An Assertion-Based Logic for Local Reasoning about Probabilistic Programs

Huiling Wu, Anran Cui *, and Yuxin Deng **

Shanghai Key Laboratory of Trustworthy Computing
East China Normal University, Shanghai, China
{51255902019, arcui}@stu.ecnu.edu.cn, yxdeng@sei.ecnu.edu.cn

Abstract. We introduce an assertion-based logic specifically designed for local reasoning about probabilistic programs featuring unbounded loops. Distribution formulas and their extensions facilitate the representation of invariants for unbounded loops in probabilistic programs. The assertions connected by separating conjunction exhibit probabilistic independence, which more intuitively displays the mutually independent properties of variables and ensures that the logic supports local reasoning. We prove the soundness of our logic and showcase its effectiveness through the formal verification of a wide range of examples including probabilistic inference in Bayesian networks and security analysis of cryptographic schemes.

Keywords: Probabilistic programs · Separation logic · Local reasoning.

1 Introduction

Probabilistic programs are ubiquitous across various domains, including reliability analysis of networks [21, 46] and cyberphysical systems [30, 37], verification of randomized algorithms [40], security of cryptographic algorithms [8], privacy protocols [10], and machine learning, particularly Bayesian network analysis [23]. With the development of these fields, there is a growing recognition of the importance of formally verifying probabilistic programs to ensure their correctness. This is particularly crucial due to the challenges posed by the probabilistic nature of their execution, necessitating rigorous verification methods to mitigate potential risks and ensure reliable performance.

Since Kozen's seminal work [35] established the semantics of probabilistic programs, different techniques have emerged to verify the correctness of programs. Early methods rely on mathematical models derived from probabilistic semantics, utilizing structures like Markov chains, Markov decision processes [29, 3], probabilistic input-output automata [45, 50], and probabilistic transition systems [28, 26]. However, the

* Huiling Wu and Anran Cui contributed equally to this work.

** Deng was supported by the National Key R&D Program of China under Grant No. 2023YFA1009403, the National Natural Science Foundation of China under Grants Nos. 62472175 and 62072176, Shanghai Trusted Industry Internet Software Collaborative Innovation Center, and the Digital Silk Road Shanghai International Joint Lab of Trustworthy Intelligent Software under Grant No. 22510750100.

$x := 0;$
while $x < 5$ **do**
  $n :=_{\$} \{\frac{1}{2} \cdot 0 + \frac{1}{2} \cdot 1\};$
  if $(n = 1)$ **then**
    $x := x + 1;$
  else
    skip;
  fi
od

Fig. 1: The program $Prog_1$

complexities of some probabilistic programs often make precise model construction difficult, hindering traditional verification methods. As an alternative, a systematic approach for formally verifying probabilistic programs without computing their semantics has received much attention. Techniques such as probabilistic process algebra [2], stochastic processes [19], and model checking [16] have been proposed.

The most mature approach in this area, which bypasses the need for computing semantics, is the expectation-based technique [36, 39]. These works primarily utilize the weakest pre-expectation calculus or the weakest precondition calculus, triggering a series of research in this direction [31, 25, 33, 32, 34, 41]. Another popular family of approaches is the assertion-based techniques, first proposed by Ramshaw [42]. These techniques enable concurrent verification of properties involving multiple probabilities and offer more intuitive specifications for loop reasoning.

Challenge. In the realm of probabilistic programs, assertion-based techniques require assessing whether an assertion is satisfied. Very often, the Boolean condition in the guard of a loop may not deterministically evaluate to true or false after an iteration; instead, it forms a Boolean distribution. This kind of loops, termed *unbounded loops* in this paper, lack a guarantee of termination within any fixed number of iterations and may potentially diverge with certain probability. The infinitary nature of unbounded loops complicates reasoning tasks, posing a significant challenge in the design of while rules and invariants for probabilistic programs. Many prior works in assertion-based systems lack adequate rules to support the inference for unbounded loops [13, 9, 38]. A few works attempt to address this issue but impose constraints on invariants [43] or introduce conditions that are challenging to verify [27, 5].

Let us consider a simple probabilistic program, called $Prog_1$, as given in Figure 1. Intuitively, the property $x = 5$ holds upon the termination of the program. However, after each iteration of the loop, there remains a non-zero probability that the Boolean condition $x < 5$ holds. Establishing a suitable invariant for this program is crucial for verifying this property.

We take advantage of distribution formulas [18] to specify properties of distribution states. An invariant for $Prog_1$ can be expressed as $x < 5 \oplus x = 5$, which intuitively means that both properties $x < 5$ and $x = 5$ are satisfied probabilistically, though the precise probabilities are not explicitly stated.

As discussed in [18], distribution formulas serve as a useful tool for constructing invariants of while loops, thus avoiding the infinite sequences of assertions discussed in [5].

Contribution. We present an assertion-based logic for local reasoning about probabilistic programs featuring unbounded loops. Our contributions are summarized as follows:

- We propose a new separation logic for discrete probabilistic programs, which uses distribution formulas to specify probabilistic behavior. These formulas play a crucial role in establishing an invariant condition in the proof rule for unbounded loops, effectively preventing the occurrence of infinite assertion sequences.
- The separating conjunction in our work can represent probabilistic independence, as inherited from [9], and is combined with a frame rule to facilitate local reasoning. We then prove the soundness of the separation logic; a key ingredient of the proof is the property that the denotation of every assertion is a closed set.
- We demonstrate the effectiveness of our logic by establishing the correctness of various probabilistic programs featuring unbounded loops. The case studies encompass examples for Bayesian networks and cryptographic schemes, highlighting the versatility and applicability of our approach.

Organization. The rest of the paper is structured as follows. In Section 2, we recall some basic notations about probability distributions. Section 3 gives the syntax and denotational semantics of a probabilistic language. Section 4 defines an assertion language and provides its semantics. In Section 5, we present a proof system for local reasoning about probabilistic programs. In Section 6, we verify the correctness of $Prog_1$ and a program for encoding a Bayesion Network. We discuss related work and compare with ours in Section 7. Finally, we conclude and discuss possible future work in Section 8.

2 Preliminaries

Let S be a countable set. A (discrete) *sub-distribution* on S is a function $\mu : S \to [0,1]$ with $\|\mu\| \triangleq \sum_{s\in S} \mu(s) \leq 1$. If $\|\mu\| = 1$, then μ is a *distribution*. We denote the set of all sub-distributions on S as **SDist[**S**]**. For any set $S' \subseteq S$, we write $\mu(S')$ for $\sum_{s\in S'} \mu(s)$. The support of a sub-distribution μ is defined as $\lceil \mu \rceil \triangleq \{s \in S \mid \mu(s) > 0\}$. If $\mu(s) = 0$ for all $s \in S$, then μ is an empty sub-distribution, denoted by ε_S.

Let $\mu_1, \mu_2 \in$ **SDist[**S**]** such that $\|\mu_1\| + \|\mu_2\| \leq 1$. The sum of $\mu_1 + \mu_2$ is also a sub-distribution in **SDist[**S**]**, defined by letting $(\mu_1 + \mu_2)(s) \triangleq \mu_1(s) + \mu_2(s)$, for all $s \in S$. Let $\mu \in$ **SDist[**S**]** and $p \in [0,1]$. Then $p \cdot \mu$ is also a sub-distribution in **SDist[**S**]**, defined by letting$(p \cdot \mu)(s) \triangleq p \cdot \mu(s)$, for all $s \in S$. Let $S' \subseteq S$ and $\mu \in$ **SDist[**S**]**. The restriction $\mu_{S'}$ of μ to S' is also a sub-distribution, defined by

$$\mu_{S'}(s) \triangleq \begin{cases} \mu(s), & \text{if } s \in S' \\ 0, & \text{otherwise.} \end{cases}$$

