1 Introduction and Motivation

Due to the probabilistic and uncertain nature of numerous real-world processes such as financial markets, robotic environments and randomized algorithms, statistical modeling and machine learning are two widely used tools in various areas of applied engineering. A statistical model can be used twofold: (a) to model the data generation process and evaluate the probability of possible outcomes, and (b) to perform (Bayesian) inference on the model using observed output data and automatically learn/adapt the model parameters.

Traditionally, such models have been created manually and evaluated by tailored mathematical approaches based on e.g., statistics, logic and algebra, which are tedious and require significant mathematical expertise. To overcome these limitations, Probabilistic Programming Languages (PPLs) allow the user to specify the probabilistic model in a designated programming language and provide means to run fully-automated inference procedures, significantly facilitating the deployment of such models [1, 2]. Moreover, probabilistic programming can be viewed as an attempt to provide machine learning procedures with domain knowledge by giving insight to how data is generated.

Example 1. To illustrate the first use-case (a), consider the biased, one-dimensional random walk shown in Figure 1a. Estimating the posterior density is non-trivial, especially since the step width is not symmetrical and the direction is chosen with some bias. A sensible query to an inference system would be “How is the probability distributed over possible values of $x$?”.
\[
x \leftarrow 0
\]
\[\text{for } i \text{ in } 1 \ldots 10 \text{ do}
\]
\[\text{direction } \leftarrow \text{Bernoulli}(0.35)
\]
\[\text{if } \text{direction } = 0 \text{ then}
\]
\[x \leftarrow x + \text{Uniform}(1; 5)
\]
\[\text{else}
\]
\[x \leftarrow x - \text{Uniform}(2; 7)
\]
\[\text{end if}
\]
\[\text{end for}
\]

(a) A biased random walk

\[\textit{lt: Expected Lifetime}
\]
\[\textit{pSmokers: smoking prevalence}
\]
\[\textit{pDisease: prevalence of severe diseases}
\]

The machine-learning view of probabilistic programming (b), can be illustrated by the problem of estimating the average life expectancy in a country based on a number of factors and given some training data\cite{1}. One could imagine to simply train some standard model such as a neuronal network, but this would hardly allow the user to include domain knowledge. In contrast, in Figure 1B, we show how one may model the problem in a PPL. By presenting evidence, the inference engine can perform Bayesian inference on the parameters \( M, V, D, S \) and \( I \) and update the belief on what the ‘real’ values are. After performing inference, the task of average life expectancy estimation (even in the presence of incomplete data) can be reduced to evaluating the program on some input data, e.g. “What is the average life expectancy in a developed country where 20 percent of the population are smokers?”.

Note that the preceding programs are not necessarily executed to answer these questions, a probabilistic program is a merely a way to formulate the model.

It is evident that full-featured probabilistic programming languages are strictly more powerful than traditional programming languages, and thus it does not come as a surprise that certain properties are undecidable (such as termination \cite{3}) or very hard to answer (such as inference \cite{1, 5, 6}). Therefore, most probabilistic programming languages have been carefully designed to syntactically only allow programs that actually can be evaluated within the chosen model (e.g., to only allow discrete variables, restriction to linear arithmetic or restrictions on distributions). Unsurprisingly, non-statically bounded loops are notoriously hard, as these cannot be unrolled to create a

\begin{verbatim}
cnt ← 1
while Bernoulli(p) = 1 do
cnt ← \text{cnt} + 1
end while
assert(cnt < 1000)
\end{verbatim}

Figure 2: A counter-example to loop unrolling.

loop-free program and are thus absent in virtually all probabilistic programming languages. The solution in classical programming languages is to unroll the loop a fixed number of times and assert that this bound is not violated, e.g., when performing bounded model checking. However, in the probabilistic setting this is not sound, consider the following counter-example:

**Example 2.** Suppose we iteratively flip a biased coin until it lands heads and count the number of throws, as in the model in Figure 2. Further, we would like to assert a certain safety property.

When unrolling the loop a finite number $N$ of times, the variable \textit{cnt} will be bounded by $N$. As such, any technique based on unrolling will report a distribution on \textit{cnt} that has a support of $\{1, \ldots, N+1\}$ and sampling-based techniques may even report a smaller set, since probabilities are exponentially decreasing in $N$. However, it is evident that the program of Figure 2 has infinitely many paths, and hence the distribution of \textit{cnt} has a strictly positive probability for every positive integer. In fact, $\textit{cnt} \sim \text{Geo}(p)$ and thus has infinite support. As such, a bounded analysis may incorrectly assert the safety property if $N$ is chosen inappropriately, showing that such an approach is not sound in general.

