Analysis of Bayesian Networks via Prob-Solvable Loops

Ezio Bartocci, Laura Kovács, and Miroslav Stanković(✉)

TU Wien, Vienna, Austria
{Ezio.Bartocci,Laura.Kovacs,Miroslav.Stankovic}@tuwien.ac.at

Abstract. Prob-solvable loops are probabilistic programs with polynomial assignments over random variables and parametrised distributions, for which the full automation of moment-based invariant generation is decidable. In this paper we extend Prob-solvable loops with new features essential for encoding Bayesian networks (BNs). We show that various BNs, such as discrete, Gaussian, conditional linear Gaussian and dynamic BNs, can be naturally encoded as Prob-solvable loops. Thanks to these encodings, we can automatically solve several BN related problems, including exact inference, sensitivity analysis, filtering and computing the expected number of rejecting samples in sampling-based procedures. We evaluate our work on a number of BN benchmarks, using automated invariant generation within Prob-solvable loop analysis.

1 Introduction

Bayesian networks (BNs) are well-established probabilistic models widely adopted to represent complex systems and to reason about their intrinsic uncertain knowledge. BNs are graphically depicted as directed acyclic graphs (DAGs) whose nodes represent random variables and edges capture conditional dependencies. Since the seminal work of [40], BNs have been extensively employed in several application domains including machine learning [22], speech recognition [44], sports betting [12], gene regulatory networks [20], diagnosis of diseases [24] and finance [39]. Part of their success is due to the inherited Bayesian inference framework enabling the prediction about the likelihood that one of several known causes is responsible for the evidence of an observed event.

Figure 1 illustrates a simple BN with two events that can cause the grass (G) to be wet: the rain (R) or an active sprinkler (S). When it rains the sprinkler is usually not active, so the rain has a direct effect on the use of the sprinkler. This dependency is provided by a *conditional probability table*, in short CPT, associated to the sprinkler random variable S. A CPT lists, for each possible combination of values of the parents' variables (one for each row of the table), the corresponding probability for the child's variable to have a certain discrete value (one for each column of the table). The random variables G, R, S of Fig. 1 are, for example, binary random variables with Bernoulli conditional distributions. However, in general BNs allow arbitrary types for their random variables and their conditional distributions.

This research was supported by the Vienna Science and Technology Fund (WWTF) under grant ICT19-018 (ProbInG), the ERC Starting Grant 2014 SYMCAR 639270 and the Austrian FWF project W1255-N23.

V. K. I. Pun et al. (Eds.): ICTAC 2020, LNCS 12545, pp. 221–241, 2020.
https://doi.org/10.1007/978-3-030-64276-1_12

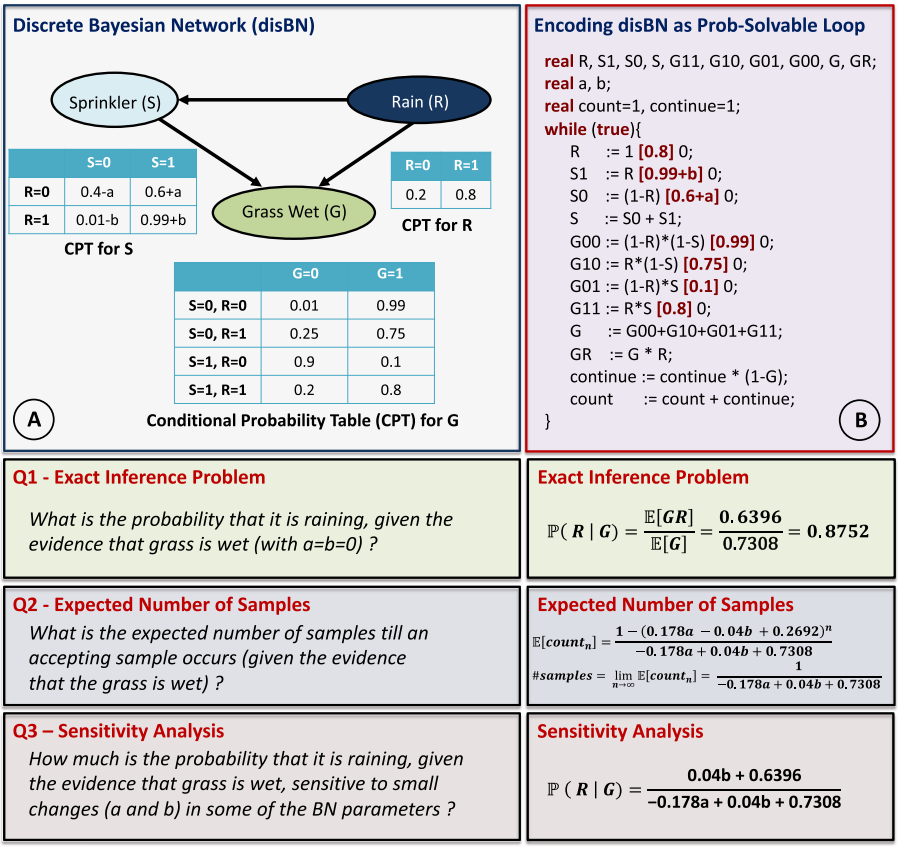

Fig. 1. Solving probabilistic inference, the expected number of samples and the sensitivity analysis for a discrete BN (disBN), by encoding the disBN as a Prob-solvable loop and computing automatically moment-based invariants (MBIs).

Probabilistic Inference. Given the BN in Fig. 1, the following can be asked:

Q1 - What is the probability that it is raining, given that the grass is wet?

The answer to this question can be obtained by solving a *probabilistic inference*, that is the problem to optimally estimate the probability of an event given an observed evidence. The works in [13,14] show that both *exact* and *approximated* (up to an arbitrary precision) methods to solve probabilistic inference are NP-hard.

How Many Samples? Approximating solutions for probabilistic inferences can be done using Monte Carlo sampling techniques [28,43]. For example, *rejection sampling* is one of the fundamental techniques for sampling from the joint (unconditional) distribution of the BN: a sample is accepted when it complies with the evidence, otherwise is rejected. Unfortunately, this method may require many samples before obtaining the

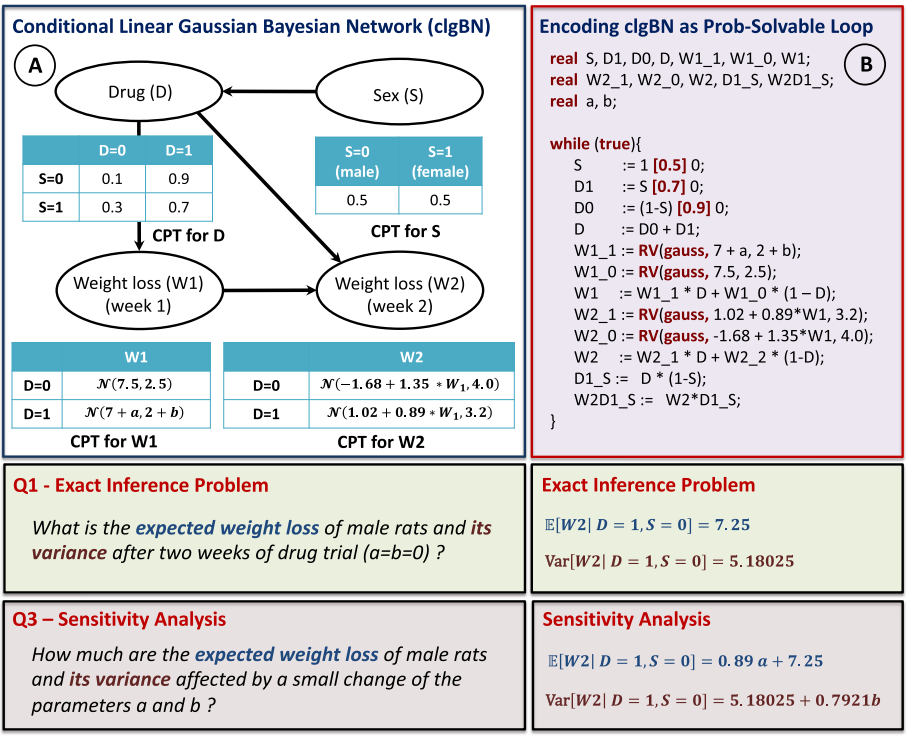

Fig. 2. Solving probabilistic inference and sensitivity analysis in a conditional linear Gaussian BN (clgBN), by encoding the clgBN as a Prob-solvable loop and computing MBIs.

first accepted samples, while most of the samples may be wasted simply because they do not satisfy the observations. Thus, an interesting question, investigated also in [7], is:

Q2 - What is the expected number of samples until an accepting sample occurs?

Sensitivity Analysis. As BN parameters are often provided manually or estimated from (incomplete) data, they are most likely to be imprecise or wrong. For example, in Fig. 1 the CPT of the random variable S contains imprecise symbolic parameters a and b. In this case, *sensitivity analysis* aims to answer the following question:

Q3 - How much does a small change in BN parameters affect probabilistic inference?

