# Probabilistic Horn Clause Verification

Aws Albarghouthi

University of Wisconsin–Madison

**Abstract.** Constrained Horn clauses have proven to be a natural intermediate language for logically characterizing program semantics and reasoning about program behavior. In this paper, we present *probabilistically constrained Horn clauses* (PCHC), which incorporate *probabilistic variables* inside otherwise traditional constrained Horn clauses. PCHC enable reasoning about probabilistic programs by encoding them as Horn clauses. Encoding probabilistic program semantics as PCHC allows us to seamlessly handle procedure calls and recursion, as well as angelic and demonic forms of nondeterminism. We formalize PCHC semantics and present a verification algorithm that can prove probabilistic safety properties of programs. We present an implementation and evaluation of our approach on a number of probabilistic programs and properties.

## 1 Introduction

*Constrained Horn Clauses* have emerged as a natural logical formalism for stating a wide spectrum of program verification and synthesis problems and solving them automatically with generic Horn clause solvers [5]. For instance, given a sequential program $P$ and a safety property $\varphi$, we can construct a set of *recursive* Horn clauses whose solution is a *safe inductive invariant* that entails correctness of $P$ with respect to $\varphi$. A key advantage in this two-tiered methodology is the clear dichotomy between the syntactic object, the program $P$, and its semantic interpretation, encoded logically as a set of Horn clauses. Thus, the generic Horn clause solver is completely unaware of the programming language of $P$. Indeed, as Grebenshchikov et al. [25] have shown, a simple Horn clause solver can be the target of a range of program models and correctness properties—including concurrent programs and liveness properties over infinite domains.

To handle richer programs and properties, such as termination and temporal properties, researchers have enriched traditional Horn clauses with additional features, such as quantifier alternation [6, 4]. In this paper, we present an extension of constrained Horn clauses to the probabilistic setting, in which variables draw their values from probability distributions. Doing so, we enable reasoning about safety properties of *probabilistic programs*: standard programs with probabilistic assignments. Probabilistic programs are used in a plethora of applications, e.g., modeling biological systems [28, 29], cognitive processes [23], cyber-physical systems [42], programs running on approximate hardware [41, 7], and randomized algorithms like privacy-preserving ones [16], amongst many others. Thus, by extending Horn clauses and their solvers to the probabilistic setting, we expand their applicability to many new domains.

We define the semantics of probabilistic Horn clauses as a probability distribution over the set of *ground derivations*. There are two key high-level advantages to reasoning about probabilistic programs in terms of probabilistic Horn clauses. The first advantage of our formulation is that it enables us to define Horn clauses over any first-order theory with an appropriate probability measure. In the simplest case, we can have propositional Horn clauses, where variables draw their values from Bernoulli distributions. In more advanced cases, for example, we can have real arithmetic formulas where variables are drawn from, e.g., Gaussian or Laplacian distributions. This provides a flexible means for encoding program semantics with appropriate first-order theories, as is standard in many hardware and software verification tools. The second advantage we gain from Horn clauses is that we can naturally encode loops, procedures, and recursion. Thus, our Horn clauses can encode probabilistic programs with recursion, a combination that is rarely addressed in the literature. Further, we extend our probabilistic semantics with *angelic* and *demonic non-determinism*. This allows us to reason about variables that receive non-deterministic values, for example, in programs with calls to unknown libraries. Angelic and demonic non-determinism allow us to compute best- and worst-case probabilities for an event.

The probabilistic safety properties (*queries*) we would like to prove about our Horn clauses are of the form, e.g., $\mathbb{P}[Q(x) \rightarrow x > 0] > 0.9$, which specifies that the probability of deriving a positive value in the relation $Q$ (which might encode, say, the return values of the program) is more than 0.9. To prove probabilistic properties, we present a verification algorithm that, like its non-probabilistic counterparts [37, 25, 24, 27], iteratively unrolls recursive Horn clauses to generate an under-approximating set of Horn clauses that encodes a *subset of the total set of possible derivations*. To compute the probability of an event in the under-approximation, we demonstrate how to encode the problem as a *weighted model counting problem* over formulas in a first-order theory [12, 11]. *The algorithm iteratively considers deeper and deeper unrollings—maintaining a lower and an upper bound on the probability of interest—until it is able to prove or disprove the property of interest.*

From a problem formulation perspective, our approach can be seen as an extension of Chistikov et al.'s probabilistic inference through model counting [12] to recursive sets of constraints. From an algorithmic perspective, one can view our approach as an extension of Sankaranarayanan et al.'s algorithm [42] to programs with recursion and non-determinism.

**Contributions** To summarize, this paper makes the following contributions:

- We present *probabilistically constrained Horn clauses* (PCHC) and define their semantics as a probability distribution over of the set of derivation sequences. Our formulation allows us to encode probabilistic safety verification problems over probabilistic programs that contain procedures and recursion.
- We extend the semantics of PCHC to encode angelic and demonic forms of non-determinism, following the semantics used by Chistikov et al. [12]. In

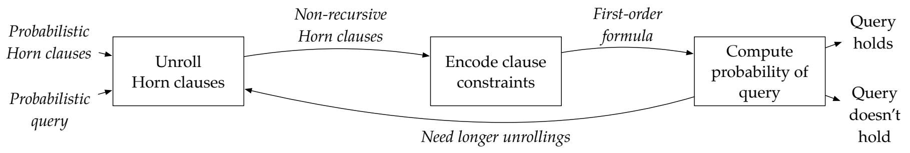

**Fig. 1.** Overview of proposed approach

the case where all variables angelically draw their values and there is no probabilistic choice, PCHC are equivalent to CHC.

- We present a verification algorithm for proving or disproving probabilistic reachability properties. Our algorithm iteratively considers larger and larger under-approximations of the Horn clauses and reduces the verification problem to weighted model counting.
- We present an implementation and evaluation of our approach on a number of probabilistic programs and properties. Our results demonstrate the utility of our approach at handling probabilistic programs with rich features such as non-determinism and recursion.

## 2 Overview

We illustrate our technique (Figure 1) on two simple examples.

**Recursive program** Consider the illustrative program in Figure 2(a). This is a recursive function that samples a real value for `x` from a Gaussian distribution (with mean 0 and standard deviation 10). If the value of `x` is negative, it recursively calls itself; otherwise, it returns `x`. The program *almost always* terminates—i.e., terminates with probability 1—and always returns a positive value.

Figure 2(b) shows a recursive Horn-like encoding of the program as a predicate $f$. There are two things to note here: First, we allow disjunctions in the body of the clause (the left hand side of the implication $\rightarrow$).[1] Second, on the left hand side of the bar ($|$) we list probabilistic variables and their corresponding probability distributions. In this case, we have $x \sim \texttt{gauss}(0, 10)$, indicating the value of the real-valued variable $x$ is drawn from a normal distribution. Observe how our clause has both probabilistic variables ($x$) and traditional ones ($y$ and $r$). In the absence of probabilistic variables, the semantics are exactly those of constrained Horn clauses.

Suppose we want to prove that the program returns a value greater than 5 with probability greater than 0.3. This is a *probabilistic safety property*, which we encode as a query of the form $\mathbb{P}[f(r) \rightarrow r > 5] > 0.3$. To prove that this query holds, we proceed as illustrated in Figure 1.

[1] While in the non-probabilistic setting we can represent the function by two clauses (one representing the base case and one the recursive call), we need to combine the two clauses in the probabilistic setting. See Section 6 for a detailed explanation.

```
fun f():
  x ~ gauss(0,10)
  if (x >= 0) ret x
  ret f()
```

(a)

$$x \sim \mathtt{gauss}(0,10) \left| \begin{array}{l} \quad x \geqslant 0 \wedge r = x \\ \vee\, x < 0 \wedge f(y) \wedge r = y \end{array} \right. \longrightarrow f(r)$$

(b)

**Fig. 2.** Example probabilistic program and its Horn clause encoding

**Unrolling Horn clauses** We begin by unrolling the recursive Horn clause into a set of non-recursive clauses. This is analogous to fixing the depth of the stack in the program. The process is standard top-down unrolling, beginning with the predicate that appears in the query—this is similar to what is implemented in constrained Horn solvers [37, 24].

