Foundations for Deductive Verification of Continuous Probabilistic Programs

From Lebesgue to Riemann and Back

KEVIN BATZ, RWTH Aachen University, Germany and University College London, United Kingdom
JOOST-PIETER KATOEN, RWTH Aachen University, Germany
FRANCESCA RANDONE, University of Trieste, Italy
TOBIAS WINKLER, RWTH Aachen University, Germany

We lay out novel foundations for the computer-aided verification of guaranteed bounds on expected outcomes of imperative probabilistic programs featuring (i) general *loops*, (ii) *continuous* distributions, and (iii) *conditioning*. To handle loops we rely on user-provided quantitative *invariants*, as is well established. However, in the realm of continuous distributions, *invariant verification* becomes extremely challenging due to the presence of *integrals* in expectation-based program semantics. Our key idea is to soundly *under-* or *over-approximate* these integrals via *Riemann sums*. We show that this approach enables the SMT-based invariant verification for programs with a fairly general control flow structure. On the theoretical side, we prove *convergence* of our Riemann approximations, and establish coRE-completeness of the central verification problems. On the practical side, we show that our approach enables to use existing automated verifiers targeting *discrete* probabilistic programs for the verification of programs involving *continuous sampling*. Towards this end, we implement our approach in the recent quantitative verification infrastructure Caesar by encoding Riemann sums in its intermediate verification language. We present several promising case studies.

CCS Concepts: • **Theory of computation** → **Probabilistic computation**; **Logic and verification**; **Invariants**; **Pre- and post-conditions**; **Program verification**; • **Mathematics of computing** → *Probabilistic inference problems.*

Additional Key Words and Phrases: probabilistic programs, deductive program verification, continuous distributions, quantitative loop invariants, weakest preexpectations, approximate integration, SMT solving

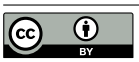
ACM Reference Format:
Kevin Batz, Joost-Pieter Katoen, Francesca Randone, and Tobias Winkler. 2025. Foundations for Deductive Verification of Continuous Probabilistic Programs: From Lebesgue to Riemann and Back. *Proc. ACM Program. Lang.* 9, OOPSLA1, Article 95 (April 2025), 28 pages. https://doi.org/10.1145/3720429

1 Introduction

Probabilistic programs (PP) are usual programs with the additional ability to

(1) *branch* their flow of control based on the outcomes of coin flips,
(2) *sample* numbers from predefined continuous and/or discrete probability distributions, and
(3) *condition* their current state on observations using dedicated `observe`-instructions.

Authors' Contact Information: Kevin Batz, RWTH Aachen University, Aachen, Germany and University College London, London, United Kingdom, kevin.batz@cs.rwth-aachen.de; Joost-Pieter Katoen, RWTH Aachen University, Aachen, Germany, katoen@cs.rwth-aachen.de; Francesca Randone, University of Trieste, Trieste, Italy, francesca.randone@units.it; Tobias Winkler, RWTH Aachen University, Aachen, Germany, tobias.winkler@cs.rwth-aachen.de.



ACM 2475-1421/2025/4-ART95
https://doi.org/10.1145/3720429

Programs with one or more of these (partly overlapping) abilities have been extensively studied in different contexts and throughout various communities: *Randomized algorithms* typically rely on ability (1) to speed up computations on average [38] or for breaking symmetries in distributed computing [33]. *Sampling* and *conditioning* (abilities (2) and (3)) are extensively employed in connection with Bayes' rule in machine learning, AI, and cognitive science to statistically infer information from observed data [28, 29, 54]. *Continuous distributions* are pivotal for these applications.

Many applications require *expressive* PP with the usual constructs from traditional programs — including unbounded `while`-loops. The latter pose a major challenge: For instance, reasoning about probabilistic termination is provably harder than reasoning about classical termination [37].

Guaranteed Bounds on Expected Outcomes via Weakest Pre-Expectations. In recent years, there has been considerable interest in analyzing expressive PP in a mathematically rigorous manner — with *hard* rather than statistical guarantees — see e.g., [2, 15, 26, 27, 43] as representative examples. Such stringent analyses are motivated by the fact that more conventional statistical techniques can be far off even for simple programs [11], and may be inappropriate for safety-critical applications.

The *weakest pre-expectation (*wp*) calculus* is a prominent means to specify and establish a broad spectrum of PP properties in a rigorous manner [35, 39, 41]. Generalizing Dijkstra's classical weakest pre-*conditions*, the central objects are so-called *expectations* f — random variables over a program's state space mapping program states to numbers — which take over the role of predicates from classical program verification. More precisely, given a probabilistic program C, an expectation f, and an initial program state σ, we have

$$\mathsf{wp}[\![C]\!]\,(f)\,(\sigma) \quad = \quad \begin{array}{c} \textit{expected value} \text{ of } f \text{ w.r.t. the distribution of } \textit{final} \text{ states} \\ \text{reached after executing } C \text{ on } \textit{initial} \text{ state } \sigma \\ \text{and aborting all executions violating an observation .} \end{array}$$

Hence, $\mathsf{wp}[\![C]\!]\,(f)$ is a map from *initial* program states to *expected final* values of f. The weakest *liberal* pre-expectation $\mathsf{wlp}[\![C]\!]\,(f)$ adds to the above quantity the probability of C's divergence[1]. Both wp and wlp can be defined by induction on the structure of C. Notice that neither of these calculi yield *conditional* expected values as one might expect in the presence of conditioning. One can, however, define *conditional weakest pre-expectations* [46, 47] as[2]

$$\mathsf{cwp}[\![C]\!]\,(f)\,(\sigma) \;=\; \frac{\mathsf{wp}[\![C]\!]\,(f)\,(\sigma)}{\mathsf{wlp}[\![C]\!]\,(1)\,(\sigma)} \;=\; \frac{\text{expected value of } f \ldots}{\text{probability of not violating an observation}}\,,$$

which indeed yields the sought-after conditional expected values. We refer to quantities such as $\mathsf{wp}[\![C]\!]\,(f)\,(\sigma)$, $\mathsf{wlp}[\![C]\!]\,(f)\,(\sigma)$, and $\mathsf{cwp}[\![C]\!]\,(f)\,(\sigma)$ as *(conditional) expected outcomes* of probabilistic programs. Reasoning about expected outcomes of loops is typically tackled by means of *quantitative loop invariants*, which are, naturally, often hard to find. However, in the presence of continuous sampling, even *verifying* a *given* candidate loop invariant poses severe challenges as reasoning about the required expected values involves possibly complex *integrals*. The aim of this paper is to lay the foundations for automated techniques tackling these challenges.

Problem Statement. We study *imperative* PP with all three abilities (1), (2), and (3). Our goal is to

> *semi-automatically verify bounds on (conditional) expected outcomes of programs featuring continuous uniform sampling, unbounded* `while`*-loops with user-provided quantitative invariants, and conditioning.*

[1] Provided f is upper-bounded by 1. See Section 4.2 for details.

[2] Notice that this quantity is undefined if $\mathsf{wlp}[\![C]\!]\,(1)\,(\sigma) = 0$ as is natural when conditioning on a probability-0-event.

"Semi-automatic" means that we focus on verification of *user-provided* quantitative loop invariants in the sense of [41]. Unlike fully automatic approaches we do not *synthesize* such invariants automatically, which is a promising direction for future work but outside the scope of this paper.

Approach. Our key idea is simple, yet powerful:

> *We replace the integrals occurring in the definition of the programs' exact weakest pre-expectations by simpler sound approximations, namely lower and upper Riemann sums.*

This results in a family of approximate *lower* and *upper Riemann* wp *transformers*, denoted by $\underline{\mathsf{wp}}^N$ and $\overline{\mathsf{wp}}^N$, where N is the number of sub-intervals used to discretize the domain of integration. Crucially, these Riemann expectation transformers ultimately give rise to automated SMT-based techniques for verifying quantitative loop invariants. Let us now consider an introductory example.

Illustrative Example: A Monte Carlo Approximator. The program shown in Figure 1 draws `M` (x, y)-samples uniformly at random from the unit square. Each time a sample lands in the quarter unit circle, the variable `count` is incremented. The program hence approximates the transcendental number $\frac{\pi}{4} \approx 0.785$ (the area of the quarter unit circle) in a Monte Carlo manner.

We will first focus on the blue fragment C_{inner} of the program from Figure 1. C_{inner} contains two uniform continuous sampling instructions. Suppose we aim to compute the expected value of `count` after executing C_{inner}. Applying the rules for determining wp's [53], we obtain the following double Lebesgue integral, where $[\ldots]$ denotes the indicator function of the enclosed predicate:

$$\mathsf{wp}[\![C_{inner}]\!]\,(\mathtt{count}) \;=\; \underbrace{\mathtt{count} + \int_0^1 \int_0^1 \left[\mathtt{x}^2 + \mathtt{y}^2 \leq 1\right] d\boldsymbol{\lambda}(x), d\boldsymbol{\lambda}(y) \;=\; \mathtt{count} + \frac{\pi}{4}}_{\text{expected final value of count in terms of its initial value}} . \qquad (\dagger)$$

This reflects the fact that, in *expectation*, C_{inner} increments `count` by $\frac{\pi}{4}$.

Symbolic integration-based tools such as PSI [26] can solve the above integral directly. However, we observe two significant issues: (i) there are integrals that cannot be evaluated symbolically in closed form, and (ii) even if all guards and assignments in the program are polynomial and distributions are restricted to uniform distributions in $[0, 1]$, the resulting integrals may evaluate to transcendental numbers, making automation notoriously difficult.

As outlined, instead of solving integrals exactly, our strategy is to soundly *under-* or *over-approximate* them by lower or upper Riemann sums. In our example, the resulting upper sum

$$\frac{1}{N^2} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \sup_{\xi \in [\frac{i}{N}, \frac{i+1}{N}]} \sup_{\zeta \in [\frac{j}{N}, \frac{j+1}{N}]} \left[\xi^2 + \zeta^2 \leq 1\right],$$

where $N \geq 1$, is a guaranteed upper bound on the double integral in (†). For instance, with $N = 16$, we can verify that the upper sum is at most 0.85, and hence that $\mathsf{wp}[\![C_{inner}]\!]\,(\mathtt{count})\,(\sigma) \leq \sigma(\mathtt{count}) + 0.85$ for all initial states σ. Crucially, to prove upper bounds on upper Riemann sums, we do *not* have to evaluate any suprema explicitly because for all $A \subseteq \mathbb{R}$ and $b \in \mathbb{R}$,

$$\sup A \leq b \qquad \text{iff} \qquad \forall a \in A\colon a \leq b\,.$$

In practice, we can thus drop the sup's in upper Riemann sums (and, dually, the inf's in lower sums) by introducing $\forall$-quantifiers, which is highly beneficial in the context of SMT-based automation.

We stress that even though we discretize the integrals' domains uniformly, our approach is *not* the same as replacing the continuous $\mathtt{unif}_{[0,1]}$ distributions by discrete uniform distributions of N point-masses. Indeed, the latter would neither yield under- nor over-approximations in general, which is, however, essential for invariant verification.

```
i := 1 ; count := 0 ;
while (i ≤ M) {
    x :≈ unif[0,1] ; y :≈ unif[0,1] ;
    if(x² + y² ≤ 1) {count := count + 1} else {skip} ;
    i := i + 1   }
```

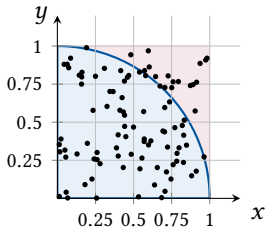


Fig. 1. Left: Monte-Carlo approximator C for π using ability (2). Right: 100 random samples.

Loops and Invariants. Now, consider the entire program C in Figure 1. We have $\mathsf{wp}[\![C]\!]\,(\mathtt{count}) = \frac{\pi}{4} \cdot \mathtt{M}$ because, intuitively, the percentage of samples landing in the quarter circle equals its area. Our techniques enable to soundly bound $\mathsf{wp}[\![C]\!]\,(\mathtt{count})$ in a semi-automated manner *for arbitrary initial values of* $\mathtt{M}$. This is done, as is standard in deductive verification, using quantitative loop invariants, which we detail in Section 6. Suffices to say here that, given a suitable invariant I, our techniques enable to verify its validity automatically. This in turn allows us to conclude that:

$$\forall \text{ initial states } \sigma\colon \quad \mathsf{wp}[\![C]\!]\,(\mathtt{count})\,(\sigma) \;\leq\; 0.85 \cdot \sigma(\mathtt{M})\ .$$

See Example 6.3 (page 15) and Example 9.1 (page 22) for details.

Contributions. In summary, this paper introduces the novel *lower* and *upper Riemann weakest pre-expectation transformers* and demonstrates their applicability to the semi-automated verification of PP with *continuous uniform sampling* and general `while`-*loops*. Put more concretely:

Verification of Invariants We show that externally provided loop invariant candidates can be verified using the Riemann wp transformers in a way that is suitable for SMT-based automation. Such invariants, once verified, imply (one-sided) bounds on wp, wlp, and cwp.

Convergence and Complexity On the more theoretical side, our approximate Riemann wp's are shown to *converge* to the exact wp's under mild assumptions as the discretization becomes finer. As a consequence, we obtain coRE-completeness[3] results for the problems of verifying upper bounds on wp and lower bounds on wlp for fairly general loopy programs.

Implementation and Case Studies We incorporate our method in the PP verification infrastructure Caesar [52] by encoding Riemann sums in its *intermediate verification language*. This enables transferring principles from classical SMT-based program verification [44], such as custom first-order theories for reasoning about exponentials, to the verification of PPs involving continuous sampling. We provide various case studies and an empirical evaluation.

Paper Structure. Section 2 provides a bird's eye view on the distinctive features and technicalities of our method. Section 3 introduces basic fixed point theory and Section 4 summarizes the existing definitions of wp, wlp, and cwp. In Section 5, we introduce our Riemann wp transformers and prove them sound. Section 6 is devoted to loop invariants. The convergence results are presented in Section 7. Towards automation, Section 8 introduces a concrete, effective syntax for expectations. Section 9 details the implementation in Caesar and presents case studies. Additional related work is surveyed in Section 10. We conclude in Section 11. An extended version of this article containing the ommitted proofs and additional material is available [10].

[3]This means that the refutation of such bounds is semi-decidable.

2 A Bird's Eye View

In this section, we explain what makes our method unique by providing a compressed overview of its characteristic features. We also discuss its limitations and point out a subtle technical challenge.

2.1 Highlights and Comparison to Other Approaches

Since we are not the first to address the problem of proving bounds on quantities expressible as w(l)p's, we now highlight five distinctive features of our approach — their combination is unique in the literature. A more in-depth review of related work is deferred to Section 10.

Of the following five items, the first two are specific to the way we handle integrals via Riemann sums, whereas the latter three tend to apply to wp-based approaches in general.

An Alternative to Moment-Based Analyses. A major thread of existing work, e.g., [2, 16, 43], circumvents explicit integration altogether. They achieve this by focusing on classes of programs and properties where, simply speaking, the analysis can soundly replace the distributions in the program by their *mean* [1, Appendix A, p. 31] or some higher moment [43]. For example, $\mathsf{wp}[\![\mathtt{x} :\approx \mathtt{unif}_{[0,1]}]\!]\,(\mathtt{xy})$ is equal to[4] $\mathsf{wp}[\![\mathtt{x} \coloneqq \frac{1}{2}]\!]\,(\mathtt{xy}) = \frac{1}{2}y$, where $\frac{1}{2}$ is the mean of $\mathtt{unif}[0,1]$. However, such mean-based analyses do not always suffice. Indeed, reconsidering the example from Section 1, the inequality $\mathsf{wp}[\![C_{inner}]\!]\,(\mathtt{count}) \sqsubseteq \mathtt{count} + 0.85$, proved correct with our approach, becomes false if we substitute $\mathtt{x} :\approx \mathtt{unif}_{[0,1]}$ and $\mathtt{y} :\approx \mathtt{unif}_{[0,1]}$ in C_{inner} by $\mathtt{x} \coloneqq \frac{1}{2}$ and $\mathtt{y} \coloneqq \frac{1}{2}$, respectively. In fact, the so-obtained program $C_{inner}{}'$ satisfies $\mathsf{wp}[\![C_{inner}{}']\!]\,(\mathtt{count}) = \mathtt{count} + 1$ as the point $(\frac{1}{2}, \frac{1}{2})$ *is certainly inside* the quarter unit circle.

General Conditional Branching. We make relatively mild assumptions regarding the shape of `if`- and `while`-guards in the programs — in practice, general Boolean combinations of the polynomial (in)equalities over all program variables are supported, as in Figure 1. This is different from, e.g., [43] which restricts to finitely-valued variables in guards or [16, 18] which restrict to linear guards. We achieve this high level of generality thanks to the simplicity of the Riemann approximation which treats integration over the resulting indicator functions in a uniform manner.

Moving Backwards: Verification and Parametric Models. We conduct a *backward analysis*: Given a random variable f over final program states (called *post-expectation* in this paper), we approximate the (conditional) weakest pre-expectation $\mathsf{wp}[\![C]\!]\,(f)$, which is *a function of the initial state*. This is dual to works like [11, 26, 58] that conduct a *forward analysis* to solve a classic Bayesian inference problem, namely to compute a program's posterior density — *a function of the final state* — for a given prior distribution. Let us contrast these two paradigms in greater detail.

Backward analysis, as done in this paper, allows verifying program properties that hold for *all initial states*. For instance, our framework supports questions such as, "*Is the posterior mean of x at most twice the initial value of y?*", expressed as $\mathsf{cwp}[C](x) \leq 2y$. The backward approach is thus well-suited for tasks such as *verification* — ensuring a program behaves as intended *for all inputs* — and *parametric analysis* — examining how expected behavior changes when program parameters are altered (parameters can be modeled as uninitialized program variables).