Probabilistic Programs. Probabilistic programs (PPs) provide a unifying framework to both encode probabilistic graphical models, such as BNs, and to implement sophisticated inference algorithms and decision making routines that can operate in real-world applications [21]. Probabilistic programming languages, such as [1,8,42] include native constructs for sampling distribution, enabling the programmer to mix deterministic and stochastic elements. However, the automated analysis of PPs implemented in these languages is still at its infancy. For example, one of the main challenges in the analysis of PPs comes with computing invariant properties summarizing PP loops. While full automation of invariant generation for PPs is in general undecidable, recent results identify classes of PPs for which invariants can automatically be computed [4,7]. In [4], we introduced a method to automatically generate moment-based invariants of so-called Prob-solvable loops with polynomial assignments over random variables and parametrised distributions. Doing so, we exploit statistical properties to eliminate probabilistic choices and turn random updates into recurrence relations over higher-order moments of program variables.

Analysis of BNs as Prob-solvable Loops. In this paper we extend Prob-solvable loops with new features essential for encoding BNs and for solving several kind of BN analysis via invariant generation over higher-order statistical moments of Prob-solvable loop variables. Figure 1(B) shows a Prob-solvable loop encoding the probabilistic behaviour of the discrete BN (disBN) illustrated in Fig. 1(A). The Prob-solvable loop of Fig. 1(B) requires one variable for each disBN node, one variable for each row of the CPT tables, one variable for each unknown parameter and some extra variables that depend on the particular BN analysis. For example, to solve exact probabilistic inference and sensitivity analysis, we require an extra variable to store the product of the random variables G and R. On the other hand, to compute the expected number of samples until an accepting sample occurs, we would need other two auxiliary variables $count$ and $continue$. Each row of each CPT is encoded as a probabilistic assignment in the Prob-solvable loop. Our approach generates moment-based invariants as quantitative invariants over higher-order moments to solve the three questions (Q1-Q3) of Fig. 1. The required Prob-solvable loop analysis requires however additional steps (e.g., calculating a limit) that are not yet supported in [4]. Moreover, while the Prob-solvable programming model of [4] can model the probabilistic behavior of disBNs, it cannot model other BN variants, such as BNs with Gaussian conditional dependencies as in Fig. 2(A). We therefore extend Prob-solvable loops with new features supporting Gaussian and uniform random variables depending on other random variables (Sect. 3) and show that these extensions allow us to solve BN problems via Prob-solvable loop reasoning (Sects. 4 and 5).

Our Contributions. (i) We prove that our extended model of Prob-solvable loops admits a decision procedure for computing moment-based invariants (Sect. 3). (ii) We provide a sound encoding of BNs as Prob-solvable loops, in particular addressing discrete BNs (disBNs), Gaussian BNs (gBNs), conditional linear Gaussian BNs (clgBNs) and dynamic BN (dynBNs) (Sect. 4). (iii) We formalize several BN problems as moment-based invariant generation tasks in Prob-solvable loops (Sect. 5). (iv) We implemented our approach in the MORA tool [6] and evaluated it on a number of examples, fully automating BN analysis via Prob-solvable loop reasoning (Sect. 5.4). Complete proofs

for the theorems in this work as well as further details are available in the extended version [5].

2 Preliminaries

We first introduce basic notions from statistics in order to reason about probabilistic systems (Sect. 2.1), and refer to [35] for further details. We then adopt basic definitions and properties of Bayesian Networks (BNs) from [40] to our setting (Sect. 2.2). Throughout this paper, let $\mathbb{N}, \mathbb{R}$ denote the set of natural and real numbers, respectively.

2.1 Probability Space and Statistical Moments

We denote random variables by capital letters $X, Y, S, R, \ldots$ and program variables by small letters $x, y, \ldots$, all possibly with indices.

Definition 1 (Probability Space). *A* probability space *is a triple (Ω, F, P), where $\Omega \neq \emptyset$ is a sample space representing the set of outcomes, $F \subset 2^{\Omega}$ is a σ-algebra representing the set of events, and $P : F \to [0, 1]$ is a probability measure with $P(\Omega) = 1$.*

We now define random variables, together with their higher-order statistical moments, in order to reason about probabilistic properties.

Definition 2 (Random Variable). *A* random variable $X : \Omega \to \mathbb{R}$ *is a measurable function from a set Ω of possible outcomes to $\mathbb{R}$. If Ω is countable, the random variable X is called* discrete*; otherwise, X is* continuous*.*

In particular, in this paper we will be interested in the following random variables:

- Random variable X with Bernoulli distribution $Bern(p)$, given by probability p, where $\Omega = \{0, 1\}$ and $X(0) = 1 - p$ and $X(1) = p$;
- Random variable X with Gaussian distribution $\mathcal{G}(\mu, \sigma^2)$, given by mean μ and variance σ^2, where $\Omega = \mathbb{R}$ and probability density function $f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$.
- Random variable X with uniform distribution $\mathcal{U}(a, b)$, given by limits a, b where $\Omega = \mathbb{R}$ and probability density function $f(x) = \begin{cases} \frac{1}{b-a} & \text{for } z \in [a, b] \\ 0 & \text{for } z \notin [a, b] \end{cases}$.

Example 1. The variables R, S, G of the BN from Fig. 1(A) are Bernoulli random variables, with variable R given by probability 0.8. Figure 2(A) features two Bernoulli random variables S and D as well as two real-valued random variables $W1$ and $W2$ drawn from a Gaussian distribution. Note that the parameters of the Gaussian distribution of $W1$ depend on the values of D, whereas for $W2$ they depend on D and $W1$.

For a given random variable X we will denote by $\Omega(X)$ the sample space of X. When working with a random variable X, the most common statistical moment of X to consider is its first-order moment, called the expected value of X.

Definition 3 (Expected Value). *An* expected value of a random variable X *defined on a probability space* (Ω, F, P) *is the Lebesgue integral:* $\mathbb{E}[X] = \int_{\Omega} X \cdot dP$. *In the special case* when Ω is discrete*, that is the outcomes are* $x_1, \ldots, x_n$ *with corresponding probabilities* $p_1, \ldots, p_n$ *and* $n \in \mathbb{N}$*, we have* $\mathbb{E}[X] = \sum_{i=1}^{n} x_i \cdot p_i$. *The expected value of* X *is often also referred to as the* mean *or* μ *of* X.

The key ingredient in analyzing and deriving properties of a random variable X is the so-called characteristic function of X.

Definition 4 (Characteristic Function). *The* characteristic function *of a random variable* X*, denoted by* $\phi_X(t)$*, is the Fourier transform of its probability density function (pdf). That is,* $\phi_X(t) = \mathbb{E}[e^{itX}]$*, with a bijective relation between probability distributions and characteristic functions.*

The characteristic function $\phi_X(t)$ of a random variable X captures the value distribution induced by X. In particular, the characteristic function $\phi_X(t)$ of X enables inferring properties about distributions given by weighted sums of X and other random variables, and thus also about statistical higher-order moments of X.

Definition 5 (Higher-Order Moments). *Let* X *be a random variable,* $c \in \mathbb{R}$ *and* $k \in \mathbb{N}$*. We write* $Mom_k[X]$ *to denote the* kth raw moment of X*, which is defined as:*

$$Mom_k[X] = \mathbb{E}[X^k]. \tag{1}$$

Remark 1. For a Bernoulli random variable X with parameter probability p, all moments of X coincide with its probability. Thus, $Mom_k[X] = P(X = 1) = p$.

Example 2. Figure 1 lists the first-order moment $\mathbb{E}[G]$ of G, as well as the first-order moment $\mathbb{E}[GR]$ of the mixed random variable GR. The second-order moment of $W2$ is used to compute the variance $Var(W2)$ of $W2$ in Fig. 2.

2.2 Probabilistic Graphical Models as Bayesian Networks

Definition 6 (Bayesian Network (BN)). *A* Bayesian network (BN) *is a directed acyclic graph (DAG) in which each node is a discrete/continuous random variable. A set of directed links or arrows connects pairs of BN nodes. If there is an arrow from a BN node* Y *to a node* X*, then* Y *is said to be a* parent *of* X.

For a random variable/node X in a BN, we write $Par(X)$ to denote the set of parents of X in the BN. Each BN node X has a *conditional probability distribution* $P(X|Par(X))$ that quantifies the effect of the parents $Par(X)$ on the node X. Dependencies in a BN can be given in different forms and we overview the most common ones. For a discrete variable X, dependencies are often given by a conditional probability table, by listing all possible values of parent variables from $Par(X)$ and the corresponding values of X. In the case of a continuous variable X, dependencies can be specified using Gaussian distributions. Another common dependency in a BN is a deterministic one, where value of a node X is determined by values of its parents from $Par(X)$; that is, a binary variable can be true iff all its (binary) parents are true, or if one of its parents is true. We overview below BN variants, studied further in Sect. 4.