Suppose we unroll to depth 1, that is, we allow only a single recursive call. We arrive at the following two non-recursive clauses:

$$C_1 : \quad x \sim \mathtt{gauss}(0,10) \left| \begin{array}{l} \quad x \geqslant 0 \wedge r = x \\ \vee\, x < 0 \wedge f'(r) \end{array} \right. \longrightarrow f(r) \tag{1}$$

$$C_2 : \quad x' \sim \mathtt{gauss}(0,10) \left| \begin{array}{l} \quad x' \geqslant 0 \wedge r' = x' \\ \vee\, x' < 0 \wedge \mathit{false} \end{array} \right. \longrightarrow f'(r') \tag{2}$$

Observe how the body of $C_1$ refers to $f'(r)$ and $C_2$ defines the predicate $f'$. The second clause, effectively, encodes a fresh clone of the function $f$. Observe also the *false* in the body of $C_2$; this indicates that no more recursive calls can be made.

**Encoding Horn clauses** Unrolling Horn clauses is effectively producing an *underapproximation* of a program's executions. Thus, if we compute the probability that the non-recursive clauses satisfy $f(r) \to r > 5$, we get a *lower bound* on the actual probability. Similarly, by computing the probability of the negation of the event, i.e., $f(r) \to r \leqslant 5$, we can derive an *upper bound* on the actual probability. (We formalize this in Section 4.) For illustration, we show how to compute a lower bound on the probability that the query holds by encoding non-recursive clauses as a *model counting* problem.

Our encoding is analogous to that used by constrained Horn solvers. The result is as follows, where $\varphi_i$ encodes clause $C_i$:

$$\varphi_1 \equiv (x \geqslant 0 \wedge r = x) \vee (x < 0 \wedge b) \qquad \varphi_2 \equiv b \Rightarrow (x' \geqslant 0 \wedge r' = x')$$

Note the introduction of the Boolean variable $b$, which indicates whether the recursive call is taken or not.

**Probability computation** Now, to compute the probability that $r > 5$, we construct the formula:

$$\varphi \equiv \exists b, r, r'.\, \varphi_1 \wedge \varphi_2 \wedge r = r' \wedge r > 5$$

```
fun q()
  flag = nondet()
  x ~ gauss(0,10)
  if (flag) x = x + 5
  else x = x - 5
  ret x
```

(a)

$$x \sim \mathtt{gauss}(0,10) \left| \begin{array}{l} \quad \mathit{flag} \wedge r = x + 5 \\ \vee \neg \mathit{flag} \wedge r = x - 5 \end{array} \right. \longrightarrow q(r)$$

(b)

**Fig. 3.** Simple non-deterministic example and its Horn clause encoding

The free variables of $\varphi$ are only $x$ and $x'$—i.e., the probabilistic variables. The constraint $r = r'$ connects the values of the two clauses. We can now compute the probability that this formula is satisfied, assuming $x$ and $x'$ get their assignments by drawing from the Gaussian distribution $\mathtt{gauss}(0, 10)$. Depending on the first-order theory we are working with, this form of *weighted* model counting requires different techniques. Since we are operating over reals, this is an integration problem. We refer the reader to Section 6 where we survey different model counting techniques.

Eliminating the quantifier from $\varphi$, we get $x \geqslant 5 \vee (x < 0 \wedge x' \geqslant 5)$. The probability of satisfying $\varphi$ is thus $\sim 0.46$.[2] Note that this is a lower bound on the actual probability of returning a value that is greater than 5. By looking at longer unrollings of the Horn clauses, we arrive closer and closer to the actual probability. For our purposes, however, we have managed to prove that the query holds with a probability greater than 0.3.

**Forms of non-determinism** In the program discussed above, the only form of non-determinism in a program's execution was probabilistic choice. In many scenarios, we also want to reason about non-deterministic events for which we cannot assign a probability. We illustrate this with the example shown in Figure 2. The variable `x` gets its value drawn from `gauss`(0,10). Then, depending on the value of the Boolean variable `flag`, which is chosen nondeterministically, `x` gets incremented or decremented by 5.

Our approach allows two different treatments of nondeterminism: *angelic* and *demonic*. In the angelic case, the intuition is as follows: an execution satisfies the property if there *exists* a set of values for nondeterministic variables that makes the execution satisfy the property. Our semantics follow those of Chistikov et al. [12]; effectively, we can think of non-determinism as being able to observe all probabilistic choices made in an execution, and then make its decision.

In our example, the probability $\mathbb{P}[q(r) \to r > 0]$ is $\sim 0.69$, assuming `flag` is chosen angelically—i.e., `flag` always takes us through the desired path, the one that increments `x` by 5. Alternatively, we can treat `flag` as a demonic variable: an execution satisfies the property if *all* values of nondeterministic variables satisfy

[2] $\mathbb{P}[\varphi] = \mathbb{P}[x \geqslant 5] + \mathbb{P}[x < 0] * \mathbb{P}[x' \geqslant 5]$. Since $x, x' \sim \mathtt{gauss}(0, 10)$, we have $\mathbb{P}[x \geqslant 5] \approx 0.308$ and $\mathbb{P}[x < 0] = 0.5$.

the property. In our example, the only executions that satisfy $q(r) \rightarrow r > 0$ are the ones where `x` draws a value that is greater than 5. This is because `flag` will demonically steer execution to the `else` branch of the conditional. Thus, in the demonic setting, the probability of the query is $\sim 0.31$.

Operationally, angelic variables are handled by existentially quantifying them in the encoding of unrolled Horn clauses; demonic variables, on the other hand, are universally quantified.

## 3 Possibilistic and Probabilistic Horn Clauses

We begin by defining required background on non-probabilistic Horn-like problems, and define their semantics in terms of derivations. This paves the way for presenting our extension to the probabilistic setting, described in Section 3.3.

### 3.1 Preliminaries

**Formulas** We assume formulas are over a fixed interpreted first-order theory $\mathcal{T}$, e.g., linear integer arithmetic. We assume we have a set $\mathcal{R}$ of uninterpreted predicate symbols. We use $\varphi$ to denote a formula in the theory $\mathcal{T}$. Given a formula $\varphi$, we use $\mathit{vars}(\varphi)$ to denote the set of free variables in $\varphi$. We say that a formula is *interpreted* if it does not contain applications of predicates in $\mathcal{R}$.

**CHC** A *constrained Horn clause* (CHC) $C$ is of the form

$$\varphi_C \rightarrow P_{n+1}(\boldsymbol{x}_{n+1})$$

where $\{P_1(\boldsymbol{x}_1), \ldots, P_n(\boldsymbol{x}_n)\}_{n \geqslant 0}$ is the set of all uninterpreted predicate applications that appear in the formula $\varphi_C$; all $P_i(\boldsymbol{x}_i)$ appear positively in $\varphi_C$ (i.e., under an even number of negations); and $\boldsymbol{x}_i$ denotes a vector of variable arguments to predicate $P_i$. We will use $P_i(\boldsymbol{x}_i) \in \varphi_C$ to denote that $P_i(\boldsymbol{x}_i)$ appears in $\varphi_C$. All free variables in a CHC are implicitly universally quantified. The left hand side of the implication ($\rightarrow$) is called the *body* of $C$, while the right hand side is its *head*. Given a clause $C$, we will use $H_C$ to denote its head $P_{n+1}(\boldsymbol{x}_{n+1})$.

**Ground instance** Given a CHC $C$ and a substitution $\sigma$, which maps every variable in $C$ to a constant, we use $\sigma C$ to denote the *ground instance* of $C$ where each variable is replaced by its respective substitution in $\sigma$, that is, $\sigma\varphi_C \rightarrow \sigma H_C$.

*Example 1.* Consider the clause

$$C : x + y > 0 \rightarrow f(x)$$

and the substitution $\sigma = [x \mapsto 1, y \mapsto 2]$. The ground instance $\sigma C$ is

$$1 + 2 > 0 \rightarrow f(1)$$

■

### 3.2 Possibilistic Horn-Clause Problems

**CHC problems** A CHC problem $\mathcal{H}$ is a tuple $(\mathcal{C}, \mathcal{Q})$, where $\mathcal{C}$ is a set of clauses $\{C_1, \ldots, C_n\}$, and the *query* $\mathcal{Q}$ is of the form $Q(\boldsymbol{x}) \rightarrow \varphi$, where $Q \in \mathcal{R}$, $\mathit{vars}(\varphi) \subseteq \boldsymbol{x}$, and there are no uninterpreted predicates in $\varphi$. We assume that $Q$ does not appear in the body of any $C \in \mathcal{C}$. Throughout the paper, we shall always use $Q$ to denote the predicate symbol appearing in the query.