Forward analyses, on the other hand, are typically motivated by probabilistic programming languages (PPLs) explicitly designed for automating Bayesian inference, see e.g., [12, 14, 28, 54]. We remark that there are some fundamental discrepancies between these PPLs and the one studied in this paper. For example, the former usually offer extensive support for general continuous distributions, but do not always support general loops. Moreover, Bayesian inference problems

[4] In general, replacing $\mathtt{x} :\approx \mathtt{unif}_{[0,1]}$ by $\mathtt{x} \coloneqq \frac{1}{2}$ is sound whenever the post-expectation is linear in x. While our approach is compatible with such optimizations, we shall not discuss them any further.

often require *soft conditioning*, which our approach only encodes as syntactic sugar (see Remark 2). Finally, some applications of Bayesian inference involve loading and processing datasets with thousands of observations. In principle, such data could be hardcoded into our programs, but the result would likely be too large for current SMT-based analysis techniques (as indicated by our experiments in Section 9.3).

In summary, our backward approach is particularly suited for proving properties of complex stochastic processes described as probabilistic programs, especially those involving unbounded loops However, it is less effective for data-intensive statistical analysis.

A Zoo of Quantities. The majority of existing methods for (semi-)automated exact analysis of PP with loops and continuous distributions focus on one of these tasks: (i) bound the posterior distribution [11, 26, 58], (ii) prove various flavors of (non-)termination, e.g., [2, 15, 17], and (iii) bound assertion violation probabilities [15, 56]. An exception is [57] which tackles *cost analysis*. In contrast, in the wp-based framework, we can express and reason about a remarkably large variety of quantities, including the following (assume that C does not contain conditioning for simplicity):

- Termination probabilities: $\mathsf{wp}[\![C]\!]\,(1)$
- The probability of terminating in a predicate[5] φ: $\mathsf{wp}[\![C]\!]\,([\varphi])$
- The expected value of variable $\mathtt{x}$ on termination: $\mathsf{wp}[\![C]\!]\,(\mathtt{x})$
- Higher moments of $\mathtt{x}$ after termination: $\mathsf{wp}[\![C]\!]\,(\mathtt{x}^k)$ for $k \geq 1$
- Expected distance between variables $\mathtt{x}$ and $\mathtt{y}$ after termination: $\mathsf{wp}[\![C]\!]\,(|\mathtt{x}-\mathtt{y}|)$

Example 2.1. Let us illustrate the above quantities. Consider the following program C:

$$\texttt{if}(\mathtt{x}=1)\ \{\{\mathtt{y}:=0\}\ [1/2]\ \{\mathtt{y}:=2\}\}\ \texttt{else}\ \{\{\mathtt{y}:=0\}\ [4/5]\ \{\mathtt{y}:=3\}\}$$

- $\mathsf{wp}[\![C]\!]\,(1) = 1$, i.e., C terminates with probability 1 for each initial state.
- $\mathsf{wp}[\![C]\!]\,([\mathtt{y}=0]) = 1/2 \cdot [\mathtt{x}=1] + 4/5 \cdot [\mathtt{x}\neq 1]$, i.e., if initially $\mathtt{x}=1$, then the probability to terminate in $\mathtt{y}=0$ is $1/2$. For all other initial states, this probability is $4/5$.
- $\mathsf{wp}[\![C]\!]\,(\mathtt{y}) = 1 \cdot [\mathtt{x}=1] + 3/5 \cdot [\mathtt{x}\neq 1]$, i.e., the expected final value of $\mathtt{y}$ is either 1 or $3/5$, depending on whether $\mathtt{x}=1$ holds initially or not.
- $\mathsf{wp}[\![C]\!]\,(\mathtt{y}^2) = 2 \cdot [\mathtt{x}=1] + 9/5 \cdot [\mathtt{x}\neq 1]$, i.e., the second moment of $\mathtt{y}$ on termination is either 2 or $9/5$, depending on whether $\mathtt{x}=1$ holds initially or not.
- $\mathsf{wp}[\![C]\!]\,(|\mathtt{x}-\mathtt{y}|) = [\mathtt{x}=1] \cdot (1/2 \cdot |\mathtt{x}| + 1/2 \cdot |\mathtt{x}-2|) + [\mathtt{x}\neq 1] \cdot (4/5 \cdot |\mathtt{x}| + 1/5 \cdot |\mathtt{x}-3|)$, which is the expected difference between the final values of $\mathtt{x}$ and $\mathtt{y}$ in terms of their initial values.

The above weakest pre-expectations can be determined by applying the rules in Table 1 (page 10).

Conditioning, Everywhere. Following [47], we allow conditioning in the form of (hard) `observe`-statements at arbitrary places in the program — even inside loops. All of the above quantities can be generalized sensibly to the conditional setting. We can go further: For instance, the probability to never violate an `observe`, even if the program does not terminate, is given by $\mathsf{wlp}[\![C]\!]\,(1)$.

2.2 Limitations and Assumptions

As mentioned, to analyze loops, we assume *user-provided invariants*. Moreover, we restrict to native support for continuous *uniform* distributions and discrete coin flips. Further, while we support two-sided bounds for loop-free programs, the kind of invariants we study in this paper can only prove *upper* bounds on wp and cwp, and *lower* bounds on wlp. We conjecture that our approach extends to invariant-based methods for the other directions [31] as well, but leave the details for future work. To keep the presentation simple, we only discuss invariant verification for *non-nested loops*. The case of general loop structures, including sequential and/or nested, can be dealt with as

[5] Recall that $[\varphi]$ is the indicator function of the predicate φ.

in [6]. We assume non-negative real-valued program variables and do not support data structures. Soft-conditioning is only supported indirectly. See Remarks 1 to 3 for more details.

2.3 From Lebesgue Integrals to Riemann Sums and Back: A Technical Challenge

The Lebesgue integral is *the* standard integral in probability theory. There is good reason for this: Lebesgue integrals can be defined for a broader class of functions than, say, Riemann integrals, and have favorable mathematical properties like Monotone Convergence Theorems.

Therefore, to define the semantics of PPs, most works (e.g., [23, 53]) have resorted to Lebesgue integrals, and we do not question that this is the way to go. To illustrate the usefulness of the Lebesgue integral as opposed to the Riemann integral[6], consider the following program D:

$$\mathtt{x} \coloneqq 0\,;\ \mathtt{y} \coloneqq 1\,;$$
$$\mathtt{while}\ (\mathtt{y}\cdot\mathtt{z} \neq \mathtt{x})\ \{\mathtt{y} \coloneqq \mathtt{y}+1\,;\ \mathtt{x} \coloneqq 0\,;\ \mathtt{while}\ (\mathtt{y}\cdot\mathtt{z} \neq \mathtt{x} \wedge \mathtt{x} < \mathtt{y})\ \{\mathtt{x} \coloneqq \mathtt{x}+1\}\ \}$$

This program "searches" for non-negative integers x and y such that $\mathtt{z} = \frac{\mathtt{x}}{\mathtt{y}}$ and stops once it finds such x and y. It thus terminates if and only if z is a *rational number* in $[0,1]$ initially (recall that we allow actual *real*-valued variables). In symbols, we have $\mathsf{wp}[\![D]\!]\,(1) = [\mathtt{z} \in \mathbb{Q} \wedge 0 \leq \mathtt{z} \leq 1]$ — this is the well-known *Dirichlet function* restricted to the unit interval. Now, consider the program $\mathtt{z} :\approx \mathtt{unif}_{[0,1]}\,;\ D$. Its termination probability is zero, the Lebesgue integral of the Dirichlet function over the interval $[0,1]$. Intuitively, this is because almost all real numbers are irrational, i.e., the rational numbers have Lebesgue measure zero. However, the Dirichlet function is *not Riemann-integrable* — it is "too discontinuous". As a consequence, one cannot easily define wp — neither in general nor in this specific example — relying on Riemann integrals.

In light of the above example, the following question arises naturally:

> *How sensible is an analysis of loops based on Riemann sums given the fact that* wp*'s of loops are not Riemann-integrable in general, not even for programs using only polynomial arithmetic and constant post-expectations?*

This question has at least two dimensions.

(1) Regarding *soundness*, we prove that our Riemann wp's are *always* sound under- or over-approximations, i.e., the following inequalities hold for all $N \geq 1$ in a very general setting:

$$\forall \text{ initial states } \sigma\colon \quad \underline{\mathsf{wp}}^N[\![C]\!]\,(f)\,(\sigma) \;\leq\; \mathsf{wp}[\![C]\!]\,(f)\,(\sigma) \;\leq\; \overline{\mathsf{wp}}^N[\![C]\!]\,(f)\,(\sigma)$$

and similarly for wlp. This works because the *lower Riemann integral* — the limit of the lower Riemann sums as the discretization becomes fines — is a lower bound on the Lebesgue integral, and similarly for the upper Riemann integral. This is always true, even if the lower and upper integral are different, i.e., if the function at hand is not Riemann-integrable.

(2) Will the Riemann approximation always converge (in some sense) to the exact wp as the discretization becomes finer? We prove that, under mild assumptions such as polynomial arithmetic in the program, the answer is *yes* for loop-free programs. As one of our theoretical main results for loopy programs, we show that (Theorem 7.6)

$$\sup_{n\geq 1} \underline{\mathsf{wp}}^n[\![C^n]\!]\,(f) \;=\; \mathsf{wp}[\![C]\!]\,(f) \qquad \text{but} \qquad \inf_{n\geq 1} \overline{\mathsf{wp}}^n[\![C^n]\!]\,(f) \overset{\text{in general}}{\neq} \mathsf{wp}[\![C]\!]\,(f)\ , \qquad (\ddagger)$$

where C^n arises from C by unfolding all loops up to depth n by taking a simultaneous limit of the unfolding depth and the fineness of the lower Riemann sum approximation (the n in $\underline{\mathsf{wp}}^n$). Via equation ($\ddagger$) we *recover* the exact wp — defined in terms of *Lebesgue* integrals — as a limit of our approximate wp's based on lower Riemann sums for a general class of probabilistic loops.

[6]The Riemann integral is the limit of both the lower and upper Riemann sums as $N \to \infty$, provided the two limits coincide.

3 Preliminaries

In this section, we set up our notations and treat the fixed point-theoretic foundations we rely on.

3.1 General Notation

$\mathbb{N}$ is the set of non-negative integers. The set of non-negative *extended reals* is $\overline{\mathbb{R}}_{\geq 0} = \mathbb{R}_{\geq 0} \cup \{\infty\}$. We adopt the following standard conventions: For all $x \in \overline{\mathbb{R}}_{\geq 0}$, we let $x \leq \infty$ and $x + \infty = \infty + x = \infty$. Moreover, we define $\infty \cdot 0 = 0 \cdot \infty = 0$ and $x \cdot \infty = \infty \cdot x = \infty$ for all $x > 0$. Given $a, b \in \mathbb{R}$, we denote by $[a, b]$ the real closed interval with endpoints a and b. The set of truth values is $\mathbb{B} = \{\mathsf{false}, \mathsf{true}\}$. Given a predicate $\varphi \colon A \to \mathbb{B}$ over a set A, we define the *Iverson bracket*

$$[\varphi] \colon A \to \{0,1\},\ a \mapsto \begin{cases} 0 & \text{if } \varphi(a) = \mathsf{false}\,, \\ 1 & \text{if } \varphi(a) = \mathsf{true}\,. \end{cases}$$

For example, assuming that it is understood from context that x is a real number, then $[x \in \mathbb{Q}]$ denotes the *Dirichlet function* that sends every $x \in \mathbb{R}$ to 1 if x is rational, and to 0 otherwise.

Lambda notation $\lambda x.E$, where E is some mathematical expression with free variable x, is used to introduce unnamed functions whose domain and codomain will be clear from the context.

3.2 Fixed Point Theory

We introduce the foundations from fixed point theory [59, Section 5.4] required to sensibly reason about the semantics of loops and to obtain suitable notions of quantitative loop invariants.

Let $(L, \preceq)$ and $(M, \sqsubseteq)$ be partial orders. A function $f \colon L \to M$ is called *monotonic*, if for all $a \preceq b$, we have $f(a) \sqsubseteq f(b)$. An *ω-chain* in $(L, \preceq)$ is a monotonic function $a \colon \mathbb{N} \to L$ (where $\mathbb{N}$ is ordered by the usual $\leq$ relation), i.e., an ω-chain is a non-decreasing sequence of elements $a(0) \preceq a(1) \preceq \ldots$ from L. The partial order $(L, \preceq)$ is an *ω-complete partial order* (ω-cpo, for short), if all ω-chains a in L have a *supremum* (least upper bound) $\sup_{i \in \mathbb{N}} a(i)$ in $(L, \preceq)$. An ω-cpo $(L, \preceq)$ with an element $\bot \in L$ satisfying $\bot \preceq a$ for all a is called *ω-cpo with bottom*. A function $f \colon L \to M$ between ω-cpos $(L, \preceq)$ and $(M, \sqsubseteq)$ is called *ω-continuous*, if for all ω-chains a in L it holds that $\sup_{i \in \mathbb{N}} f(a(i)) = f(\sup_{i \in \mathbb{N}} a(i))$. Let $f \colon L \to L$ be a function where L is an arbitrary set. An element $a \in L$ satisfying $f(a) = a$ is called a *fixed point* of f. The following is often attributed to Kleene:

Theorem 3.1 (Kleene's Fixed Point Theorem [59, Theorem 5.11]). *Let $(L, \preceq)$ be an ω-cpo with bottom and let $f \colon L \to L$ be ω-continuous. Then f has a least fixed point $\mathrm{lfp}\, f \in L$ and $\mathrm{lfp}\, f = \sup_{i \in \mathbb{N}} f^i(\bot)$ where $f^i(\bot)$ denotes the i-fold application of f to $\bot$.*

A partial order $(L, \preceq)$ is said to be *ω-cocomplete* (ω-cocpo) if all *ω-cochains*, i.e., non-*increasing* sequences $a(0) \succeq a(1) \succeq \ldots$ in L have an infimum $\inf_{i \in \mathbb{N}} a(i) \in L$. An ω-cocpo with a greatest element $\top$ is called ω-cocpo *with top*. Note that $(L, \preceq)$ is an ω-cocpo (with top) iff the reversed partial order $(L, \succeq)$ is an ω-cpo (with bottom). Similarly, a function $f \colon L \to M$ between ω-cocpos $(L, \preceq)$ and $(M, \sqsubseteq)$ is called *ω-cocontinuous*, if $\inf_{i \in \mathbb{N}} f(a(i)) = f(\inf_{i \in \mathbb{N}} a(i))$ for all non-increasing a. Note that for an ω-cocpo $(L, \preceq)$ with top, Theorem 3.1 reads as follows: If $f \colon L \to L$ is ω-cocontinuous, then f has a *greatest* fixed point $\mathrm{gfp}\, f = \inf_{i \in \mathbb{N}} f^i(\top)$.

A partial order is called *ω-bicomplete* (ω-bicpo), if it is both an ω-cpo and ω-cocpo. Similarly, a function f between ω-bicpos is called *ω-bicontinuous*, if it is both ω-continuous and ω-cocontinuous.

A partial order $(L, \preceq)$ is a *complete lattice* if for all $A \subseteq L$ there exists $\sup A \in L$ and $\inf A \in L$. Note that every complete lattice is an ω-bicpo with bottom $\bot = \sup \emptyset$ and top $\top = \inf \emptyset$.

Theorem 3.2 (Knaster-Tarski Theorem [59, Theorems 5.15 and 5.16]). *Let $f \colon L \to L$ be a monotonic function on the complete lattice $(L, \preceq)$. Then f has a* least *and* greatest fixed point $\mathrm{lfp}\, f \in L$

and $\operatorname{gfp} f \in L$. *Moreover, for all* $a \in L$ *it holds that*

$$f(a) \preceq a \quad \textit{implies} \quad \operatorname{lfp} f \preceq a \qquad \textit{and} \qquad a \preceq f(a) \quad \textit{implies} \quad a \preceq \operatorname{gfp} f\,.$$

The above implications are often referred to as *Park (co)induction* [48].

4 Weakest Pre-Expectations for Probabilistic Programs

In this section we define programming language pWhile and its weakest pre-expectation semantics based on *Lebesgue* integrals as defined in [53].

4.1 Program Syntax

For the rest of the paper we fix a finite[7] set $V = \{\mathtt{x}, \mathtt{y}, \ldots\}$ of program variables. A *(program) state* is a variable valuation $\sigma \in \mathbb{R}^V_{\geq 0}$, where $\mathbb{R}^V_{\geq 0}$ is a shorthand for the set of functions $V \to \mathbb{R}_{\geq 0}$. Notice that our program variables range over the *non-negative* reals. The restriction to non-negative variables is for technical convenience and not essential, see Remark 3.

To obtain a well-defined weakest pre-expectation semantics of programs involving continuous sampling, we need to have some measure-theoretic fundamentals in mind, provided in [10]. Suffice it to say here that we consider the standard Borel σ-algebra and Lebesgue measure $\boldsymbol{\lambda}$ on $\mathbb{R}^V_{\geq 0}$. Lebesgue integrals of a measurable function $f\colon \mathbb{R} \to \overline{\mathbb{R}}_{\geq 0}$ over a measurable set $A \subseteq \mathbb{R}$ are denoted by $\int_A f \, d\boldsymbol{\lambda}$, or $\int_A f(x)\, d\boldsymbol{\lambda}(x)$. We explicitly allow Lebesgue integrals to evaluate to ∞.