Definition 7 (Variants of Bayesian Networks).

- *A* discrete Bayesian Network (disBN) *is a BN whose variables are discrete.*
- *A* Gaussian Bayesian Network (gBN) *is a BN whose dependencies are given by Gaussian distributions in which, for any BN node X, we have $P(X|Par(X)) = \mathcal{G}(\mu_X, \sigma_X^2)$, with $\mu_X = \alpha_X + \sum_{k=1}^{m_X} \beta_{X,k} Y_{X,k}$,, $Par(X) = \{Y_1, \cdots, Y_{m_X}\}$ and σ_X^2 is fixed.*
- *A* conditional linear Gaussian Bayesian Network (clgBN) *is a BN in which (i) continuous nodes X cannot be parents of discrete nodes Y; (ii) the local distribution of each discrete node Y is a conditional probability table (CPT); (iii) the local distribution of each continuous node X is a set of Gaussian distributions, one for each configuration of the discrete parents Y, with the continuous parents acting as regressors.*
- *A* dynamic Bayesian Network (dynBN) *is a structured BN consisting of a series of time slices that represent the state of all the BN nodes X at a certain time t. For each time-slice, a dependency structure between the variables X at that time is defined by intra-time-slice edges. Additionally, there are edges between variables from different slices—inter-time-slice edges, with their directions following the direction of time.*

Example 3. A disBN encoding the probabilistic model of the grass getting wet is shown in Fig. 1(A). Figure 2(A) lists a clgBN, describing a weight loss process in a drug trial performed on rats. The (Gaussian) random variables encoding weight loss for weeks 1 and 2 are respectively denoted with $W1$ and $W2$.

3 Programming Model: Extending Prob-solvable Loops

We introduce our programming model extending the class of *Prob-solvable loops* [4], allowing us to encode and analyze BN properties in Sect. 4. In particular, we extend [4] to support Prob-solvable loops with symbolic random variables encoding dependencies among other (random) variables, where Gaussian and uniform random variables can linearly depend on other program variables, encoding this way common BN dependencies. To this end, we consider probabilistic while-programs as introduced in [30,37] and restrict this class of programs to probabilistic programs with polynomial updates among random variables. We write $x := e_1[p]e_2$ to denote that the probability of the program variable x being updated with expression e_1 is $p \in [0, 1]$, whereas the probability of x being updated with expression e_2 is $1 - p$. In the sequel, whenever we refer to a Prob-solvable loop/program, we mean a program as defined below.

Definition 8 (Prob-solvable Loop). *Let $m \in \mathbb{N}$ and $x_1, \ldots, x_m$ denote real-valued program variables. A* Prob-solvable loop *with variables $x_1, \ldots, x_m$ is a probabilistic program of the form*

$$I; \texttt{while(true)}\{U\}, \tag{2}$$

where:

- *(Initialization) I is a sequence of initial assignments over $x_1, \ldots, x_m$. That is, I is an assignments sequence $x_1 := c_1; x_2 := c_2; \ldots; x_m := c_m$, with $c_i \in \mathbb{R}$ representing a number drawn from a known distribution [1] - in particular, c_i can be a real constant.*

[1] A known distribution is a distribution with known and computable moments.

Algorithm 1. Moment-Based Invariants (MBIs) of Prob-solvable Loops

Input: Prob-solvable loop $\mathcal{P}$ with variables $\{x_1, \ldots, x_m\}$, and $k \geq 1$
Output: MBIs of $\mathcal{P}$ of degree k
Assumptions: $n \in \mathbb{N}$ is an arbitrary loop iteration of $\mathcal{P}$

1: Extract moment-based recurrence relations of $\mathcal{P}$, for $i = 1, \ldots, m$:

$$\mathbb{E}[x_i(n+1)] = p_i \cdot \mathbb{E}\big[a_i x_i(n) + P_i(x_1(n), \ldots, x_{i-1}(n))\big] \\ +(1-p_i) \cdot \mathbb{E}\big[b_i x_i(n) + Q_i(x_1(n), \ldots, x_{i-1}(n))\big].$$

2: $MBRecs = \{\mathbb{E}[x_i(n+1)] \mid i = 1, \ldots, m\}$ ▷ initial set of moment-based recurrences
3: $S := \{x_1^k, \ldots, x_m^k\}$ ▷ initial set of monomials of $\mathbb{E}$-variables
4: **while** $S \neq \emptyset$ **do**
5: $M := \prod_{i=1}^{m} x_i^{\alpha_i} \in S$, where $\alpha_i \in \mathbb{N}$
6: $S := S \setminus \{M\}$
7: $M' = M[x_i^{\alpha_i} \leftarrow upd_i]$, for each $i = m, \ldots, 1$ ▷ replace each $x_i^{\alpha_i}$ in M with upd_i

where upd_i denotes:
$p_i \cdot \big(a_i x_i + P_i(x_1, \ldots, x_{i-1})\big)^{\alpha_i} + (1-p_i) \cdot \big(b_i x_i + Q_i(x_1, \ldots, x_{i-1})\big)^{\alpha_i}$

8: Rewrite M' as $M' = \sum N_j$ for monomials N_j over $x_1, \ldots, x_m$
9: **Simplify moment-based recurrence** $\mathbb{E}[M(n+1)] = \mathbb{E}[\sum N_j]$ **using (5)-(6)**
▷ $M(n+1)$ denotes $\prod_{i=1}^{m} x_i(n+1)^{\alpha_i}$
10: $MBRecs = MBRecs \cup \{\mathbb{E}[M(n+1)]\}$
▷ add $\mathbb{E}[M(n+1)]$ to the set of moment-based recurrences
11: **for** each monomial N_j in M **do**
12: **if** $\mathbb{E}[N_j] \notin MBRecs$ **then** ▷ there is no moment-based recurrence for N_j
13: $S = S \cup \{N_j\}$ ▷ add N_j to S
14: **end while**
15: $MBI = \{\mathbb{E}[x_i(n)^k] - f_{x_i,k}(n) = 0 \mid i = 1, \ldots, m\}$
▷ $f_{x_i,k}(n)$ is the closed form solution of $E[x_i^k]$
16: **return** MBIs of $\mathcal{P}$ for the kth moments of $x_1, \ldots, x_m$

– *(Update) U denotes a sequence of m random updates, each update of the form:*

$$x_i := a_i x_i + P_i(x_1, \ldots, x_{i-1}) \; [p_i] \; b_i x_i + Q_i(x_1, \ldots, x_{i-1}), \tag{3}$$

or, in case of a deterministic assignment,

$$x_i := a_i x_i + P_i(x_1, \ldots, x_{i-1}), \tag{4}$$

where $a_i, b_i \in \mathbb{R}$ are constants and $P_i, Q_i \in \mathbb{R}[x_1, \ldots, x_{i-1}]$ are polynomials over program variables $x_1, \ldots, x_{i-1}$.
– *(Dependencies) The coefficients a_i, b_i and the coefficients of P_i and Q_i in the variable assignments (3)-(4) of x_i can be drawn from a random distribution as long as the moments of this distribution are known and either they are (i) Gaussian or uniform distributions linearly depending on x_i and other random variables x_j with $j \neq i$; or (ii) other known distributions independent from $x_1, \ldots, x_m$.*

Note that Prob-solvable loops support parametrised distributions, for example one may have the uniform distribution $\mathcal{U}(d_1, d_2)$ with arbitrary $d_1 < d_2 \in \mathbb{R}$ symbolic

constants. Similarly, the probabilities p_i in the probabilistic updates (3) can be symbolic constants. The restriction on random variable dependencies from Definition 8 extends [4] by allowing parameters of Gaussian and uniform random variables x_i in Prob-solvable loop to be specified using previously updated program variables x_j and to depend on x_i linearly. In Theorem 1 we prove that this extension maintains the existence and computability of higher-order statistical moments of Prob-solvable loops, allowing us to derive all *moment-based invariants* of Prob-solvable loops of degree $k \geq 1$.

Definition 9 (Moment-Based Invariants (MBIs)). *Let $\mathcal{P}$ be a Prob-solvable loop and $n \in \mathbb{N}$ denote an arbitrary loop iteration of $\mathcal{P}$. Consider $k \in \mathbb{N}$ with $k \neq 0$. A* moment-based invariant (MBI) of degree k over x_i *of $\mathcal{P}$ is $\mathbb{E}[x_i(n)^k] = f_{x_i,k}(n)$, where $f_{x_i,k} : \mathbb{N} \to \mathbb{R}$ of n is a closed form expression for the kth (raw) higher-order moment of x_i, such that $f_{x_i,k}(b)$ depends only on n and the initial variable values of $\mathcal{P}$.*

In what follows, we consider an arbitrary Prob-solvable loop $\mathcal{P}$ and formalize our results relative to $\mathcal{P}$. Further, we reserve $n \in \mathbb{N}$ to denote an arbitrary loop iteration of $\mathcal{P}$. Note that MBIs of $\mathcal{P}$ yield functional representations of the kth higher-order moments of loop variables x_i at n. Hence, the MBIs $\mathbb{E}[x_i(n)^k] = f_{x_i,k}(n)$ are valid and invariant. In Algorithm 1 we show that MBIs of Prob-solvable loops can always be computed. As in [4], the main ingredient of Algorithm 1 are so-called $\mathbb{E}$-*variables* for capturing expected values and other higher-order moments of loop variables of $\mathcal{P}$.