Let S_1 and S_2 be two countable sets, $\mu_1 \in$ **SDist[**S_1**]**, and $\mu_2 \in$ **SDist[**S_2**]**. The *joint sub-distribution* $\mu_1 \otimes \mu_2 \in$ **SDist[**$S_1 \times S_2$**]** is defined by letting

$$\mu_1 \otimes \mu_2(s_1, s_2) \triangleq \mu_1(s_1) \cdot \mu_2(s_2),$$

$$
\begin{array}{llll}
(\textbf{Aexp}) & a & ::= & r \mid x, y, \ldots \mid f_m(a, \ldots, a) \\
(\textbf{Bexp}) & b & ::= & \textbf{true} \mid \textbf{false} \mid P_m(a, \ldots, a) \mid b \wedge b \mid \neg b \\
(\textbf{Com}) & c & ::= & \textbf{skip} \mid \textbf{abort} \mid x := a \mid x :=_{\$} d_A \mid c;\ c \\
 & & & \mid \textbf{if}\ b\ \textbf{then}\ c\ \textbf{else}\ c\ \textbf{fi} \mid \textbf{while}\ b\ \textbf{do}\ c\ \textbf{od}
\end{array}
$$

Fig. 2: Syntax of PIMP

for any $s_1 \in S_1$ and $s_2 \in S_2$. Let S_1 and S_2 be two countable sets and $\mu \in \textbf{SDist}[S_1 \times S_2]$. The two projection operators $\pi_{S_1}(\cdot)$ and $\pi_{S_2}(\cdot)$ yield two sub-distributions in the sets $\textbf{SDist}[S_1]$ and $\textbf{SDist}[S_2]$, called the *first* and *second marginals* respectively, defined by

$$\pi_{S_1}(\mu)(s_1) \triangleq \sum_{s_2 \in S_2} \mu(s_1, s_2) \qquad \pi_{S_2}(\mu)(s_2) \triangleq \sum_{s_1 \in S_1} \mu(s_1, s_2).$$

The events in S_1 and S_2 are mutually independent if any $\mu \in \textbf{SDist}[S_1 \times S_2]$ can be factored as $\pi_{S_1}(\mu) \otimes \pi_{S_2}(\mu)$.

3 A Probabilistic Imperative Language

In this section, we introduce the syntax and denotational semantics of a probabilistic language, known as PIMP, obtained by extending the imperative language in [49] with random assignments.

3.1 Syntax

The syntax of PIMP is given in Figure 2. Let $\mathbb{Q}$ be the set of rational numbers, and **Var** be the set of variables. Arithmetic operators (e.g., $+$, $-$, $*$, etc.) are denoted by f_m, and Boolean predicates (e.g., $=$, $\leq$, $\geq$, etc.) are denoted by P_m, where m is the arity of f or P. The set **Aexp** of arithmetic expressions includes constant rational numbers $r \in \mathbb{Q}$, variables x, y, ... in **Var**, as well as other arithmetic expressions constructed by arithmetic operators f_m. Similarly, the set **Bexp** of Boolean expressions encompasses Boolean constants **true** and **false**, and other Boolean expressions created by Boolean predicates P_m and Boolean connectives such as $\wedge$ and $\neg$. We collectively refer to arithmetic expressions a and Boolean expressions b as expressions, denoted by e. Here we use d_A to represent a distribution on the finite set A consisting of arithmetic expressions. The set **Com** includes **skip** (no-op command), **abort** (halt command), the classical assignment, the random assignment, the sequential statement, the conditional statement and the **while**-loop statement.

3.2 Partial States and Distribution States

Recall that a classical state is a mapping from **Var** to $\mathbb{Q}$. We introduce a notion of partial state in order to facilitate local reasoning.

Definition 1. *Let X be any subset of* **Var**. *A classical partial state on X is a function $\sigma : X \to \mathbb{Q}$, where $\sigma(x)$ represents the value of the classical variable x.*

We abbreviate a classical (partial) state as a state hereafter. We denote the set of all states on X by Σ_X and the domain of any state $\sigma \in \Sigma_X$ by $\mathrm{dom}(\sigma) \triangleq X$. Note that the set $\Sigma_\emptyset$ has only one element $\mathbf{0} : \emptyset \to \mathbb{Q}$. We denote the set of all states by Σ. Suppose $X \subseteq Y \subseteq$ **Var**. For any $\sigma \in \Sigma_Y$, we use the notation σ_X to denote the state obtained by restricting the state σ to the domain Σ_X, such that $\sigma_X(x) = \sigma(x)$. The updated state denoted by $\sigma[a/x]$ is defined as

$$\sigma[a/x](y) \triangleq \begin{cases} a & \text{if } y = x \\ \sigma(y) & \text{otherwise.} \end{cases}$$

Let the set of (sub)-distribution states on X be $\mathbf{SDist}[\Sigma_X]$. For any $\mu \in \mathbf{SDist}[\Sigma_X]$, its domain $\mathrm{dom}(\mu) \triangleq X$. For simplicity, we abbreviate (sub)-distribution states as distribution states hereafter. We denote the set of all distribution states by

$$\Delta = \{\mu \in \mathbf{SDist}[\Sigma_X] \mid X \subseteq \mathbf{Var}\}.$$

The updated distribution state $\mu[a/x]$ is defined by letting

$$(\mu[a/x])(\sigma) \triangleq \sum_{\sigma' \in \lceil \mu \rceil} \{\mu(\sigma') \mid \sigma'[a/x] = \sigma\}.$$

Definition 2. *Let X and Y be two sets of disjoint variables, $\mu_1 \in$ **SDist**$[\Sigma_X]$ and $\mu_2 \in$ **SDist**$[\Sigma_Y]$. We write $\mu_1 \otimes \mu_2$ for a new distribution state in the set **SDist**$[\Sigma_{X \cup Y}]$, which satisfies $(\mu_1 \otimes \mu_2)(\sigma) = \mu_1(\sigma_X) \cdot \mu_2(\sigma_Y)$, for any $\sigma \in \Sigma_{X \cup Y}$.*

Suppose $X \subseteq Y \subseteq$ **Var**. We project a distribution state $\mu \in \mathbf{SDist}[\Sigma_Y]$ to a distribution state $\pi_X(\mu) \in \mathbf{SDist}[\Sigma_X]$ via the projection operator π in the following way:

$$\pi_X(\mu)(\sigma) \triangleq \sum_{\sigma' \in \lceil \mu \rceil} \{\mu(\sigma') \mid \sigma'_X = \sigma\}.$$

If any distribution μ in $\mathbf{SDist}[\Sigma_{X_1 \cup X_2}]$ can be factored as $\pi_{X_1}(\mu) \otimes \pi_{X_2}(\mu)$, we say the variables in X_1 and X_2 are mutually independent, denoted by $X_1 \perp X_2$.

3.3 Denotational Semantics

We interpret expressions in both partial states and distribution states. For any expression e, we use $\mathbf{V}(e)$ to denote the set of variables occurring in e. The denotation of an arithmetic expression a is a mapping of type $\Sigma_X \to \mathbb{Q}$, and that of a Boolean expression b is a mapping of type $\Sigma_{X'} \to \{\mathbf{true}, \mathbf{false}\}$, where $\mathbf{V}(a) \subseteq X$ and $\mathbf{V}(b) \subseteq X'$. Given a distribution state μ, the evaluation of expressions under μ is defined inductively in Figure 3 and most of them are self-explanatory. Note that for an expression e and distribution state μ, the notation $[\![e]\!]_\mu$ stands for a distribution over the values $[\![e]\!]_\sigma$ for all $\sigma \in \lceil \mu \rceil$.

$$
\begin{array}{ll}
[\![n]\!]_\sigma & = n \\
[\![x]\!]_\sigma & = \sigma(x) \\
[\![f_m(a,...,a)]\!]_\sigma & = f_m([\![a]\!]_\sigma,...,[\![a]\!]_\sigma) \\
[\![\textbf{true}]\!]_\sigma & = \textbf{true} \\
[\![\textbf{false}]\!]_\sigma & = \textbf{false} \\
[\![P_m(a,...,a)]\!]_\sigma & = P_m([\![a]\!]_\sigma,...,[\![a]\!]_\sigma) \\
[\![b_1 \wedge b_2]\!]_\sigma & = [\![b_1]\!]_\sigma \wedge [\![b_2]\!]_\sigma \\
[\![\neg b]\!]_\sigma & = \neg([\![b]\!]_\sigma) \\
[\![a]\!]_\mu & = d_{\{[\![a]\!]_\sigma \mid \sigma \in \lceil \mu \rceil\}}, \text{ where } d([\![a]\!]_\sigma) = \sum_{\sigma' \in \lceil \mu \rceil, [\![a]\!]_{\sigma'} = [\![a]\!]_\sigma} \mu(\sigma') \\
[\![b]\!]_\mu & = d_{\{[\![b]\!]_\sigma \mid \sigma \in \lceil \mu \rceil\}}, \text{ where } d([\![b]\!]_\sigma) = \sum_{\sigma' \in \lceil \mu \rceil, [\![b]\!]_{\sigma'} = [\![b]\!]_\sigma} \mu(\sigma')
\end{array}
$$