Now let $\mathcal{E}$ be a set of measurable functions of type $\mathbb{R}^V_{\geq 0} \to \mathbb{R}_{\geq 0}$ and let $\mathcal{G}$ be a set of measurable functions of type $\mathbb{R}^V_{\geq 0} \to \mathbb{B}$. Elements of $\mathcal{E}$ and $\mathcal{G}$ are called *arithmetic expressions* and *guards*, resp.

Definition 4.1 (Probabilistic Programs). Programs C in the set $\mathsf{pWhile}(\mathcal{E}, \mathcal{G})$ of programs with arithmetic expressions from $\mathcal{E}$ and guards from $\mathcal{G}$ adhere to the following grammar:

$$\begin{array}{rcll}
C & ::= & \mathtt{skip} & \text{(effectless program)} \\
& \mid & \mathtt{diverge} & \text{(nonterminating program)} \\
& \mid & \mathtt{x} \coloneqq E & \text{(assignment; } \mathtt{x} \in V, E \in \mathcal{E}\text{)} \\
& \mid & \mathtt{observe}(\varphi) & \text{(conditioning; } \varphi \in \mathcal{G}\text{)} \\
& \mid & \mathtt{x} :\approx \mathtt{unif}_{[0,1]} & \text{(sample from real interval } [0,1]\text{; } \mathtt{x} \in V\text{)} \\
& \mid & \mathtt{if}(\varphi)\ \{C\}\ \mathtt{else}\ \{C\} & \text{(conditional choice; } \varphi \in \mathcal{G}\text{)} \\
& \mid & \{C\}\ [p]\ \{C\} & \text{(probabilistic choice; } p \in [0,1] \cap \mathbb{Q}\text{)} \\
& \mid & C\,;\,C & \text{(sequential composition)} \\
& \mid & \mathtt{while}\ (\varphi)\ \{C\} & \text{(while loop; } \varphi \in \mathcal{G}\text{)}
\end{array}$$

If $\mathcal{E}$ and $\mathcal{G}$ are the sets of *all* measurable functions of the corresponding type, we write pWhile instead of $\mathsf{pWhile}(\mathcal{E}, \mathcal{G})$. A program not containing while-loops is called *loop-free*. △

Let us briefly go over each construct, all of which are standard. `skip` does nothing. `diverge` is a non-terminating program, i.e., behaves like `while` (`true`) {`skip`}. $\mathtt{x} \coloneqq E$ assigns the value of the arithmetic expression E evaluated in the current state to variable `x`. $\mathtt{x} :\approx \mathtt{unif}_{[0,1]}$ assigns to `x` a value drawn from the continuous uniform $[0,1]$-distribution. $\mathtt{observe}(\varphi)$ *conditions* the program execution on the guard φ being true. $\{C_1\}\ [p]\ \{C_2\}$ executes C_1 with probability p, otherwise C_2. The assumption $p \in [0,1] \cap \mathbb{Q}$ is to avoid defining a dedicated syntax for probabilities later on in Section 8. $\mathtt{if}(\varphi)\ \{C_1\}\ \mathtt{else}\ \{C_2\}$, $C_1\,;\,C_2$, and $\mathtt{while}\ (\varphi)\ \{C\}$ are standard conditional choices, sequential compositions, and while-loops, respectively. See [10] for additional remarks.

[7] Once V is fixed we can only write programs with at most $|V|$ distinct variables. However, as we never make any assumptions about the size of V, our theory applies to programs with arbitrarily many variables. Previous work [53] has considered an infinite V, but this requires defining a measure space on the infinite-dimensional $\mathbb{R}^V_{\geq 0}$, which is somewhat more involved.

Table 1. Inductive definition of weakest (liberal) pre-expectations for post-expectation f [53].

C	$\mathsf{wp}[\![C]\!]\,(f)$ where $f \in \mathbb{E}_{meas}$	$\mathsf{wlp}[\![C]\!]\,(f)$ where $f \in \mathbb{E}^{\leq 1}_{meas}$
`skip`	f	f
`diverge`	0	1
$\mathtt{x} \coloneqq E$	$f[\mathtt{x}/E]$	$f[\mathtt{x}/E]$
$\mathtt{x} :\approx \mathtt{unif}_{[0,1]}$	$\lambda\sigma.\, \int_{[0,1]} f(\sigma[\mathtt{x} \mapsto \xi])\, d\boldsymbol{\lambda}(\xi)$	$\lambda\sigma.\, \int_{[0,1]} f(\sigma[\mathtt{x} \mapsto \xi])\, d\boldsymbol{\lambda}(\xi)$
$\mathtt{observe}(\varphi)$	$[\varphi] \cdot f$	$[\varphi] \cdot f$
$\mathtt{if}(\varphi)\ \{C_1\}\ \mathtt{else}\ \{C_2\}$	$[\varphi] \cdot \mathsf{wp}[\![C_1]\!]\,(f) + [\neg\varphi] \cdot \mathsf{wp}[\![C_2]\!]\,(f)$	$[\varphi] \cdot \mathsf{wlp}[\![C_1]\!]\,(f) + [\neg\varphi] \cdot \mathsf{wlp}[\![C_2]\!]\,(f)$
$\{C_1\}\ [p]\ \{C_2\}$	$p \cdot \mathsf{wp}[\![C_1]\!]\,(f) + (1-p) \cdot \mathsf{wp}[\![C_2]\!]\,(f)$	$p \cdot \mathsf{wlp}[\![C_1]\!]\,(f) + (1-p) \cdot \mathsf{wlp}[\![C_2]\!]\,(f)$
$C_1\,;\, C_2$	$\mathsf{wp}[\![C_1]\!]\,(\mathsf{wp}[\![C_2]\!]\,(f))$	$\mathsf{wlp}[\![C_1]\!]\,(\mathsf{wlp}[\![C_2]\!]\,(f))$
$\mathtt{while}\ (\varphi)\ \{B\}$	$\mathrm{lfp}\,\lambda Y.\, [\neg\varphi] \cdot f + [\varphi] \cdot \mathsf{wp}[\![B]\!]\,(Y)$	$\mathrm{gfp}\,\lambda Y.\, [\neg\varphi] \cdot f + [\varphi] \cdot \mathsf{wlp}[\![B]\!]\,(Y)$

Remark 1 (More General Distributions). In theory, the restriction to uniform $[0,1]$-distributions does not limit expressiveness: Arbitrary distributions can be simulated by sampling from $[0,1]$ and applying the inverse *cumulative distribution function* (CDF) of the target distribution [53]. However, to obtain decidability results, we further restrict the syntax of arithmetic expressions and guards to a class of functions expressible in first-order (FO) real arithmetic, see Section 8. Distributions whose inverse CDF belongs to this class include triangular, trapezoidal, U-quadratic, and Kumaraswamy distributions. On the other hand, distributions with transcendental inverse CDF (Gaussian, Laplace, etc.) do not reside in this class. We mention two future directions to address this issue: (i) leverage heuristics implemented in modern SMT solvers to discharge the generated verification conditions, even if they do not belong to a decidable theory, (ii) soundly over/under-approximate the transcendental inverse CDF by FO-expressible algebraic functions.

Remark 2 (Soft Conditioning). Our syntax has native support for *hard conditioning* (`observe`). Bounded *soft conditioning* (scoring) — multiplying the current execution with a weight in $[0,1]$ — can be simulated using $\mathtt{x} :\approx \mathtt{unif}_{[0,1]}$ and `observe`, see [53, Lemma 5] for details.

4.2 Weakest Pre-Expectation Semantics

We now unify the weakest pre-expectation calculi for continuous probabilistic programs from [53] with the calculi proposed in [47]. The latter calculi take *renormalization* — for conditional expected outcomes — into account. The central objects these calculi operate on are *expectations*:

Definition 4.2 (Expectations). We distinguish between the following sets of functions:

(1) The set of *expectations* is $\mathbb{E} = \{f \mid f \colon \mathbb{R}^V_{\geq 0} \to \overline{\mathbb{R}}_{\geq 0}\}$.
(2) The set of *1-bounded expectations* is $\mathbb{E}^{\leq 1} = \{f \mid f \colon \mathbb{R}^V_{\geq 0} \to [0,1]\}$.
(3) The set of *measurable (1-bounded) expectations* $\mathbb{E}_{meas}$ ($\mathbb{E}^{\leq 1}_{meas}$) is the subset of $\mathbb{E}$ ($\mathbb{E}^{\leq 1}$) containing exactly the Borel-measurable functions.

We equip all of these sets with the partial order $\sqsubseteq$ defined as $f \sqsubseteq g$ iff $\forall \sigma \in \mathbb{R}^V_{\geq 0} \colon f(\sigma) \leq g(\sigma)$. △

Crucially, we have (see [53, Lemma 2] and [10]):

Lemma 4.3. *$(\mathbb{E}, \sqsubseteq)$ and $(\mathbb{E}^{\leq 1}, \sqsubseteq)$ are complete lattices. $(\mathbb{E}_{meas}, \sqsubseteq)$ and $(\mathbb{E}^{\leq 1}_{meas}, \sqsubseteq)$ are ω-bicpos with bottom ($0 = \lambda\sigma.0$) and top ($\infty = \lambda\sigma.\infty$ and $1 = \lambda\sigma.1$, respectively).*

The arithmetic operations + (addition) and · (multiplication) on $\mathbb{E}$ are defined pointwise, i.e., $\forall \sigma \in \mathbb{R}^V_{\geq 0} \colon (f+g)(\sigma) = f(\sigma) + g(\sigma)$, and analogously for multiplication. These operations preserve

measurability [49, Theorem 11.18], i.e., $\mathbb{E}_{meas}$ is closed under + and ·. Moreover, addition (for both arguments) and multiplication by constants[8] are ω-bicontinuous functions.

For state $\sigma \in \mathbb{R}^V_{\geq 0}$, program variable $\mathsf{x} \in V$, $\xi \in \mathbb{R}_{\geq 0}$, we define the *updated state as*

$$\sigma[\mathsf{x} \mapsto \xi] \;=\; \lambda \mathsf{y}. \begin{cases} \xi & \text{if } \mathsf{y} = \mathsf{x} \\ \sigma(\mathsf{y}) & \text{otherwise} \end{cases} .$$

Further, given an expectation $f \in \mathbb{E}$, a variable $\mathsf{x} \in V$, and an arithmetic expression $E \colon \mathbb{R}^V_{\geq 0} \to \mathbb{R}_{\geq 0}$ we define the *substitution of* x *by* E *in* f as the expectation $f[\mathsf{x}/E] = \lambda\sigma. f(\sigma[\mathsf{x} \mapsto E(\sigma)])$. If we have syntactic expressions for f and E, then we can obtain a syntactic representation of $f[\mathsf{x}/E]$ by substituting all free occurrences of x in f by E in a capture-avoiding manner, see Section 8.

Definition 4.4 (Weakest (Liberal) Pre-expectation Transformers [53]*).* For all $C \in$ pWhile, the *weakest (liberal) pre-expectation transformers* $\mathsf{wp}[\![C]\!] \colon \mathbb{E}_{meas} \to \mathbb{E}_{meas}$ and $\mathsf{wlp}[\![C]\!] \colon \mathbb{E}^{\leq 1}_{meas} \to \mathbb{E}^{\leq 1}_{meas}$ are defined inductively on the structure of C according to the rules in Table 1. △

Let us briefly explain the inductive definition in Table 1. The effectless program `skip` leaves the post-expectation unchanged. An assignment $\mathsf{x} := E$ substitutes the expression E for the variable x in the post-expectation f, while uniform assignment $\mathsf{x} :\approx \mathtt{unif}_{[0,1]}$ integrates the expectation over all possible values of x in the interval $[0, 1]$, capturing the averaging effect of drawing x uniformly. $\mathtt{observe}(\varphi)$ scales the expectation by the Iverson bracket of the guard φ. Proper renormalization is performed in a second step, see Section 4.4. For $\mathtt{if}(\varphi)\ \{C_1\}\ \mathtt{else}\ \{C_2\}$ and the probabilistic choice $\{C_1\}\ [p]\ \{C_2\}$, the weakest (liberal) pre-expectation is a weighted sum of the wp's from both branches, either weighted by $[\varphi]$ and $[\neg\varphi]$ in the former case, or by the probabilities p and $1-p$ in the latter. The w(l)p of a sequential composition $C_1\,;\, C_2$ is function composition $\mathsf{w(l)p}[\![C_1]\!] \circ \mathsf{w(l)p}[\![C_2]\!]$. Notably, C_2 is evaluated *before* C_1 — w(l)p's are thus computed in a backward manner. Note that wp and wlp only differ in the handling of divergence and loops. The `diverge` command, representing non-termination, sets the wp to 0 and the wlp to 1. The $\mathtt{while}\ (\varphi)\ \{C\}$ loop involves computing the *least* fixed point (lfp) for wp and the *greatest* fixed point (gfp) for wlp. The difference between wp and wlp is further explained in Section 4.3.

Remark 3 (On Non-Negativity). We follow the classic line of work on weakest pre-expectations [39, 41] and restrict attention to *non-negative expectations* (see [37] for a discussion of the mixed-sign case). This means that we can only reason about expected values of non-negative random variables measured in a program's final state. As a consequence, in order to reason about the expected final value of a program variable x on termination, we have to assume that x is unsigned. We have opted to ensure this by simply requiring *all* variables to be unsigned real numbers.

Given a loop $C = \mathtt{while}\ (\varphi)\ \{B\}$ and $f \in \mathbb{E}_{meas}$, we denote by

$$^{\mathsf{wp}}_{C}\Phi_f \colon \mathbb{E}_{meas} \to \mathbb{E}_{meas}\,, \qquad {}^{\mathsf{wp}}_{C}\Phi_f(g) \;=\; [\varphi] \cdot \mathsf{wp}[\![B]\!]\,(g) + [\neg\varphi] \cdot f$$

the wp-*characteristic function of* C *w.r.t.* f (analogously for $f' \in \mathbb{E}^{\leq 1}_{meas}$ and ${}^{\mathsf{wlp}}_{C}\Phi_{f'}$). Hence, weakest pre-expectations of loops can be denoted more concisely as

$$\mathsf{wp}[\![C]\!]\,(f) \;=\; \mathrm{lfp}\ {}^{\mathsf{wp}}_{C}\Phi_f \qquad \text{and} \qquad \mathsf{wlp}[\![C]\!]\,(f') \;=\; \mathrm{gfp}\ {}^{\mathsf{wlp}}_{C}\Phi_{f'}\,.$$

Due to the fixed points in Table 1 it is not immediately obvious that $\mathsf{wp}[\![C]\!]$ and $\mathsf{wlp}[\![C]\!]$ are well-defined. This will we be ensured by Kleene's Theorem 3.1 as shown in the next lemma (parts of which have been proved in [53]).

[8] Formally, for every $g \in \mathbb{E}$ these are the functions $\lambda f. f + g$ and $\lambda f. f \cdot g$.

Theorem 4.5 (Well-definedness of wp and wlp). *For all programs $C \in \mathsf{pWhile}$, $\mathsf{wp}[\![C]\!]$ and $\mathsf{wlp}[\![C]\!]$ are well-defined. In particular, $\mathsf{wp}[\![C]\!]$ is ω-continuous and $\mathsf{wlp}[\![C]\!]$ is ω-cocontinuous.*

4.3 Probabilistic Termination: wp vs. wlp

In general, for all $C \in \mathsf{pWhile}$ and $f \in \mathbb{E}^{\leq 1}_{meas}$ we have $\mathsf{wp}[\![C]\!]\,(f) \sqsubseteq \mathsf{wlp}[\![C]\!]\,(f)$ since the former relies on a least and the latter on a greatest fixed point. More specifically, we have [53, Section 5]

$$\mathsf{wp}[\![C]\!]\,(f) + \mathsf{wlp}[\![C]\!]\,(0) \;=\; \mathsf{wlp}[\![C]\!]\,(f)\;. \tag{1}$$

We call C *almost-surely terminating* (AST) if $\mathsf{wlp}[\![C]\!]\,(0) = 0$, i.e., if the program does not admit any initial state for which the program's infinite runs that do not violate any `observe` have positive probability mass. It follows from equation (1) that C is AST iff $\mathsf{wp}[\![C]\!]\,(f) = \mathsf{wlp}[\![C]\!]\,(f)$.

4.4 Conditional Weakest Pre-Expectations

For a program $C \in \mathsf{pWhile}$, an expectation $f \in \mathbb{E}_{meas}$ and a state σ, following [47], we define:

$$\mathsf{cwp}[\![C]\!]\,(f)\,(\sigma) \;=\; \begin{cases} \frac{\mathsf{wp}[\![C]\!](f)(\sigma)}{\mathsf{wlp}[\![C]\!](1)(\sigma)} & \text{if } \mathsf{wlp}[\![C]\!]\,(1)\,(\sigma) \neq 0 \\ \text{undefined} & \text{else.} \end{cases}$$

The above definition factors out the probability mass of runs violating an `observe`, i.e., divides by $\mathsf{wlp}[\![C]\!]\,(1)\,(\sigma)$, see Section 4.3. $\mathsf{cwp}[\![C]\!]\,(f)\,(\sigma)$ is thus the expected value of f after termination of C started with initial state σ, *conditioned* on all `observe` statements in C being successful.