Definition 10 ($\mathbb{E}$-variables of Prob-solvable Loops [4]). *An* $\mathbb{E}$-variable *of $\mathcal{P}$ is an expected value of a monomial over the random variables x_i of $\mathcal{P}$.*

Using Definition 10, in Algorithm 1 we compute $\mathbb{E}$-variables based on expected values $\mathbb{E}[x_i(n)]$ of loop variables x_i, as well as using higher-order and mixed moments of $\mathcal{P}$, such as $\mathbb{E}[x_i^k(n)]$ or $\mathbb{E}[x_i x_j(n)]$ (lines 3 and 9 of Algorithm 1). To this end, Algorithm 1 resembles the approach of [4] and extends it to handle Prob-solvable loops with dependencies among random variables drawn from Gaussian/uniform distributions (line 9 of Algorithm 1). More specifically, Algorithm 1 uses *moment-based recurrences over E-variables* from [4], describing the expected values $\mathbb{E}[x_i(n)]$ of x_i as functions of other E-variables (line 2 of Algorithm 1). To this end, note that Prob-solvable loop updates from (3)-(4) over x_i yield linear recurrences with constant coefficients over $\mathbb{E}[x_i(n)]$, by using the following simplification rules over $\mathbb{E}$-variables:

$$\begin{array}{lll} \mathbb{E}[expr_1 + expr_2] & \to \mathbb{E}[expr_1] + \mathbb{E}[expr_2] & \\ \mathbb{E}[expr_1 \cdot expr_2] & \to \mathbb{E}[expr_1] \cdot \mathbb{E}[expr_2], & \text{if } expr_1, expr_2 \text{ are independent} \\ \mathbb{E}[c \cdot expr_1] & \to c \cdot \mathbb{E}[expr_1] & \\ \mathbb{E}[c] & \to c & \\ \mathbb{E}[\mathcal{D} \cdot expr_1] & \to \mathbb{E}[\mathcal{D}] \cdot \mathbb{E}[expr_1] & \end{array} \tag{5}$$

where $c \in \mathbb{R}$ is a constant, $\mathcal{D}$ is a known independent distribution, and $expr_1$, $expr_2$ are polynomial expressions over random variables. Yet, to address our Prob-solvable loop extensions compared to [4], in addition to (5) we need to ensure that dependencies among the random variables of $\mathcal{P}$ yield also moment-based recurrences. We achieve this by introducing the following two simplification rules over random variables with Gaussian/uniform distributions:

$$\begin{array}{ll} \mathcal{G}(expr_1, \sigma^2) & \to expr_1 + \mathcal{G}(0, \sigma^2), \\ \mathcal{U}(expr_1, expr_2) & \to expr_1 + (expr_2 - expr_1)\mathcal{U}(0, 1), \end{array} \tag{6}$$

for arbitrary polynomial expressions $expr_1$, $expr_2$ over random variables. Using (6) in addition to (5), moment-based recurrences of Prob-solvable loops can always be computed as linear recurrences with constant coefficients over $\mathbb{E}$-variables (line 9 of Algorithm 1), implying thus the existence of closed form solutions of $\mathbb{E}$-variables and hence of MBIs of $\mathcal{P}$, as formalized below.

Theorem 1 (Moment-Based Invariants (MBIs) of Prob-solvable Loops). *Let $\mathcal{P}$ be a Prob-solvable loop with variables $\{x_1, \ldots, x_m\}$ and consider $k \in \mathbb{N}$ with $k \geq 1$. Algorithm 1 is sound and terminating, yielding MBIs of degree k of $\mathcal{P}$.*

Proof. We first prove correctness of the simplification rules (6), from which the soundness and termination of Algorithm 1 follows. Recall that there is a one-to-one correspondence between probability distributions and characteristic functions $\mathbb{E}[e^{itX}]$ of a random variable X. In particular, the characteristic function of a Gaussian distribution with parameters μ and σ^2 is $e^{i\mu t - \frac{1}{2}\sigma^2 t^2}$, and thus the characteristic function of $\mathcal{G}(expr_1, \sigma^2)$ is $\mathbb{E}[e^{it\mathcal{N}(expr_1, \sigma^2)}]$. Then,

$$\mathbb{E}\left[e^{it\mathcal{N}(expr_1,\sigma^2)}\right] = \mathbb{E}\left[\int e^{it\mathcal{N}(y,\sigma^2)} f(y)dy\right] = \iint e^{itx} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-y)^2}{2\sigma^2}} f(y)dxdy$$

$$= \int e^{itx} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x)^2}{2\sigma^2}} dx \int e^{ity} f(y)dy$$
$$= \mathbb{E}\left[e^{it\mathcal{N}(0,\sigma^2)}\right] \cdot \mathbb{E}\left[e^{it \cdot expr_1}\right] = \mathbb{E}\left[e^{it\left(\mathcal{N}(0,\sigma^2) + expr_1\right)}\right]$$

by change of limits for $x \in \mathbb{R}$, where f is the probability density function of the random variable $expr_1$. Note that $\mathbb{E}\left[e^{it\left(\mathcal{N}(0,\sigma^2)+expr_1\right)}\right]$ corresponds to the characteristic function of $expr_1 + \mathcal{G}(0, \sigma^2)$, and hence the simplification rule $\mathcal{G}(expr_1, \sigma^2) \rightarrow expr_1 + \mathcal{G}(0, \sigma^2)$ of (6) is correct. The correctness of the simplification rule of (6) over uniform distributions can be established in a similar way.

Further, observe that polynomial expressions remain polynomial after applications of (6) (line 9 of Algorithm 1). Once Gaussian and uniform distributions depending on loop variables are replaced using (6), we are left with independent known distributions and polynomial expressions over random variables for which (5) can further be used, as in [4]. As Algorithm 1 extends [4] only with (6) (line 9 of Algorithm 1), using results of [4], we conclude that Algorithm 1 is both sound and terminating. □

Example 4. Consider the Prob-solvable loop in Fig. 2(B). An example of $\mathbb{E}$-variable would be $\mathbb{E}[W2^2]$, for which an MBI $\mathbb{E}[W2^2] = 4.01408a^2 + 53.83168a + 4.01408b + 250.3172$ is computed using Algorithm 1.

Remark 2. While Prob-solvable loops are non-deterministic, with trivial loop guards of $true$, we note that probabilistic loops bounded by a number of iterations (such as $n := 0; while(n < 1000)\{n := n + 1\}$) can be encoded as Prob-solvable loops.

4 Encoding BNs as Prob-solvable Loops

In this section we argue that Prob-solvable loops offer a natural way for encoding BNs, enabling further BN analysis via Prob-solvable loop reasoning in Sect. 5.

4.1 Modeling Local Probabilistic Models of BNs as Prob-solvable Loop Updates

A BN is fully specified by its local dependencies. We consider common local probabilistic models and encode these models as Prob-solvable loop instances, as follows.

Deterministic Dependency. We first explore local probabilistic models specifying deterministic dependency, that is when the values of BN nodes X are determined by the values of the parent variables from $Par(X)$. For example, when X is binary-valued, such a deterministic dependency can be a Boolean expression. On the other hand, when X is continuous, deterministic dependency can be a function over $Par(X)$.

For a continuous variable X whose value is given by a polynomial $Q(Par(X))$, encoding deterministic dependencies as a Prob-solvable loop update is straightforward: we simply set $X = Q(Par(X))$.

For a discrete random variable X, let $[X = x]$ be the expression such that $[X = x] = 1$ if $X = x$ and 0 otherwise. Note that when X is binary-valued, we have $[X = 1] = X$ and $[X = 0] = 1 - X$. It follows that, in general, for a discrete variable X with possible values $x = 0, 1, \cdots, k$, we have $[X = x] = \prod_{\substack{0 \leq i < k \\ i \neq x}} \frac{X - i}{x - i}$. Furthermore, let $[(X, Y) = (x, y)] = [X = x] \cdot [Y = y]$. Then, $[(X, Y) = (x, y)] = 1$ iff $X = x \wedge Y = y$, and 0 otherwise. Finally, we write $[X \neq x]$ to denote $1 - [X = x]$. Observe that $[X = x]$ and $[X \neq x]$ are polynomials in X, providing thus a natural way to specify deterministic dependencies as updates (3)-(4) of Prob-solvable loops (see Algorithm 2).