**Semantics** Intuitively, a CHC problem's semantics are defined by the *least solution* (interpretation) of the predicates in $\mathcal{R}$ that satisfies all clauses $\mathcal{C}$. We say that a query $\mathcal{Q}$ *holds iff* in the least solution of $\mathcal{H}$, all elements of $Q$ satisfy $\varphi$, i.e., that $\forall \boldsymbol{x}.\, Q(\boldsymbol{x}) \rightarrow \varphi$. In program terms, $\varphi$ is the set of *safe states*.

**Derivation sequences** We shall define least solutions in terms of *derivation sequences*. Given a problem $\mathcal{H} = (\mathcal{C}, \mathcal{Q})$, a derivation sequence $d$ is a finite sequence of ground instances of clauses in $\mathcal{C}$:

$$\sigma_1 C_{i_1}, \sigma_2 C_{i_2}, \ldots, \sigma_n C_{i_n}$$

where:

1. For all $j \in [1, n]$, each ground predicate in the body of $\sigma_j C_{i_j}$ should appear as the head of a $\sigma_k C_{i_k}$, for some $k < j$; otherwise, it is replaced by *false*.
2. For all $j \in [1, n]$, $\sigma_j \varphi_{C_{i_j}}$ is satisfiable.

For a derivation sequence $d$, we shall use $\boldsymbol{c}_d$ to denote the vector of constants in the head of ground instance $\sigma_n C_{i_n}$.

It follows that a query $Q(\boldsymbol{x}) \rightarrow \varphi$ holds *iff* for every derivation sequence $d$ that ends with $Q$ as the head of the last ground instance, we have $\sigma\varphi$ is satisfiable, where $\sigma = [\boldsymbol{x} \mapsto \boldsymbol{c}_d]$. For conciseness, we use $d \models \varphi$ to denote that $d$ derives a value that satisfies $\varphi$.

*Example 2.* Consider the two clauses

$$C_1 : x > 0 \vee g(x) \rightarrow f(x)$$
$$C_2 : f(x) \wedge y = x + 1 \rightarrow f(y)$$

Consider the two substitutions $\sigma_1 = [x \mapsto 1]$ and $\sigma_2 = [x \mapsto 1, y \mapsto 2]$. The sequence $d = \sigma_1 C_1, \sigma_2 C_2$ is a derivation sequence. ■

***n*-derivations** To pave the way for our probabilistic semantics, we shall redefine what it means for a query to hold in terms of *n-derivations*: the set of derivations of length $\leqslant n$. We define all such derivations by first *unrolling* the set of clauses $\mathcal{C}$ to a new non-recursive set $\mathcal{C}_n$. This is shown in Algorithm 1.

The algorithm unrolls the set of clauses $\mathcal{C}$ in a top-down fashion, beginning with the predicate appearing in the query $\mathcal{Q}$. In UNROLL, we use $\mathcal{C}(P_i^k)$ to denote the set of all clauses in $\mathcal{C}$ whose head is an application of $P_i$. We use the superscript $k$ to denote *primed* values of a predicate symbol; primes are used to ensure that the resulting unrolling is not recursive. The function $\mathit{fresh}(C, P_i^k)$ takes a clause $C \in \mathcal{C}$ and returns a new clause where the predicate in the head

```
Require: $n > 0$
function UNROLL($(\mathcal{C}, \mathcal{Q}), n$)
    $rels \leftarrow \{Q\}$ //$\mathcal{Q} = Q(\boldsymbol{x}) \rightarrow \varphi$
    $\mathcal{C}_n \leftarrow \emptyset$
    for $i$ from 1 to $n$ do
        $cls \leftarrow \{fresh(C, P_i^k) \mid C \in \mathcal{C}(P_i^k), P_i^k \in rels\}$
        $\mathcal{C}_n \leftarrow \mathcal{C}_n \cup \{cls\}$
        $rels \leftarrow \{P_i^k \mid P_i^k \in \varphi_C, C \in cls\}$
    return $\mathcal{C}_n$
```

**Algorithm** 1: Unrolling a set of Horn clauses

of $C$ is replaced with $P_i^k$, and all occurrences of predicate symbols in the body are given fresh (unused) superscripts.

We assume that all clauses in $\mathcal{C}_n$ have mutually disjoint sets of variables. We also assume that $\mathcal{C}_n \subseteq \mathcal{C}_{n+1}$, for all $n \geqslant 1$—that is, UNROLL always picks canonical names for variables and predicates. We use $\boldsymbol{x}_\infty$ to be the vector of all variables appearing in all clauses in the (potentially infinite) set $\mathcal{C}_\infty = \bigcup_{n=1}^{\infty} \mathcal{C}_n$. Variables in $\boldsymbol{x}_\infty$ are ordered canonically, e.g., in order of generation in UNROLL.

*Example 3.* Recall the Horn clause problem in Figure 2 from Section 2. The problem was unrolled for $n = 2$, resulting in the two clauses $C_1$ and $C_2$, shown in Formulas 1 and 2. ■

**Queries and $n$-derivations** Given a (potentially infinite) set of clauses $\mathcal{C}' \subseteq \mathcal{C}_\infty$, we shall use $\sigma\mathcal{C}'$, where $\sigma$ maps each variable in $\boldsymbol{x}_\infty$ to a constant, to denote the set of all derivation sequences that $(i)$ are formed from ground instances in $\{\sigma C \mid C \in \mathcal{C}'\}$ and $(ii)$ end in a clause with $Q(\boldsymbol{c})$ in the head.

The following theorem formalizes what it means for a query to hold in terms of $\infty$-derivations.

**Theorem 1.** *A query $\mathcal{Q} = Q(\boldsymbol{x}) \rightarrow \varphi$ holds* iff $\{\sigma \mid d \in \sigma\mathcal{C}_\infty$ *and* $d \not\models \varphi\} = \emptyset$.

The idea is that a query holds *iff* there does not exist a substitution $\sigma$ that results in a derivation $d$ that falsifies the formula $\varphi$.

## 3.3 Probabilistic Horn-Clause Problems

Now that we have defined traditional Horn clause problems and their semantics, we are ready to define *probabilistically constrained Horn clauses* (PCHC).

**PCHC problems** A PCHC problem $\mathcal{H}^p$ is a tuple of the form $(\mathcal{C}^p, \mathcal{Q}^p)$ (for clarity, we drop the superscript $p$ below):

- Each clause $C \in \mathcal{C}$ is defined as in CHC: $\varphi_c \rightarrow H_C$. However, the set of unbound variables that appear in $C$ are divided into two disjoint vectors: $\boldsymbol{x}_C^p$ and $\boldsymbol{x}_C^a$. We call $\boldsymbol{x}_C^p$ the set of *probabilistic variables*, whose values are drawn from a *joint probability distribution* $\mathcal{D}_C$. The variables $\boldsymbol{x}_C^a$ are *angelic variables.*

– The probabilistic query $\mathcal{Q}$ is a pair of the form $(Q(\boldsymbol{x}) \rightarrow \varphi, \theta)$, where $\theta \in [0,1)$. We would like to prove that the probability of deriving an element of $Q$ that satisfies $\varphi$ is greater than $\theta$.

**Semantics of PCHC** In CHC problems, the semantics are such that an element is either derived or not; here, an element is derived with a probability. The following semantics of PCHC problems are inspired by Kozen's seminal work on the semantics of probabilistic programs [31].

Using the UNROLL procedure in Algorithm 1, we analogously unroll clauses $\mathcal{C}$ into sets $\mathcal{C}_1 \subseteq \mathcal{C}_2 \subseteq \ldots$. The set of variables $\boldsymbol{x}_\infty$ appearing in $\mathcal{C}_\infty = \bigcup_{n=1}^{\infty} \mathcal{C}_n$ is broken into two disjoint vectors $\boldsymbol{x}^p_\infty$ and $\boldsymbol{x}^a_\infty$, where $\boldsymbol{x}^p_\infty$ denotes all *probabilistic variables*, and $\boldsymbol{x}^a_\infty$ denotes all *angelic variables*. We treat the variables $\boldsymbol{x}^p_\infty$ as random variables distributed according to their respective distributions in $\{\mathcal{D}_C\}$. We shall assume existence of a *probability space* $(\Omega, \mathcal{F}, \mathbb{P})$ where *outcomes* $\Omega$ are valuations of $\boldsymbol{x}^p_\infty$, defined through substitutions $\sigma^p$; *events* $\mathcal{F}$ are sets of substitutions; and $\mathbb{P}$ is a probability measure over sets of substitutions. We assume existence of events $\Sigma^\phi_n \in \mathcal{F}$ defined as

$$\Sigma^\phi_n = \{\sigma^p \mid \exists \sigma^a, d \text{ s.t. } d \in \sigma^a(\sigma^p \mathcal{C}_n) \text{ and } d \models \phi\},$$

for $n \in [1, \infty]$ and interpreted formula $\phi$ with free variables in $\boldsymbol{x}$ (the variables in the query $\mathcal{Q}$). That is, $\Sigma^\phi_n$ is the set of all substitutions to probabilistic variables that yield derivations of length $\leqslant n$ and satisfy $\phi$. Observe the substitution $\sigma^a$: this is used to pick values for the angelic variables $\boldsymbol{x}^a_n$. We note that when $n$ is $\infty$, the definition of $\Sigma^\phi_\infty$ is as defined above, using the set $\mathcal{C}_\infty$.