2 Aim of the Thesis

The proposed thesis aims to alleviate some of the mentioned issues and conduct research in the area of \textit{exact inference techniques for probabilistic programs}, to highlight their strengths and show their limitations. Special focus will be placed on the ability to handle looping constructs that add considerable complexity when performing inference. The planned areas for research are as follows.

**Theory.** On the theory side, the main goal is to enable exact inference for classes of loopy programs. This problem can be reduced to the task of describing the distribution of the loop and reasoning over the remaining
program when replacing the program with its distribution. As such, we propose the following topics to be handled:

1. **Classification of loops that can be analyzed.**
   We aim to classify loops according to their properties, which allow us to determine whether we can automatically and efficiently compute their distribution. Discriminating factors might be properties such as finite-stateness, discreteness, continuity or to restrict randomization to loop bounds.

2. **Exploration of methods that can be utilized.**
   After providing a loop classification, we will investigate possibilities to extract the distribution from loops where this is feasible. Different classifications will require different approaches and theoretical foundations. One may use probabilistic model checkers such as Storm \[7\] to analyze finite-state loops, or utilize recurrence equation-based methodologies \[8\] to characterize certain infinite-state programs. We envision to use a combination of such approaches in the thesis.

3. **Extraction of joint probability distributions from moments.** It has been shown that certain classes of potentially unbounded loops can be described by their moments \[8\]. For variables with finite domain, a finite number of moments are sufficient to describe their marginal distribution. However, to perform exact Bayesian inference, it is necessary to characterize the joint distribution. Now, note that any expected value can be written as a linear equation involving the joint probability distribution. We aim to characterize conditions that guarantee that the joint distribution can be extracted from moments over program variables, and thus enable inference for certain loops.

**Empirical.** As shown in the preceding parts of the proposal, even if general loops are out-of-reach, specific loops can still be managed. After the theoretical investigations, the work aims to evaluate the applicability of theoretical advances in practical tools and existing techniques. Depending on the restriction placed on loops to be analyzable, existing works that may profit from our advances are Polar \[8\], Dice \[4\], SPPL \[9\], PSI \[10\]; calculus-based methods such as \[11\ \[12\]; and formal power series-based methods \[13\]. We will investigate the applicability of our approach to these frameworks and show which novel programs can be handled.
3 Methods

To achieve the goal described in the previous section, the following steps will be performed:

1. Literature Review.
   It will be necessary to acquire the necessary foundational knowledge and explore existing approaches in probabilistic programming literature, program analysis and on probabilistic model checking.

2. Comparison of Different Approaches.
   We will compare existing approaches from the literature, explore their strengths and weaknesses, and describe which restrictions are placed on programs to be handled efficiently.

3. Theoretical Contributions.
   As described in the previous section, we will investigate methods to characterize probabilistic loops, classify them and, if applicable, present ways to compute their probability distribution.

   After the previous step, we will investigate how existing approaches can be supplemented with our findings.

4 State of the Art

We will now first provide a general overview of probabilistic programming, and then Section 4.1 investigates existing probabilistic loop analysis techniques.

There are several approaches to the task of performing inference in probabilistic programs. The most popular approaches are approximate, such as importance sampling [2, 14], various flavors of Markov chain Monte Carlo (MCMC) [2, 14, 15], and variational inference [16, 17, 14, 2, 18, 19], which in principle also can handle looping constructs. There is no lack of tools implementing such inference algorithms, such as (Open)BUGS and its successor JAGS [20], Church [21], Edward [22], Infer.NET [23], PyMC [24], Stan [25], and WebPPL [26].

However, these methods provide an approximation of the posterior distribution and typically do not provide any accuracy guarantees. Moreover, modern, well-performing approaches such as Hamiltonian Monte Carlo
or Automatic Differentiation Variational Inference \cite{18} require the model to be almost-everywhere differentiable, which prohibits the analysis of discrete models.

In contrast, exact techniques provide an analytic expression that describes the posterior distribution or at least provide bounds. The approaches here are plenty as well, including path enumeration \cite{28, 29, 30}, techniques based on knowledge compilation \cite{31, 32}, such as (weighted) model counting for discrete programs \cite{33, 34, 35, 36, 4}, and the related weighted model integration for continuous and hybrid programs \cite{6, 37, 38, 39, 10, 41, 9}. The spectrum of approaches is further broadened by symbolic, integration-based approaches such as PSI \cite{10}, its higher-order variant \( \lambda \text{PSI} \) \cite{42}, Hakaru \cite{43} and hybrid approaches \cite{40}.