5 Approximate Riemann Weakest Pre-Expectations

We start by recalling lower and upper Riemann sums and integrals.[9] Let $[a,b] \subseteq \mathbb{R}$ and $N \geq 1$. A *partition* of $[a,b]$ is a tuple of at least $N+1$ real numbers $P = (x_0, x_1, \ldots, x_N)$ such that

$$a = x_0 \;<\; x_1 \;<\; \ldots \;<\; x_N = b\;.$$

The set of all partitions of the interval $[a,b]$ is denoted $\mathfrak{P}[a,b]$. For a bounded $f\colon [a,b] \to \mathbb{R}$ and a partition $P = (x_0, \ldots, x_N) \in \mathfrak{P}[a,b]$, we define the *lower* and *upper sums* of f w.r.t. P as

$$L_{f,P} \;=\; \sum_{i=1}^{N} (x_i - x_{i-1}) \inf_{\xi \in [x_{i-1}, x_i]} f(\xi) \qquad \text{and} \qquad U_{f,P} \;=\; \sum_{i=1}^{N} (x_i - x_{i-1}) \sup_{\xi \in [x_{i-1}, x_i]} f(\xi)\;.$$

Note that $U_{f,P}$ and $L_{f,P}$ are well-defined real numbers because f is bounded.

Definition 5.1 (Riemann integral). Let $f\colon [a,b] \to \mathbb{R}$ be bounded. The *lower-* and *upper Riemann integrals* of f are defined as follows:

$$\underline{\int_a^b} f(x)\,dx \;=\; \sup\left\{L_{f,P} \mid P \in \mathfrak{P}[a,b]\right\} \qquad \text{and} \qquad \overline{\int_a^b} f(x)\,dx \;=\; \inf\left\{U_{f,P} \mid P \in \mathfrak{P}[a,b]\right\}\;.$$

If the upper integral equals the lower integral, then the common value is written $\int_a^b f(x)\,dx$ and called *the* Riemann integral of f. In this case, f is called *Riemann-integrable*. △

[9] The definition of an integral in terms of these sums is in fact commonly attributed to Darboux, not to Riemann. However, Darboux's integral is equivalent to Riemann's which, rather than considering lower and upper sums, relies on evaluating f at sample points within the intervals of a partition of the integration domain. We refer to [13, Chapter 3, Theorems 3.3.1 and 3.3.2] for an in-depth comparison. In this paper, we consistently use Darboux's definitions, but refer to them nonetheless as Riemann sums and integrals, since the latter terminology is more widespread.

Finally, we introduce the following terminology: (i) For $g\colon D \to \mathbb{R}$, $D \subseteq \mathbb{R}$, we say that g is Riemann-integrable *on an interval* $[c,d] \subset D$ if the restriction $g\colon [c,d] \to \mathbb{R}$ is Riemann-integrable. (ii) In this paper we often consider functions of the form $g\colon D^V \to \mathbb{R}$ where V is a finite set, e.g. the set of program variables. We then say that g is Riemann-integrable on $[c,d] \subseteq D$ w.r.t. some $\mathtt{x} \in V$ if *for all* $\sigma \in D^V$ the function $\lambda\xi.g(\sigma[\mathtt{x} \mapsto \xi])$ of type $[c,d] \to \mathbb{R}$ is Riemann-integrable.

5.1 $\underline{\mathrm{wp}}^N$ and $\overline{\mathrm{wp}}^N$: Lower and Upper Riemann Weakest Pre-Expectations

We are now ready to define our approximate expectation transformers. They arise from the standard transformers defined in Table 1 by replacing the *Lebesgue* integrals in the $\mathtt{unif}_{[0,1]}$ case by a lower or upper Riemann sum. Formally:

Definition 5.2 (Lower and Upper Riemann w(l)p*-Transformers).* For all $C \in$ pWhile and integers $N \geq 1$, the expectation transformers $\underline{\mathrm{wp}}^N[\![C]\!]\colon \mathbb{E} \to \mathbb{E}$ and $\overline{\mathrm{wp}}^N[\![C]\!]\colon \mathbb{E} \to \mathbb{E}$ are defined by induction over the structure of C as in Table 1, with the only exception that if C is $\mathtt{x} :\approx \mathtt{unif}_{[0,1]}$, then for all $f \in \mathbb{E}$ we set[10]

$$\underline{\mathrm{wp}}^N[\![\mathtt{x} :\approx \mathtt{unif}_{[0,1]}]\!]\,(f) \;=\; \frac{1}{N}\sum_{i=0}^{N-1} \inf_{\xi \in [\frac{i}{N}, \frac{i+1}{N}]} f[\mathtt{x}/\xi] \qquad \text{and, similarly,}$$

$$\overline{\mathrm{wp}}^N[\![\mathtt{x} :\approx \mathtt{unif}_{[0,1]}]\!]\,(f) \;=\; \frac{1}{N}\sum_{i=0}^{N-1} \sup_{\xi \in [\frac{i}{N}, \frac{i+1}{N}]} f[\mathtt{x}/\xi]\,.$$

The lower and upper Riemann weakest *liberal* pre-expectation transformers $\underline{\mathrm{wlp}}^N[\![C]\!]\colon \mathbb{E}^{\leq 1} \to \mathbb{E}^{\leq 1}$ and $\overline{\mathrm{wlp}}^N[\![C]\!]\colon \mathbb{E}^{\leq 1} \to \mathbb{E}^{\leq 1}$ are defined analogously. △

Our Riemann transformers approximate the Lebesgue integral in the definition of wp (and wlp) by a lower or upper Riemann sum. We work with the partitions $0 < \frac{1}{N} < \frac{2}{N} < \ldots < 1$ of the unit interval for the sake of concreteness. Note that these partitions are *not* successive refinements of each other (see [10] for definitions). Thus, increasing N by 1 does *not necessarily* yield "better" approximations, i.e., *we might have* $\underline{\mathrm{wp}}^N[\![C]\!]\,(f) \not\sqsubseteq \underline{\mathrm{wp}}^{N+1}[\![C]\!]\,(f)$.

Given a loop $C = \mathtt{while}\ (\varphi)\ \{B\}$, $N \geq 1$, $f \in \mathbb{E}$, and $\mathcal{T} \in \{\overline{\mathrm{wp}}^N, \underline{\mathrm{wp}}^N\}$, we define the *$\mathcal{T}$-characteristic function of C w.r.t. f* as

$${}^{\mathcal{T}}_{C}\Phi_f\colon \mathbb{E} \to \mathbb{E}, \qquad {}^{\mathcal{T}}_{C}\Phi_f(g) \;=\; [\varphi]\cdot \mathcal{T}[\![B]\!]\,(g) + [\neg\varphi]\cdot f\,.$$

For $g \in \mathbb{E}^{\leq 1}$ and $\mathcal{T} \in \{\overline{\mathrm{wlp}}^N, \underline{\mathrm{wlp}}^N\}$, ${}^{\mathcal{T}}_{C}\Phi_g\colon \mathbb{E}^{\leq 1} \to \mathbb{E}^{\leq 1}$ is defined analogously. Note that unlike wp and wlp, the approximate transformers are defined on the full sets of expectations $\mathbb{E}$ and $\mathbb{E}^{\leq 1}$, respectively, not just on the measurable ones. All transformers from Definition 5.2 are well-defined: this follows from the Knaster-Tarski Theorem 3.2 using that $\mathbb{E}$ and $\mathbb{E}^{\leq 1}$ are complete lattices and monotonicity of the transformers, see Lemma 5.3 below.

5.2 Healthiness Properties of $\underline{\mathrm{wp}}^N$ and $\overline{\mathrm{wp}}^N$

Lemma 5.3 (Monotonicity of the Riemann w(l)p-Transformers). *For all $C \in$ pWhile and integers $N \geq 1$, the functions $\underline{\mathrm{wp}}^N[\![C]\!]$, $\overline{\mathrm{wp}}^N[\![C]\!]$, $\underline{\mathrm{wlp}}^N[\![C]\!]$, and $\overline{\mathrm{wlp}}^N[\![C]\!]$ are monotonic w.r.t. $\sqsubseteq$.*

Notably, the Riemann w(l)p-transformers do not possess the same continuity properties as their Lebesgue counterparts from Section 4.2. For instance, due to the presence of infima in the lower Riemann sum, $\underline{\mathrm{wp}}^N[\![C]\!]$ is *not* ω-continuous in general (see [10] for a counter-example).

[10] Notice that $\inf_{\xi\in[a,b]} f[\mathtt{x}/\xi]$ is the same as $\lambda\sigma.\,\inf_{\xi\in[a,b]} f(\sigma[\mathtt{x}\mapsto\xi])$.

Lemma 5.4 (Soundness of the Riemann w(l)p-Transformers). *For all programs $C \in \mathsf{pWhile}$, post-expectations $f \in \mathbb{E}_{meas}$, $g \in \mathbb{E}^{\leq 1}_{meas}$, and integers $N \geq 1$,*

$$\underline{\mathsf{wp}}^N[\![C]\!]\,(f) \;\sqsubseteq\; \mathsf{wp}[\![C]\!]\,(f) \;\sqsubseteq\; \overline{\mathsf{wp}}^N[\![C]\!]\,(f) \quad \text{and}$$
$$\underline{\mathsf{wlp}}^N[\![C]\!]\,(g) \;\sqsubseteq\; \mathsf{wlp}[\![C]\!]\,(g) \;\sqsubseteq\; \overline{\mathsf{wlp}}^N[\![C]\!]\,(g)\ .$$

For *conditional* weakest pre-expectations (Section 4.4), we immediately get the following:

Corollary 5.5. *For all programs $C \in \mathsf{pWhile}$, post-expectations $f \in \mathbb{E}_{meas}$, and integers $N \geq 1$*

$$\frac{\underline{\mathsf{wp}}^N[\![C]\!]\,(f)\,(\sigma)}{\overline{\mathsf{wlp}}^N[\![C]\!]\,(1)\,(\sigma)} \;\leq\; \mathsf{cwp}[\![C]\!]\,(f)\,(\sigma) \;\leq\; \frac{\overline{\mathsf{wp}}^N[\![C]\!]\,(f)\,(\sigma)}{\underline{\mathsf{wlp}}^N[\![C]\!]\,(1)\,(\sigma)}$$

for all $\sigma \in \mathbb{R}^V_{\geq 0}$ such that none of $\underline{\mathsf{wlp}}^N[\![C]\!]\,(1)\,(\sigma)$, $\mathsf{wlp}[\![C]\!]\,(1)\,(\sigma)$, and $\overline{\mathsf{wlp}}^N[\![C]\!]\,(1)\,(\sigma)$ is zero.

We stress that Lemma 5.4 and Corollary 5.5 hold even for non-Riemann-integrable post-expectations.

6 Invariant-Based Reasoning for Loops

Deductive probabilistic program verification techniques typically bound expected outcomes of loops by means of *quantitative loop invariants*. Intuitively, a quantitative loop invariant I is an expectation whose pre-expectation w.r.t. *one loop iteration* does not increase (or decrease, depending on whether one wants to establish upper or lower bones, see below). While quantitative loop invariant-based reasoning can simplify the verification of loops significantly, the continuous setting poses challenges: Computing the pre-expectation of I w.r.t. a loop's body requires reasoning about possibly complex Lebesgue integrals — involving, e.g., indicator functions of predicates — arising from the continuous sampling instructions. We now develop invariant-based proof rules for our *Riemann pre-expectations* (Definition 5.2), which yield sound bounds on the *Lebesgue pre-expectations* (Definition 4.4). We will see in Sections 8 and 9 that these proof rules give rise to SMT-based techniques for verifying bounds on Lebesgue pre-expectations of loops in a semi-automated fashion.

Our first insight is that — due to $(\mathbb{E}, \sqsubseteq)$ and $(\mathbb{E}^{\leq 1}, \sqsubseteq)$ being complete lattices and the Riemann expectation transformers being monotonic by Lemma 5.3 — bounds on *Riemann* pre-expectations of loops can be established via Park induction (Theorem 3.2):

Lemma 6.1. *Let $C = \texttt{while}\ (\varphi)\ \{B\} \in \mathsf{pWhile}$ and $N \geq 1$. Then:*

(1) For all $f, I \in \mathbb{E}$, we have $\underbrace{\overline{\mathsf{wp}}^N_C\Phi_f(I) \sqsubseteq I}_{I \text{ is } \overline{\mathsf{wp}}^N\text{-superinvariant of } C \text{ w.r.t. } f}$ *implies* $\overline{\mathsf{wp}}^N[\![C]\!]\,(f) \sqsubseteq I$.

(2) For all $f, I \in \mathbb{E}^{\leq 1}$, we have $\underbrace{I \sqsubseteq \underline{\mathsf{wlp}}^N_C\Phi_f(I)}_{I \text{ is } \underline{\mathsf{wlp}}^N\text{-subinvariant of } C \text{ w.r.t. } f}$ *implies* $I \sqsubseteq \underline{\mathsf{wlp}}^N[\![C]\!]\,(f)$.

More colloquially stated, *super*invariants yield *upper* bounds on *upper* Riemann pre-expectations of loops and, dually, *sub*invariants yield *lower* bounds on *lower liberal* Riemann pre-expectations. Notice that establishing the premise of the above proof rules only requires reasoning about the loop's body and avoids explicitly computing Lebesgue integrals.

It thus follows from the soundness of our Riemann expectation transformers (Lemma 5.4) that the above proof rules yield sound bounds on *Lebesgue* pre-expectations.

Theorem 6.2. *Let $C = \texttt{while}\ (\varphi)\ \{B\} \in \mathsf{pWhile}$ and $N, N' \geq 1$. We have:*

(1) If $I \in \mathbb{E}$ is a $\overline{\mathsf{wp}}^N$-superinvariant of C w.r.t. $f \in \mathbb{E}_{meas}$, then $\mathsf{wp}[\![C]\!]\,(f) \sqsubseteq I$.

(2) If $I \in \mathbb{E}^{\leq 1}$ *is a* $\underline{\mathrm{wlp}}^N$*-subinvariant of* C *w.r.t.* $f \in \mathbb{E}^{\leq 1}_{meas}$*, then* $I \sqsubseteq \mathrm{wlp}[\![C]\!]\,(f)$*.*

(3) If $I \in \mathbb{E}$ *is a* $\overline{\mathrm{wp}}^N$*-superinvariant of* C *w.r.t.* $f \in \mathbb{E}_{meas}$ *and* $J \in \mathbb{E}^{\leq 1}$ *is a* $\underline{\mathrm{wlp}}^{N'}$*-subinvariant of* C *w.r.t.* 1*, then, for all* $\sigma \in \mathbb{R}^V_{\geq 0}$*,* $J(\sigma) > 0$ *implies that* $\mathrm{cwp}[\![C]\!]\,(f)\,(\sigma)$ *is defined and*

$$\mathrm{cwp}[\![C]\!]\,(f)\,(\sigma) \;\leq\; \frac{I(\sigma)}{J(\sigma)}\,.$$

It is important to note that in Lemma 6.1 we admit *arbitrary* (1-bounded) post-expectations whereas in Theorem 6.2 we have to restrict to *measurable* (1-bounded) post-expectations in order for the Lebesgue pre-expectations $\mathrm{wp}[\![C]\!]\,(f)$ and $\mathrm{wlp}[\![C]\!]\,(f)$ to be well-defined. The quantitative loop invariant I, on the other hand, must *not* necessarily be measurable since there is no need to plug I into a Lebesgue expectation transformer.

Example 6.3. Consider the Monte Carlo π-approximator C from Figure 1. The expectation

$$I \;=\; \mathtt{count} + [\mathtt{i} \leq \mathtt{M}] \cdot \big(0.85 \cdot ((\mathtt{M} \dot{-} \mathtt{i}) + 1)\big)$$

is a $\overline{\mathrm{wp}}^{16}$-superinvariant of C w.r.t. count (here $\dot{-}$ denote the *monus* operator defined as $\mathrm{t}_1 \dot{-} \mathrm{t}_2 = \max(\mathrm{t}_1 - \mathrm{t}_2, 0)$, detailed in Section 8.1). Hence, we get by Theorem 6.2, $\mathrm{wp}[\![C]\!]\,(\mathtt{count}) \sqsubseteq I$, i.e., if initially $\mathtt{count} = 0$, $\mathtt{i} = 1$, then $0.85 \cdot \mathtt{M}$ upper-bounds the expected final value of count.

7 Convergence of The Riemann Weakest Pre-Expectations

We now address the question if and under which conditions $\underline{\mathrm{wp}}^N[\![C]\!]\,(f)$ and $\overline{\mathrm{wp}}^N[\![C]\!]\,(f)$ converge to $\mathrm{wp}[\![C]\!]\,(f)$ as N goes to ∞. We focus on *pointwise* convergence. It is easy to see that in general, pointwise convergence does not hold: consider for example the simple program $C = \mathtt{x} :\approx \mathtt{unif}_{[0,1]}$ and the expectation $f = [\mathtt{x} \in \mathbb{Q}]$, for which we have $\mathrm{wp}[\![C]\!]\,(f) = 0$. However, for every $N \geq 1$ it holds that $\overline{\mathrm{wp}}^N[\![C]\!]\,(f) = 1$. Our goal is thus to identify a *Riemann-suitable* subset of our framework, by mildly restricting the class of allowed expectations.

We first recall two fundamental properties of the Riemann integral. The first is a well-known characterization of Riemann integrability, the second reveals that Lebesgue integrals conservatively generalize Riemann integrals. In the following we will use $\boldsymbol{\lambda}$ to denote the Lebesgue measure. For the formal definition and properties of $\boldsymbol{\lambda}$ we refer the reader to [10].

Theorem 7.1 (Riemann-Lebesgue Theorem [49, Thm. 11.33]). *Let* $f\colon [a,b] \to \mathbb{R}$ *be bounded. Consider the set* $D = \{x \in [a,b] \mid f \text{ is discontinuous in } x\}$*. Then* f *is Riemann-integrable if and only if* $\boldsymbol{\lambda}(D) = 0$*, i.e.,* f *is* continuous almost everywhere.