Conditional Probability Tables – CPTs. As shown in Fig. 1(A), a common way to specify BN dependencies among discrete variables is CPTs, with each CPT line representing a possible assignment of values of a BN node X to $Par(X)$. A CPT for X can be turned into Prob-solvable loop updates, as follows.

We represent values of X with integers. For simplicity, assume that X is binary-valued. Let $Par(X) = \{Y_1, \cdots, Y_k\}$ denote the parents of X. For each line L in the CPT for X we introduce a new variable X_L. Each line L specifies values for $Par(X)$; for example, $Y_1 = y_1, \cdots, Y_k = y_k$. Let $p_L := P(X = 1|L)$ and define

$$X_L = \prod_{0 < i \leq k} [Y_i = y_i]\, [p_L]\, 0, \tag{7}$$

encoding that the value of X_L is 0 if the values of Y_i are not specified in the respective CPT line L; otherwise the value of X_L is 1 with probability p_L. We then set

$$X = \sum_{L \in CPT} X_L. \tag{8}$$

Example 5. Using (7)-(8), the disBN of Fig. 1(A) is encoded as a Prob-solvable loop in Fig. 1(B). While the parameters of S and G are not directly visible from the disBN, these parameters are given by the expected values of S and G in the Prob-solvable loop of Fig. 1(B). Note that Fig. 1(B) also features a GR variable corresponding to a Bernoulli random variable depending on G and R, such that GR is 1 iff both G and R are 1. The program variable $continue$ samples a sequence of Bernoulli random variables (one for each iteration n), while the random variable $count$ represents a geometric distribution encoding the sum of $continue$ values.

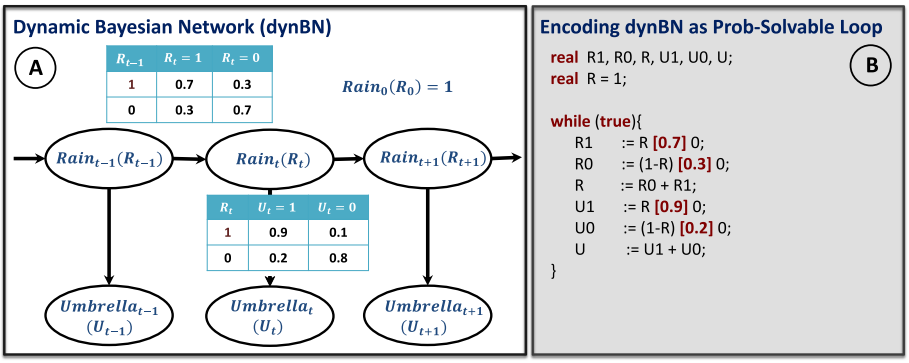

Fig. 3. In Fig. 3(B) we give the Prob-solvable loop encoding of the dynBN from Fig. 3(A).

Linear Dependency for Gaussian Variables. A local probabilistic model for a Gaussian random variable with continuous parents (as introduced in Definition 7) can be encoded as a Prob-solvable loops update, as follows:

$$X = RV(gauss,\ \alpha_X + \sum_{Y \in Par(X)} \beta_{X,Y} \cdot Y,\ \sigma_X^2), \tag{9}$$

where $\alpha_X, \beta_{X,Y}$ are constants, σ_X^2 is fixed and $RV(gauss, \mu, \sigma^2)$ denotes a Gaussian random variable drawn from a Gaussian distribution $\mathcal{G}(\mu, \sigma^2)$.

Conditional Linear Gaussian Dependency. By combining BN dependencies on discrete and continuous variables for a Gaussian random variable X, we can model conditional linear Gaussian dependencies for X. Let D be the joint distribution of the discrete parents of X and for each $d \in D$ let $\mathcal{G}_d$ be the Gaussian distribution associated with condition d (here $\mathcal{G}_d$ may depend on the values of continuous parents $Par(X)$ of X, as discussed in Sect. 3). The conditional linear Gaussian dependency for X can be modeled as the following Prob-solvable loop update:

$$\sum_{d \in \Omega(D)} [D = d] \cdot \mathcal{G}_d. \tag{10}$$

Example 6. Figure 2(B) shows the Prob-solvable loop encoding of the clgBN of Fig. 2(A). The random variables, W_1 and W_2 are given by conditional linear Gaussian dependency and encoded using (10). For simplicity, $W1$ and $W2$ are further split into variables $W1\_1$ and $W1\_2$, and $W2\_1$ and $W2\_2$, respectively, representing different values of $W1$ and $W2$ based on the value of D. Further, $D1\_S$ is a binary variable which is 1 iff D is 1 and S is 0, and $W2D1\_S$ represents the expected value of $W2 \cdot D1\_S$.

Temporal Dependencies in DynBNs. Dependencies in dynBNs are given by intra- and inter-time-slice edges. While the encoding of these dependencies is similar to the BN dependencies discussed above, there are two restrictions on the structure of the dynBNs ensuring that dynBNs can be encoded as Prob-solvable loops. First, dependency of a dynBN variable X on itself must be represented by a linear function. This restriction

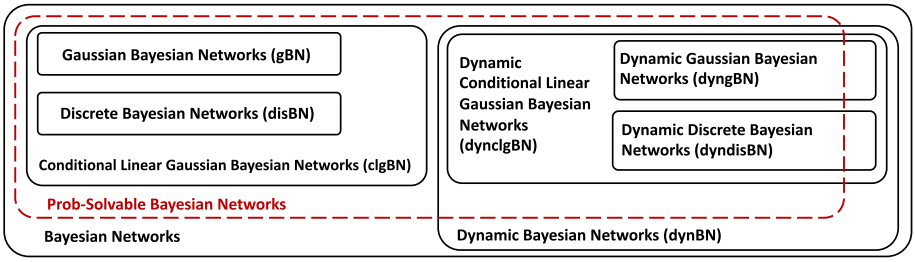

Fig. 4. BN hierarchy.

could be lifted for discrete variables, as discussed in Lemma 1. Second, a variable X can only depend on itself in previous time-slice and current time-slice variables.

Example 7. Figure 3(B) lists the Prob-solvable loop corresponding to Fig. 3(A). The Bernoulli random variables R and U are encoded using (7)-(8). The parameters of R and U change across iterations, corresponding to parameters in different time-slices of the dynBN; their concrete values are given by the expected values of R and U.

Algorithm 2. Encoding BN variants as Prob-solvable loops

Input: BN
Output: Prob-solvable program
Notation: LPM denoting a local probabilistic model

$Nodes :=$ topologically ordered set of BN nodes
for X in $Nodes$ **do**
  if LPM of X is CPT **then**
    for each line L in the CPT **do** Set X_L as in (7)
    Set X as in (8)
  if LPM of X is a linear dependency for Gaussian variables **then** Set X as in (9)
  if LPM of X is a conditional linear Gaussian dependency **then** Set X as in (10)

4.2 Encoding BNs as Prob-solvable Loops

Section 4.1 encoded common local probabilistic models of BN dependencies as Prob-solvable loop updates. Since BNs are DAGs, BN nodes can be ordered in such a way that each BN node X depends only on previous BN variables— its parents $Par(X)$. Hence, BNs can be encoded as Prob-solvable loops, as shown in Algorithm 2 and stated below.

Theorem 2. *Every BN and dynBN*[2] *with local probabilistic models given by CPT or (conditional linear) Gaussian dependencies can be encoded as a Prob-solvable loop. In particular, disBNs, gBNs and clgBNs can be encoded as Prob-solvable loops.*

[2] subject to the restriction on structure of dynBN as discussed in Sect. 4.1.

Based on Algorithm 2 and Theorem 2, we complete this section by defining the following class of BNs, in relation to Prob-solvable loops.

Definition 11 (Prob-solvable Bayesian Networks). *A* Prob-solvable Bayesian Network (PSBN) *is a BN which can be encoded as a Prob-solvable loop.*

The relation and expressivity of PSBNs, and hence Prob-solvable loops, compared to BN variants is visualized in Fig. 4.

5 Automatic BN Analysis via Prob-solvable Loop Reasoning

We now show that several BN challenges can be automatically solved by generating moment-based invariants of Prob-solvable loops encoding the respective BNs. To this end, (i) we consider exact inference, sensitivity analysis, filtering and computing the expected number of rejecting samples in sampling-based BN procedures and (ii) formalize these BN problems as reasoning tasks within Prob-solvable loop analysis. We then (iii) encode BNs as Prob-solvable loop $\mathcal{P}$ using Algorithm 2 and (iv) generate moment-based invariants of $\mathcal{P}$ using Algorithm 1. We address steps (i)-(ii) in Sects. 5.1-5.3, and report on the automation of our work in Sect. 5.4.