Fig. 3: Evaluation for expressions

$$
\begin{array}{rl}
[\![\textbf{skip}]\!]_\mu = & \mu \\
[\![\textbf{abort}]\!]_\mu = & \varepsilon \\
[\![x := a]\!]_\mu = & \sum_{\sigma \in \lceil \mu \rceil} \mu(\sigma) \cdot \sigma[[\![a]\!]_\sigma / x] \\
[\![x :=_\$ d_A]\!]_\mu = & \sum_{a \in A} d(a) \cdot [\![x := a]\!]_\mu \\
[\![c_0; c_1]\!]_\mu = & [\![c_1]\!]_{[\![c_0]\!]_\mu} \\
[\![\textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1 \textbf{ fi}]\!]_\mu = & [\![c_0]\!]_{(b?\mu)} + [\![c_1]\!]_{(\neg b?\mu)} \\
[\![\textbf{while } b \textbf{ do } c \textbf{ od}]\!]_\mu = & \lim_{n \to \infty} [\![(\textbf{if } b \textbf{ then } c \textbf{ fi})^n; \textbf{if } b \textbf{ then abort fi}]\!]_\mu
\end{array}
$$

Fig. 4: Denotational semantics for commands

The denotation of a command c is a mapping of type $\textbf{SDist}[\Sigma_{\textbf{Var}}] \to \textbf{SDist}[\Sigma_{\textbf{Var}}]$, presented in Figure 4. Here $b?$ is a function from Δ to Δ, which satisfies

$$
b?\mu(\sigma) = \begin{cases} \mu(\sigma), & \textbf{iff} \quad [\![b]\!]_\sigma = \textbf{true} \\ 0, & \textbf{iff} \quad [\![b]\!]_\sigma = \textbf{false}. \end{cases}
$$

The **skip** command does not change any states and **abort** maps any input distribution state to the empty distribution state ε. The classical assignment statement changes the value of variable x in every state of the input distribution and the random assignment statement may assign any expression $a \in A$ to variable x with probability $d(a)$, where A is a finite set.

As usual, the denotation of $c_0; c_1$ is the composition of these commands. For the conditional statement, we need to divide the distribution state μ into two parts $b?\mu$ and $\neg b?\mu$, then combine the semantics c_0 and c_1 in the two parts, respectively. Following the

$$\begin{array}{ll} \text{(Deterministic formulas)} & P ::= \mathbf{true} \mid \mathbf{false} \mid P_m(a,...,a) \mid P \wedge P \mid \neg P \mid \exists x.P(x) \\ \text{(Probabilistic formulas)} & \phi ::= P \mid \oplus_{i\in I} p_i \phi_i \mid \oplus_{i\in I} \phi_i \mid \phi \odot \phi \mid \phi \wedge \phi \mid \phi \vee \phi \end{array}$$

Fig. 5: Syntax of assertions

work of Barthe et al. [5], we define the semantics of a loop (**while** b **do** c **od**) as the limit of its lower approximations, where the n-th lower approximation of $[\![\mathbf{while}\ b\ \mathbf{do}\ c\ \mathbf{od}]\!]_\mu$ is $[\![(\mathbf{if}\ b\ \mathbf{then}\ c\ \mathbf{fi})^n; \mathbf{if}\ b\ \mathbf{then\ abort\ fi}]\!]_\mu$. The statement (**if** b **then** c **fi**) is a shorthand for (**if** b **then** c **else skip fi**) and $c^{n+1} = c; c^n$ with $c^0 \equiv \mathbf{skip}$. By [5, Definition 7], the sequence $([\![(\mathbf{if}\ b\ \mathbf{then}\ c\ \mathbf{fi})^n; \mathbf{if}\ b\ \mathbf{then}\ \text{abort}\ \mathbf{fi}]\!]_\mu)_{n\in\mathbb{N}}$ is convergent; its limit exists because the sequence is strictly increasing and bounded.

4 Assertions

Assertions in our logic contain two categories of formulas: deterministic and probabilistic ones. The latter are characterized by distribution formulas, which draw inspiration from the work of [18] to describe the properties of distributions.

4.1 Syntax of Assertions

In Figure 5, we give the syntax of assertions. Deterministic formulas can be Boolean constants, built from arithmetic expressions by Boolean predicates, or formed by other deterministic formulas using connectives $\wedge$, $\neg$ and $\exists$. Probabilistic formulas include deterministic formulas, distribution formulas, and other probabilistic formulas built by some connectives such as $\odot$, $\wedge$ and $\vee$. Recall distribution formulas in the form $\oplus_{i\in I} p_i \phi_i$ from [18], which consist of various formulas ϕ_i composed by the connective $\oplus$, with each formula weighted by p_i. These weights satisfy the condition $\sum_{i\in I} p_i = 1$ and the index set I is finite. The formula $\oplus_{i\in I} \phi_i$ is a relaxed form of $\oplus_{i\in I} p_i \phi_i$ as the individual probabilities p_i are not important. The connective $\odot$ describes the properties of two independent sets of variables. Other connectives are standard.

4.2 Semantics of Assertions

We first formally describe the combining operation on distribution states following the Kripke resource semantics, where the set of possible worlds forms a partial, pre-order commutative monoid $\mathcal{M} = (M, \circ, \mathbf{I}, \sqsubseteq)$.

Definition 3 ([20]). *A (partial) Kripke resource monoid consists of a set M of possible worlds, a partial binary combining operation $\circ : M \times M \to M$, an element $\mathbf{I} \in M$, and a pre-order $\sqsubseteq$ on M, such that the monoid operation*

- *has identity $\mathbf{I}$: for all $x \in M$, we have $\mathbf{I} \circ x = x \circ \mathbf{I} = x$;*
- *is associative: $x \circ (y \circ z) = (x \circ y) \circ z$, where both sides are either defined and equal, or both undefined; and*

$$
\begin{array}{ll}
[\![\mathbf{true}]\!] & = \Sigma \\
[\![\mathbf{false}]\!] & = \emptyset \\
[\![P_m(a_1,...,a_n)]\!] & = \{\sigma \in \Sigma_X \mid P_m([\![a_1]\!]_\sigma,...,[\![a_n]\!]_\sigma) = \mathbf{true} \\
 & \quad \text{and} \bigcup_{i=1,...,n} \mathbf{V}(a_i) = X\} \\
[\![P_1 \wedge P_2]\!] & = [\![P_1]\!] \cap [\![P_2]\!] \\
[\![\neg P]\!] & = \Sigma \setminus [\![P]\!] \\
[\![\exists x.P(x)]\!] & = \{\sigma \mid \exists r \in \mathbb{Q}, \sigma \in [\![P[r/x]]\!]\}
\end{array}
$$

Fig. 6: Semantics of deterministic formulas

– *is compatible with pre-order: if $x \sqsubseteq y$ and $x' \sqsubseteq y'$ and if both $x \circ x'$ and $y \circ y'$ are defined, then $x \circ x' \sqsubseteq y \circ y'$.*

Proposition 1. *Let Δ be the set of all distribution states and $\mathbf{I}$ be the element in **SDist[$\Sigma_\emptyset$]** such that $\mathbf{I}(\mathbf{0}) = 1$. The partial binary combining operation $\circ$ is defined as:*

$$\mu \circ \mu' \triangleq \begin{cases} \mu \otimes \mu' & \textit{if } \mathrm{dom}(\mu) \cap \mathrm{dom}(\mu') = \emptyset \\ \text{undefined} & \textit{otherwise}. \end{cases}$$

The pre-order $\sqsubseteq$ is defined as

$$\mu \sqsubseteq \mu' \quad \textbf{iff} \quad \mathrm{dom}(\mu) \subseteq \mathrm{dom}(\mu') \text{ and } \pi_{\mathrm{dom}(\mu)}(\mu') = \mu.$$

Then $(\Delta, \circ, \mathbf{I}, \sqsubseteq)$ is a Kripke resource monoid.