The following theorem states two key properties of $\Sigma^\phi_n$ that we exploit later.

**Theorem 2.** *(a) For all $n \in [1, \infty)$, $\Sigma^\phi_n \subseteq \Sigma^\phi_{n+1}$. (b) $\Sigma^\phi_\infty = \bigcup_{n=1}^{\infty} \Sigma^\phi_n$.*

*Proof.* (a) Suppose that $\Sigma^\varphi_n \not\subseteq \Sigma^\varphi_{n+1}$. Then, there must be an assignment $\sigma_p \in \Sigma^\varphi_n$ such that $\sigma_p \notin \Sigma^\varphi_{n+1}$. By construction, we know that there exists a sequence of unique clauses $C_1, \ldots, C_k \in \mathcal{C}_n$, and a substitution $\sigma_a$ to $\boldsymbol{x}_a$ such that derivation $d = \sigma_p\sigma_a C_1, \ldots, \sigma_p\sigma_a C_k$, clause $C_k$ is of the form ... $\rightarrow Q(\boldsymbol{x})$, and $d \models \varphi$. By monotonicity of UNROLL, we know that $\mathcal{C}_n \subseteq \mathcal{C}_{n+1}$. So, we know that $d \in \sigma_p\sigma_a\mathcal{C}_{n+1}$ and therefore $\sigma_p \in \Sigma^\varphi_{n+1}$.

(b) The $\Leftarrow$ direction is similar to the above proof, since $\mathcal{C}_n \subseteq \mathcal{C}_\infty$. The $\Rightarrow$ direction: Take any $\sigma_p \in \Sigma^\varphi_\infty$, then there is a $\sigma'_p$ that is $\sigma_p$ with a substitution for all $\boldsymbol{x}^a_\infty$ such that there is a derivation $d = \sigma'_p C_1, \ldots, \sigma'_p C_k$ such that $d \models \varphi$. By monotonicity of UNROLL, we know that $\{C_1, \ldots, C_k\} \in \mathcal{C}_l$, for some $l \in [1, \infty)$. Therefore, $d \in \sigma'_p \mathcal{C}_l$ and $\sigma_p \in \Sigma^\varphi_l$. ■

*Example 4.* Recall the function `f` and its associated PCHC problem in Figure 2. In Section 2, we considered an unrolling with $n = 2$. The set $\Sigma^{r>5}_2$ is the following set of substitutions to $\boldsymbol{x}^p_\infty = (x, x', x'', \ldots)$—in our unrolling in Section 2, we only have $x$ and $x'$; variables $x'', \ldots$ appear in longer unrollings:

$$\Sigma^{r>5}_2 = \{[x \mapsto c, x' \mapsto c', x'' \mapsto c'', \ldots] \mid c \geqslant 5 \vee (c < 0 \wedge c' \geqslant 5)\}$$

Note that only $x$ and $x'$ are constrained in the substitutions, since they are the only ones that appear in unrollings of length 2. We computed that $\mathbb{P}[\Sigma^{r>5}_2]$ is $\sim 0.46$, as the values of $c$ and $c'$ are drawn from $\mathsf{gauss}(0, 10)$. ■

The following definition formalizes what it means for a query to hold.

**Definition 1.** *A query* $\mathcal{Q} = (Q(\boldsymbol{x}) \to \varphi, \theta)$ *holds* iff $\mathbb{P}\left[\Sigma_{\infty}^{\varphi}\right] > \theta$.

The intuition is as follows: take the set of all substitutions $\sigma^p$ that can derive an element that satisfies $\varphi$ (for some $\sigma^a$), and compute the probability of picking a substitution in that set.

We assume that $\mathbb{P}\left[\Sigma_{\infty}^{true}\right] = 1$. In other words, *almost all* substitutions of the probabilistic variables result in a derivation. This is analogous to the *almost-sure termination* property of probabilistic programs, which stipulates that a program terminates with probability 1.

## 4 Probabilistic Horn Clause Verification

**Overview** The high-level idea underlying our algorithm is as follows. Ideally, we would like to compute the probability of picking a substitution $\sigma^p$ that results in a derivation $d \models \varphi$, for some assignment $\sigma^a$ of the angelic variables. However, $\sigma^p$ is over an infinite set of variables. To make the problem manageable, we begin by considering substitutions that result in derivations $|d| \leqslant n$, for some fixed $n$. By doing so, we compute a *lower bound* on the probability, since we consider a *subset* of all possible derivations. By iteratively increasing the value $n$—i.e., look at longer and longer derivations—we converge to the actual probability of the event of interest.

Since for all $n \in [1, \infty)$, $\Sigma_n^{\varphi} \subseteq \Sigma_{n+1}^{\varphi}$, we have the fact that $\mathbb{P}[\Sigma_n^{\varphi}] \leqslant \mathbb{P}[\Sigma_{n+1}^{\varphi}]$. Our algorithm iteratively increases $n$, computing the probability $\mathbb{P}[\Sigma_n^{\varphi}]$ at each step, until it can prove that $\mathbb{P}[\Sigma_n^{\varphi}] > \theta$. Additionally, as we will see, the algorithm can disprove such properties, i.e., prove that $\mathbb{P}[\Sigma_n^{\varphi}] \leqslant \theta$, by maintaining an upper bound on $\mathbb{P}[\Sigma_n^{\varphi}]$.

**Encoding derivations** The primary step in making the algorithm practical is to characterize the set $\Sigma_n^{\varphi}$ and figure out how to compute the probability of picking an element in that set. We make the observation that the set $\Sigma_n^{\varphi}$ can be characterized as the set of models of an interpreted formula $\Psi_n^{\varphi}$ in the first-order theory $\mathcal{T}$. Then, the probability becomes that of picking a satisfying assignment of $\Psi_n^{\varphi}$. This is a *model counting* problem, where models additionally have a probability of occurrence. In what follows, we present an encoding of $\Psi_n^{\varphi}$. In Section 6, we discuss different mechanisms for model counting.

Our encoding algorithm is presented in Algorithm 2, as the primary function ENC, which is similar to other encodings of Horn clauses, e.g., [37]. Given a set of clauses $\mathcal{C}_n$ and a query $\mathcal{Q} = (Q(\boldsymbol{x}) \to \varphi, \theta)$, ENC encodes the clauses in a top-down recursive fashion, starting with the predicate $Q(\boldsymbol{x})$. In each recursive call to $\text{ENC}_C(P(\boldsymbol{x}))$, it encodes all clauses where $P(\boldsymbol{x})$ is the head of the clause. Uninterpreted predicates $P_i(\boldsymbol{x}_i)$ in the body of a clause are replaced with fresh Boolean variables, which indicate whether a predicate is set to true or false in a derivation. Finally, all angelic variables $\boldsymbol{x}_n^a$ and freshly introduced Boolean variables $\boldsymbol{b}$ are existentially quantified, leaving us with a formula where the

```
function ENC($\mathcal{C}_n, \mathcal{Q}$)
    return $\exists \boldsymbol{b}, \boldsymbol{x}_n^a.\, \varphi \wedge \text{ENC}_C(Q(\boldsymbol{x}))$

function $\text{ENC}_C(P(\boldsymbol{x}))$
    $D \leftarrow \emptyset$
    for all $\varphi_C \rightarrow P(\boldsymbol{x}') \in \mathcal{C}_n$ do
        map each $P_i(\boldsymbol{x}_i) \in \varphi_C$ to fresh Bool variable $b_i$
        $\phi \leftarrow \varphi_C[b_i/P_i(\boldsymbol{x}_i)] \wedge \bigwedge_{P_i(\boldsymbol{x}_i) \in \varphi_C} b_i \Rightarrow \text{ENC}_C(P_i(\boldsymbol{x}_i))$
        $D \leftarrow D \cup \{\phi \wedge \boldsymbol{x} = \boldsymbol{x}'\}$
    return $\bigvee D$
```

**Algorithm** 2: Encoding of a set of clauses (input $\mathcal{C}_n$ is accessible by $\text{ENC}_C$)

```
1:  function VERIFY($\mathcal{C}, \mathcal{Q}$)
2:      for $n \in [1, \infty)$ do
3:          $\mathcal{C}_n \leftarrow$ UNROLL($\mathcal{C}, n$)
4:          $\Psi_n^\varphi \leftarrow$ ENC($\mathcal{C}_n, \mathcal{Q}$)
5:          ▷ prove that the query holds
6:          if $\mathbb{P}[\Psi_n^\varphi] > \theta$ then
7:              return $\mathcal{Q}$ holds
8:          ▷ prove that the query does not hold
9:          if $1 - \mathbb{P}[\Psi_n^{\neg\varphi}] \leqslant \theta$ then
10:             return $\mathcal{Q}$ does not hold
```

**Algorithm** 3: Verification algorithm

only variables are the probabilistic ones. (Recall the encoding in Section 2 for a concrete example.)