5.1 Exact Inference in BNs

Common queries on BN properties address (i) the probability distributions of BN nodes X, for example by answering what is $P(X = x)$ or $P(X < c)$; (ii) the conditional probabilities of BN nodes X, Y, such as $P(X = x|Y = y)$; or (iii) the expected values and higher-order moments of BN nodes X, Y, for instance $\mathbb{E}[X], \mathbb{E}[X^2], \mathbb{E}[X|Y = y]$ and $\mathbb{E}[X^2|Y = y]$. Here we focus on (iii) but show that, in some BN variants, queries related to (ii) can also be solved by our work.

Exact Inference in disBNs. In the case when a BN node X is binary-valued, we have $\mathbb{E}[X] = P(X = true)$. Furthermore, for any higher-order moment of X we also have $Mom_k[X] = P(X = true)$. For non-binary-valued but discrete BN node X, with values from $\{0, \ldots, m\}$, the higher-order moments of X are also computable. Moreover, the first $m-1$ moments are sufficient to fully specify probabilities $P(X = i)$, for $i \in \{0, \ldots, m-1\}$, as proven below.

Lemma 1. *The probabilities, and hence the higher-order moments, of a discrete random variable X over $\{0, \ldots, m-1\}$ are specified by the first $m-1$ higher-order moments of X.*

Proof. Let $p_i := P(X = i)$, for $i \in \{0, \ldots, m-1\}$. Then, $\sum_{0 \leq i < m} i^k p_k = Mom_k(X)$, yielding $m-1$ linear equations over $p_0, \cdots, p_{m-1}$, with $k \in \{1, \cdots, m-1\}$. As we also have $\sum_{0 \leq i < m} p_i = 1$, we have a linear system of m linearly independent equations, implying the existence of a unique solution which specifies the distribution of X. □

For computing conditional expected values and higher-order moments, we show next that deriving $\mathbb{E}[X^k|D = i]$ is reduced to the problem of computing $\frac{\mathbb{E}[X^k \cdot [D=i]]}{\mathbb{E}[[D=i]]}$.

Lemma 2. *If $D = i$ with non-zero probability, we have* $\mathbb{E}[X^k|D = i] = \frac{\mathbb{E}[X^k \cdot [D=i]]}{\mathbb{E}[[D=i]]}$.

Proof. By partition properties for expected values, we have

$$\mathbb{E}[X^k[D = i]] = \mathbb{E}[X^k[D = i]|D = i]P(D = i) + \mathbb{E}[X^k[D = i]|D \neq i]P(D \neq i).$$

As $[D = i] = 1$ iff $D = i$, we derive $\mathbb{E}[X^k[D = i]|D = i] = \mathbb{E}[X^k|D = i]$ and $\mathbb{E}[X^k[D = i]|D \neq i] = 0$. Therefore, $\mathbb{E}[X^k|D = i] = \mathbb{E}[X^k|D = i]P(D = i)$. Since $P(D = i) \neq 0$, we conclude $\mathbb{E}[X^k|D = i] = \frac{\mathbb{E}[X^k \cdot [D=i]]}{\mathbb{E}[[D=i]]}$. □

Exact Inference in gBNs. Recall that a Gaussian distribution is specified by its first two moments, that is by its mean μ and variance σ^2. As all nodes in a gBN are Gaussian random variables, the first two moments of gBN nodes are sufficient to analyse gBN behaviour. Further, $\mathbb{E}[X]$ and $\mathbb{E}[X^2]$ of a gBN node X are computable using Algorithm 1.

Exact Inference in clgBNs. As continuous variables X in clgBNs are Gaussian random variables, the means and variance of X are also computable using Algorithm 1. However, clgBNs might also include discrete variables D, whose (conditional) higher-order moments can be computed as in Lemmas 1-2. Further, for a continuous variable X and a discrete variable D in a clgBN, we have

$$\mathbb{E}[X|D = i] = \frac{\mathbb{E}[X^k \cdot [D = i]]}{\mathbb{E}[[D = i]]},$$

allowing us, for example, to derive $\mathbb{E}[W2|D = 1] = 7.25 + 0.89a$ in Fig. 2.

Exact Inference in dynBNs. As dynBNs are infinite in nature, (infinite) Prob-solvable loops are suited to reason about dynBN inferences, such as (i) long-term behaviour or prediction and (ii) filtering and smoothing. A related problem is characterizing the dynBN behaviour after n iterations, and in particular for $n \to \infty$.

(i) Prediction and long-term behaviour in dynBNs By modeling dynBNs as Prob-solvable loops, we can compute/predict higher-order moments $\mathbb{E}[X_n^k]$ of dynBN nodes X using Algorithm 1, for an arbitrary n. Further, thanks to the existence of $\mathbb{E}[X_n^k]$ for Prob-solvable loops, we conclude that $\lim_{n\to\infty} \mathbb{E}[X_n^k]$ is also computable. Moreover, Algorithm 1 computes higher-order moments/MBIs in $O(1)$ time w.r.t. n, which is not the case of the $O(n)$ approach of the standard Forward algorithm.

(ii) Filtering and prediction in dynBNs Predicting next dynBNs states X_{t+1} given all observations $e_1, \ldots, e_{t+1}$ until time t can be expressed as $P(X_{t+1}|e_1, \ldots, e_{t+1})$, which in turn can be rewritten using Bayes' rule under the sensor Markov assumption (the evidence e_t depends only on program variables X_t from the same time-slice), as follows:

$$P(X_{t+1}|e_1, \ldots, e_{t+1}) = P(e_{t+1}|X_{t+1}) \cdot \sum_{x_t} P(X_{t+1}|x_t) \cdot P(x_t|e_1, \ldots, e_t),$$

where $P(e_{t+1}|X_{t+1})$ and $P(X_{t+1}|x_t)$ are specified by the BN, assuming discrete-valued observation variables. Filtering and prediction in dynBNs is thus computable using MBIs of Prob-solvable loops.

5.2 Number of BN Samples Until Positive BN Instance

As pointed out in [7], an interesting question about BNs is "Given a Bayesian network with observed evidence, how long does it take in expectation to obtain a single sample that satisfies the observations?". A related, though arguably simpler, question would require giving the expected number of positive instances (samples satisfying the observation) in N samples of BNs. Both of these questions can be answered using standard results from probability theory.

Lemma 3. *Given the probability p of a BN observation, the expected number of positive BN instances in N samples is pN. Further, the expected number of BN samples until the first positive BN instance is $\frac{1}{p}$.*

Although the lemma can be proven using standard techniques from probability, we note that the results can also be obtained using Prob-solvable loop reasoning, by relying on Algorithm 1, as illustrated next.

Example 8. For inferring the expected number of positive instances in N samples in Fig. 1, we first encode the observation in the BN as a new variable $GR = G \cdot R$, capturing the observation that the grass is wet and there was rain. We then transform the BN into a dynBN adding an inter-time-slice counter update $count = count + GR$. The expected number of positive instances is then the prediction $\mathbb{E}[count_n]$ for $n = N$.

For answering the question of [7], we again encode the observation first as above, e.g. $GR = G \cdot R$. We use a boolean variable to indicate whether there has been a positive instance $continue = continue \cdot [GR = 0]$, which is initiated as 1 (or $true$) and updated to 0 once $GR = 1$ and stays 0 thereafter. Finally, we update a loop counter as long as there was no positive instance observed with $count = count + continue$. The expected number of samples until the first positive instance is the long-term behaviour of $count$, i.e. $\lim_{n\to\infty} \mathbb{E}[count_n]$.

5.3 Sensitivity Analysis in BNs

As BNs rely on network parameters, a challenging task is to understand to what extent does a small change in a network parameter affect the outcome of particular BN query. This task is referred to as sensitivity analysis in BNs. More precisely, we would like to compute $P(X|e)$ and $\mathbb{E}[X|e]$ for a random variable X and evidence e as functions of a BN parameter(s) θ. For doing so, we note that Prob-solvable loops may use symbolic coefficients. Thus, replacing concrete BN probabilities with symbolic parameters and solving BN queries as discussed in Sect. 5.1, allow us to automate sensitivity analysis in BNs by computing MBIs of the respective Prob-solvable loops, using Algorithm 1.

Example 9. A sensitivity analysis in Fig. 2 could measure the effect of parameters of weight loss in week 1 on the conditional expectation $\mathbb{E}[W2|D = 1]$. That is, we compute $\mathbb{E}[W2|D = 1]$ as a function of parameters of $W1$. In this case, we introduce symbolic parameters a and b adjusting the parameters of weight loss in week 1 ($W1\_1$) when the drug was administered. Using Algorithm 1, we compute the MBIs $\mathbb{E}[W2^k \cdot D], \mathbb{E}[D]$, from which we have, for $k = 1$, $\mathbb{E}[W2|D = 1] = \frac{\mathbb{E}[W2 \cdot D]}{\mathbb{E}[D]} = 0.89a + 7.25$, answering the respective sensitivity analysis of Fig. 2.