The semantics of a deterministic formula P on partial states is denoted by $[\![P]\!]$, as shown in Figure 6. For a deterministic formula P and a partial state σ, we say $\sigma \models P$ if and only if $\sigma \in [\![P]\!]$. For a deterministic formula P and a distribution state μ, we say

$$\mu \models P \quad \textbf{iff} \quad \forall \sigma \in \lceil \mu \rceil,\ \sigma \models P.$$

The semantics of a probabilistic formula ϕ, denoted by $[\![\phi]\!]$, is given in Figure 7. The denotation of $\oplus_{i \in I} p_i \cdot \phi_i$ is the set including all the distribution states that are linear combinations of some μ_i with weight p_i, and each μ_i is in the denotation of ϕ_i. The denotation of $\oplus_{i \in I} \phi_i$ comprises all the distribution states that are in the denotation of $\oplus_{i \in I} p_i \cdot \phi_i$ for some arbitrary weights p_i such that their sum is 1. The denotation for the formula $\phi_1 \odot \phi_2$ contains all the up-closures of distribution states that dominate the combination of two distributions μ_1 and μ_2 with disjoint domains such that μ_1 and μ_2 are in the denotations of ϕ_1 and ϕ_2, respectively. Finally, other types of formulas are self-explanatory. The satisfaction relation between distribution state μ and probabilistic formula ϕ is defined as: $\mu \models \phi$ if and only if $\mu \in [\![\phi]\!]$.

We then propose the following important property to prove the soundness of our logic. The proof is proceeded by induction on the structure of ϕ.

Proposition 2. *For any probabilistic formula ϕ, its denotation $[\![\phi]\!]$ is a closed set.*

$$
\begin{aligned}
[\![P]\!] &= \{\mu \mid \mu \models P\} \\
[\![\oplus_{i\in I} p_i \cdot \phi_i]\!] &= \{\mu \mid \exists \mu_1 \cdots \exists \mu_m.\ [(\bigwedge_{i\in I} (\mu_i \models \phi_i) \wedge (\|\mu_i\| = \|\mu\|)) \\
&\qquad \wedge \mu = \sum_{i\in I} p_i \cdot \mu_i]\},\ \text{for } I = \{1,\ldots,m\} \\
[\![\oplus_{i\in I} \phi_i]\!] &= \{\mu \mid \exists p_1 \cdots p_m.\ \sum_{i\in I} p_i = 1 \wedge \mu \models \oplus_{i\in I} p_i \cdot \phi_i\} \\
[\![\phi_1 \odot \phi_2]\!] &= \{\mu \mid \exists \mu_1, \mu_2.\ (\mu_1 \circ \mu_2 \sqsubseteq \mu) \wedge \mu_1 \in [\![\phi_1]\!] \wedge \mu_2 \in [\![\phi_2]\!]\} \\
[\![\phi_1 \wedge \phi_2]\!] &= [\![\phi_1]\!] \cap [\![\phi_2]\!] \\
[\![\phi_1 \vee \phi_2]\!] &= [\![\phi_1]\!] \cup [\![\phi_2]\!]
\end{aligned}
$$

Fig. 7: Semantics of probabilistic formulas

5 Proof System

In this section, we present a proof system which consists of some rules to reason about the correctness of PIMP programs. As usual, we use Hoare triples of the form $\{\phi_1\}\ c\ \{\phi_2\}$, where c is a program fragment and ϕ_1, ϕ_2 are two assertions.

The rules are given in Figure 8. Some of them are adapted from [18] and we just give explanations for several new rules. The [RAssn] rule is used for the random assignment, which is different from rule [DAssn] used for the deterministic assignment. If the formula $\phi_a[a/x]$ holds at the current state for any a in a finite set A, the post-condition $\oplus_{a\in A} d(a) \cdot \phi_a$ holds after executing the random assignment. The [While] rule is derived from [18], where ϕ_0 and ϕ_1 in the invariant ϕ are the assertions presented in Figure 5. Note that the invariant ϕ represents a special form of distribution formulas with arbitrary probability coefficients. This indicates that we allow the probability coefficients of the distributions to differ after each loop, as long as the support sets remain the same. This weaker property enables us to handle a broader class of unbounded while loops.

The [OFrame] rule in our work is similar to the [QFrame] rule in the work of [18], with the distinction lying in the side condition. Let $\mathbf{V}(c)$ be the set of all variables occurring in program c and $\mathbf{MV}(c)$ the set of all variables modified by program c. Besides, given a formula ϕ, we denote the set of all free variables in ϕ by $\mathbf{FV}(\phi)$. The condition $\mathbf{FV}(\phi_3) \cap \mathbf{V}(c)$ stipulates that both the modified and read variables in c are restricted from appearing as free in assertion ϕ_3, while rule [QFrame] in [18] allows read variables in the side condition. This distinction from rule [QFrame] is required for handling the unique semantics of the $\odot$ connective in the current work. Here the assertion $\phi_1 \odot \phi_2$ denotes the independence relation between the variables in $\mathbf{FV}(\phi_1)$ and $\mathbf{FV}(\phi_2)$ for any assertions ϕ_1, ϕ_2. Consequently, even if the free variables in ϕ_3 are not modified by the program c, being read by c may still result in the non-independence between $\mathbf{FV}(\phi_2)$ and $\mathbf{FV}(\phi_3)$ after the execution of c. This point is illustrated further in Example 1.

Example 1. Let $\phi_1 = \mathbf{true}$, $\phi_2 = (x = 0) \oplus (x = 1)$, $\phi_3 = \frac{1}{2}(y = 0) \oplus \frac{1}{2}(y = 1)$ and let c be the program in Figure 9. We encounter a situation where $\{\phi_1\}\ c\ \{\phi_2\}$ holds, and

$$\frac{}{\{\phi\}\ \mathbf{skip}\ \{\phi\}}[\text{Skip}] \qquad \frac{}{\{\phi\}\ \mathbf{abort}\ \{\mathbf{false}\}}[\text{Abort}]$$

$$\frac{}{\{\phi[a/x]\}\ x := a\ \{\phi\}}[\text{DAssn}] \qquad \frac{}{\{\bigwedge_{a\in A}\phi_a[a/x]\}\ x :=_{\$} d_A\ \{\oplus_{a\in A} d(a)\cdot\phi_a\}}[\text{RAssn}]$$

$$\frac{\{\phi_0\}\ c_0\ \{\phi_1\} \quad \{\phi_1\}\ c_1\ \{\phi_2\}}{\{\phi_0\}\ c_0; c_1\ \{\phi_2\}}[\text{Seq}]$$

$$\frac{\{\phi_1\wedge b\}\ c_1\ \{\phi_1'\} \quad \{\phi_2\wedge\neg b\}\ c_2\ \{\phi_2'\}}{\{p(\phi_1\wedge b)\oplus(1-p)(\phi_2\wedge\neg b)\}\ \mathbf{if}\ b\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2\ \mathbf{fi}\ \{p\phi_1'\oplus(1-p)\phi_2'\}}[\text{Cond}]$$

$$\frac{\phi = (\phi_0\wedge b)\oplus(\phi_1\wedge\neg b) \quad \{\phi_0\wedge b\}\ c\ \{\phi\}}{\{\phi\}\ \mathbf{while}\ b\ \mathbf{do}\ c\ \mathbf{od}\ \{\phi_1\wedge\neg b\}}[\text{While}]$$

$$\frac{\phi_0\Rightarrow\phi_1 \quad \{\phi_1\}\ c\ \{\phi_2\} \quad \phi_2\Rightarrow\phi_3}{\{\phi_0\}\ c\ \{\phi_3\}}[\text{Conseq}]$$

$$\frac{\{\phi_1\}\ c\ \{\phi_1'\} \quad \{\phi_2\}\ c\ \{\phi_2'\}}{\{\phi_1\wedge\phi_2\}\ c\ \{\phi_1'\wedge\phi_2'\}}[\text{Conj}] \qquad \frac{\{\phi_1\}\ c\ \{\phi_1'\} \quad \{\phi_2\}\ c\ \{\phi_2'\}}{\{\phi_1\vee\phi_2\}\ c\ \{\phi_1'\vee\phi_2'\}}[\text{Disj}]$$

$$\frac{\{\phi_1\}\ c\ \{\phi_2\} \quad \mathbf{FV}(\phi_3)\cap\mathbf{V}(c)=\emptyset}{\{\phi_1\odot\phi_3\}\ c\ \{\phi_2\odot\phi_3\}}[\text{OFrame}] \qquad \frac{\{\phi_1\}\ c\ \{\phi_2\} \quad \mathbf{FV}(\phi_3)\cap\mathbf{MV}(c)=\emptyset}{\{\phi_1\wedge\phi_3\}\ c\ \{\phi_2\wedge\phi_3\}}[\text{Frame}]$$

$$\frac{\forall i\in I.\ \{\phi_i\}\ c\ \{\phi_i'\} \qquad \sum_{i\in I}p_i = 1}{\{\oplus_{i\in I}p_i\cdot\phi_i\}\ c\ \{\oplus_{i\in I}p_i\cdot\phi_i'\}}[\text{Sum}] \qquad \frac{\forall r.\ r\in\mathbb{Q}\ \wedge\ \{P(r)\}\ c\ \{\phi'\}}{\{\exists x.\ P(x)\}\ c\ \{\phi'\}}[\text{Exists}]$$