For a fixed $n$, we shall treat the set of models of $\Psi_n^\varphi$ as a set of substitutions to $\boldsymbol{x}_n^p$. The following theorem states that the set of models of $\Psi_n^\varphi$ is the same as the set of substitutions in $\Sigma_n^\varphi$.

**Theorem 3.** *For all $n \geqslant 1$, $\Psi_n^\varphi = \Sigma_n^\varphi$. (We assume that variables that are not in $\Psi_n^\varphi$ but in $\boldsymbol{x}_\infty^p$ can take any value in models of $\Psi_n^\varphi$.)*

**Iterative probability approximation algorithm** Algorithm 3 shows our overall algorithm. For now, ignore the gray lines 9 and 10. As discussed above, it iteratively increases the value of $n$ attempting to prove that the query holds.

The algorithm VERIFY (without lines 9-10) is sound, that is, only returns correct solutions. VERIFY is also complete, relative to existence of an oracle for computing $\mathbb{P}[\Psi_n^\varphi]$ and assuming $\mathbb{P}[\Sigma_\infty^\varphi] > \theta$.

**Theorem 4.** VERIFY *is sound. If $\mathbb{P}[\Sigma_\infty^\varphi] > \theta$, then* VERIFY *terminates.*

*Proof.* Soundness follows from Theorem 3. Suppose that the query holds, then we know, from Theorem 2, that $\lim_{n\to\infty} \mathbb{P}[\Sigma_n^\varphi] > \theta$ and $\forall i \in \mathbb{N}.\, \mathbb{P}[\Sigma_i^\varphi] \subseteq \mathbb{P}[\Sigma_{i+1}^\varphi]$. By definition of limit, we know that there exists an $n$ such that $\mathbb{P}[\Sigma_n^\varphi] > \theta$. ■

**Disproving queries with upper bounds** The algorithm so far is only able to prove that a query holds—it cannot prove that a query does not hold, because it only computes lower bounds on the probability. Now consider the entire VERIFY algorithm, i.e., including lines 9 and 10, which also computes upper bounds. We now provide a sufficient condition for making the algorithm complete in both directions—proving and disproving that a query holds.

The restriction is as follows: for any query $(Q(\boldsymbol{x}) \to \varphi, \theta)$,

$$\Sigma^{\varphi}_{\infty} \cap \Sigma^{\neg\varphi}_{\infty} = \emptyset$$

Effectively, this ensures that derivations are completely dictated by the probabilistic variables; in program terms, this is (roughly) like ensuring that the only source of non-determinism in a program is probabilistic choice. Now, we can compute an *upper bound* for $\mathbb{P}[\Sigma^{\varphi}_{\infty}]$ by simply computing the value of $1 - \mathbb{P}[\Sigma^{\neg\varphi}_{n}]$, for any $n \in [1, \infty)$. Thus, if $1 - \mathbb{P}[\Psi^{\neg\varphi}_{n}] \leqslant \theta$, we know that the query does not hold. If we perform this check at every iteration of VERIFY, we ensure that the algorithm terminates if $\mathbb{P}[\Sigma^{\varphi}_{\infty}] < \theta$. Notice that if $\mathbb{P}[\Sigma^{\varphi}_{\infty}] = \theta$, the upper bound might come asymptotically close to $\theta$ but never get to it.

**Theorem 5.** VERIFY *is sound. If* $\mathbb{P}[\Sigma^{\varphi}_{\infty}] \neq \theta$*, then* VERIFY *terminates.*

*Proof.* By definition, $\mathbb{P}[\Sigma^{\varphi}_{\infty}] + \mathbb{P}[\Sigma^{\neg\varphi}_{\infty}] = 1$. Therefore, $\mathbb{P}[\Sigma^{\varphi}_{\infty}] = 1 - \mathbb{P}[\Sigma^{\neg\varphi}_{\infty}]$. Since $\mathbb{P}[\Sigma^{\neg\varphi}_{n}] \leqslant \mathbb{P}[\Sigma^{\neg\varphi}_{\infty}]$, for any $n \in [1, \infty)$, we know that $\mathbb{P}[\Sigma^{\varphi}_{\infty}] \leqslant 1 - \mathbb{P}[\Sigma^{\neg\varphi}_{n}]$, thus ensuring soundness. Termination follows from the fact that $\lim_{n\to\infty} \mathbb{P}[\Psi^{\neg\varphi}_{n}] \geqslant 1 - \theta$, assuming $\mathbb{P}[\Sigma^{\varphi}_{\infty}] < \theta$. ■

## 5 Angels and Demons

We now discuss extensions of PCHC problems with *demonic non-determinism.*

Analogous to angelic variables, we add a set of demonic variables $\boldsymbol{x}^{d}_{C}$ for every clause $C$. That is, now, every clause $C$ has free variables divided amongst three disjoint sets: $\boldsymbol{x}^{p}_{C}$, $\boldsymbol{x}^{a}_{C}$, and $\boldsymbol{x}^{d}_{C}$. We now redefine $\Sigma^{\varphi}_{n}$ as follows:

$$\Sigma^{\varphi}_{n} = \{\sigma^{p} \mid \forall \sigma^{d}.\, \exists \sigma^{a}, d.\, d \in \sigma^{d}(\sigma^{a}(\sigma^{p}\mathcal{C}_{n})) \text{ and } d \models \varphi\}$$

In other words, we can only add $\sigma^{p}$ to the set if *every* assignment to demonic variables leads to a derivation of an element in $\varphi$.

Notice that the alternation of quantifiers indicates that demonic non-determinism is resolved first, followed by angelic non-determinism. We can also consider arbitrary quantifier alternations, by dividing demonic and angelic variables into sets of variables that get resolved in a certain order. For our purposes, we will restrict our attention to cases where demonic non-determinism is resolved first. Informally, demonic variables can maliciously pick substitutions $\sigma^{d}$ such that there is no $\sigma^{a}$ that results in a derivation. If we flipped the quantifiers to $\exists \sigma^{a}.\, \forall \sigma^{d}$, then, effectively, the angelic variables get to divine a substitution for $\sigma^{a}$ such that no matter what substitution $\sigma^{d}$ the demonic variables are possessed with, a derivation $d \models \varphi$ exists.

**Implementing non-determinism** In the demonic case, we can construct the formula $\Psi_n^\varphi$ just as in Algorithm 2, but we quantify out the demonic variables: we use $\forall \boldsymbol{x}_n^d. \Psi_n^\varphi$. The intuition behind the choice of quantifier directly follows from the definition of $\Sigma_n^\varphi$ above.

In presence of demonic non-determinism, VERIFY loses its termination guarantee in Theorem 4.

## 6 Algorithmic Details

In this section, we discuss some of the subtleties of PCHC problems. We then discuss instantiations of our approach with various model counting techniques.

### 6.1 Decomposition and non-determinism

We now discuss key design decisions in encoding and verification.