Table 1. BN analysis via Prob-solvable loop reasoning within `Mora`.

BN BN Problem	MBIs	BN Solutions
Grass – Fig. 1 (disBN) #nodes: 3, #edges: 3, #parameters: 7, #variables in Prob-solvable encoding: 9		
Q1: $P(R\|G)$	$0.72s$	$P(R\|G) = 0.8752$
Q2: Number of samples	$1.24s$	$\#samples = 1.37$
Q3: Sensitivity analysis	$0.82s$	$\frac{0.04b+0.6396}{-0.178a+0.04b+0.7308}$
Alarm [41] (disBN) #nodes: 5, #edges: 4, #parameters: 10, #variables in Prob-solvable encoding: 13		
Q1: $P(B\|A)$	$0.83s$	$P(B\|A) = 0.373551$
Q1: $P(EQ\|M)$	$1.01s$	$P(EQ\|M) = 0.0358809$
Q1: $P(\neg EQ \wedge \neg B\|A \wedge J)$	$1.53s$	$P(\neg EQ \wedge \neg B\|A \wedge J) = 0.396195$
Q1: $P(EQ \wedge \neg B\|M \wedge J)$	$1.43s$	$P(EQ \wedge \neg B\|M \wedge J) = 0.175492$
Q2: Number of samples (for $M \wedge J$)	$1.91s$	$\#samples = 19.978$
Q3: Sensitivity analysis (all of above)	$3.36s$	$P(B\|A) = \frac{b(0.01q+0.94)}{-0.279bq+0.939b+0.289q+0.001}, \cdots$
Asia [34] (disBN) #nodes: 8, #edges: 8, #parameters: 18, #variables in Prob-solvable encoding: 24		
Q1: $P(Asia, Lung\|Dysp)$	$2.25s$	$P(Asia, Lung\|Dysp) = 0.00045596785$
Q2: Number of samples	$2.85s$	$\#samples = 1818.1818$
Q3: Sensitivity analysis	$3.76s$	$P(Asia, Lung\|Dysp) = \frac{0.192a+0.29625b+0.0221625}{0.992a+0.62b+48.6054}$
Marks [36] (gBN) #nodes: 3, #edges: 3, #parameters: 6, #variables in Prob-solvable encoding: 5-6		
Q1: Marks - expected values	$0.05s$	$\mathbb{E}[Stat] = 41.688, \cdots$
Q3: Marks - sensitivity analysis EVs	$0.12s$	$\mathbb{E}[Stat] = 0.99\mu_{al}c + 1.0669\mu_{al} - 3.57c - 12.2967, \cdots$
Q1: Marks - second moments	$0.11s$	$\mathbb{E}[Stat^2] = 2035.718, \cdots$
Q3: Marks - sensitivity 2nd moments	$0.28s$	$\mathbb{E}[Stat^2] = 0.9801\mu_{al}^2c^2 + \cdots + 0.0961\sigma_{an} + 438.4063, \cdots$
Q1: Average - expected values	$0.08s$	$\mathbb{E}[AverageMark] = 46.271$
Q3: Average - sensitivity EV	$0.18s$	$\mathbb{E}[AverageMark] = 0.33\mu_{al}c + 1.01897\mu_{al} - 1.19c - \cdots$
Q1: Average - second moments	$0.13s$	$\mathbb{E}[AverageMark^2] = 2673.160$
Q3: Average - sensitivity 2nd moment	$0.46s$	$\mathbb{E}[AverageMark^2] = 0.1089\mu_{al}^2c^2 + \cdots + 0.190678\sigma_{an}$
Rats [17] – Fig. 2 (clgBN) #nodes: 4, #edges: 4, #parameters: 11, #variables in Prob-solvable encoding: 10		
Q1: $\mathbb{E}[W2\|D]$	$0.49s$	$\mathbb{E}[W2\|D] = 15.02$
Q3: $\mathbb{E}[W2\|D]$ sensitivity	$0.72s$	$\mathbb{E}[W2\|D] = 15.02 + 2.24a$
Q1: $\mathbb{E}[W2^2\|D]$	$1.05s$	$\mathbb{E}[W2^2\|D] = 242.8356$
Q3: $\mathbb{E}[W2^2\|D]$ sensitivity	$1.35s$	$\mathbb{E}[W2^2\|D] = 242.8356 + 5.0176b + 67.2896a + 5.0176a^2$
Umbrella [41] (dynBN) #nodes: 2, #edges: 2, #parameters: 3, #variables in Prob-solvable encoding: 6		
Q1: Prediction	$0.56s$	$\mathbb{E}[R] = \frac{1}{2}((2/5)^n + 1)$
Q1: Long-term behaviour		$\mathbb{E}[R] \to \frac{1}{2}$ as $n \to \infty$
Q3: Prediction - sensitivity	$1.15s$	$\mathbb{E}[R] = \frac{(r-1)(r-0.3)^n - 0.3}{r-1.3}$
Q3: Long-term - sensitivity		$\mathbb{E}[R] \to \frac{0.3}{1.3-r}$ as $n \to \infty$

5.4 Implementation and Experiments

We automated BN analysis via Prob-solvable loop reasoning by extending and using our tool `Mora` [6]. To this end, we first manually encoded BNs as Prob-solvable loops using Algorithm 2. We then extended `Mora` to support our extended programming model of Prob-solvable loops and integrated Algorithm 1 within `Mora`[3] to generate MBIs of Prob-solvable loops, solving thus the BN problems of Sects. 5.1–5.3. As benchmarks, we used 28 BN-related problems for 6 BNs taken from [17,29,34,36,41]. Table 1 summarizes our experiments. For each example of Table 1, we list the BN queries we considered, that is probabilistic inference (Q1), number of BN samples (Q2) and sensitivity analysis (Q3) as introduced in Sect. 1 and discussed in Sects. 5.1–5.3. Column 2 of Table 1 shows the time needed by `Mora` to compute moment-based invariants (MBIs) solving the respective BN problems. The last column of Table 1 gives our derived solutions for the considered BN queries. Our experiments were run on a MacBook Pro 2017 with 2.3 GHz Intel Core i5 and 8GB RAM.

6 Related Work

The classical approach to analyze probabilistic models is based on probabilistic model checking [2]. However, approaches [15,27,33] cannot yet handle unbounded and real variables that are required for example to encode Gaussian BNs, nor do they support invariant generation, which is a key step in our work.

In the context of probabilistic programs (PPs), a formal semantics for PPs was first introduced in [30], together with a deductive calculus to reason about expected running time of PPs [31]. This approach was further refined and extended in [37], by introducing weakest pre-expectations based on the weakest precondition calculus of [16]. While [37] infers quantitative invariants only over expected values of program variables, our moment-based invariants yield quantitative invariants over arbitrary higher-order moments, including expected values. Further, the setting of [37] considers PPs where the stochastic inputs are restricted to discrete distributions with finite support. To encode Gaussian BNs it is however necessary to handle also continuous distributions with infinite support, as described in our work.

The first semi-automatic and complete method synthesizing the linear quantitative invariants needed by [37] was introduced in [26]. To this end, PP loops are annotated with linear template invariants and constraint solving is used to find concrete values of the template parameters. Further extensions for template-based non-linear quantitative invariant generation have been proposed in [11,18]. A related line of research is given in [3], where martingales and user-provided hints are used to compute quantitative invariants of PPs. The recent work of [32] generalizes the use of martingales in conjunction with templates for computing higher-order moments of program variables, with the overall goal of approximating runtimes of randomized programs. Unlike these works, our approach extends Prob-solvable loops from [4] and provides a fully automated approach for deriving non-linear invariants over higher-order moments.

Several techniques infer runtimes and expected values of PPs, see e.g. [9,10,19,23,38]. To the best of our knowledge, however only [7] targets explicitly BNs on the source code level, by using a weakest precondition calculus similar to [25,37]. The PPs

[3] https://github.com/probing-lab/mora.

addressed in [7] are expressed in the *Bayesian Network Language* (BNL) fragment of the *probabilistic Guarded Command Language (pGCL)* of [37]. The main restriction of BNL is that loops prohibit undesired data flow across multiple loop iterations: it is not possible to assign to a variable the value of the same variable or another variable at the previous iteration. Furthermore, BNL does not natively allow to draw samples from Gaussian distribution, allowing thus only discrete BNs to be encoded in BNL. In contrast to [7], in our work we use Prob-solvable loops, as a subclass of PPs, to allow polynomial updates over random variables and parametric distributions. Variable updates of Prob-solvable loops can involve coefficients from Bernoulli, Gaussian, uniform and other distributions, whereas variable updates drawn from Gaussian and uniform distributions can depend on other program variables. Compared to [7], we thus support reasoning about (conditional linear) Gaussian BNs and our PPs also allow data flow across loop iterations which is necessary to encode dynamic BNs.

