Verifying Probabilistic Programs Using a Hoare like Logic

Tim Quatmann
Seminar on Probabilistic Programs WS 2014/15
RWTH Aachen, Lehrstuhl für Informatik 2
Supervisor: Nils Jansen

February 2, 2015

Abstract

Using Hoare logic is a common way to formally proof correct behavior of a computer system. So-called Hoare triples of the form $\{\, p \,\}$ `s` $\{\, q \,\}$ are used to verify that the postcondition q holds after executing a program `s`, providing that the precondition p was true at the beginning. The original Hoare logic is only applicable for deterministic programs. However, this class of programs is not sufficient for certain tasks. Various application areas like computer vision, cryptographic protocols, and biology motivate the usage of probabilistic programs. In this work, we will discuss how the idea of Hoare logic can be extended to verify claims regarding probabilistic programs. For this purpose, a language for probabilistic programs $\mathcal{L}_{\text{pw}}$ is introduced. Probabilistic predicates are defined in order to make claims about the state of a probabilistic program. Finally, a derivation system pH is introduced which can be used to formally proof the correctness of a Hoare triple.

1 Introduction

Unexpected behavior of computer systems can have serious consequences, especially for critical applications like airplanes or medical equipment. A method to formally proof the correctness of a program is very helpful to rule out such situations. One common approach to achieve this is the usage of *Hoare logic.* The idea is to express predicates with respect to the variables of the considered program. Depending on the current state (i.e., the assignment of the variables) these predicates evaluate to either *true* or *false*. We then formulate so-called Hoare triples of the form $\{\, p \,\}$ `s` $\{\, q \,\}$, where p is a predicate which is called the precondition, q is a predicate called postcondition, and `s` is a program. A Hoare triple is said to hold if for all states of the program where the precondition p evaluates to *true*, the postcondition q is also *true* after executing the program `s`. Consider, for instance, the Hoare triples $\{\, \textit{true} \,\}\ x := x \cdot 3\ \{\, x \bmod 6 = 0 \,\}$ and $\{\, x \bmod 2 = 0 \,\}\ x := x \cdot 3\ \{\, x \bmod 6 = 0 \,\}$. While the first Hoare triple does not hold (e.g., when initially $x = 1$), the second one holds since every even number multiplied with

3 is divisible by 6. For more complex programs, a derivation system H can be used to verify whether a Hoare triple holds.

The described approach is designed to work for *deterministic programs*, i.e., programs where there is no choice between different program flows. However, various application areas such as computer vision, cryptographic protocols, machine learning, and biology make use of systems that exhibit probabilistic behavior. In such cases, *probabilistic programs* can be used to model the system. One advantage of probabilistic programs is that they can be expressed in a way similar to widely known programming languages like C or Java. Moreover, other famous models for probabilistic systems such as Markov Chains [2] or Bayesian Networks [6] can be encoded as probabilistic programs [5]. Exemplary applications like skill rating in online games, population models in biology, or sensitivity analysis regarding erroneous test data in medicine are presented in [5].

In probabilistic programs, a (virtual) coin is flipped and the subsequent steps depend on the outcome. Such a coin flip is denoted by $\mathtt{s} \oplus_r \mathtt{s}'$ with the interpretation that the program $\mathtt{s}$ will be executed with probability r and the program $\mathtt{s}'$ will be executed with probability $1 - r$. Consider, for instance, the following program:

$$x := 0 \oplus_{0.2} x := 1$$

After this step $x = 0$ holds with probability 0.2 and $x = 1$ holds with probability 0.8. Hoare logic is not directly applicable for probabilistic programs since a predicate like $x \bmod 2 = 0$ only evaluates to *true* with a certain probability. In this work, we will deal with this problematic.

The topics of this work originate from [3]. A more detailed description can also be found in the PhD thesis of J.I. den Hartog [4]. However, we are going to focus on the actual application of the described notions. Involved definitions that are of less importance for our purposes will therefore be omitted in favor of comprehensive examples.

After a brief introduction to Hoare logic for non-probabilistic programs in Section 2, we will introduce probabilistic choices for programs and the notion of probabilistic states in Section 3. Section 4 presents predicates that evaluate to *true* or *false* on probabilistic states. A derivation system to prove the correctness of Hoare triples for probabilistic programs is given in Section 5. Finally, Section 6 concludes the presented ideas.

2 Hoare Logic for Deterministic Programs

Before we investigate the application of Hoare logic for probabilistic programs, we briefly present the non-probabilistic case. An extensive treatment of Hoare logic can be found in [1].

We make use of a simple, but seminal language for programs, so-called *while programs*. Let Var be a fixed set of variables. For simplicity, we assume that the value of the variables range over a fixed set Val (usually, non-negative integers). Expressions as well as Boolean conditions are terms over the values in Val and the variables in Var. The set of all Expressions is denoted by Exp, whereas the set of Boolean conditions is denoted by BC. We want to abstract away from the internal details of expressions and Boolean conditions. Therefore, we will not define a precise syntax of the elements in Exp and BC. Instead of that, it is assumed that (given the value of the variables) an expression evaluates to a value in Val and a Boolean condition evaluates to either *true* or *false*. We can now define the syntax of while programs.