Moreover, Dijkstra’s deterministic weakest-precondition style reasoning has been extended to reason about expected values \cite{44, 11, 45, 12}. This allows a characterization of expected values over random variables by means of a logic calculus based on the equivalents of pre- and postcondition. Similarly, there has been recent work to perform forward reasoning based on formal power series \cite{13}.

In between approximate and exact approaches, there are also methods that aim to approximate the posterior distribution by characterizing distributions over finite sets of intervals \cite{46, 47}, providing quantitative estimates and convergence guarantees.

There is also the related and mature field of probabilistic model checking (PMC), which answers quantitative queries over probabilistic models such as Markov Chains and Markov Decisions processes posed in an LTL/CTL style query language \cite{48}. Even though the aims of PMCs and PPLs are different, there are relation that can be exploited to either perform inference by model checking \cite{49} or vice-versa \cite{50, 51}, since semantics of many PPLs can be formulated as a Markov Chain.

Even though there is a plethora of existing work, the vast majority is syntactically restricted to statically bounded loops to allow unrolling. In the following section, we highlight the few existing approaches, their strengths and their shortcomings.

4.1 The State of Probabilistic Loops Analysis

In general, there is no approach that can handle arbitrary probabilistic loops, since the problem is already undecidable for deterministic loops.

Sampling based approaches can theoretically handle loops, but the probability of long-running executions may be close to zero (see Example 2), hence MCMC techniques may require very long time to explore these regions in the
probability space. The works of [28, 29, 47] in some sense allow approximate inference on loops, by increasing the number of iterations to analyze, akin to bounded model checking, but cannot provide an exact characterization. In contrast, the work of [30], proposes to peel off a single iteration of the loop until a fixpoint is (approximately) reached, however no convergence guarantees exist. Finally, the work of [52] and [53] consider recursive programs over finite domains, which allows them to build finite equation systems over method calls. However, the resulting equation systems may be very large and are not guaranteed to be efficiently solvable, let alone exactly.

One approach that focuses on potentially unbounded probabilistic loops is presented in [8], where a program is considered as describing a recurrence relation over the moments of program variables. If these recurrence relations can be solved (i.e., if the recurrences are linear), then all moments can be computed. However, severe restrictions are based on variables influencing the control flow and no observations are possible. Also, for variables over infinite domains, the extraction of a closed-form distribution may require infinitely many moments.

The already mentioned calculus-based approaches such as [44, 45, 11, 12, 13] can theoretically perform inference on arbitrary probabilistic programs, however they are severely limited by the necessity to compute fixed points or come up with appropriate loop invariants, which is unfortunately undecidable [3].

6 Conclusion on State of the Art

The existing body of work is either restricted to statically bounded loops, is not able to perform exact inference on a large class of loops, or depends on user-provided invariants. With this thesis work, we aim to close some of these holes and provide exact inference methods for special classes of probabilistic loops.

5 Relevance to the Curricula of Computer Engineering

The area of probabilistic programming bridges the disciplines of machine learning, statistics, programming languages and program analysis. As such, it builds on various skills taught in the computer engineering bachelor and master programs. The most relevant courses that allow me to conduct this research and those that are related to this field of research are briefly summarized below:
• 199.100 Verification of Probabilistic Programs
• 184.741 Program and System Verification
• 181.144 Computer-Aided Verification
• 184.774 Automated Deduction
• 107.A04 Probability Theory and Stochastic Processes for Computer Science
• 104.271 Discrete Mathematics
• 185.291 Formal Methods in Computer Science
• 185.A48 Compilers
• 184.703 Program Analysis

Moreover, due to the wide application of probabilistic models and inference techniques in cyber-physical systems, research in these fields will also pave the way for more advanced paradigms and more powerful cyber-physical systems, touching another fundamental area of computer engineering. As such, this thesis is related to diverse fields of computer engineering and investigates state-of-the-art research questions.

References


[43] Praveen Narayanan, Jacques Carette, Wren Romano, Chung-chieh Shan, and Robert Zinkov. Probabilistic inference by program transformation in hakaru (system description). In Oleg Kiselyov and Andy