Theorem 7.2 ([4, Theorem 1.7.1]). *Let* $f\colon [a,b] \to \mathbb{R}$ *be bounded and Riemann-integrable. Then* $\int_a^b f(x)\,dx = \int_{[a,b]} f(x)\,d\boldsymbol{\lambda}(x)$*, i.e., the Riemann integral coincides with the Lebesgue integral.*

7.1 Convergence of Approximation for Loop-free Programs

By Theorems 7.1 and 7.2, the Riemann integral coincides with the Lebesgue integral for "sufficiently continuous" functions. There is thus hope to establish convergence if we restrict our setting to almost everywhere continuous expectations. However, there is an additional complication: A function needs to be *bounded* on the integration domain in order to be Riemann-integrable – otherwise the upper sum could be "stuck" at ∞. We address this issue using the concept of *local boundedness*: A function $f\colon \mathbb{R}^V_{\geq 0} \to \mathbb{R} \cup \{\pm\infty\}$ is called *locally bounded* if

$$\text{for all compact } K \subseteq \mathbb{R}^V_{\geq 0}\colon \quad \sup_{\sigma \in K} |f(\sigma)| \;<\; \infty\,.$$

We recall that a subset K of $\mathbb{R}^V$ is compact iff it is closed and bounded. Many common functions such as polynomials are locally bounded. An example of a function that is *not* locally bounded is $[x > 0] \cdot \frac{1}{x}$. Based on these considerations, we define the following:

Definition 7.3 (Riemann-suitable). A combination $(\mathcal{E}, \mathcal{G}, \mathcal{F})$ of a class $\mathcal{E}$ of locally bounded arithmetic expressions, a class $\mathcal{G}$ of guards, and a class $\mathcal{F} \subseteq \mathbb{E}$ of locally bounded expectations if for all $f \in \mathcal{F}$ and $\mathsf{x} \in V$, f is Riemann-integrable on $[0,1]$ w.r.t. x. Moreover, $\mathcal{F}$ contains the constant expectations $0, 1$ and satisfies the following closure properties:

(i) $\mathcal{F}$ is closed under taking infima and suprema over closed subintervals of $[0,1]$, i.e., for all $f \in \mathcal{F}$, $\mathsf{x} \in V$, and $[a,b] \subseteq [0,1]$ we have $\inf_{\xi \in [a,b]} f[\mathsf{x}/\xi] \in \mathcal{F}$ and $\sup_{\xi \in [a,b]} f[\mathsf{x}/\xi] \in \mathcal{F}$.
(ii) For all $f \in \mathcal{F}$, $E \in \mathcal{E}$ and $\mathsf{x} \in V$, we have $f[\mathsf{x}/E] \in \mathcal{F}$.
(iii) For all $f, g \in \mathcal{F}$ and $p \in [0,1] \cap \mathbb{Q}$, we have $p \cdot f + (1-p) \cdot g \in \mathcal{F}$.
(iv) For all $f, g \in \mathcal{F}$ and $\varphi \in \mathcal{G}$, we have $[\varphi] \cdot f + [\neg\varphi] \cdot g \in \mathcal{F}$. △

The conditions (i) - (iv) in the above definition ensure that $\mathcal{F}$ is closed under $\underline{\mathrm{wp}}^N[\![C]\!]$ and $\overline{\mathrm{wp}}^N[\![C]\!]$ for *loop-free* C (see (2) in Lemma 7.4 below). We remark that it is not immediately clear if any "interesting" Riemann-suitable instantiations exist. Fortunately, the answer is *yes*, as we show in Section 8. In the Riemann-suitable setting, we can show the following convergence result:

Lemma 7.4 (Convergence – Loop-free Case). *Let $(\mathcal{E}, \mathcal{G}, \mathcal{F})$ be Riemann-suitable. Then for all loop-free $C \in \mathsf{pWhile}(\mathcal{E}, \mathcal{G})$ and post-expectations $f \in \mathcal{F}$ the following holds:*

$$\forall N \geq 1\colon \quad \underline{\mathrm{wp}}^N[\![C]\!](f) \in \mathcal{F} \quad \text{and} \quad \overline{\mathrm{wp}}^N[\![C]\!](f) \in \mathcal{F}\,. \tag{2}$$

$$\text{If } f \in \mathbb{E}_{meas}\colon \quad \mathrm{wp}[\![C]\!](f) \;=\; \sup_{N \geq 1} \underline{\mathrm{wp}}^N[\![C]\!](f) \;=\; \inf_{N \geq 1} \overline{\mathrm{wp}}^N[\![C]\!](f)\,. \tag{3}$$

An analogous wlp-version of Lemma 7.4 is stated in [10].

7.2 Convergence of Approximation for Programs with Loops

Recall the *Dirichlet program* D from Section 2. For all N we have $\overline{\mathrm{wp}}^N[\![D]\!](1) = 1$, showing that Lemma 7.4 does not hold for loopy programs in general. The added difficulty stems from the fact that the semantics of loops is itself a limit. Specifically, we can imagine to "unroll" every loop in a program C up to a certain depth $\ell \in \mathbb{N}$, denoted C^ℓ (see [10] for the formal definition), and view the semantics C as the limit of the semantics of C^ℓ when ℓ tends to infinity. Notably, wp and wlp behave differently with respect to this limit: when increasing ℓ the (non-liberal) pre-expectation of f increases, while the liberal pre-expectation decreases. This is formalized in the next lemma:

Lemma 7.5. *For all $C \in \mathsf{pWhile}$ and post-expectations $f \in \mathbb{E}_{meas}$ and $g \in \mathbb{E}^{\leq 1}_{meas}$, $\mathrm{wp}[\![C^\ell]\!](f)$ and $\mathrm{wlp}[\![C^\ell]\!](g)$ are non-decreasing (non-increasing, respectively) sequences in $\ell \in \mathbb{N}$ and it holds that*

$$\sup_{\ell \in \mathbb{N}} \mathrm{wp}[\![C^\ell]\!](f) \;=\; \mathrm{wp}[\![C]\!](f) \qquad \text{and} \qquad \inf_{\ell \in \mathbb{N}} \mathrm{wlp}[\![C^\ell]\!](f) \;=\; \mathrm{wlp}[\![C]\!](f)\,.$$

It should now be clear that, when considering the convergence of $\overline{\mathrm{wp}}^N[\![C]\!](f)$ for a program C with loops, we are implicitly considering two limits, the one over N and the one over ℓ. Unfortunately, when the two limits have different monotonic behaviors (i.e., for $\overline{\mathrm{wp}}^N[\![C]\!]$, where the limit in ℓ is the supremum but the limit in N is the infimum, and symmetrically for $\underline{\mathrm{wlp}}^N[\![C]\!]$) convergence cannot be guaranteed. Intuitively, this explains why the next result only holds for $\underline{\mathrm{wp}}^N$ and $\overline{\mathrm{wlp}}^N$.

Theorem 7.6 (Convergence – General Case). *Let $(\mathcal{E}, \mathcal{G}, \mathcal{F})$ be Riemann-suitable. Then for all $C \in \mathsf{pWhile}(\mathcal{E}, \mathcal{G})$ and post-expectations $f \in \mathcal{F} \cap \mathbb{E}_{meas}$ and $g \in \mathcal{F} \cap \mathbb{E}^{\leq 1}_{meas}$ it holds that:*

$$\sup_{n \geq 1} \underline{\mathrm{wp}}^n[\![C^n]\!](f) \;=\; \mathrm{wp}[\![C]\!](f) \quad \text{and} \quad \mathrm{wlp}[\![C]\!](g) \;=\; \inf_{n \geq 1} \overline{\mathrm{wlp}}^n[\![C^n]\!](g)\,.$$

Note that Theorem 7.6 ensures that when considering a Riemann suitable class, the L.H.S. of the inequality in Corollary 5.5 converges to the exact conditional weakest pre-expectation cwp.

8 Effective Verification of pWhile-Programs

In the previous sections, we have assumed that guards and arithmetical expressions in programs as well as the resulting (pre/post)-expectations are purely mathematical objects. To enable *effective*, i.e., automated, verification we now define, inspired by [8, 52], concrete syntaxes for these objects.

8.1 A Formal Language of Expressions

Let LV be a countably infinite set of logical variables ranged over by $x, y, \ldots$, etc. The sets Terms and Guards of (syntactic) *terms* and *guards* are defined as follows:

$$\mathsf{t} \;::=\; q \in \mathbb{Q}_{\geq 0} \mid x \in LV \mid \mathsf{t} + \mathsf{t} \mid \mathsf{t} \dot{-} \mathsf{t} \mid \mathsf{t} \cdot \mathsf{t} \qquad\qquad \varphi \;::=\; \mathsf{t} < \mathsf{t} \mid \neg\varphi \mid \varphi \wedge \varphi$$

Note that the other standard comparison relations $\leq, \neq, =, >, \geq$ and Boolean connectives $\vee$, $\rightarrow$, $\leftrightarrow$ can be expressed in terms of $<, \neg, \wedge$. We allow further syntactic sugar such as t^2 for $\mathsf{t}\cdot\mathsf{t}$ and $1 \leq x \leq 2$ for $1 \leq x \wedge x \leq 2$, etc. Additional parentheses are admitted to clarify the order of precedence; to minimize the use of parentheses we assume that $\cdot$ takes precedence over $+$ and $\dot{-}$, and $\neg$ binds stronger than the binary Boolean connectives, as is standard. We let $\mathsf{free}(\mathsf{t})$ and $\mathsf{free}(\varphi)$ be the variables occurring in term t and guard φ.

For every $LV' \supseteq \mathsf{free}(\mathsf{t})$, the semantics $[\![\mathsf{t}]\!] \colon \mathbb{R}_{\geq 0}^{LV'} \to \mathbb{R}_{\geq 0}$ of $\mathsf{t} \in \mathsf{Terms}$ is standard except that $\dot{-}$ is interpreted as "monus", i.e., $[\![\mathsf{t}_1 \dot{-} \mathsf{t}_2]\!] = \max(\mathsf{t}_1 - \mathsf{t}_2, 0)$. The semantics of $\varphi \in \mathsf{Guards}$ can be viewed similarly as a function $[\![\varphi]\!] \colon \mathbb{R}_{\geq 0}^{LV'} \to \mathbb{B}$ for every $LV' \supseteq \mathsf{free}(\varphi)$. For $\sigma \in \mathbb{R}_{\geq 0}^{LV'}$, we write $\sigma \models \varphi$ and $\sigma \not\models \varphi$ to indicate that $[\![\varphi]\!](\sigma) = \mathsf{true}$ and $[\![\varphi]\!](\sigma) = \mathsf{false}$, respectively.

Note that every $\mathsf{t} \in \mathsf{Terms}$ without monus is a non-negative polynomial in the — finitely many — variables $\mathsf{free}(\mathsf{t})$ with rational coefficients, possibly written in (partially) factorized form, whereas terms with monus can be seen as a piecewise defined non-negative polynomial. Similarly, every $\varphi \in \mathsf{Guards}$ is a Boolean combination of (in)equations between such (piecewise) polynomials.

We are now ready to define our expression language similar to [8]. The restriction to infima and suprema over *compact* intervals $[a, b]$ resembles the definition of lower and upper Riemann sums.

Definition 8.1 (The Expression Language Expr*).* The set of Expr of (syntactic) *expressions* is defined according to the following grammar:

$$\mathsf{f} \quad ::= \quad \mathsf{t} \mid [\varphi] \cdot \mathsf{f} \mid q \cdot \mathsf{f} \mid \mathsf{f} + \mathsf{f} \mid \mathfrak{S}_{[a,b]}\, x \colon \mathsf{f} \mid \mathfrak{l}_{[a,b]}\, x \colon \mathsf{f}$$

Here, $\mathsf{t} \in \mathsf{Terms}$, $\varphi \in \mathsf{Guards}$, $q \in \mathbb{Q}_{\geq 0}$, $a, b \in \mathbb{Q}_{\geq 0}$, $a \leq b$, and $x \in LV$. △

We allow $[\varphi]$ as syntactic sugar for $[\varphi] \cdot 1$ and adopt the usual rules regarding parentheses and orders of precedence. For $\mathsf{f} \in \mathsf{Expr}$ we let $\mathsf{free}(\mathsf{f})$ be the variables in f that are not bound by a $\mathfrak{S}$ (supremum) or $\mathfrak{l}$ (infimum) "quantifier". As in standard first-order logic, it is possible for a variable to have both a free and a non-free occurrence in f. We may write $\mathsf{f}(x_1, \ldots, x_n)$ to indicate that f contains at most the pairwise distinct free variables $x_1, \ldots, x_n \in LV$.

Definition 8.2 (Semantics of Expressions). We define the semantics of expressions $[\![\mathsf{f}]\!] \colon \mathbb{R}_{\geq 0}^{LV'} \to \mathbb{R}_{\geq 0}$ for every finite $LV' \subseteq LV$ with $\mathsf{free}(\mathsf{f}) \subseteq LV'$ inductively as follows. For all $\sigma \in \mathbb{R}_{\geq 0}^{LV'}$:

- $[\![\mathsf{t}]\!](\sigma)$ is defined in the standard manner, see above.
- $[\![[\varphi] \cdot \mathsf{f}]\!](\sigma) = [\![\mathsf{f}]\!](\sigma)$ if $\sigma \models \varphi$; $[\![[\varphi] \cdot \mathsf{f}]\!](\sigma) = 0$ if $\sigma \not\models \varphi$.
- $[\![q \cdot \mathsf{f}]\!](\sigma) = q \cdot [\![\mathsf{f}]\!](\sigma)$.
- $[\![\mathsf{f}_1 + \mathsf{f}_2]\!](\sigma) = [\![\mathsf{f}_1]\!](\sigma) + [\![\mathsf{f}_2]\!](\sigma)$
- $[\![\mathfrak{S}_{[a,b]}\, x \colon \mathsf{f}]\!](\sigma) = \sup_{\xi \in [a,b]} [\![\mathsf{f}]\!](\sigma[x \mapsto \xi])$.

- $[\![\mathcal{L}_{[a,b]}\, x\colon\ \mathsf{f}]\!](\sigma) = \inf_{\xi \in [a,b]} [\![\mathsf{f}]\!](\sigma[x \mapsto \xi])$.

In the last two cases, if x is not already in the domain of σ, we tacitly assume that the operation $\sigma[x \mapsto \xi]$ extends the domain of σ by x. △

It is not immediately clear that $[\![\mathsf{f}]\!]$ is well-defined due to the suprema in Definition 8.2 — we have to ensure that those exist in $\mathbb{R}_{\geq 0}$. We show this now.

Lemma 8.3. *For every* $\mathsf{f} \in \mathsf{Expr}$ *and every finite* $LV' \subseteq LV$ *with* $\mathsf{free}(\mathsf{f}) \subseteq LV'$ *the semantics* $[\![\mathsf{f}]\!] \colon \mathbb{R}_{\geq 0}^{LV'} \to \mathbb{R}_{\geq 0}$ *is a well-defined locally bounded function (cf. Section 7.1).*

Example 8.4. $\mathsf{f} = \mathcal{C}_{[0,5]}\, x\colon \left[w = x^2\right] \cdot x$ denotes the function $[\![\mathsf{f}]\!] = \lambda\sigma.[0 \leq \sigma(w) \leq 25] \cdot \sqrt{\sigma(w)}$.

8.2 Properties of Syntactic Expressions

The set FO of first-order formulae[11] is defined according to the following grammar:

$$\psi \quad ::= \quad \varphi \mid \exists x\colon \psi \mid \forall x\colon \psi \mid \neg\psi \mid \psi \wedge \psi \mid \psi \vee \psi \mid \psi \to \psi$$

where φ is a Boolean combination of polynomial (in)equalities (similar to the guards defined in Section 8.1 but with standard minus instead of "monus"), and $x \in LV$. As usual, $\mathsf{free}(\psi)$ is the set of free variables in ψ, and we write $\psi(x_1, \ldots, x_n)$ to indicate that ψ contains at most the (pairwise distinct) free variables $x_1, \ldots, x_n \in LV$. An FO formula ψ is called *quantifier-free* if it does not contain any $\exists$ and $\forall$ quantifiers. Moreover, ψ is said to be in *prenex normal form* (PNF) if $\psi = Q_1 x_1\colon \ldots Q_n x_n\colon \psi'$ where $n \geq 0$, $Q_1, \ldots, Q_n \in \{\exists, \forall\}$, and ψ' is quantifier-free. Similarly, ψ is in *existential (universal) prenex form* if for all $1 \leq i \leq n$ we have $Q_i = \exists$ ($Q_i = \forall$, respectively).

The semantics of a formula $\psi \in \mathsf{FO}$ is standard and can be viewed as a function $[\![\psi]\!] \colon \mathbb{R}^{LV'} \to \mathbb{B}$ for all $LV' \subseteq LV$ with $\mathsf{free}(\psi) \subseteq LV'$. For $\sigma \in \mathbb{R}^{LV'}$ we write $\sigma \models \psi$ to indicate that σ is a model of ψ, i.e., the *sentence* (formula without free variables) obtained by substituting every free occurrence of $x \in \mathsf{free}(\psi)$ in ψ by $\sigma(x)$ evaluates to true in $\mathbb{R}$. A formula ψ is called *satisfiable* if it has a model, *unsatisfiable* if it does not have a model, and *valid* if $\neg\psi$ is unsatisfiable.

The decision problem of checking whether a given *quantifier-free* $\psi \in \mathsf{FO}$ is satisfiable is called QF_NRA[12]. It is known that QF_NRA $\in$ PSPACE. Checking the satisfiability of general FO formulae is decidable as well, but has higher complexity.