Fig. 8: Inferences rules for PIMP programs

additionally, $\mathbf{FV}(\phi_3)\cap\mathbf{MV}(c)=\emptyset$. Despite these conditions, the Hoare triple

$$\{\phi_1\odot\phi_3\}\ c\ \{\phi_2\odot\phi_3\}$$

is still invalid. This is because the variables x and y are clearly not independent. Actually, we get the Hoare triple $\{\phi_1\odot\phi_3\}\ c\ \{\frac{1}{2}(x=0\wedge y=0)\oplus\frac{1}{2}(x=1\wedge y=1)\}$ in Figure 9, where the post-condition is not equal to $\phi_2\odot\phi_3$.

The [Frame] rule adheres to the conventions of classical Hoare logic. Since the assertion $\phi_1\wedge\phi_2$ does not have any particular constraints between the variables in the assertions ϕ_1 and ϕ_2, the side condition of the free variables in ϕ_3 remains unmodified by the program c, which is enough. Rule [Exists] is inspired by the work of [17].

Figure 10 displays several rules for auxiliary reasoning. Some already appear in [18], we only explain the rules that are new in our work. The [OdotD] rule aids in distributing $\odot$ into the connective $\oplus$. The connective operator $\oplus$ is commutative and associative according to rules [OplusC] and [OplusA].

We write $\vdash\{\phi_1\}\ c\ \{\phi_2\}$ if the Hoare triple $\{\phi_1\}\ c\ \{\phi_2\}$ can be derived from the rules of our proof system. Semantically, the Hoare triple is valid, written as $\models\{\phi_1\}\ c\ \{\phi_2\}$, if for any distribution state μ, $\mu\models\phi_1$ implies $[\![c]\!]_\mu\models\phi_2$.

Theorem 1 (Soundness). *For any assertions ϕ_1, ϕ_2 and any program c, we have that $\vdash\{\phi_1\}\ c\ \{\phi_2\}$ implies $\models\{\phi_1\}\ c\ \{\phi_2\}$.*

$\{\mathbf{true} \odot \frac{1}{2}(y=0) \oplus \frac{1}{2}(y=1)\}$

$\{\frac{1}{2}(y=0) \oplus \frac{1}{2}(y=1)\}$ [OdotE]

if $(y=0)$ **then**

$x := 0;$

else

$x := 1;$

fi

$\{\frac{1}{2}(x=0 \wedge y=0) \oplus \frac{1}{2}(x=1 \wedge y=1)\}$ [Cond]

Fig. 9: Example 1

$$\frac{}{\phi \vdash \mathbf{true}}\,[\text{PT}] \qquad \frac{}{\phi \odot \mathbf{true} \dashv\vdash \phi}\,[\text{OdotE}]$$

$$\frac{}{(\oplus_{i\in I} p_i\phi_i) \odot (\oplus_{j\in J} q_j\phi_j) \vdash \oplus_{i\in I, j\in J} p_i q_j (\phi_i \odot \phi_j)}\,[\text{OdotD}]$$

$$\frac{}{\phi_1 \odot \phi_2 \dashv\vdash \phi_2 \odot \phi_1}\,[\text{OdotC}] \qquad \frac{}{\phi_1 \odot (\phi_2 \odot \phi_3) \dashv\vdash (\phi_1 \odot \phi_2) \odot \phi_3}\,[\text{OdotA}]$$

$$\frac{}{\phi_1 \odot \phi_2 \vdash \phi_1 \wedge \phi_2}\,[\text{OdotO}] \qquad \frac{}{\phi_1 \odot (\phi_2 \wedge \phi_3) \vdash (\phi_1 \odot \phi_2) \wedge (\phi_1 \odot \phi_3)}\,[\text{OdotOC}]$$

$$\frac{}{\phi_1 \oplus \phi_2 \dashv\vdash \phi_2 \oplus \phi_1}\,[\text{OplusC}] \qquad \frac{}{\phi_1 \oplus (\phi_2 \oplus \phi_3) \dashv\vdash (\phi_1 \oplus \phi_2) \oplus \phi_3}\,[\text{OplusA}]$$

$$\frac{}{p_0 \cdot P \oplus p_1 \cdot P \oplus p_2 \cdot P' \dashv\vdash (p_0 + p_1) \cdot P \oplus p_2 \cdot P'}\,[\text{OMerg}]$$

$$\frac{}{\oplus_{i\in I} p_i \cdot \phi_i \vdash \oplus_{i\in I} \phi_i}\,[\text{Oplus}] \qquad \frac{\forall i \in I,\ \phi_i \vdash \phi_i'}{\oplus_{i\in I} p_i \cdot \phi_i \vdash \oplus_{i\in I} p_i \cdot \phi_i'}\,[\text{OCon}]$$

Fig. 10: Rules for entailment reasoning

Proof. We prove by induction that all the rules in the proof system are sound. Here we only consider rule [While], which is the most difficult case.

Suppose $\phi = (\phi_0 \wedge b) \oplus (\phi_1 \wedge \neg b)$ and $\models \{\phi_0 \wedge b\}\ c\ \{\phi\}$. We aim to prove that $\models \{\phi\}$ **while** b **do** c **od** $\{\phi_1 \wedge \neg b\}$.

For any $n \geq 0$, we write **while**^{n} for the n-th iteration of the while loop, i.e.

$$(\textbf{if } b \textbf{ then } c \textbf{ fi})^n; \textbf{if } b \textbf{ then abort fi}.$$

We claim that if $\mu \models \phi$ then $[\![\textbf{while}^n]\!]_\mu \models \phi_1 \wedge \neg b$ for any $n \geq 0$. This can be proved by induction on n. Note that if $\mu \models \phi$ then there exist some p, μ_0 and μ_1 such that $\mu = p\mu_0 + (1-p)\mu_1$ with $\mu_0 \models \phi_0 \wedge b$ and $\mu_1 \models \phi_1 \wedge \neg b$.

– $n = 0$. In this case we have that

$$\begin{aligned}[\![\mathbf{while}^0]\!]_\mu &= [\![\mathbf{if}\ b\ \mathbf{then\ abort\ fi}]\!]_\mu \\ &= p[\![\mathbf{if}\ b\ \mathbf{then\ abort\ fi}]\!]_{\mu_0} + (1-p)[\![\mathbf{if}\ b\ \mathbf{then\ abort\ fi}]\!]_{\mu_1} \\ &= p\varepsilon + (1-p)\mu_1 \\ &= (1-p)\mu_1.\end{aligned}$$

Since $\mu_1 \models \phi_1 \wedge \neg b$, it implies that $(1-p)\mu_1 \models \phi_1 \wedge \neg b$.