7 Conclusion

We extend the class of Prob-solvable loops with variable updates over Gaussian and uniform random variables depending on other program variables. We show that moment-based invariants (MBIs) in Prob-solvable loops can always be computed as quantitative invariants over higher-order moments of loop variables. We further encode BN variants as Prob-solvable loops, allowing us to turn several BN problems into the problem of computing MBIs of Prob-solvable loops. In particular, we automate the BN analysis of exact inference, sensitivity analysis, filtering and computing the expected number of rejecting samples in sampling-based procedures via Prob-solvable loop reasoning. As future work, we plan to further extend the class of Prob-solvable loops with more complex flow and arithmetic and address termination analysis of such loops.

References

1. Ai, J., et al: HackPPL: a universal probabilistic programming language. In: Proceeding of MAPL@PLDI, pp. 20–28 (2019)
2. Baier, C., Katoen, J.P.: Principles of Model Checking. The MIT Press, Cambridge (2008)
3. Barthe, G., Espitau, T., Ferrer Fioriti, L.M., Hsu, J.: Synthesizing probabilistic invariants via doob's decomposition. In: Chaudhuri, S., Farzan, A. (eds.) CAV 2016. LNCS, vol. 9779, pp. 43–61. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-41528-4_3
4. Bartocci, E., Kovács, L., Stankovič, M.: Automatic generation of moment-based invariants for prob-solvable loops. In: Chen, Y.-F., Cheng, C.-H., Esparza, J. (eds.) ATVA 2019. LNCS, vol. 11781, pp. 255–276. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-31784-3_15
5. Bartocci, E., Kovács, L., Stankovič, M.: Analysis of bayesian networks via prob-solvable loops. arXiv preprint arXiv:2007.09450 (2020)
6. Bartocci, E., Kovács, L., Stankovič, M.: MORA - automatic generation of moment-based invariants. TACAS 2020. LNCS, vol. 12078, pp. 492–498. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45190-5_28
7. Batz, K., Kaminski, B.L., Katoen, J.-P., Matheja, C.: How long, O Bayesian network, will I sample thee? In: Ahmed, A. (ed.) ESOP 2018. LNCS, vol. 10801, pp. 186–213. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-89884-1_7
8. Bingham, E., et al.: Pyro: deep universal probabilistic programming. J. Mach. Learn. Res. **20**, 1–6 (2019)

9. Brázdil, T., Kiefer, S., Kucera, A., Vareková, I.H.: Runtime analysis of probabilistic programs with unbounded recursion. J. Comput. Syst. Sci. **81**(1), 288–310 (2015)
10. Celiku, O., McIver, A.: Compositional specification and analysis of cost-based properties in probabilistic programs. In: Fitzgerald, J., Hayes, I.J., Tarlecki, A. (eds.) FM 2005. LNCS, vol. 3582, pp. 107–122. Springer, Heidelberg (2005). https://doi.org/10.1007/11526841_9
11. Chen, Y.-F., Hong, C.-D., Wang, B.-Y., Zhang, L.: Counterexample-guided polynomial loop invariant generation by lagrange interpolation. In: Kroening, D., Păsăreanu, C.S. (eds.) CAV 2015. LNCS, vol. 9206, pp. 658–674. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21690-4_44
12. Constantinou, A.C., Fenton, N.E., Neil, M.: pi-Football: a Bayesian network model for forecasting association football match outcomes. Knowl. Based Syst. **36**, 322–339 (2012)
13. Cooper, G.F.: The computational complexity of probabilistic inference using bayesian belief networks. Artif. Intell. **42**(2–3), 393–405 (1990)
14. Dagum, P., Luby, M.: Approximating probabilistic inference in bayesian belief networks is NP-hard. Artif. Intell. **60**(1), 141–153 (1993)
15. Dehnert, C., Junges, S., Katoen, J.-P., Volk, M.: A storm is coming: a modern probabilistic model checker. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10427, pp. 592–600. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63390-9_31
16. Dijkstra, E.W.: Guarded commands, nondeterminacy and formal derivation of programs. Commun. ACM **18**(8), 453–457 (1975)
17. Edwards, D.: Introduction to Graphical Modelling. Springer Science & Business Media, New York (2012)
18. Feng, Y., Zhang, L., Jansen, D.N., Zhan, N., Xia, B.: Finding polynomial loop invariants for probabilistic programs. In: D'Souza, D., Narayan Kumar, K. (eds.) ATVA 2017. LNCS, vol. 10482, pp. 400–416. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68167-2_26
19. Fioriti, L.M.F., Hermanns, H.: Probabilistic termination: soundness, completeness, and compositionality. In: Proceedings of POPL, pp. 489–501 (2015)
20. Friedman, N., Linial, M., Nachman, I., Pe'er, D.: Using Bayesian networks to analyze expression data. J. Comput. Biol. **7**(3–4), 601–620 (2000)
21. Ghahramani, Z.: Probabilistic machine learning and artificial intelligence. Nature **521**, 452–459 (2015)
22. Heckerman, D.: A tutorial on learning wihtbayesian networks. In: Innovations in Bayesian Networks: Theory and Applications, Studies in Computational Intelligence, vol. 156, pp.33–82. Springer (2008). https://doi.org/10.1007/978-3-540-85066-3_3
23. Hehner, E.: A probability perspective. Formal Aspects Comput. **23**(4), 391–419 (2011). 10.1007/s00165-010-0157-0
24. Jiang, X., Cooper, G.: A Bayesian spatio-temporal method for disease outbreak detection. J. Am. Med. Inform. Assoc. **17**(4), 462–471 (2010)
25. Kaminski, B.L., Katoen, J.-P., Matheja, C., Olmedo, F.: Weakest precondition reasoning for expected run–times of probabilistic programs. In: Thiemann, P. (ed.) ESOP 2016. LNCS, vol. 9632, pp. 364–389. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49498-1_15
26. Katoen, J.-P., McIver, A.K., Meinicke, L.A., Morgan, C.C.: Linear-invariant generation for probabilistic programs: automated support for proof-based methods. In: Cousot, R., Martel, M. (eds.) SAS 2010. LNCS, vol. 6337, pp. 390–406. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15769-1_24
27. Katoen, J., Zapreev, I.S., Hahn, E.M., Hermanns, H., Jansen, D.N.: The ins and outs of the probabilistic model checker MRMC. Perform. Eval. **68**(2), 90–104 (2011)
28. Koller, D., Friedman, N.: Probabilistic Graphical Models - Principles and Techniques. MIT Press, Cambridge (2009)
29. Korb, K., Nicholson, A.: Bayesian Artificial Intelligence, 2nd edn. Chapman and Hall, Boca Raton (2010)

30. Kozen, D.: Semantics of probabilistic programs. J. Comput. Syst. Sci. **22**(3), 328–350 (1981)
31. Kozen, D.: A probabilistic PDL. J. Comput. Syst. Sci. **30**(2), 162–178 (1985)
32. Kura, S., Urabe, N., Hasuo, I.: Tail probabilities for randomized program runtimes via martingales for higher moments. In: Vojnar, T., Zhang, L. (eds.) TACAS 2019. LNCS, vol. 11428, pp. 135–153. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17465-1_8
33. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_47
34. Lauritzen, S.L., Spiegelhalter, D.J.: Local computation with probabilities on graphical structures and their application to expert systems (with discussion). Roy. Stat. Soc. B (Stat. Methodol.) **50**(2), 157–224 (1988)
35. Lin, G.L.: Characterizations of Distributions via Moments. Indian Statistical Institute, Kolkata (1992)
36. Mardia, K.V., Kent, J.T., Bibby, J.M.: Multivariate Analysis. Academic Press, Cambridge (1979)
37. McIver, A., Morgan, C.: Abstraction Refinement and Proof for Probabilistic Systems. Monographs in Computer Science. Springer, New York (2005)
38. Monniaux, D.: An abstract analysis of the probabilistic termination of programs. In: Cousot, P. (ed.) SAS 2001. LNCS, vol. 2126, pp. 111–126. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-47764-0_7
39. Neapolitan, R., Jiang, X.: Probabilistic Methods for Financial and Marketing Informatics. Morgan Kaufmann, San Francisco (2010)
40. Pearl, J.: Bayesian Networks: A model of self-activated memory for evidential reasoning. In: Proceedings of Cognitive Science Society, pp. 329–334 (1985)
41. Russell, S.J., Norvig, P.: Artificial Intelligence - A Modern Approach. Pearson Education, London (2010)
42. Tran, D., Hoffman, M.D., Saurous, R.A., Brevdo, E., Murphy, K., Blei, D.M.: Deep Probabilistic Programming. CoRR abs/1701.03757 (2017)
43. Yuan, C., Druzdzel, M.J.: Importance sampling algorithms for bayesian networks: principles and performance. Math. Comput. Model. **43**(9), 1189–1207 (2006)
44. Zweig, G., Russell, S.J.: Speech recognition with dynamic bayesian networks. In: Proceedings of AAAI, pp. 173–180 (1998)