**Decomposition** Consider the following Horn clause $C$, where $\texttt{unif}(0,10)$ is the uniform distribution over reals between 0 and 10.

$$x \sim \texttt{unif}(0,10) \mid x \leqslant 1 \lor x \geqslant 8 \to f(x)$$

Suppose we decompose the clause $C$ into two clauses, $C_1$ and $C_2$, by splitting the disjunction:

$$x_1 \sim \texttt{unif}(0,10) \mid x_1 \leqslant 1 \to f(x_1)$$
$$x_2 \sim \texttt{unif}(0,10) \mid x_2 \geqslant 8 \to f(x_2)$$

In the non-probabilistic setting, this transformation would result in a semantically equivalent set of clauses. In our setting, however, we get a semantically different set of clauses. This is because we duplicate the probabilistic variables, resulting in two *independent* variables, $x_1$ and $x_2$. Suppose we want to compute the probability that $f(x) \to$ *true*. In the first case, the answer is 0.3. In the second case, the answer is 0.28.[3]

*Remark 1.* We could alternatively just assume that $x_1$ and $x_2$ are the same variable in $C_1$ and $C_2$. This view drastically complicates the semantics: we now have variable sharing between clauses, and the semantics of unrollings need to take that into account. Given that we also have recursion, we have to reason about which instances of clauses in the unrolling are sharing variables and which are not (for instance, in Section 2, when we unrolled the recursive clause, we constructed a new copy with an independent variable $x'$). To simplify the semantics of PCHC, we opted to enrich the formulas that can appear in Horn clauses, rather than encode and manage probabilistic variable independence explicitly.

[3] Since $\mathbb{P}[x_1 \leqslant 1 \lor x_2 \geqslant 8] = 1 - \mathbb{P}[x_1 > 1 \land x_2 < 8] = 1 - 0.9 * 0.8 = 0.28$

**Non-determinism** We now discuss a related issue. One might wonder: why not compute the probability for each subset of the clauses separately and sum the answers? In program terms, we can view this as computing the probability of individual program paths separately, e.g., as in Sankaranarayanan et al.'s algorithm [42].

Unfortunately, in our setting, non-determinism does not allow us to decompose the problem. Consider the following example:

$$x \sim \texttt{unif}(0,10) \mid (b \wedge (x \geqslant 2 \iff r)) \vee (\neg b \wedge (x \leqslant 4 \iff r)) \rightarrow f(r)$$

where $b$ and $r$ are angelic Boolean variables. This encodes the following program:

```
    x ~ unif(0,10)
    b = nondet()
π1  if (b) ret x >= 2
π2  else ret x >= 4
```

Suppose we have the query $f(r) \rightarrow r = \mathit{true}$. The probability that the query holds is 8/10, because angelic nondeterminism always leads us through the then branch of the conditional. Consider the approach where we compute the probability for one disjunct at a time—i.e., one program path at a time. The following clause defines path $\pi_1$, which takes the then branch of the conditional. The clause satisfies the query with a probability of 8/10:

$$x \sim \texttt{unif}(0,10) \mid (b \wedge (x \geqslant 2 \iff r)) \rightarrow f(r)$$

The following clause, encoding $\pi_2$, satisfies the query with probability 6/10:

$$x \sim \texttt{unif}(0,10) \mid (\neg b \wedge (x \geqslant 4 \iff r)) \rightarrow f(r)$$

*Adding the two probabilities results in 14/10.* Approaches that divide the program into paths and sum up the results assume that different paths are mutually exclusive—i.e., the program is deterministic. In non-deterministic programs, the events of taking different paths are *not* mutually exclusive, therefore, we cannot simply add the probability of the two events. Our approach considers both paths simultaneously through encoding, resolving the non-determinism and discovering that the probability of the query is 8/10.

## 6.2 Model counting modulo probability spaces

**Overview** In Section 4, we assumed existence of an oracle that, given a formula $\Psi_n^\varphi$, can compute the probability that it is satisfied, assuming the values of variables $\boldsymbol{x}_n^p$ are drawn from some probability distribution. Suppose, for instance, that $\Psi_n^\varphi$ is a propositional formula and that there is a joint probability distribution $p(\boldsymbol{x}_n^p)$. Then, we define

$$\mathbb{P}[\Psi_n^\varphi] = \sum_{\boldsymbol{c} \in \{0,1\}^n} \mathbb{1}([\boldsymbol{x}_n^p \mapsto \boldsymbol{c}]\Psi_n^\varphi)\ p(\boldsymbol{c})$$

where the *indicator function* $\mathbb{1}(\phi)$ is 1 if $\phi$ is *true* and 0 otherwise.

Suppose, alternatively, that $\Psi_n^p$ is a formula over real linear arithmetic and there is a joint probability density function $p(\boldsymbol{x}_n^p)$. Then, we define

$$\mathbb{P}[\Psi_n^\varphi] = \int \mathbb{1}(\Psi_n^\varphi)\ p(\boldsymbol{x}_n^p)\ d\boldsymbol{x}_n^p$$

That is, we integrate over region $\Psi_n^\varphi \subseteq \mathbb{R}^n$, weighted by the probability density.

The above problems are hard, for instance, in the propositional setting, the counting problem is #P-complete. Nonetheless, there are efficient approaches for various first-order theories; we survey prominent techniques below. Our algorithm, of course, is agnostic to the technique used for computing probabilities.

**Approximate guarantees** Approximate techniques come in two flavors: $(i)$ statistical approaches that utilize *concentration inequalities*, and $(ii)$ PTIME randomized approximation schemes (PRAS) with access to an NP oracle (e.g., a SAT solver) [43]. Both approaches provide $(\epsilon, \delta)$ guarantees, where they produce a result that is within a multiplicative or additive error of $\epsilon$ from the exact result, with a probability $1 - \delta$ of being correct. Recently, there has been progress in practical PRAS algorithms [12, 9, 2], due to developments in SAT and SMT solvers.

**Hard guarantees** Other approaches attempt to produce *exact* answers. For instance, recent work has utilized *cone decomposition* [14, 3] to integrate polynomial probability density functions over linear real arithmetic formulas. Other work considered over- and under-approximating linear real and integer arithmetic formulas as a set of cubes to produce upper and lower bounds on the probability [42].

## 7 Implementation and Evaluation

**Implementation** We have implemented a prototype of our technique that $(i)$ takes programs in a simple Python-like language with procedure calls, probabilistic assignments and non-deterministic ones; $(ii)$ converts the program and a query of interest to a PCHC problem; and $(iii)$ verifies the query. All programs are encoded in linear real arithmetic.

Recall that we need to compute $\mathbb{P}[\Psi_n^\varphi]$ at every iteration of VERIFY, and that $\Psi_n^\varphi$ is quantified. To do so, we apply a simple Monte-Carlo-based sampling approach that proves $\mathbb{P}[\Psi_n^\varphi] > \theta$ with a 0.99 confidence—using *Hoeffding's concentration inequality*. Specifically, the approach draws an assignment for the probabilistic variables, substitutes the assignment in the formula $\Psi_n^\varphi$, and checks whether the result is SAT. Given that the formula is quantified, this is an expensive process. If the formula is existentially quantified, then evaluating a sample is NP-complete. One could also eliminate the quantifier, and then each sample evaluation is just a simplification of the formula. We have, however, found that quantifier elimination degrades performance in this case, and it is better to evaluate each sample with a call the SMT solver. On the other hand, for universally quantified formulas, we have found that it is very important to perform quantifier elimination first, as iteratively calling the SMT solver to evaluate samples

| Benchmark | Iters | $\theta$ | Time | QETime | Description |
|---|---|---|---|---|---|
| simple | 1 | 0.4 | 6.3 | 0.0 | Example from Figure 2 |
| | 2 | 0.6 | 12.6 | 0.0 | |
| | 4 | 0.8 | 25.4 | 0.0 | |
| simp-ang | 1 | 0.4 | 6.6 | 0.0 | Example from Figure 2 with angelic determinism for the choice of x's distribution |
| | 2 | 0.6 | 13.5 | 0.0 | |
| | 3 | 0.8 | 20.4 | 0.0 | |
| simp-dem | 4 | 0.1 | 45.5 | 20.9 | Example from Figure 2 with demonic determinism for the choice of x's distribution |
| | 4 | 0.2 | 45.4 | 20.9 | |
| | 5 | 0.3 | 530.2 | 505.9 | |
| mc91 | 2 | 0.2 | 6.7 | 0.0 | McCarthy91 function with a distribution on possible inputs |
| | 2 | 0.3 | 6.6 | 0.0 | |
| | – | 0.4 | – | – | |
| mc91-approx | 2 | 0.4 | 6.8 | 0.0 | McCarthy91 function with an approximate adder |
| | 3 | 0.5 | 14.0 | 0.0 | |
| | 6 | 0.6 | 51.4 | 0.0 | |
| mc91-equiv | 3 | 0.7 | 15.2 | 0.0 | Prob. of equivalence between an approx. and exact `mc91`—using self-composition |
| | 4 | 0.8 | 25.9 | 0.0 | |
| | – | 0.9 | – | – | |
| chat-fig2 | 3 | 0.2 | 7.3 | 1.0 | From Chatterjee et al. [10]: random walk with demonic and angelic nondeterminism |
| | 3 | 0.3 | 7.3 | 1.0 | |
| | – | 0.4 | – | – | |
| chat-rw1 | 6 | 0.1 | 38.2 | 0.0 | From Chatterjee et al. [10]: 1-dimensional random walk |
| | 8 | 0.2 | 57.8 | 0.0 | |
| | 11 | 0.3 | 90.9 | 0.0 | |
| chat-rw2-dem | 3 | 0.1 | 14.7 | 2.3 | From Chatterjee et al. [10]: 2-dimensional random walk with demonic non-determinism |
| | 3 | 0.2 | 14.6 | 2.3 | |
| | – | 0.3 | – | – | |

**Table 1.** Experimental results: Iters is the number of iterations; $\theta$ is the query threshold; Time (s) is the total running time; and QETime is quantifier elimination time.

on universally quantified formulas is infeasible. As such, we only perform quantifier elimination in the presence of demonic nondeterminism—which requires universal quantifiers. We use Redlog [1] for quantifier elimination.

