{assert}\ \varphi](\infty) \ \text{valid}
\end{array}
$$

$\leadsto$ Lower bound reasoning reduces to checking validity

$\leadsto$ Upper bound reasoning dually reduces to checking covalidity

Towards a verification infrastructure for probabilistic programs

1. **What are quantitative assertions?**

   $\rightsquigarrow$ **HeyLo formulae**, e.g. $(\text{\textit{ʖ}}\, x_1.\ \ldots\ \text{\textit{ʖ}}\, x_n.\ \triangle(\Phi_f(I) \Rightarrow I)) \sqcap I$

2. **What is an intermediate language for probabilistic program verification?**

   $\rightsquigarrow$ **HeyVL** $\approx$ pGCL $+$ dual Boogie-like verification-specific commands

3. **What can be encoded in such a language?**

4. **What automation is available?**

Example: preexpectation calculi

$\mathrm{wp}[C](\varphi) \quad = \quad$ expected value of φ after termination of C

$\mathrm{wlp}[C](\varphi) \quad = \quad \mathrm{wp}[C](\varphi) +$ probability that C does not terminate

$\rightsquigarrow$ straightforward to encode in HeyVL for loop-free pGCL programs C

Example

$$\underbrace{\texttt{if}\ (b)\ \{\ C_1\ \}\ \texttt{else}\ \{\ C_2\ \}}_{\text{pGCL}} \quad \rightsquigarrow \quad \underbrace{\{\,\texttt{assume}\ ?(b);\ C_1\,\}\ \ \texttt{[]}\ \ \{\,\texttt{assume}\ ?(\neg b);\ C_2\,\}}_{\text{HeyVL}}$$

Example: Encoding Park Induction for Partial Correctness

Given: $\texttt{while}\ (b)\ \{\ C\ \}$ with modified variables $x_1, \ldots, x_n$

Characteristic function: $\Phi_\psi(I) \;=\; [b] \cdot \mathrm{wlp}[C](I) \;+\; [\neg b] \cdot \psi$

Proof rule: If $I \preceq \Phi_\psi(I)$ then $\mathrm{wlp}[\texttt{while}\ (b)\ \{\ C\ \}](\psi) \;=\; \mathit{gfp}(\Phi_\psi) \;\succeq\; I$

```
assert $I$;
havoc $x_1, \ldots, x_n$;
validate;
assume $I$;
if ($b$) {
    $C$;
    assert $I$;
    assume ?(false)
} else { }   /// $\psi$
```

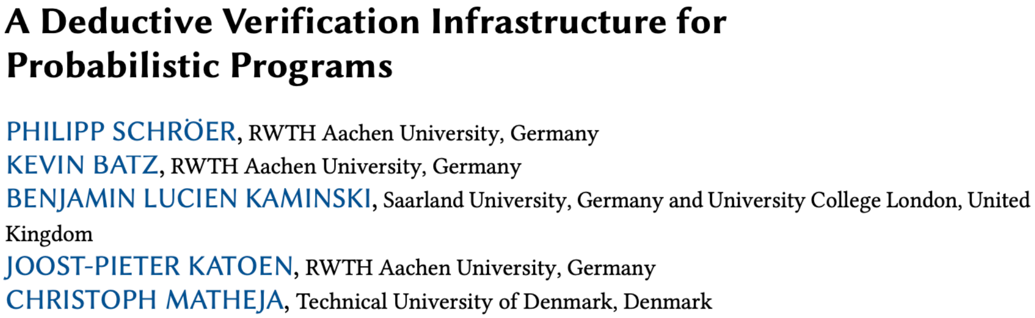
Soundness of HeyVL encoding

$$\begin{aligned} & \mathrm{wlp}[\texttt{while}\ (b)\ \{\ C\ \}](\psi) \\ \succeq \quad & \mathrm{vc}[\mathit{encoding}](\psi) \\ = \quad & \text{ʅ}\, x_1.\ \ldots\ \text{ʅ}\, x_n.\ \triangle(I \Rightarrow \Phi_\psi(I)) \;\sqcap\; I \\ = \quad & \begin{cases} I, & \text{if } \Phi_\psi(I) \succeq I \\ 0, & \text{otherwise} \end{cases} \end{aligned}$$

Some proof rules encoded in HeyVL

Problem	Verification Technique	Source
LPROB	wlp + Park induction	[McIver & Morgan, 2005]
	wlp + latticed k-induction	[OOPSLA 2023]
UPROB	wlp + ω-invariants	[Kaminski, 2019]
UEXP	wp + Park induction	[McIver & Morgan, 2005]
	wp + latticed k-induction	[CAV 2021]
	wp + ω-invariants	[Kaminski, 2019]
LEXP	wp + Optional Stopping Theorem	[Hark et al., 2019]
CEXP	wp + conditioning	[Olmedo et al., 2018]
UERT	ert calculus + UEXP rules	[ESOP, 2016]
LERT	ert calculus + ω-invariants	[ESOP, 2016]
AST	parametric super-martingales	[McIver et al., 2018]
PAST	program analysis with martingales	[Chakarov & Sankaranarayanan, 2013]

Details



A Deductive Verification Infrastructure for Probabilistic Programs

PHILIPP SCHRÖER, RWTH Aachen University, Germany
KEVIN BATZ, RWTH Aachen University, Germany
BENJAMIN LUCIEN KAMINSKI, Saarland University, Germany and University College London, United Kingdom
JOOST-PIETER KATOEN, RWTH Aachen University, Germany
CHRISTOPH MATHEJA, Technical University of Denmark, Denmark

[OOPLSA 2023]