– $n = k+1$. We infer that

$$\begin{aligned}[\![\mathbf{while}^{k+1}]\!]_\mu &= [\![\mathbf{if}\ b\ \mathbf{then}\ c\ \mathbf{fi};\ \mathbf{while}^k]\!]_\mu \\ &= p[\![\mathbf{if}\, b\, \mathbf{then}\, c\, \mathbf{fi};\ \mathbf{while}^k]\!]_{\mu_0} + (1-p)[\![\mathbf{if}\, b\, \mathbf{then}\, c\, \mathbf{fi};\ \mathbf{while}^k]\!]_{\mu_1} \\ &= p[\![\mathbf{while}^k]\!]_{[\![c]\!]_{\mu_0}} + (1-p)[\![\mathbf{while}^k]\!]_{\mu_1} \\ &= p[\![\mathbf{while}^k]\!]_{[\![c]\!]_{\mu_0}} + (1-p)\mu_1.\end{aligned}$$

Note that $[\![c]\!]_{\mu_0} \models \phi$. Therefore, it follows from the induction hypothesis that

$$[\![\mathbf{while}^k]\!]_{[\![c]\!]_{\mu_0}} \models \phi_1 \wedge \neg b.$$

As in the last case, we have $(1-p)\mu_1 \models \phi_1 \wedge \neg b$. Then we immediately have that $[\![\mathbf{while}^{k+1}]\!]_\mu \models \phi_1 \wedge \neg b$.

Thus we have completed the proof of the claim. Finally, by Proposition 2, we know that the denotation $[\![\phi_1 \wedge \neg b]\!]$ for $\phi_1 \wedge \neg b$ is a closed set. This implies that if a sequence $\{\nu_n\}$ of distributions is in the set $[\![\phi_1 \wedge \neg b]\!]$ and the sequence has a limit $\lim_{n\to\infty} \nu_n$, then the limit is still in $[\![\phi_1 \wedge \neg b]\!]$. So we can obtain $[\![\mathbf{while}]\!]_\mu = \lim_{n\to\infty}[\![\mathbf{while}^n]\!]_\mu \models \phi_1 \wedge \neg b$. □

6 Case Studies

6.1 A Simple Program

Recall the program $Prog_1$ given in the introduction, which has an unbounded loop. The correctness of this program can be specified by the following Hoare triple:

$$\{\mathbf{true}\}\ Prog_1\ \{x = 5\}.$$

We outline the correctness proof in Figure 11. Although this example is straightforward, it effectively illustrates the advantages of our logic when reasoning about loops involving an unknown number of iterations.

6.2 Bayesian Network

First of all, we use the symbol $\mathbb{P}_\mu(P)$ to represent the probability of deterministic formula P being satisfied by μ, and $\mathbb{P}_\mu(P|Q)$ to represent the conditional probability of formula P given Q. That is,

$$\mathbb{P}_\mu(P) \triangleq \frac{1}{\|\mu\|} \sum_{\sigma \in \lceil \mu \rceil} \{\mu(\sigma) \mid \sigma \models P\},$$

$$\mathbb{P}_\mu(P|Q) \triangleq \frac{\mathbb{P}_\mu(P \wedge Q)}{\mathbb{P}_\mu(Q)}.$$

$\{\mathbf{true}\}$
$\{0=0\}$ [Conseq]
$x := 0;$
$\{x=0\}$ [DAssn]
$\{x<5 \oplus x=5\}$ [Conseq]
while $x<5$ **do**
  $\{x<5\}$ [Conseq]
  $\{x<5 \odot \mathbf{true}\}$ [OdotE]
  $\Longrightarrow \{0=0 \wedge 1=1\}$ [Conseq]
  $n :=_{\$} \{\frac{1}{2}\cdot 0 + \frac{1}{2}\cdot 1\};$
  $\Longleftarrow \{\frac{1}{2}\cdot(n=0) \oplus \frac{1}{2}\cdot(n=1)\}$ [RAssn]
  $\{(x<5)\odot(\frac{1}{2}\cdot n=0 \oplus \frac{1}{2}\cdot n=1)\}$ [OFrame]
  $\{\frac{1}{2}((x<5)\odot n=0)\oplus\frac{1}{2}((x<5)\odot n=1)\}$ [OdotD]
  $\{\frac{1}{2}((x<5)\wedge n=0)\oplus\frac{1}{2}((x<5)\wedge n=1)\}$ [OdotO OCon]
  if $(n=1)$ **then**
    $\{(x<5)\wedge n=1\}$ [Conseq]
    $\{(x+1<5)\oplus(x+1=5)\}$ [Conseq]
    $x := x+1;$
    $\{(x<5\oplus x=5)\}$ [DAssn]
  else
    $\{(x<5)\wedge n=0\}$ [Conseq]
    $\{x<5\}$ [Conseq]
    skip;
    $\{x<5\}$ [Skip]
  fi
  $\{\frac{1}{2}(x<5\oplus x=5)\oplus\frac{1}{2}(x<5)\}$ [Cond]
  $\{(x<5\oplus x=5)\oplus(x<5)\}$ [Conseq Oplus]
  $\{(x<5)\oplus(x=5)\}$ [OplusC OplusA OMerg]
od
$\{x=5\}$ [While]

Fig. 11: Proof outline for program $Prop_1$

Lemma 1. *Let μ be a distribution state, if $\mu \models \oplus_{i\in I} p_i \cdot P_i$ for some p_i and P_i, then $\mathbb{P}_\mu(P_i) \geq p_i$.*

For reasoning about Bayesian networks at the source code level, Batz et al. translated the Bayesian network, along with observations, into the *Bayesian Network Language* (BNL) in [11]. Then they derived dedicated proof rules to determine exact expected outcomes and runtimes of such loops based on the weakest pre-expectation calculus. We borrow the *Bayesian Network* (BN) depicted in Figure 12 from [11]. The network consists of four binary random variables, including exam difficulty (D), student preparation (P), achieved grade (G), and resulting mood (M), with inherent dependencies shown in Figure 12. Each node x_v has a conditional probability table, with rows for the random variable values and columns for the corresponding probabilities of x_v.

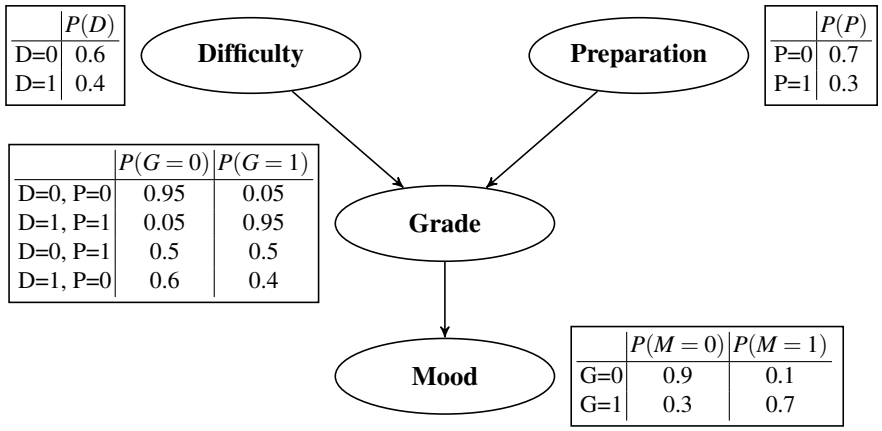

Fig. 12: A Bayesian network

We make some minor adjustments to the syntax of BNL fragment to align it with our language, and the entire program is named *BN*, as shown in Figure 13. We can still use this *BN* to compute the probability of a student having a bad mood ($x_M = 0$), after receiving a bad grade ($x_G = 0$) for an easy exam ($x_D = 0$), given that he was well-prepared ($x_P = 1$).