Lemma 8.5 (Expr to FO). *For every* $\mathsf{f}(x_1, \ldots, x_n) \in \mathsf{Expr}$ *there exists an FO formula* $\psi_{\mathsf{f}}(x_1, \ldots, x_n, y)$ *encoding* $[\![\mathsf{f}]\!]$ *in the sense that for all* $\sigma \in \mathbb{R}_{\geq 0}^{\mathsf{free}(\mathsf{f})}$ *and* $r \in \mathbb{R}_{\geq 0}$*, we have that* $\sigma, r \models \psi_{\mathsf{f}}$ *iff* $[\![\mathsf{f}]\!](\sigma) = r$*. For quantifier-free* f*, we can construct* ψ_{f} *in existential prenex form in linear time.*

FO-expressible functions in the sense of Lemma 8.5 have been extensively studied [22]. Let $\mathsf{FO}_{\mathbb{R}}$ be defined like FO with the only difference that *arbitrary real numbers* are allowed as coefficients in the polynomials. A set $A \subseteq \mathbb{R}^n$, $n \geq 1$, is called *semi-algebraic* if there exists an $\mathsf{FO}_{\mathbb{R}}$ formula $\psi(x_1, \ldots, x_n)$ such that $A = \{\sigma \in \mathbb{R}^n \mid \sigma \models \psi\}$. A function $f\colon \mathbb{R}^n \to \mathbb{R}$ is called semi-algebraic if its graph $\{(\sigma, f(\sigma)) \mid \sigma \in \mathbb{R}^n\}$ is a semi-algebraic set.

Lemma 8.6 (e.g., [22, Exercise 2.22]). *Every semi-algebraic function* $f\colon \mathbb{R} \to \mathbb{R}$ *has at most finitely many discontinuities. In particular,* f *is almost everywhere continuous and hence Riemann-integrable on every interval* $[a, b] \subset \mathbb{R}$ *where* f *is bounded.*

By combining local boundedness from Lemma 8.3, the translation to FO from Lemma 8.5, and the fact that semi-algebraic functions are Riemann-integrable (Lemma 8.6), we obtain:

[11] In this paper, *first-order formula* always refers to first-order logic over the signature of polynomial arithmetic with comparison; we interpret such formulae over the fixed structure $\mathbb{R}$.

[12] Quantifier-free non-linear real arithmetic, a term coined by the SMT-community [5].

Theorem 8.7. *For all* $\mathsf{f} \in \mathsf{Expr}$, *intervals* $[a,b] \subset \mathbb{R}_{\geq 0}$, *finite* $LV' \subseteq LV$ *with* $\mathsf{free}(\mathsf{f}) \subseteq LV'$, *and variables* $x \in LV'$, *the function* $[\![\mathsf{f}]\!] \colon \mathbb{R}_{\geq 0}^{LV'} \to \mathbb{R}_{\geq 0}$ *is Riemann-integrable on* $[a,b]$ *w.r.t.* x.

In analogy to standard FO, we say that an expression $\mathsf{f} \in \mathsf{Expr}$ is in *prenex normal form* (PNF) if

$$\mathsf{f} \;=\; \mathcal{Q}^1_{[a_1,b_1]}\, x_1 \colon \ldots \mathcal{Q}^n_{[a_n,b_n]}\, x_n \colon \mathsf{f}'$$

where $n \geq 0$, $\mathcal{Q}^1, \ldots, \mathcal{Q}^n \in \{\mathcal{U}, \mathcal{E}\}$, and f' is quantifier-free.

Lemma 8.8 (Prenex Normal Form [8, Lemma 8.1]). *Every* $\mathsf{f} \in \mathsf{Expr}$ *can be transformed into an equivalent* $\mathsf{f}' \in \mathsf{Expr}$ *in PNF with the same free variables. Moreover, if* f *is* $\mathcal{U}$*-free (*$\mathcal{E}$*-free), then* f' *is* $\mathcal{U}$*-free (*$\mathcal{E}$*-free, respectively) as well. This transformation can be done in linear time.*

Next, we formalize one of our key insights: In order to check a "*quantitative entailment*" $[\![\mathsf{f}]\!] \sqsubseteq [\![\mathsf{g}]\!]$ such that suprema occur only in f and infima only in g, it is *not* actually necessary to evaluate them exactly (which amounts to solving optimization problems). Instead, one can, loosely speaking, replace sup and inf by $\forall$-quantifiers. This idea is based on the following elementary observation: For arbitrary sets $A, B \subseteq \mathbb{R}$, we have $\sup A \leq \inf B$ iff $\forall a \in A, b \in B \colon a \leq b$.

Lemma 8.9 (Checking Quantitative Entailments, cf. [52, Section 5.2]). *Let*

$$\mathsf{f} \;=\; \mathcal{E}_{[a_1,b_1]}\, x_1 \colon \ldots \mathcal{E}_{[a_n,b_n]}\, x_n \colon \mathsf{f}' \qquad \textit{and} \qquad \mathsf{g} \;=\; \mathcal{U}_{[c_1,d_1]}\, z_1 \colon \ldots \mathcal{U}_{[c_m,d_m]}\, z_m \colon \mathsf{g}'$$

be $\mathcal{U}$*-free (*$\mathcal{E}$*-free, respectively) expressions in PNF. Further, let* $\psi_{\mathsf{f}'}(\mathsf{free}(\mathsf{f}'), y_{\mathsf{f}})$ *and* $\psi_{\mathsf{g}'}(\mathsf{free}(\mathsf{g}'), y_{\mathsf{g}})$ *be the FO formulae in existential prenex form encoding* f' *and* g' *from Lemma 8.5. Then:*

$$[\![\mathsf{f}]\!] \sqsubseteq [\![\mathsf{g}]\!]$$

$$\textit{iff} \quad \left(\bigwedge_{\substack{x \in \mathsf{free}(\mathsf{f}) \\ \cup\, \mathsf{free}(\mathsf{g})}} x \geq 0 \;\wedge\; \bigwedge_{i=1}^{n} a_i \leq x_i \leq b_i \;\wedge\; \bigwedge_{i=1}^{m} c_i \leq z_i \leq d_i \;\wedge\; \psi_{\mathsf{f}'} \wedge \psi_{\mathsf{g}'} \right) \;\to\; y_{\mathsf{f}} \leq y_{\mathsf{g}} \quad \textit{is valid} \tag{4}$$

As a consequence, the decision problem "$[\![\mathsf{f}]\!] \sqsubseteq [\![\mathsf{g}]\!]$*?" for* f *and* g *is linear-time reducible to* QF_NRA.

Example 8.10. Reconsider the expression $\mathsf{f} = \mathcal{E}_{[0,5]}\, x \colon \left[w = x^2\right] \cdot x$ from Example 8.4. Suppose we wish to check whether $[\![\mathsf{f}]\!]$ is upper-bounded by the constant function $[\![\mathsf{g}]\!]$, $\mathsf{g} = 4$. Since f and g have the form required by Lemma 8.9 we can achieve this by considering the FO formula

$$\big(w \geq 0 \wedge 0 \leq x \leq 5 \wedge (w = x^2 \to y = x) \wedge (w \neq x^2 \to y = 0) \wedge y' = 4\big) \;\to\; y \leq y'$$

which is *not* valid as witnessed by the assignment $\{x \mapsto 5, w \mapsto 25, y \mapsto 5, y' \mapsto 4\}$. Hence $[\![\mathsf{f}]\!] \not\sqsubseteq 4$.

8.3 Decidability and Complexity of pWhile Verification

We now assume (w.l.o.g.) that the set of logical variables LV used in syntactic expressions contains our fixed set V of program variables.

Definition 8.11 (Representable and Syntactic Expectations). We define the following terminology:

- An expectation $f \in \mathbb{E}$ is called *representable* if $f = [\![\mathsf{f}]\!]$ for some $\mathsf{f} \in \mathsf{Expr}$ with $\mathsf{free}(\mathsf{f}) \subseteq V$. The set of all representable (1-bounded) expectations is denoted $\mathbb{E}_{\mathsf{Expr}}$ ($\mathbb{E}^{\leq 1}_{\mathsf{Expr}}$, respectively).
- A *syntactic expectation* is an expression $\mathsf{f} \in \mathsf{Expr}$ whose free variables are program variables, i.e., $\mathsf{free}(\mathsf{f}) \subseteq V$. A *1-bounded* syntactic expectation $\mathsf{Expr}^{\leq 1}$ is $\mathsf{f} \in \mathsf{Expr}$ such that $[\![\mathsf{f}]\!] \in \mathbb{E}^{\leq 1}$. The set of all 1-bounded syntactic expectations is denoted $\mathsf{Expr}^{\leq 1}$. △

Membership in $\mathsf{Expr}^{\leq 1}$ is decidable by encoding $\mathsf{f}(x_1, \ldots, x_n)$ as the FO formula $\psi_{\mathsf{f}}(x_1, \ldots, x_n, y)$ from Lemma 8.5 and checking validity of $(x_1 \geq 0 \wedge \ldots \wedge x_n \geq 0 \wedge \psi_{\mathsf{f}}(x_1, \ldots, x_n, y)) \to y \leq 1$.

Our next theorem implies that if we restrict our framework to representable expectations and only allow the Terms and Guards defined in Section 8.1 as the arithmetic and Boolean, respectively, expressions in our programs, then we obtain the convergence guarantees from Section 7.

Theorem 8.12 (Riemann-suitability of Expr). *The combination* $(\mathbb{E}_{\mathsf{Expr}}, \mathsf{Terms}, \mathsf{Guards})$ *with* Terms *and* Guards *as defined in Section 8.1 is Riemann-suitable (see Definition 7.3).*

8.3.1 Verifying Loop-free Programs. Since Expr is effectively closed under weakest (liberal) Riemann pre-expectation (formalized in [10]), it follows that we can effectively prove upper and lower bounds on weakest (liberal) pre-expectations of loop-free programs:

Theorem 8.13 (Bounds on Loop-free w(l)p). *For all loop-free* $C \in \mathsf{pWhile}(\mathsf{Terms}, \mathsf{Guards})$ *and integers* $N \geq 1$ *the following decision problems can be effectively translated*[13] *to* QF_NRA*: Given ...*

(1) ... an $\overline{\mathsf{a}}$*-free* $\mathsf{f} \in \mathsf{Expr}$ *and an* a*-free* $\mathsf{g} \in \mathsf{Expr}$*, is* $[\![\mathsf{g}]\!] \sqsubseteq \underline{\mathsf{wp}}^N[\![C]\!]([\![\mathsf{f}]\!])$ *?*
If yes, then $[\![\mathsf{g}]\!] \sqsubseteq \mathsf{wp}[\![C]\!]([\![\mathsf{f}]\!])$*, provided* $[\![\mathsf{f}]\!] \in \mathbb{E}_{meas}$*.*

(2) ... an a*-free* $\mathsf{f} \in \mathsf{Expr}$ *and an* $\overline{\mathsf{a}}$*-free* $\mathsf{g} \in \mathsf{Expr}$*, is* $\overline{\mathsf{wp}}^N[\![C]\!]([\![\mathsf{f}]\!]) \sqsubseteq [\![\mathsf{g}]\!]$ *?*
If yes, then $\mathsf{wp}[\![C]\!]([\![\mathsf{f}]\!]) \sqsubseteq [\![\mathsf{g}]\!]$*, provided* $[\![\mathsf{f}]\!] \in \mathbb{E}_{meas}$*.*

(3) ... an $\overline{\mathsf{a}}$*-free* $\mathsf{f} \in \mathsf{Expr}^{\leq 1}$ *and an* a*-free* $\mathsf{g} \in \mathsf{Expr}^{\leq 1}$*, is* $[\![\mathsf{g}]\!] \sqsubseteq \underline{\mathsf{wlp}}^N[\![C]\!]([\![\mathsf{f}]\!])$ *?*
If yes, then $[\![\mathsf{g}]\!] \sqsubseteq \mathsf{wlp}[\![C]\!]([\![\mathsf{f}]\!]))$*, provided* $[\![\mathsf{f}]\!] \in \mathbb{E}^{\leq 1}_{meas}$*.*

(4) ... an a*-free* $\mathsf{f} \in \mathsf{Expr}^{\leq 1}$ *and an* $\overline{\mathsf{a}}$*-free* $\mathsf{g} \in \mathsf{Expr}^{\leq 1}$*, is* $\overline{\mathsf{wlp}}^N[\![C]\!]([\![\mathsf{f}]\!]) \sqsubseteq [\![\mathsf{g}]\!]$ *?*
If yes, then $\mathsf{wlp}[\![C]\!]([\![\mathsf{f}]\!]) \sqsubseteq [\![\mathsf{g}]\!]$*, provided* $[\![\mathsf{f}]\!] \in \mathbb{E}^{\leq 1}_{meas}$*.*

Moreover, for fixed C and f, the size of the syntactic expectation representing $\mathcal{T}[\![C]\!]([\![\mathsf{f}]\!])$ for $\mathcal{T} \in \{\underline{\mathsf{wp}}^N, \overline{\mathsf{wp}}^N, \underline{\mathsf{wlp}}^N, \overline{\mathsf{wlp}}^N\}$ is in $O\left(N^k\right)$, where k is the number of `unif`-statements in C.
Using Theorem 8.13 and Corollary 5.5, it is straightforward to derive bounds on cwp as well.

8.3.2 Verifying Loops. We now state how to apply the proof rules from Section 6 effectively:

Theorem 8.14 (Verification of Invariants). *Let* $C = \texttt{while}\,(\varphi)\,\{B\} \in \mathsf{pWhile}(\mathsf{Terms}, \mathsf{Guards})$ *such that* B *is loop-free. Further, let* $\mathsf{I}, \mathsf{f} \in \mathsf{Expr}$ *and* $\mathsf{J}, \mathsf{g} \in \mathsf{Expr}^{\leq 1}$ *all be quantifier-free, and let* $N \geq 1$ *be an integer. Then we can ...*

(1) ... decide if $[\![\mathsf{I}]\!]$ *is a* $\overline{\mathsf{wp}}^N$*-superinvariant of* C *w.r.t.* $[\![\mathsf{f}]\!]$ *by a reduction to* QF_NRA*.*
If the superinvariant property holds, then $\mathsf{wp}[\![C]\!]([\![\mathsf{f}]\!]) \sqsubseteq [\![I]\!]$*, provided* $[\![\mathsf{f}]\!] \in \mathbb{E}_{meas}$

(2) ... decide if $[\![\mathsf{J}]\!]$ *is a* $\underline{\mathsf{wlp}}^N$*-subinvariant of* C *w.r.t.* $[\![\mathsf{g}]\!]$ *by a reduction to* QF_NRA*.*
If the subinvariant property holds, then $[\![J]\!] \sqsubseteq \mathsf{wlp}[\![C]\!]([\![\mathsf{g}]\!])$*, provided* $[\![\mathsf{g}]\!] \in \mathbb{E}^{\leq 1}_{meas}$*.*

Theorem 8.15 (Complexity of Verification Problems). *The following decision problems are* coRE*-complete*[14]*: Given an arbitrary (not necessarily loop-free)* $C \in \mathsf{pWhile}(\mathsf{Terms}, \mathsf{Guards})$ *and ...*

(1) ... $\mathsf{f}, \mathsf{g} \in \mathsf{Expr}$ *such that* $[\![\mathsf{f}]\!] \in \mathbb{E}_{meas}$*, does it hold that* $\mathsf{wp}[\![C]\!]([\![\mathsf{f}]\!]) \sqsubseteq [\![\mathsf{g}]\!]$ *?*
(2) ... $\mathsf{f}, \mathsf{g} \in \mathsf{Expr}^{\leq 1}$ *such that* $[\![\mathsf{f}]\!] \in \mathbb{E}^{\leq 1}_{meas}$*, does it hold that* $[\![\mathsf{g}]\!] \sqsubseteq \mathsf{wlp}[\![C]\!]([\![\mathsf{f}]\!])$ *?*

The proof of membership in coRE heavily relies on the convergence results established in Theorem 7.6. The coRE-hardness proofs are standard [36] and follow from coRE-hardness of the non-halting problem. See [10] for details.

[13] In polynomial time if we assume C to be of constant size and N given in unary. In general, an expression encoding $\underline{\mathsf{wp}}^N[\![C]\!]([\![\mathsf{f}]\!])$ for a given f can be exponential in the size of C — consider n sequential `if-else` constructs. However, our reduction is of practical interest as QF_NRA is supported by many SMT solvers.
[14] I.e., Π^0_1-complete in the arithmetical hierarchy.

9 Implementation and Case Studies

We have automated our techniques using a modern deductive verifier for probabilistic programs. We first describe how to integrate our techniques within that verifier in Section 9.1. We then describe our case studies, i.e., programs and specifications we have verified, in Section 9.2. Finally, we evaluate our approach empirically on these case studies in Section 9.3.

9.1 Implementation

Caesar[15] [52] is a recent expectation-based automated deductive verifier targeting *discrete* and possibly *nondeterministic*[16] probabilistic programs. As input, Caesar takes programs written in an intermediate verification language called HeyVL — a quantitative analogue of Boogie [40]. HeyVL provides a programmatic means to describe verification conditions such as the validity of quantitative loop invariants. These verification conditions are offloaded to an SMT solver, which enables the semi-automated verification of said discrete probabilistic programs. Caesar does *not* support continuous sampling instructions. We will, however, now demonstrate that with our approach based on Riemann sums, Caesar *can* readily be used to verify bounds on expected outcomes of continuous probabilistic programs in a semi-automated fashion.