*Remark 2.* We opted for an approximate approach to model counting because our formulas are over quantified LRA and non-trivial distributions, like Gaussians, that established exact volume computation tools are unable to handle. For example, LattE [14]—which is used in a number of tools [22, 3]—can only integrate a piecewise polynomial function over a polyhedron.

**Benchmarks** We collected a set of benchmarks that is meant to exercise the various features of our approach. Table 1 shows the list of benchmarks along with a description of each. The family of benchmarks `simple*` are variants of the illustrative example in Figure 2, where we enrich it with angelic and demonic forms of nondeterminism with which it decides the distribution to draw the value of `x` from. We then consider the classic McCarthy91, `mc91`, recursive function, where we impose a distribution on the possible inputs and compute the probability that the return value is greater than 91. We consider an *approximate* version of `mc91`, where the *adder* may flip the least significant bit from 0 to 1 with a small probability. `mc91-equiv` computes the probability that the approximate version `mc91-approx` returns the same result as the exact version `mc91`. The family of benchmarks `chat-*` are random-walk programs taken from Chatterjee et al. [10] (who are interested in termination). The programs contain demonic and angelic nondeterminism (in case of `chat-fig2`, both). The query we check is about the probability that the random walk ends in a certain region on the grid.

**Evaluation and discussion** Table 1 shows the results of running our algorithm with a 10 minute timeout per benchmark ('–' indicates timeout). For each benchmark, we pick three values for $\theta$ that gradually increase the difficulty of the verification process by forcing VERIFY to perform more iterations. Most benchmarks complete within 90 seconds.

We found two primary sources of difficulty: The first difficulty is dealing with universal quantifiers. Consider, for instance, `simp-dem` with $\theta = 0.3$. Here, VERIFY needs to unroll the recursion up to depth 5, resulting in a difficult formula for quantifier elimination, as shown by the time taken for quantifier elimination. Similarly, `chat-fig2` and `chat-rw2-dem` (both of which contain demonic nondeterminism) timeout at larger values of $\theta$ while waiting for quantifier elimination to complete. In the future, we plan on investigating efficient underapproximations of quantifier elimination [26] that result in good-enough lower bounds for probabilities. The second source of difficulty is the exponential explosion in the size of the unrolling—and therefore the encoding—which occurs in problems like `mc91`. In the non-probabilistic case, recent work [33] has dealt with this problem by merging procedure calls on different execution paths to limit the explosion. It would be interesting to investigate such technique in the probabilistic setting.

To the best of our knowledge, there are no automated verification/analysis tools that can handle the range of features demonstrated in our benchmark suite. In the next section, we survey existing works and describe the differences.

## 8 Related work and Discussion

**Probabilistic program analysis** There is a plethora of work on analyzing probabilistic programs. Abstraction-based techniques employ abstract domains to compute approximations of probability distributions [38, 39, 13]. By unrolling program executions, our approach does not lose precision due to abstraction. The closest approach to ours is that of Sankaranyanan et al. [42] where computing the probability of an event is reduced to summing probabilities of event occurrences on individual paths. First, our intermediate language of Horn clauses allows natural handling of recursive calls; additionally, we handle nondeterminism, which, as discussed in Section 6, is handled unsoundly in the path-based technique. Similarly, Sampson et al. [41] perform path-based unrolling, but do not provide whole-program guarantees.

Other techniques for program analysis include axiomatic and exact ones. Exact techniques, like PSI [21], involve finding a closed-form for the return values of a given program, using rewrite rules and symbolic execution. To our knowledge, none of the existing exact techniques can handle nondeterminism and/or recursive procedures. Axiomatic techniques synthesize expectation invariants from which a post-condition of interest may be deduced [30, 36, 8]. Compared to our approach, these techniques do not handle procedures, are not guaranteed to prove properties of the form in this paper, and are restricted in terms of variable types and distributions used. Axiomatic approaches, however, excel at characterizing the probability of an event in terms of inputs. It would be very interesting to study expectation invariants in the context of PCHC. Note that our semantics of non-determinism are slightly different from those used by McIver and Morgan [36]; we discuss this more below when describing Luckow et al.'s work [35].

**Probabilistic and statistical model checking** Compared to probabilistic model checking, our approach allows encoding semantics of arbitrary recursive programs, as long they fit in an appropriate first-order theory. Probabilistic model checkers like PRISM [32] are often restricted to reasoning about finite-state Markov chains. Statistical model checking [34] applies statistical testing to prove properties with high confidence. We applied statistical testing in our evaluation to compute probabilities with high confidence, where our testing was over quantified formulas encoding Horn-clause unrollings.

**Model counting** Like other recent techniques, our approach reduces probabilistic analysis to a form of model counting. Chistikov et al. [12] apply a similar technique to encode single-procedure programs and use approximate model counting with an NP oracle. Our approach can be viewed as a generalization of Chistikov et al.'s formulation to programs with procedures and recursive calls.

A number of other analysis techniques for probabilistic programs employ model counting [22, 19, 35]. The closest work to ours in that space is that of Luckow et al. [35]. There, the program also involves non-determinism in the form of a sequence of Boolean variables (a schedule) and the goal is to find an assignment that maximizes/minimizes probability of an event. There are two key differences with our work: First, we do not only admit Boolean non-deterministic variables, but we can also handle, e.g., real-valued non-determinism. Second, our non-determinism semantics are slightly different: We follow Chistikov et al. [12], where non-determinism follows probabilistic choice. In the future, we plan to investigate the alternate form of non-determinism used by Luckow et al. [35], where non-deterministic variables are resolved first. In such case, the weighted model counting problem turns into E-MAJSAT (which is in $\text{NP}^{\text{PP}}$), where the goal is to find a satisfying assignment to the non-deterministic variables that maximizes the weighted model count of the formula.

**Probabilistic Horn clauses** In artificial intelligence and databases, Horn clauses have been extended to the probabilistic setting, e.g., [20, 15]. The semantics and usage are quite different from our setting. Probabilities are usually associated at the level of the clause—e.g., a rule applies with a 0.75 probability. Our approach incorporates probabilistic variables with the clauses themselves and is over infinite domains, e.g., reals.

**Probabilistic recursive models** There have been a number of proposals for probabilistic models that involve recursion. For instance, probabilistic pushdown [17] automata and recursive Bayesian networks [40]. In probabilistic pushdown automata, and equivalently recursive Markov chains [18, 44], variable domains are finite and probabilities are applied only on transitions. Our approach allows for infinite domains and probabilistic choice allows encoding probabilistic control-flow transitions in a program as well as probabilistic assignments.

**Horn clause solving** As discussed throughout the paper, our algorithmic contribution adapts existing Horn clause solving algorithms to the probabilistic setting. Specifically, most existing algorithms, e.g., HSF [24, 25] and Duality [37], employ an tree unrolling of Horn clauses, but are concerned with finding inductive invariants as opposed to probability bounds.

## 9 Conclusion

We introduced probabilistically constrained Horn clauses (PCHC), and presented an algorithm for proving/disproving probabilistic queries. Our semantics incorporated a form of angelic/demonic non-determinism, where, effectively, angelic/demonic variables can look into the future. This is, for instance, different from the semantics used by McIver and Morgan [36]. In the future, we plan to handle such semantics by extending techniques like Luckow et al. [35] to our Horn-clause setting. Another interesting avenue for future work is to incorporate some form of loop summarization, so that we can reduce probabilistic inference over infinitely many derivations to a fixed set, therefore avoiding iterative unrolling.