In [11] the conditional probability given an observation on $P = 1$ is computed as:

$$\mathbb{P}(D = 0, G = 0, M = 0 | P = 1) = \frac{\mathbb{P}(D = 0, G = 0, M = 1, P = 1)}{\mathbb{P}(P = 1)}.$$

However, according to our semantics, this property can be expressed by

$$\{\mathbf{true}\}\ BN\ \{x_P = 1 \wedge F_1\},$$

where F_1 is

$$\begin{aligned}
&(\tfrac{0.0054}{0.3}(x_G = 0 \wedge x_D = 1 \wedge x_M = 0) \oplus \tfrac{0.0006}{0.3}(x_G = 0 \wedge x_D = 1 \wedge x_M = 1)\\
&\oplus \tfrac{0.0342}{0.3}(x_G = 1 \wedge x_D = 1 \wedge x_M = 0) \oplus \tfrac{0.0798}{0.3}(x_G = 1 \wedge x_D = 1 \wedge x_M = 1)\\
&\oplus \tfrac{0.081}{0.3}(x_G = 0 \wedge x_D = 0 \wedge x_M = 0) \oplus \tfrac{0.009}{0.3}(x_G = 0 \wedge x_D = 0 \wedge x_M = 1)\\
&\oplus \tfrac{0.027}{0.3}(x_G = 1 \wedge x_D = 0 \wedge x_M = 0) \oplus \tfrac{0.063}{0.3}(x_G = 1 \wedge x_D = 0 \wedge x_M = 1)).
\end{aligned}$$

$x_P := 0;$
while $x_P \neq 1$ **do**
  $x_D :=_{\$} \{0.6 \cdot 0 + 0.4 \cdot 1\};$
  $x_P :=_{\$} \{0.7 \cdot 0 + 0.3 \cdot 1\};$
  if $(x_D = 0 \wedge x_P = 0)$
    $x_G :=_{\$} \{0.95 \cdot 0 + 0.05 \cdot 1\};$
  else if $(x_D = 1 \wedge x_P = 1)$
    $x_G :=_{\$} \{0.05 \cdot 0 + 0.95 \cdot 1\};$
  else if $(x_D = 0 \wedge x_P = 1)$
    $x_G :=_{\$} \{0.5 \cdot 0 + 0.5 \cdot 1\};$
  else if $(x_D = 1 \wedge x_P = 0)$
    $x_G :=_{\$} \{0.6 \cdot 0 + 0.5 \cdot 1\};$
  fi
  if $(x_G = 0)$
    $x_M :=_{\$} \{0.9 \cdot 0 + 0.1 \cdot 1\};$
  else if $(x_G = 1)$
    $x_M :=_{\$} \{0.3 \cdot 0 + 0.7 \cdot 1\};$
  fi
od

Fig. 13: Probabilistic program *BN*

Then by Lemma 1, we get $\mathbb{P}_\mu(x_D = 0, x_G = 0, x_M = 0) \geq \frac{0.081}{0.3}$ when $x_P = 1$.

With our program logic, we can reason about the correctness of other interesting examples such as encodings of cryptographic schemes including one-time pad, private information retrieval, and oblivious transfer. They are omitted due to lack of space.

7 Related Work

In addition to the approaches mentioned above, there are many other well-established methods for the verification of probabilistic programs, such as martingale-based methods [14, 15, 48] or symbolic integration-based methods [44, 22, 12]. Here, we mainly focus on probabilistic assertions within probabilistic programs.

7.1 Comparison of assertion-based work

In this sub-section, we compare our work with the most relevant assertion-based approaches in the literature, specifically focusing on Ellora [5], PSL [9], and Lilac [38].

Barthe et al. introduce ELLORA, a novel assertion logic that integrates probabilistic independence and other distribution law properties into first-order logic. This extension enables assertions to directly describe the probabilities of events and offers comparable expressive power to expectation-based approaches. However, Barthe et al. also recognize ELLORA's limitations in handling conditional control flow in independence logic. To address this issue, they later introduce PSL, a refined logic that draws from separation logic and treats independence in a structural manner. The separating conjunction

in PSL models probabilistic independence by combining distributions over disjoint domains of random variables, with the restriction that random variables cannot appear on both sides of the conjunction. Moreover, the frame rule imposes extra side conditions to capture the program's data-flow properties, which can make the application of the frame rule cumbersome. In contrast, the [OFrame] rule in our work simplifies these conditions to a simpler constraint that is easier to verify while still sufficient for the verification of relevant examples. Li et al. introduce Lilac, a modal separation logic that innovatively reinterprets the separating conjunction by combining probability spaces, analogous to the disjoint union of heap fragments in ordinary separation logic. This work models probabilistic systems using standard objects from probability theory, where the measurability of a random variable mirrors ownership in heap logic. Furthermore, they expand the application scope of probabilistic separation logic by incorporating conditional independence.

7.2 pRHL

Another method for reasoning about the properties of probabilistic programs involves establishing logical relations to recast reasoning problems as program equivalence problems. The equivalence is characterized using a step-indexed biorthogonal logical relation constructed over an operational semantics.

One prominent work in this trend is the relational Hoare logic pRHL proposed by Barthe et al. [7]. The logic compares a pair of commands from a given pre-relation to a given post-relation of sub-distributions over states in the language of pWHILE programs. It is widely used for verifying cryptographic protocols, such as private information retrieval and multi-party computation. While pRHL is very useful for verifying the equivalence between two commands, we can also analyze program equivalence using our Hoare logic. Our definition of program equivalence is more targeted at state transitions caused by variables modified by program fragments. Furthermore, logical relations are less well-suited for proving intricate post-conditions such as the convergence of distributions and the specific probability value of the occurrence of an event. Another restriction of pRHL is that the coupled programs must execute synchronously. This is a very strict requirement, though in [24] they propose a way to relax the requirement by introducing the presampling steps and storing the results on a tape.

7.3 Conditional independence

Li et al. [38] propose a logic that supports conditioning and continuous random variables, integrating a substructural approach to independence. They add a new modality to handle conditional independence, which is based on disintegration theory. The verification process involves expressing probabilistic programs in a monadic style, similar to Haskell's style of pure functional language.

Several prior systems have explored the concept of conditional independence. For example, Batz et al. [11] employ the weakest precondition style to analyze both expected outcomes and expected runtimes of a syntactic fragment of pGCL, which is called the Bayesian Network Language (BNL). Simultaneously, Barthe et al. [6] investigate a relational type system named PrivInfer, designed for Bayesian inference within

a functional programming language. In another direction, Bao et al. extend the standard logic of bunched implications, which underlies separation logic, to handle conditional independence through the introduction of doubly-bunched implications [4]. On the foundational side, Ackerman et al. [1] delve into computability issues for (conditional) independence, motivated in part by exchangeable sequences closely related to independence. Language-based investigations of exchangeable sequences are explored by Staton et al. [47].

8 Conclusion and Future Work

Our work builds upon the previous work [18]. Initially, we developed a method for the local reasoning of quantum programs. However, recognizing the broader applicability of our methodology, we now adapt it to the probabilistic setting. Proposition 2 is newly added, which is not explicitly stated in [18], but is crucial to prove the soundness of the [While] rule. Besides, since we are no longer dealing with quantum states, the semantics of assertions involving separation conjunction is changed to represent probabilistic independence, which also necessitated a change in the side condition of the [OFrame] rule. Moreover, we now allow conjunction and disjunction of distribution formulas, which is not the case in [18] because the syntax of assertions given there is more restricted. In summary, we have introduced a probabilistic separation logic tailored for handling unbounded loops, with distribution formulas serving as useful invariants. Our interpretation of separating conjunction aligns with the concept of probabilistic independence, enhancing the versatility of our logic. We have proved the soundness of the logic and its effectiveness is illustrated through the formal verification of several intricate examples from probabilistic inference of Bayesian network to cryptographic schemes.

As to the future work, we plan to analyze more randomized algorithms and cryptographic protocols, addressing specific challenges in these domains through the proposed logic. Additionally, further theoretical development and embedding the logic into a proof assistant are also being considered.