Our key insight is that our Riemann expectation transformers can be expressed as expectation transformers denoted by *discrete* probabilistic programs featuring angelic (resp. demonic) *nondeterministic choices* for $\mathbb{R}_{\geq 0}$-valued intervals. The latter features *are* supported by Caesar: HeyVL supports the discrete fragment of pWhile and, amongst others, the two statements

(1) $\mathtt{x} \coloneqq [E_1, E_2]$, which, on program state σ, assigns a nondeterministically chosen value from the $\mathbb{R}_{\geq 0}$-valued interval $[E_1(\sigma), E_2(\sigma)]$ to the variable $\mathtt{x}$, and
(2) $\mathtt{x} :\approx \mathtt{discrete\_unif}\,(N)$, which, given a (constant) natural number $N \geq 1$, samples a value from the set $\{0, \ldots, N-1\}$ uniformly at random and assigns the result to the variable $\mathtt{x}$.

Notice that the latter statement is syntactic sugar for pWhile as it can be simulated by binary probabilistic choices. The statement $\mathtt{x} \coloneqq [E_1, E_2]$, on the other hand, is not syntactic sugar as nondeterminism is not supported in pWhile.

Now, given a (discrete but possibly nondeterministic) program C, Caesar employs a transformer $\mathrm{vc}[\![C]\!] \colon \mathbb{E} \to \mathbb{E}$, which is defined[17] as in Table 1 and where for the additional statements, we have

$$\mathrm{vc}[\![\mathtt{x} \coloneqq [E_1, E_2]]\!]\,(f) \;=\; \lambda\sigma. \sup_{\xi \in [E_1(\sigma), E_2(\sigma)]} f(\sigma[x \mapsto \xi])$$

$$\mathrm{vc}[\![\mathtt{x} :\approx \mathtt{discrete\_unif}\,(N)]\!]\,(f) \;=\; \frac{1}{N} \cdot \sum_{i=0}^{N-1} f[\mathtt{x}/i]\;.$$

Notice that $\mathrm{vc}[\![\mathtt{x} \coloneqq [E_1, E_2]]\!]\,(f)$ resolves the nondeterministic choice of $\mathtt{x}$ *angelically* by returning the *supremum* of all values obtained from evaluating f at σ where $\mathtt{x}$ is set to some value from $[E_1(\sigma), E_2(\sigma)]$. The demonic counterpart is obtained from replacing sup by inf. Now let $\mathtt{j}$ be a fresh program variable and observe that for all $N \geq 1$ and all $f \in \mathbb{E}$, we have

$$\underbrace{\overline{\mathrm{wp}}^{N}[\![\mathtt{x} :\approx \mathtt{unif}_{[0,1]}]\!]\,(f)}_{\text{defined in this paper}} \;=\; \underbrace{\mathrm{vc}[\![\mathtt{j} :\approx \mathtt{discrete\_unif}\,(N)\;;\; \mathtt{x} \coloneqq [\tfrac{\mathtt{j}}{N}, \tfrac{\mathtt{j}+1}{N}]]\!]\,(f)}_{\text{expressible in HeyVL and thus supported by Caesar}}\;.$$

[15] https://www.caesarverifier.org/
[16] Here we refer to pure nondeterminism which is to be resolved angelically or demonically.
[17] We do not need to require f to be measurable since C is does not contain continuous sampling. Moreover, as is standard for deductive verifiers, loops need to be annotated with quantitative loop invariants; see Example 9.1 on page 22.

```
while (i ≤ M) {                          while (i ≤ M) {
    y :≈ unif_[0,1] ; x := x + y;            y :≈ unif_[0,1] ; observe(y ≤ 1/2) ; x := x + y;
    i := i + 1   }                           i := i + 1   }
```

Fig. 2. Generating the standard Irwin-Hall distribution (left) and a variant with conditioning (right).

Hence, the upper Riemann pre-expectation of $\mathtt{x} :\approx \mathtt{unif}_{[0,1]}$ w.r.t. f corresponds to the maximal — under all possible resolutions of the nondeterminism — expected final value of f obtained from (i) sampling one of the N intervals occurring in the Riemann sums uniformly at random and (ii) assigning to $\mathtt{x}$ a nondeterministically chosen value from that sampled interval. *Lower* Riemann pre-expectations can be expressed analogously by resolving the nondeterminism *demonically*.

Example 9.1. Reconsider the Monte Carlo π-approximator C and the superinvariant I of C w.r.t. `count` from Example 6.3. We can provide Caesar with the following annotated loop:

$$
\begin{aligned}
&\mathtt{@invariant}\left(\mathtt{count} + [\mathtt{i} \le \mathtt{M}] \cdot \left(0.85 \cdot ((\mathtt{M} \dot{-} \mathtt{i}) + 1)\right)\right)\\
&\mathtt{while}\ (\mathtt{i} \le \mathtt{M})\ \{\\
&\qquad \mathtt{j}_1 :\approx \mathtt{discrete\_unif}\,(16)\ ;\ \mathtt{x} := [\tfrac{\mathtt{j}_1}{16}, \tfrac{\mathtt{j}_1+1}{16}]\,;\\
&\qquad \mathtt{j}_2 :\approx \mathtt{discrete\_unif}\,(16)\ ;\ \mathtt{y} := [\tfrac{\mathtt{j}_2}{16}, \tfrac{\mathtt{j}_2+1}{16}]\,;\\
&\qquad \mathtt{if}(\mathtt{x}^2 + \mathtt{y}^2 \le 1)\ \{\mathtt{count} := \mathtt{count} + 1\}\ \mathtt{else}\ \{\mathtt{skip}\}\,;\ \mathtt{i} := \mathtt{i} + 1 \quad \}
\end{aligned}
$$

We can then instruct Caesar to check whether I is a superinvariant of this loop w.r.t. `count`. Since the loop body encodes appropriate upper Riemann pre-expectations, Caesar offloads the quantitative entailments corresponding to Theorem 8.14 to an SMT solver to check them automatically. For lower Riemann pre-expectations and subinvariants (for wlp), Caesar can be used analogously.

9.2 Case Studies

In what follows, we describe the programs and specifications we have verified using Caesar with the SMT solver Z3 [24] as back-end. All case studies (including Example 9.1) have been verified within 12 seconds on an Apple M2. The precise inputs to Caesar are provided in [10]. Both the verified bounds and the required parition sizes N where determined manually by guessing the respective weakest pre-expectations and increasing N until Caesar reports success.

9.2.1 The Irwin-Hall Distribution. The Irwin-Hall distribution (parameterized in `M`) [34] is the sum of `M` independent and identically distributed random variables, each of which is distributed uniformly over $[0, 1]$. The loop C depicted in Figure 2 (left) models this family of distributions: On each iteration, C samples a value for `y` uniformly at random and adds the result to the variable `x`. Hence, if initially $\mathtt{i} = 1$, $\mathtt{x} = 0$, and $\mathtt{M} \in \mathbb{N}$, then the final distribution of `x` indeed corresponds to the Irwin-Hall distribution with parameter `M`.

We now aim to upper-bound the expected final value of `x`, i.e., the expectation of the Irwin-Hall distribution, for *arbitrary* values of `M`. Towards this end, we employ our encoding from Section 9.1 and use Caesar to automatically verify that the expectation $I = \mathtt{x} + [\mathtt{i} \le \mathtt{M}] \cdot \left(1.1 \cdot \frac{(\mathtt{M} \dot{-} \mathtt{i})+1}{2}\right)$ is a $\overline{\mathsf{wp}}^{10}$-superinvariant of C w.r.t. `x`. Hence, by Theorem 6.2, we get $\mathsf{wp}[\![C]\!]\,(\mathtt{x}) \sqsubseteq I$ and thus

$$\text{for initial } \sigma \text{ with } \sigma(\mathtt{i}) = 1,\ \sigma(\mathtt{x}) = 0,\ \sigma(\mathtt{M}) \in \mathbb{N} \text{ with } \sigma(\mathtt{M}) \ge 1\text{: } \mathsf{wp}[\![C]\!]\,(\mathtt{x})\,(\sigma) \le 1.1 \cdot \frac{\sigma(\mathtt{M})}{2}\ .$$

Now consider a variant of the Irwin-Hall distribution where we condition each of the `M` random variables to take a value in $[0, 1/2]$. This situation is modeled by the loop C depicted in Figure 2

```
while (x > 0) {
    y :≈ unif_[0,1] ; y := (b ∸ a) · y + a ;
    if(y ≤ (a + b)/2) {diverge} else {skip} ;
    x := x ∸ 1   }
```

```
while (h ≤ t) {
    {x :≈ unif_[0,1] ; x := 10 · x ; h := h + x} [1/2] {skip} ;
    t := t + 1 ;
    count := count + 1   }
```

Fig. 3. A probably diverging loop (left) and a race between tortoise and hare (right) [16].

(right), where we employ conditioning inside of the loop. We aim to upper-bound $\mathrm{cwp}[\![C]\!]\,(\mathtt{x})$, i.e., the *conditional expected final value of* $\mathtt{x}$. For that, we use Caesar to verify that the following expectation is a $\overline{\mathrm{wp}}$[19]-superinvariant of C w.r.t. $\mathtt{count}$ (resp. $\underline{\mathrm{wlp}}$[2]-subinvariant of C w.r.t. 1):

$$I \;=\; \mathtt{x} + [\mathtt{i} \le \mathtt{M}] \cdot \Big(1.5 \cdot \frac{(\mathtt{M} \dot{-} \mathtt{i}) + 1}{8}\Big) \qquad \text{and} \qquad J \;=\; [\mathtt{i} \le \mathtt{M}] \cdot 0.5^{(\mathtt{M} \dot{-} \mathtt{i}) + 1} + [\mathtt{i} > \mathtt{M}] \cdot 1\,.$$

Hence, we get by Theorem 6.2 that $\mathrm{cwp}[\![C]\!]\,(\mathtt{x})\,(\sigma)$ is well-defined for all $\sigma \in \mathbb{R}^V_{\ge 0}$ and

$$\text{for initial } \sigma \text{ with } \sigma(\mathtt{i}) = 1,\ \sigma(\mathtt{x}) = 0,\ \sigma(\mathtt{M}) \in \mathbb{N} \text{ with } \sigma(\mathtt{M}) \ge 1\text{: } \mathrm{cwp}[\![C]\!]\,(\mathtt{x})\,(\sigma) \le \frac{1.5 \cdot \sigma(\mathtt{M})}{8 \cdot 0.5^{\sigma(\mathtt{M})}}\,.$$

Notice that, even though exponential functions like $0.5^{\mathtt{M}}$ are not supported by our decidable language of syntactic expectations from Section 8, we can use Caesar to reason about it by means of standard user-defined domain declaration for exponentials (see [52, Section 5.1] for details). Decidability of the corresponding entailments involving such domains is, however, not guaranteed.

9.2.2 Reasoning about Diverging Programs. Weakest *liberal* pre-expectations enable us to lower-bound divergence probabilities of programs involving continuous sampling. Consider the loop C depicted in Figure 3 (left). In each iteration, we sample uniformly from the interval $[\mathtt{a}, \mathtt{b}]$ (notice that $\mathtt{a}$, $\mathtt{b}$ are uninitialized variables and hence correspond to parameters) and assign the result to $\mathtt{y}$. Whenever we sample a value in $[0, \frac{\mathtt{a}+\mathtt{b}}{2}]$, the program diverges. Using Caesar, we verify that

$$I \;=\; [\mathtt{a} \le \mathtt{b}] \cdot (1 \dot{-} 0.5^{\mathtt{x}})$$

is a $\underline{\mathrm{wlp}}$[2]-subinvariant of C w.r.t. 0. Hence, we have $I \sqsubseteq \mathrm{wlp}[\![C]\!]\,(0)$ by Theorem 6.2, i.e., whenever $\mathtt{a} \le \mathtt{b}$, then C diverges on initial state σ with probability at least $1 - 0.5^{\sigma(\mathtt{x})}$.

9.2.3 Race between Tortoise and Hare. The loop C depicted in Figure 3 (right) is adapted from [16] and models a race between a tortoise ($\mathtt{t}$) and a hare ($\mathtt{h}$). As long as the hare did not overtake the tortoise, the hare flips a fair coin to decide whether to move or not. If the hare decides to move, it samples a distance uniformly at random from $[0, 10]$. The tortoise always moves exactly one step.

We now upper-bound the expected final value of variable $\mathtt{count}$. Towards this end, we use Caesar to verify that the expectation $I = \mathtt{count} + [\mathtt{h} \le \mathtt{t}] \cdot 3.012 \cdot \big((\mathtt{t} \dot{-} \mathtt{h}) + 2\big)$ is a $\overline{\mathrm{wp}}$[16]-superinvariant of C w.r.t. $\mathtt{count}$. Hence, we get $\mathrm{wp}[\![C]\!]\,(\mathtt{count}) \sqsubseteq I$ by Theorem 6.2, i.e., if C is executed on an initial state σ with $\sigma(\mathtt{count}) = 0$ and $\sigma(\mathtt{h}) \le \sigma(\mathtt{t})$, then the expected final value of $\mathtt{count}$ is at most $3.012 \cdot (\sigma(\mathtt{t}) - \sigma(\mathtt{h}) + 2)$.

9.3 Experimental Evaluation

In this section, we empirically evaluate the scalability of our approach based on the programs from the previous sections, i.e., our case studies and the Monte-Carlo approximator from Figure 1. For each of these programs and post-expectations, we verify the tightest upper bound (up to a precision of 3 decimal places) on the respective weakest pre-expectation for increasing values of N (except for `diverging` where $N = 2$ already yields the precise weakest pre-expectation).

Table 2. Experimental results. Time in seconds. TO = 180 s, MO = 8 GB.

Program	N	Time	|AST|	Post	Bound
`MonteCarlo` (Fig. 1)	2	0.03	552	`count`	$1.000 \cdot \mathtt{M}$
	4	0.05	1634		$0.938 \cdot \mathtt{M}$
	8	0.38	5448		$0.875 \cdot \mathtt{M}$
	16	10.05	19676		$0.837 \cdot \mathtt{M}$
	32	TO	74526		–
`IrwinHall` (Fig. 2 (left))	2	0.02	187	`x`	$0.750 \cdot \mathtt{M}$
	4	0.02	309		$0.625 \cdot \mathtt{M}$
	8	0.03	553		$0.565 \cdot \mathtt{M}$
	10	0.08	716		$0.55 \cdot \mathtt{M}$
	16	2.15	1041		$0.532 \cdot \mathtt{M}$
	32	TO	2017		–
`diverging` (Fig. 3 (left))	2	0.22	162	0	$[\mathtt{a} \leq \mathtt{b}] \cdot (1 \dot{-} 0.5^{\mathtt{x}})$
`TortoiseHare` (Fig. 3 (right))	2	0.02	2	`count`	∞
	4	.03	400		$8 \cdot (\mathtt{t} \dot{-} \mathtt{h} + 2)$
	8	.015	680		$3.38 \cdot (\mathtt{t} \dot{-} \mathtt{h} + 2)$
	16	100.09	1240		$3.012 \cdot (\mathtt{t} \dot{-} \mathtt{h} + 2)$
	32	TO	2160		–

Our empirical results are depicted in Table 2. Column **Program** depicts the respective program, N denotes the partition size for the Riemann pre-expectation, **Time** denotes the verification time in seconds (i.e., verification condition generation and time for SMT solving) required by Caesar, **|AST|** denotes the size of the formula offloaded to the SMT solver Z3 (i.e., the number of nodes in the formula's abstract syntax tree), **Post** denotes the post-expectation, and **Bound** depicts the upper (lower) bound on the (liberal) weakest pre-expectation. For the `diverging` benchmark, we lower-bound a liberal weakest pre-expectation. All other benchmarks upper-bound a non-liberal weakest pre-expectation. The bounds were inferred manually in a binary search-like fashion.

With our encoding of Riemann pre-expectations described in Section 9.1, Caesar allows verifying increasingly sharper bounds when increasing the partition size N The size of the generated formula for the verification conditions mostly increases moderately when increasing N. This is to be expected since the formula size grows polynomially in N for a fixed program and a fixed post-expectation (see Section 8.3.1). The time required for verification, however, can increase drastically when doubling the value of N (see entries of Table 2 where $N = 32$).

10 Related Work

Integral Approximation. Our approach relies on under- and over-approximation of Lebesgue integrals via Riemann sums, achieved by discretizing the domain of the continuous uniform distributions in the program. Recently, [25] proposed an efficient *bit blasting* discretization method for continuous mixed-gamma distributions (which subsume the uniform distributions) for programs with bounded (`for`) loops. In [11], integral approximation is used to derive guaranteed bounds on a program's posterior distribution. Following up on this, [58] extended the computation of such bounds to a class of programs with soft conditioning (*score-at-end* programs). Previously, [51] and [3] used integral approximations to bound the satisfaction probabilities of properties like safety and fairness.

However, these approaches are not able to deal with unbounded loops in full generality. AQUA [32] uses a different discretization method, where the posterior of a given program is approximated by quantizing the state space. However, this approximation does not necessarily provide an upper or lower bound of the true distribution, and unbounded loops are not supported. Finally, [55] propose a method to derive upper and lower bounds on central moments of cost accumulators.