## References

1. Redlog. `http://www.redlog.eu/`
2. Belle, V., Van den Broeck, G., Passerini, A.: Hashing-based approximate probabilistic inference in hybrid domains. In: Proceedings of the 31st Conference on Uncertainty in Artificial Intelligence (UAI) (2015)
3. Belle, V., Passerini, A., den Broeck, G.V.: Probabilistic inference in hybrid domains by weighted model integration. In: Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015. pp. 2770–2776 (2015), `http://ijcai.org/Abstract/15/392`
4. Beyene, T.A., Popeea, C., Rybalchenko, A.: Solving existentially quantified horn clauses. In: International Conference on Computer Aided Verification. pp. 869–882. Springer (2013)
5. Bjørner, N., Gurfinkel, A., McMillan, K., Rybalchenko, A.: Horn clause solvers for program verification. In: Fields of Logic and Computation II, pp. 24–51. Springer (2015)
6. Bjørner, N., McMillan, K., Rybalchenko, A.: On solving universally quantified horn clauses. In: International Static Analysis Symposium. pp. 105–125. Springer (2013)
7. Carbin, M., Kim, D., Misailovic, S., Rinard, M.C.: Verified integrity properties for safe approximate program transformations. In: Proceedings of the ACM SIGPLAN 2013 workshop on Partial evaluation and program manipulation. pp. 63–66. ACM (2013)

8. Chakarov, A., Sankaranarayanan, S.: Probabilistic program analysis with martingales. In: International Conference on Computer Aided Verification. pp. 511–526. Springer (2013)
9. Chakraborty, S., Fremont, D., Meel, K., Seshia, S., Vardi, M.: Distribution-aware sampling and weighted model counting for sat (2014)
10. Chatterjee, K., Fu, H., Novotnỳ, P., Hasheminezhad, R.: Algorithmic analysis of qualitative and quantitative termination problems for affine probabilistic programs. ACM SIGPLAN Notices 51(1), 327–342 (2016)
11. Chavira, M., Darwiche, A.: On probabilistic inference by weighted model counting. Artificial Intelligence 172(6-7), 772–799 (2008)
12. Chistikov, D., Dimitrova, R., Majumdar, R.: Approximate counting in SMT and value estimation for probabilistic programs. In: Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings. pp. 320–334 (2015), `http://dx.doi.org/10.1007/978-3-662-46681-0_26`
13. Cousot, P., Monerau, M.: Probabilistic abstract interpretation. In: Programming Languages and Systems, pp. 169–193. Springer (2012)
14. De Loera, J., Dutra, B., Koeppe, M., Moreinis, S., Pinto, G., Wu, J.: Software for exact integration of polynomials over polyhedra. ACM Communications in Computer Algebra 45(3/4), 169–172 (2012)
15. De Raedt, L., Kersting, K.: Probabilistic inductive logic programming. In: Probabilistic Inductive Logic Programming, pp. 1–27. Springer (2008)
16. Dwork, C.: Differential privacy. In: Automata, languages and programming, pp. 1–12. Springer (2006)
17. Esparza, J., Kucera, A., Mayr, R.: Model checking probabilistic pushdown automata. In: Logic in Computer Science, 2004. Proceedings of the 19th Annual IEEE Symposium on. pp. 12–21. IEEE (2004)
18. Etessami, K., Yannakakis, M.: Recursive markov chains, stochastic grammars, and monotone systems of nonlinear equations. In: Annual Symposium on Theoretical Aspects of Computer Science. pp. 340–352. Springer (2005)
19. Filieri, A., Păsăreanu, C.S., Visser, W.: Reliability analysis in symbolic pathfinder. In: Proceedings of the 2013 International Conference on Software Engineering. pp. 622–631. IEEE Press (2013)
20. Fuhr, N.: Probabilistic datalogâĂŤa logic for powerful retrieval methods. In: Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval. pp. 282–290. ACM (1995)
21. Gehr, T., Misailovic, S., Vechev, M.: Psi: Exact symbolic inference for probabilistic programs. In: Computer aided verification. Springer (2016)
22. Geldenhuys, J., Dwyer, M.B., Visser, W.: Probabilistic symbolic execution. In: Proceedings of the 2012 International Symposium on Software Testing and Analysis. pp. 166–176. ACM (2012)
23. Goodman, N.D.: The principles and practice of probabilistic programming. ACM SIGPLAN Notices 48(1), 399–402 (2013)
24. Grebenshchikov, S., Gupta, A., Lopes, N.P., Popeea, C., Rybalchenko, A.: Hsf (c): a software verifier based on horn clauses. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 549–551. Springer (2012)
25. Grebenshchikov, S., Lopes, N.P., Popeea, C., Rybalchenko, A.: Synthesizing software verifiers from proof rules. ACM SIGPLAN Notices 47(6), 405–416 (2012)

26. Gulwani, S., McCloskey, B., Tiwari, A.: Lifting abstract interpreters to quantified logical domains. In: ACM SIGPLAN Notices. vol. 43, pp. 235–246. ACM (2008)
27. Gurfinkel, A., Kahsai, T., Komuravelli, A., Navas, J.A.: The seahorn verification framework. In: International Conference on Computer Aided Verification. pp. 343–361. Springer (2015)
28. Heath, J., Kwiatkowska, M., Norman, G., Parker, D., Tymchyshyn, O.: Probabilistic model checking of complex biological pathways. Theoretical Computer Science 391(3), 239–257 (2008)
29. Jha, S.K., Clarke, E.M., Langmead, C.J., Legay, A., Platzer, A., Zuliani, P.: A bayesian approach to model checking biological systems. In: International Conference on Computational Methods in Systems Biology. pp. 218–234. Springer (2009)
30. Katoen, J.P., McIver, A.K., Meinicke, L.A., Morgan, C.C.: Linear-invariant generation for probabilistic programs. In: Static Analysis, pp. 390–406. Springer (2010)
31. Kozen, D.: Semantics of probabilistic programs. Journal of Computer and System Sciences 22(3), 328–350 (1981)
32. Kwiatkowska, M., Norman, G., Parker, D.: Prism 4.0: Verification of probabilistic real-time systems. In: Computer aided verification. pp. 585–591. Springer (2011)
33. Lal, A., Qadeer, S.: Dag inlining: a decision procedure for reachability-modulo-theories in hierarchical programs. In: ACM SIGPLAN Notices. vol. 50, pp. 280–290. ACM (2015)
34. Legay, A., Delahaye, B., Bensalem, S.: Statistical model checking: An overview. In: International Conference on Runtime Verification. pp. 122–135. Springer (2010)
35. Luckow, K., Păsăreanu, C.S., Dwyer, M.B., Filieri, A., Visser, W.: Exact and approximate probabilistic symbolic execution for nondeterministic programs. In: Proceedings of the 29th ACM/IEEE international conference on Automated software engineering. pp. 575–586. ACM (2014)
36. McIver, A., Morgan, C.C.: Abstraction, refinement and proof for probabilistic systems. Springer Science & Business Media (2006)
37. McMillan, K.L., Rybalchenko, A.: Solving constrained horn clauses using interpolation. Tech. Rep. MSR-TR-2013-6 (2013)
38. Monniaux, D.: Abstract interpretation of probabilistic semantics. In: Static Analysis, pp. 322–339. Springer (2000)
39. Monniaux, D.: An abstract monte-carlo method for the analysis of probabilistic programs. In: ACM SIGPLAN Notices. vol. 36, pp. 93–101. ACM (2001)
40. Pfeffer, A., Koller, D.: Semantics and inference for recursive probability models. In: AAAI/IAAI. pp. 538–544 (2000)
41. Sampson, A., Panchekha, P., Mytkowicz, T., McKinley, K.S., Grossman, D., Ceze, L.: Expressing and verifying probabilistic assertions. ACM SIGPLAN Notices 49(6), 112–122 (2014)
42. Sankaranarayanan, S., Chakarov, A., Gulwani, S.: Static analysis for probabilistic programs: inferring whole program properties from finitely many paths. In: ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '13, Seattle, WA, USA, June 16-19, 2013. pp. 447–458 (2013), `http://doi.acm.org/10.1145/2462156.2462179`
43. Stockmeyer, L.: On approximation algorithms for# p. SIAM Journal on Computing 14(4), 849–861 (1985)
44. Wojtczak, D., Etessami, K.: Premo: an analyzer for probabilistic recursive models. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 66–71. Springer (2007)