Towards a verification infrastructure for probabilistic programs

1. **What are quantitative assertions?**

   $\leadsto$ **HeyLo formulae**, e.g. $(\reflectbox{$\mathcal{S}$} x_1.\ \ldots\ \reflectbox{$\mathcal{S}$} x_n.\ \triangle(\Phi_f(I) \Rightarrow I)) \sqcap I$

2. **What is an intermediate language for probabilistic program verification?**

   $\leadsto$ **HeyVL** $\approx$ pGCL + dual Boogie-like verification-specific commands

3. **What can be encoded in such a language?**

   $\leadsto$ **many proof rules based on expectations or supermartingales**

4. **What automation is available?**

Caesar: an SMT-backed verifier for HeyVL

$\sim$10k LOC of Rust code

- Verification condition generator
- Recursive procedures, mathematical data types, ...
- Frontend for simple weakest preexpectation calculi

Performance is competitive with specialized tools demonstrating new proof rules

- Custom rewritings for dealing with ∞
- Quantifier elimination for ɭ , Ƨ

Caesar enables rapid prototyping of new proof rules for probabilistic programs

What are concrete programs verified with Caesar?

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

- Try to send N packets via a lossy channel
- Transmitting a single packet fails with probability p
- Attempt at most F retransmissions per packet; otherwise abort

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

- Verified properties: upper bounds on the expected number of transmissions
- Encoded technique: Latticed k-Induction [CAV 2021]

Variant of Random Walk

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

- Verified property: almost-sure termination
- Encoded technique: parametric supermartingales [McIver et al., 2017]

Coupon Collector's Problem

```
while $(0 < x)$ {
    $i := N + 1$;
    while $(0 < x < i)$ {
        $i :\approx$ unif(1, $N$)
    }
    $x := x - 1$
}
```

- Verified property: expected runtime $\leq N \cdot \mathcal{H}(N) = N \cdot \sum_{k=1}^{N} 1/k$
- Encoded technique: expected runtime calculus [JACM 2018]

Conclusion

An infrastructure for automating verification of probabilistic programs

1. **What are quantitative assertions?**

   $\rightsquigarrow$ **HeyLo formulae**, e.g. $(\mathcal{L}\, x_1.\ \ldots\ \mathcal{L}\, x_n.\ \triangle(\Phi_f(I) \Rightarrow I)) \sqcap I$

2. **What is an intermediate language for probabilistic program verification?**

   $\rightsquigarrow$ **HeyVL** $\approx$ pGCL + dual Boogie-like verification-specific commands

3. **What can be encoded in such a language?**

   $\rightsquigarrow$ **many proof rules based on expectations or supermartingales**

4. **What automation is available?**

   $\rightsquigarrow$ **Caesar**, an SMT-backed verification tool for HeyVL

Further developments

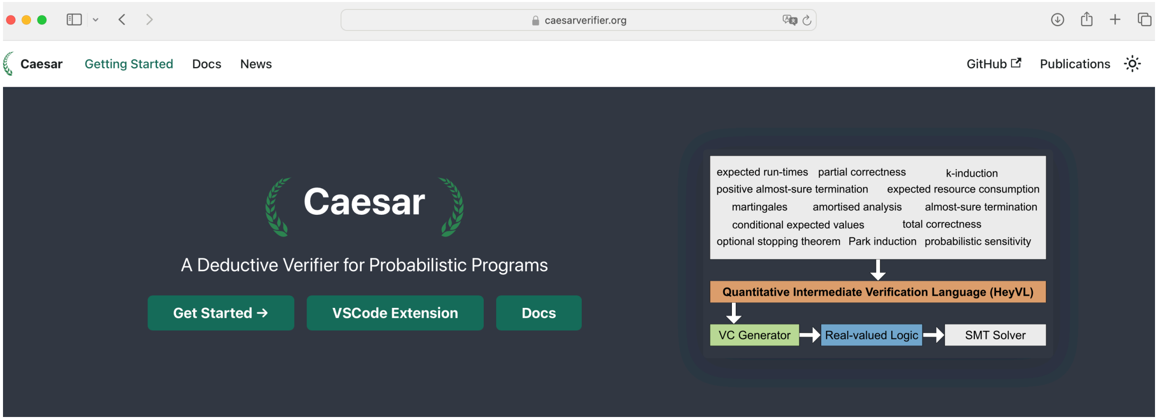
Follow-up works

- HeyVL semantics as (infinite-state) stochastic games [AISOLA 2024]
- Reasoning about continuous distributions with HeyVL [Batz et al., 2025]
- DIREC project: encoding of the relational preexpectation calculus [POPL 2021]
- Lean formalization (WIP)
  - $\rightsquigarrow$ Interactive verification backend
  - $\rightsquigarrow$ Simplify mechanization of soundness proofs

Future work

- Improve automation, e.g. better quantifier elimination [Batz et al., 2025]
- Alternative backends for HeyVL, e.g. Storm
- Leverage stochastic indepence à la probabilistic Hoare logics like PSL, Lilac, Bluebell

Thanks for listening



caesarverifier.org

Caesar Getting Started Docs News GitHub Publications

Caesar

A Deductive Verifier for Probabilistic Programs

Get Started → VSCode Extension Docs

expected run-times
partial correctness
k-induction
positive almost-sure termination
expected resource consumption
martingales
amortised analysis
almost-sure termination
conditional expected values
total correctness
optional stopping theorem
Park induction
probabilistic sensitivity
Quantitative Intermediate Verification Language (HeyVL)
VC Generator
Real-valued Logic
SMT Solver

caesarverifier.org