References

1. Ackerman, N.L., Avigad, J., Freer, C.E., Roy, D.M., Rute, J.M.: On the computability of graphons. arXiv preprint arXiv:1801.10387 (2018)
2. Andova, S.: Probabilistic process algebra. Phd thesis, Mathematics and Computer Science (2002). https://doi.org/10.6100/IR561343
3. Baier, C., Kwiatkowska, M., Norman, G.: Computing probability lower and upper bounds for LTL formulae over sequential and concurrent Markov chains. Tech. rep., University of Birmingham (1998)
4. Bao, J., Docherty, S., Hsu, J., Silva, A.: A bunched logic for conditional independence. In: Proceedings of the 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2021). pp. 1–14. IEEE (2021)
5. Barthe, G., Espitau, T., Gaboardi, M., Grégoire, B., Hsu, J., Strub, P.Y.: An assertion-based program logic for probabilistic programs. In: Proceedings of the 27th European Symposium on Programming Programming Languages and Systems (ESOP 2018). pp. 117–144. Springer (2018)

6. Barthe, G., Farina, G.P., Gaboardi, M., Arias, E.J.G., Gordon, A., Hsu, J., Strub, P.Y.: Differentially private bayesian programming. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 68–79 (2016)
7. Barthe, G., Grégoire, B., Zanella Béguelin, S.: Formal certification of code-based cryptographic proofs. In: Proceedings of the 36th annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 90–101 (2009)
8. Barthe, G., Grégoire, B., Zanella Béguelin, S.: Probabilistic relational Hoare logics for computer-aided security proofs. In: Proceedings of the 11th International Conference on Mathematics of Program Construction. pp. 1–6. Springer (2012)
9. Barthe, G., Hsu, J., Liao, K.: A probabilistic separation logic. Proceedings of the ACM on Programming Languages **4**(POPL), 1–30 (2019)
10. Barthe, G., Köpf, B., Olmedo, F., Zanella Beguelin, S.: Probabilistic relational reasoning for differential privacy. In: Proceedings of the 39th annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 97–110 (2012)
11. Batz, K., Kaminski, B.L., Katoen, J.P., Matheja, C.: How long, O Bayesian network, will I sample thee? a program analysis perspective on expected sampling times. In: Proceedings of the 27th European Symposium on Programming Languages and Systems (ESOP 2018). pp. 186–213. Springer (2018)
12. Beutner, R., Ong, C.H.L., Zaiser, F.: Guaranteed bounds for posterior inference in universal probabilistic programming. In: Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation. pp. 536–551 (2022)
13. Chadha, R., Cruz-Filipe, L., Mateus, P., Sernadas, A.: Reasoning about probabilistic sequential programs. Theoretical Computer Science **379**(1-2), 142–165 (2007)
14. Chakarov, A., Sankaranarayanan, S.: Probabilistic program analysis with martingales. In: Computer Aided Verification: 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings 25. pp. 511–526. Springer (2013)
15. Chatterjee, K., Novotnỳ, P., Žikelić, Đ.: Stochastic invariants for probabilistic termination. In: Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages. pp. 145–160 (2017)
16. De Alfaro, L.: Formal verification of probabilistic systems. Stanford university (1998)
17. Den Hartog, J., de Vink, E.P.: Verifying probabilistic programs using a Hoare like logic. International Journal of Foundations of Computer Science **13**(03), 315–340 (2002)
18. Deng, Y., Wu, H., Xu, M.: Local reasoning about probabilistic behaviour for classical-quantum programs. In: Proceedings of the 25th International Conference Verification, Model Checking, and Abstract Interpretation (VMCAI 2024). Lecture Notes in Computer Science, vol. 14500, pp. 163–184. Springer (2024)
19. DArgenio, P.R., Katoen, J.P., Brinksma, E.: General purpose discrete event simulation using. In: Proc. of 6th International Workshop on Process Algebras and Performance Modeling, PAPM. vol. 98, pp. 85–102 (1998)
20. Galmiche, D., Mery, D., Pym, D.: The semantics of BI and resource tableaux. Mathematical Structures in Computer Science **15**(6), 10331088 (2005)
21. Gehr, T., Misailovic, S., Tsankov, P., Vanbever, L., Wiesmann, P., Vechev, M.: Bayonet: probabilistic inference for networks. ACM SIGPLAN Notices **53**(4), 586–602 (2018)
22. Gehr, T., Misailovic, S., Vechev, M.: Psi: Exact symbolic inference for probabilistic programs. In: Computer Aided Verification: 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part I 28. pp. 62–83. Springer (2016)
23. Ghahramani, Z.: Probabilistic machine learning and artificial intelligence. Nature **521**(7553), 452–459 (2015)
24. Gregersen, S.O., Aguirre, A., Haselwarter, P., Tassarotti, J., Birkedal, L.: Asynchronous probabilistic couplings in higher-order separation logic. Proc. ACM Program. Lang. **8**(POPL) (2024)

25. Gretz, F., Katoen, J.P., McIver, A.: Operational versus weakest pre-expectation semantics for the probabilistic guarded command language. Performance Evaluation **73**, 110–132 (2014)
26. Hartog, D.J.J.: Comparative semantics for a process language with probabilistic choice and non-determinism. Vrije Universitt (1998)
27. den Hartog, J.I.: Probabilistic extensions of semantical models. Ph.D. thesis, Vrije Universiteit Amsterdam (2002)
28. den Hartog, J.I., de Vink, E.P.: Mixing up nondeterminism and probability: A preliminary report. Electronic Notes in Theoretical Computer Science **22**, 88–110 (1999)
29. Hermanns, H.: Interactive Markov chains. Springer (2002)
30. Holtzen, S., Junges, S., Vazquez-Chanlatte, M., Millstein, T., Seshia, S.A., Van den Broeck, G.: Model checking finite-horizon Markov chains with probabilistic inference. In: International Conference on Computer Aided Verification. pp. 577–601. Springer (2021)
31. Hurd, J.: Formal verification of probabilistic algorithms. Tech. rep., University of Cambridge, Computer Laboratory (2003)
32. Kaminski, B.L.: Advanced weakest precondition calculi for probabilistic programs. Ph.D. thesis, RWTH Aachen University (2019)
33. Kaminski, B.L., Katoen, J.P., Matheja, C.: Inferring covariances for probabilistic programs. In: International Conference on Quantitative Evaluation of Systems. pp. 191–206. Springer (2016)
34. Kaminski, B.L., Katoen, J.P., Matheja, C., Olmedo, F.: Weakest precondition reasoning for expected runtimes of randomized algorithms. Journal of the ACM (JACM) **65**(5), 1–68 (2018)
35. Kozen, D.: Semantics of probabilistic programs. In: 20th Annual Symposium on Foundations of Computer Science (SFCS 1979). pp. 101–114. IEEE (1979)
36. Kozen, D.: A probabilistic pdl. In: Proceedings of the fifteenth annual ACM symposium on Theory of computing. pp. 291–297 (1983)
37. Lee, E.A., Seshia, S.A.: Introduction to embedded systems: A cyber-physical systems approach. MIT press (2016)
38. Li, J.M., Ahmed, A., Holtzen, S.: Lilac: a modal separation logic for conditional probability. Proceedings of the ACM on Programming Languages **7**(PLDI), 148–171 (2023)
39. McIver, A., Morgan, C.: Abstraction, refinement and proof for probabilistic systems. Springer (2005)
40. Motwani, R., Raghavan, P.: Randomized algorithms. Cambridge university press (1995)
41. Ngo, V.C., Carbonneaux, Q., Hoffmann, J.: Bounded expectations: resource analysis for probabilistic programs. ACM SIGPLAN Notices **53**(4), 496–512 (2018)
42. Ramshaw, L.H.: Formalizing the analysis of algorithms. Ph.D. thesis, Stanford University (1979)
43. Rand, R., Zdancewic, S.: VPHL: A verified partial-correctness logic for probabilistic programs. Electronic Notes in Theoretical Computer Science **319**, 351–367 (2015)
44. Sankaranarayanan, S., Chakarov, A., Gulwani, S.: Static analysis for probabilistic programs: inferring whole program properties from finitely many paths. In: Proceedings of the 34th ACM SIGPLAN conference on Programming language design and implementation. pp. 447–458 (2013)
45. Segala, R.: Modeling and verification of randomized distributed real-time systems. Ph.D. thesis, MIT (1996)
46. Smolka, S., Kumar, P., Kahn, D.M., Foster, N., Hsu, J., Kozen, D., Silva, A.: Scalable verification of probabilistic networks. In: Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation. pp. 190–203 (2019)
47. Staton, S., Stein, D., Yang, H., Ackerman, N.L., Freer, C.E., Roy, D.M.: The Beta-Bernoulli process and algebraic effects. In: Proceedings of the 45th International Colloquium on Automata, Languages, and Programming (ICALP 2018). LIPIcs, vol. 107, pp. 141:1–141:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2018)

48. Wang, P., Fu, H., Goharshady, A.K., Chatterjee, K., Qin, X., Shi, W.: Cost analysis of nondeterministic probabilistic programs. In: Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation. pp. 204–220 (2019)
49. Winskel, G.: The Formal Semantics of Programming Languages - An Introduction. The MIT Press (1993)
50. Wu, S.H., Smolka, S.A., Stark, E.W.: Composition and behaviors of probabilistic I/O automata. Theoretical Computer Science **176**(1-2), 1–38 (1997)