Loop Invariant Analysis. The main purpose of the current work is to enable automatic verification of inductive invariants for continuous probabilistic programs. Reasoning about probabilistic loops via quantitative invariants was first introduced by McIver and Morgan [41] for probabilistic programs with discrete probabilistic choice and extended by various authors to different settings [30, 37, 53]. A generalization of this framework to programs featuring continuous distribution was proposed in [15] and uses super-martingale ranking functions as the probabilistic analogue of classic ranking functions. The super-martingale approach has been specialized to qualitative and quantitative termination problems for various classes of programs such as polynomial programs [17], affine programs admitting a linear ranking super-martingale [18], and non-deterministic probabilistic programs [2, 21]. Besides termination, this approach has been successfully applied also to cost analysis [57] and equivalence refutation [20]. While building on the martingale approach, [19] overcomes the necessity to compute ranking martingales, by introducing the notion of stochastic invariants indicators, that can be used to prove termination in a sound and relatively complete way. A different generalization was proposed in [16], where expectation invariants are proposed to introduce a generalized fixed-point framework for programs with continuous distributions and unbounded loops. However, this analysis rules out invariants expressed as Iverson brackets and is restricted to piecewise linear assertion guards and updates. Similarly, [56], restricts to affine programs and invariants to find exponential bounds on assertion violation probabilities.

Exact Inference. Exact inference on probabilistic programs involving continuous distributions is especially hard because of the challenge of computing densities via integrals. Exact symbolic inference engines like [26, 27, 45, 50] only support deterministically bounded loops. Even within this constrained context, the occurrence of non-simplified integrals in the output can significantly impede the potential to perform appropriate analysis or further computations. A different, yet still exact approach is pursued in [43], where the focus is on the computation of moments as functions of the loop iteration. In order to compute the moments exactly, the syntax of the programs is restricted quite severely, though parametric models are supported. Under the same syntactic restrictions, it has been shown in [42] that termination probabilities can be computed automatically.

11 Conclusion

We tackled the semi-automatic verification of user-provided quantitative invariants for probabilistic programs involving continuous distributions, conditioning, and potentially unbounded loops. Our approach is to approximate the Lebesgue integrals appearing in a program's weakest pre-expectation semantics by lower and upper Riemann sums. This simple idea allows us to prove sound bounds on expected outcomes by verifying invariants automatically with SMT-solvers. We demonstrated that this method can be readily integrated in the Caesar verification infrastructure [52].

There are many opportunities for future work. A natural direction is to integrate our approach to invariant verification with techniques for *fully automatic invariant generation* such as [7]. While we can already verify invariants for non-trivial benchmarks, there is some room for improvement on the engineering side. This includes more sophisticated, multi-dimensional, and adaptive partitions of the integration domain, and a combination with direct integration methods, where applicable. We also plan to extend the front end of Caesar with dedicated support for continuous distributions.

Due to its simplicity, we are confident that our approach is amenable to generalizations such as non-determinism and strategy synthesis as in [6], and programs with pointers and data structures.

Acknowledgments

The authors thank Philipp Schroer for his support with Caesar, Mirco Tribastone for his help with the initial conceptualization of the paper, and the anonymous referees for their detailed comments. This work was partially funded by the DFG GRK 2236 UnRAVeL, the EU's Horizon 2020 programme under the Marie Skłodowska-Curie grant No. 101008233 (MISSION), the ERC AdG No. 787914 (FRAPPANT), and the PNRR project iNEST (Interconnected Nord-Est Innovation Ecosystem) funded by the European Union Next-GenerationEU (Piano Nazionale di Ripresa e Resilienza (PNRR) – Missione 4 Componente 2, Investimento 1.5 – D.D. 1058 23/06/2022, ECS_00000043).

Data Availability Statement

A software artifact comprising the Caesar verifier, the encodings of the case studies from Section 9, and instructions for running the experiments is available [9].

References

[1] Sheshansh Agrawal, Krishnendu Chatterjee, and Petr Novotný. 2017. Lexicographic Ranking Supermartingales: An Efficient Approach to Termination of Probabilistic Programs. *CoRR* abs/1709.04037 (2017). arXiv:1709.04037 http://arxiv.org/abs/1709.04037

[2] Sheshansh Agrawal, Krishnendu Chatterjee, and Petr Novotný. 2018. Lexicographic ranking supermartingales: an efficient approach to termination of probabilistic programs. *Proc. ACM Program. Lang.* 2, POPL (2018), 34:1–34:32. doi:10.1145/3158122

[3] Aws Albarghouthi, Loris D'Antoni, Samuel Drews, and Aditya V. Nori. 2017. FairSquare: probabilistic verification of program fairness. *Proc. ACM Program. Lang.* 1, OOPSLA (2017), 80:1–80:30. doi:10.1145/3133904

[4] Robert B Ash and Catherine A Doléans-Dade. 2000. *Probability and measure theory*. Academic press.

[5] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. 2016. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`.

[6] Kevin Batz, Tom Jannik Biskup, Joost-Pieter Katoen, and Tobias Winkler. 2024. Programmatic Strategy Synthesis: Resolving Nondeterminism in Probabilistic Programs. *Proc. ACM Program. Lang.* 8, POPL (2024), 2792–2820. doi:10.1145/3632935

[7] Kevin Batz, Mingshuai Chen, Sebastian Junges, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. 2023. Probabilistic Program Verification via Inductive Synthesis of Inductive Invariants. In *TACAS (2) (Lecture Notes in Computer Science, Vol. 13994)*. Springer, 410–429. doi:10.1007/978-3-031-30820-8_25

[8] Kevin Batz, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. 2021. Relatively complete verification of probabilistic programs: an expressive language for expectation-based reasoning. *Proc. ACM Program. Lang.* 5, POPL (2021), 1–30. doi:10.1145/3434320

[9] Kevin Batz, Joost-Pieter Katoen, Francesca Randone, and Tobias Winkler. 2025. *Artifact for Paper Foundations for Deductive Verification of Continuous Probabilistic Programs*. doi:10.5281/zenodo.15175355

[10] Kevin Batz, Joost-Pieter Katoen, Francesca Randone, and Tobias Winkler. 2025. Foundations for Deductive Verification of Continuous Probabilistic Programs: From Riemann to Lebesgue and Back. *CoRR* abs/2502.19388 (2025). https://arxiv.org/abs/2502.19388

[11] Raven Beutner, C.-H. Luke Ong, and Fabian Zaiser. 2022. Guaranteed bounds for posterior inference in universal probabilistic programming. In *PLDI*. ACM, 536–551. doi:10.1145/3519939.3523721

[12] Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul A. Szerlip, Paul Horsfall, and Noah D. Goodman. 2019. Pyro: Deep Universal Probabilistic Programming. *J. Mach. Learn. Res.* 20 (2019), 28:1–28:6. https://jmlr.org/papers/v20/18-403.html

[13] Frank E Burk. 2007. *A garden of integrals*. Vol. 31. American Mathematical Soc. doi:10.7135/UPO9781614442097

[14] Bob Carpenter, Andrew Gelman, Matthew D. Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. 2017. Stan: A Probabilistic Programming Language. *Journal of Statistical Software* 76, 1 (2017), 1–32. doi:10.18637/jss.v076.i01

[15] Aleksandar Chakarov and Sriram Sankaranarayanan. 2013. Probabilistic Program Analysis with Martingales. In *CAV (Lecture Notes in Computer Science, Vol. 8044)*. Springer, 511–526. doi:10.1007/978-3-642-39799-8_34

[16] Aleksandar Chakarov and Sriram Sankaranarayanan. 2014. Expectation Invariants for Probabilistic Program Loops as Fixed Points. In *SAS (Lecture Notes in Computer Science, Vol. 8723)*. Springer, 85–100. doi:10.1007/978-3-319-10936-7_6

[17] Krishnendu Chatterjee, Hongfei Fu, and Amir Kafshdar Goharshady. 2016. Termination Analysis of Probabilistic Programs Through Positivstellensatz's. In *CAV (1) (Lecture Notes in Computer Science, Vol. 9779)*. Springer, 3–22.

doi:10.1007/978-3-319-41528-4_1

[18] Krishnendu Chatterjee, Hongfei Fu, Petr Novotný, and Rouzbeh Hasheminezhad. 2016. Algorithmic analysis of qualitative and quantitative termination problems for affine probabilistic programs. In *POPL*. ACM, 327–342. doi:10.1145/2837614.2837639

[19] Krishnendu Chatterjee, Amir Kafshdar Goharshady, Tobias Meggendorfer, and Dorde Zikelic. 2022. Sound and Complete Certificates for Quantitative Termination Analysis of Probabilistic Programs. In *CAV (1) (Lecture Notes in Computer Science, Vol. 13371)*. Springer, 55–78. doi:10.1007/978-3-031-13185-1_4

[20] Krishnendu Chatterjee, Ehsan Kafshdar Goharshady, Petr Novotný, and Dorde Zikelic. 2024. Equivalence and Similarity Refutation for Probabilistic Programs. *Proc. ACM Program. Lang.* 8, PLDI (2024), 2098–2122. doi:10.1145/3656462

[21] Krishnendu Chatterjee, Petr Novotný, and Dorde Zikelic. 2017. Stochastic invariants for probabilistic termination. In *POPL*. ACM, 145–160. doi:10.1145/3009837.3009873

[22] Michel Coste. 2000. An introduction to semialgebraic geometry.

[23] Fredrik Dahlqvist, Alexandra Silva, and Dexter Kozen. 2020. Semantics of probabilistic programming: A gentle introduction. *Foundations of Probabilistic Programming* (2020), 1–42.

[24] Leonardo Mendonça de Moura and Nikolaj S. Bjørner. 2008. Z3: An Efficient SMT Solver. In *TACAS (Lecture Notes in Computer Science, Vol. 4963)*. Springer, 337–340. doi:10.1007/978-3-540-78800-3_24

[25] Poorva Garg, Steven Holtzen, Guy Van den Broeck, and Todd D. Millstein. 2024. Bit Blasting Probabilistic Programs. *Proc. ACM Program. Lang.* 8, PLDI (2024), 865–888. doi:10.1145/3656412

[26] Timon Gehr, Sasa Misailovic, and Martin T. Vechev. 2016. PSI: Exact Symbolic Inference for Probabilistic Programs. In *CAV (1) (Lecture Notes in Computer Science, Vol. 9779)*. Springer, 62–83. doi:10.1007/978-3-319-41528-4_4

[27] Timon Gehr, Samuel Steffen, and Martin T. Vechev. 2020. λPSI: exact inference for higher-order probabilistic programs. In *PLDI*. ACM, 883–897. doi:10.1145/3385412.3386006

[28] Noah D Goodman and Andreas Stuhlmüller. 2014. The Design and Implementation of Probabilistic Programming Languages. http://dippl.org. Accessed: 2024-10-15.

[29] Andrew D. Gordon, Thomas A. Henzinger, Aditya V. Nori, and Sriram K. Rajamani. 2014. Probabilistic programming. In *FOSE*. ACM, 167–181. doi:10.1145/2593882.2593900

[30] Friedrich Gretz, Joost-Pieter Katoen, and Annabelle McIver. 2014. Operational versus weakest pre-expectation semantics for the probabilistic guarded command language. *Perform. Evaluation* 73 (2014), 110–132. doi:10.1016/J.PEVA.2013.11.004

[31] Marcel Hark, Benjamin Lucien Kaminski, Jürgen Giesl, and Joost-Pieter Katoen. 2020. Aiming low is harder: induction for lower bounds in probabilistic program verification. *Proc. ACM Program. Lang.* 4, POPL (2020), 37:1–37:28. doi:10.1145/3371105

[32] Zixin Huang, Saikat Dutta, and Sasa Misailovic. 2022. Automated quantized inference for probabilistic programs with AQUA. *Innov. Syst. Softw. Eng.* 18, 3 (2022), 369–384. doi:10.1007/S11334-021-00433-3

[33] Alon Itai and Michael Rodeh. 1990. Symmetry breaking in distributed networks. *Inf. Comput.* 88, 1 (1990), 60–87. doi:10.1016/0890-5401(90)90004-2

[34] N.L. Johnson, S. Kotz, and N. Balakrishnan. 1995. *Continuous Univariate Distributions, Volume 2*. Wiley.

[35] Benjamin Lucien Kaminski. 2019. *Advanced weakest precondition calculi for probabilistic programs*. Ph. D. Dissertation. RWTH Aachen University, Germany. doi:10.18154/RWTH-2019-01829

[36] Benjamin Lucien Kaminski and Joost-Pieter Katoen. 2015. On the Hardness of Almost-Sure Termination. In *Mathematical Foundations of Computer Science 2015 - 40th International Symposium, MFCS 2015, Milan, Italy, August 24-28, 2015, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 9234)*, Giuseppe F. Italiano, Giovanni Pighizzini, and Donald Sannella (Eds.). Springer, 307–318. doi:10.1007/978-3-662-48057-1_24

[37] Benjamin Lucien Kaminski and Joost-Pieter Katoen. 2017. A weakest pre-expectation semantics for mixed-sign expectations. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*. IEEE Computer Society, 1–12. doi:10.1109/LICS.2017.8005153

[38] Richard M. Karp. 1991. An introduction to randomized algorithms. *Discret. Appl. Math.* 34, 1-3 (1991), 165–201.

[39] Dexter Kozen. 1983. A Probabilistic PDL. In *STOC*. ACM, 291–297. doi:10.1145/800061.808758

[40] K. Rustan M. Leino. 2008. *This Is Boogie 2*.

[41] Annabelle McIver and Carroll Morgan. 2005. *Abstraction, Refinement and Proof for Probabilistic Systems*. Springer. doi:10.1007/B138392

[42] Marcel Moosbrugger, Ezio Bartocci, Joost-Pieter Katoen, and Laura Kovács. 2022. The probabilistic termination tool amber. *Formal Methods Syst. Des.* 61, 1 (2022), 90–109. doi:10.1007/S10703-023-00424-Z

[43] Marcel Moosbrugger, Miroslav Stankovic, Ezio Bartocci, and Laura Kovács. 2022. This is the moment for probabilistic loops. *Proc. ACM Program. Lang.* 6, OOPSLA2 (2022), 1497–1525. doi:10.1145/3563341

[44] Peter Müller, Malte Schwerhoff, and Alexander J. Summers. 2016. Viper: A Verification Infrastructure for Permission-Based Reasoning. In *VMCAI (Lecture Notes in Computer Science, Vol. 9583)*. Springer, 41–62. doi:10.1007/978-3-662-49122-5_2

[45] Praveen Narayanan, Jacques Carette, Wren Romano, Chung-chieh Shan, and Robert Zinkov. 2016. Probabilistic Inference by Program Transformation in Hakaru (System Description). In *FLOPS (Lecture Notes in Computer Science, Vol. 9613)*. Springer, 62–79. doi:10.1007/978-3-319-29604-3_5
[46] Aditya V. Nori, Chung-Kil Hur, Sriram K. Rajamani, and Selva Samuel. 2014. R2: An Efficient MCMC Sampler for Probabilistic Programs. In *AAAI*. AAAI Press, 2476–2482. doi:10.1609/AAAI.V28I1.9060
[47] Federico Olmedo, Friedrich Gretz, Nils Jansen, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Annabelle McIver. 2018. Conditioning in Probabilistic Programming. *ACM Trans. Program. Lang. Syst.* 40, 1 (2018), 4:1–4:50. doi:10.1145/3156018
[48] David Park. 1969. Fixpoint induction and proofs of program properties. *Machine intelligence* 5 (1969).
[49] Walter Rudin. 1953. *Principles of mathematical analysis*.
[50] Feras A. Saad, Martin C. Rinard, and Vikash K. Mansinghka. 2021. SPPL: probabilistic programming with fast exact symbolic inference. In *PLDI*. ACM, 804–819. doi:10.1145/3453483.3454078
[51] Sriram Sankaranarayanan, Aleksandar Chakarov, and Sumit Gulwani. 2013. Static analysis for probabilistic programs: inferring whole program properties from finitely many paths. In *PLDI*. ACM, 447–458. doi:10.1145/2491956.2462179
[52] Philipp Schröer, Kevin Batz, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. 2023. A Deductive Verification Infrastructure for Probabilistic Programs. *Proc. ACM Program. Lang.* 7, OOPSLA2 (2023), 2052–2082. doi:10.1145/3622870
[53] Marcin Szymczak and Joost-Pieter Katoen. 2019. Weakest Preexpectation Semantics for Bayesian Inference - Conditioning, Continuous Distributions and Divergence. In *SETSS (Lecture Notes in Computer Science, Vol. 12154)*. Springer, 44–121. doi:10.1007/978-3-030-55089-9_3
[54] Jan-Willem van de Meent, Brooks Paige, Hongseok Yang, and Frank Wood. 2018. An Introduction to Probabilistic Programming. *CoRR* abs/1809.10756 (2018). arXiv:1809.10756 http://arxiv.org/abs/1809.10756
[55] Di Wang, Jan Hoffmann, and Thomas W. Reps. 2021. Central moment analysis for cost accumulators in probabilistic programs. In *PLDI*. ACM, 559–573. doi:10.1145/3453483.3454062
[56] Jinyi Wang, Yican Sun, Hongfei Fu, Krishnendu Chatterjee, and Amir Kafshdar Goharshady. 2021. Quantitative analysis of assertion violations in probabilistic programs. In *PLDI*. ACM, 1171–1186. doi:10.1145/3453483.3454102
[57] Peixin Wang, Hongfei Fu, Amir Kafshdar Goharshady, Krishnendu Chatterjee, Xudong Qin, and Wenjun Shi. 2019. Cost analysis of nondeterministic probabilistic programs. In *PLDI*. ACM, 204–220. doi:10.1145/3314221.3314581
[58] Peixin Wang, Tengshun Yang, Hongfei Fu, Guanyan Li, and C-H Luke Ong. 2024. Static Posterior Inference of Bayesian Probabilistic Programming via Polynomial Solving. *Proceedings of the ACM on Programming Languages* 8, PLDI (2024), 1361–1386. doi:10.1145/3656432
[59] Glynn Winskel. 1993. *The formal semantics of programming languages - an introduction*. MIT Press.

Received 2024-10-16; accepted 2025-02-18