Definition 2.1 (While Program)
A while program is an element of the language $\mathcal{L}_\mathrm{w}$ which is given by:

$$\mathsf{s} ::= \mathsf{skip} \mid x := e \mid \mathsf{s};\mathsf{s} \mid \mathsf{if}\ c\ \mathsf{then}\ \mathsf{s}\ \mathsf{else}\ \mathsf{s}\ \mathsf{fi} \mid \mathsf{while}\ c\ \mathsf{do}\ \mathsf{s}\ \mathsf{od}$$

where $x \in \mathrm{Var}$ is a variable, $e \in \mathrm{Exp}$ an expression, and $c \in \mathrm{BC}$ a Boolean condition. ■

The elements of $\mathcal{L}_\mathrm{w}$ are also called *statements* and exhibit the well known semantics. The statement skip denotes a program that does nothing. If we want to assign the value of an expression e to some variable x, we write $x := e$. A sequence of two statements $\mathsf{s};\mathsf{s}'$ denotes that s is executed first and after that s' is executed. Executing a statement of the form $\mathsf{if}\ c\ \mathsf{then}\ \mathsf{s}\ \mathsf{else}\ \mathsf{s}'\ \mathsf{fi}$ means that s is executed providing that the condition c evaluates to *true*. However, if c evaluates to *false*, the statement s' is executed. Finally, $\mathsf{while}\ c\ \mathsf{do}\ \mathsf{s}\ \mathsf{od}$ denotes a while loop. As long as the condition c evaluates to *true*, the program s is repeatedly executed. Typically, we use $x, y, z \in \mathrm{Var}$ as variables, $e \in \mathrm{Exp}$ as expression, and $c \in \mathrm{BC}$ as Boolean condition. Occasionally, parentheses will be added to increase readability.

Example 2.2 (A While Program)
Consider the following while program:

$$\mathsf{if}\ y = 2\ \mathsf{then}\ x := 3 \cdot x\ \mathsf{else}\ x := 2 \cdot x\ \mathsf{fi}$$

Assume that initially the value of x is 4 and the value of y is 2. Executing the given program will change x to 12 and y remains 2. ■

An assignment of the variables Var to values in Val is called a *deterministic state*.

Definition 2.3 (Deterministic State)
A deterministic state is a mapping $\sigma\colon \mathrm{Var} \to \mathrm{Val}$. The set of all deterministic states is denoted by $\mathcal{S}$. ■

In order to describe states in $\mathcal{S}$, *deterministic predicates* can be used to formulate constraints with respect to the variable assignments. These predicates are first order predicate formulae which evaluate to *true* or *false*, depending on the values of the variables given by a deterministic state σ. We say that a state $\sigma \in \mathcal{S}$ satisfies a predicate p (written $\sigma \models p$) whenever p evaluates to *true* with respect to σ.

Given a while program s for which we want to prove a claim, the idea of Hoare logic is to formulate Hoare triples of the form $\{\, p \,\}\ \mathsf{s}\ \{\, q \,\}$. Deterministic predicates are used to formulate a precondition p and a postcondition q. Assume that we start with a state $\sigma \in \mathcal{S}$ that satisfies the precondition ($\sigma \models p$). The Hoare triple $\{\, p \,\}\ \mathsf{s}\ \{\, q \,\}$ is said to hold (written $\models \{\, p \,\}\ \mathsf{s}\ \{\, q \,\}$) whenever the state σ' that is reached after executing s satisfies the postcondition ($\sigma' \models q$). In this case, we also say that the Hoare triple is valid. Note that a valid Hoare triple does not imply that the considered program terminates, i.e., that for all occurring statements of the form $\mathsf{while}\ c\ \mathsf{do}\ \mathsf{s}\ \mathsf{od}$ the condition c becomes eventually *false*. This has to be shown separately.

Example 2.4 (Predicates and Hoare Triples for a While Program)
Let us first consider the predicate $p = (x \bmod y = 0) \wedge (y = 2)$ over the integer variables x and y. It evaluates to *true* at every point in the considered program, where the variable x is even and y equals 2. After executing the program $x := 3 \cdot x$, the new value of x is divisible by 3 and still even. Therefore, the predicate $q = (x \bmod 6 = 0) \wedge (y = 2)$ will always hold in the resulting state. We can conclude that the following Hoare triple holds.

$$\{\, p \,\}\ \mathsf{s}\ \{\, q \,\} = \{\, (x \bmod y = 0) \wedge (y = 2) \,\}\ x := 3 \cdot x\ \{\, (x \bmod 6 = 0) \wedge (y = 2) \,\} \qquad \blacksquare$$

In Example 2.4 it is easy to see that the Hoare triple $\{\, p \,\}\ \mathsf{s}\ \{\, q \,\}$ holds. However, if the considered program is more complex, the derivation system H can be used to verify that a Hoare triple holds.

Definition 2.5 (Derivation System H)
The derivation system H is given by the following set of rules.

$$\{\, p \,\}\ \mathsf{skip}\ \{\, p \,\} \quad \text{(Skip)} \qquad \frac{\{\, p \wedge c \,\}\ \mathsf{s}\ \{\, q \,\} \quad \{\, p \wedge \neg c \,\}\ \mathsf{s}'\ \{\, q \,\}}{\{\, p \,\}\ \mathsf{if}\ c\ \mathsf{then}\ \mathsf{s}\ \mathsf{else}\ \mathsf{s}'\ \mathsf{fi}\ \{\, q \,\}}\ \text{(If)}$$

$$\{\, p[x/e] \,\}\ x := e\ \{\, p \,\} \quad \text{(Assign)} \qquad \frac{\{\, p \wedge c \,\}\ \mathsf{s}\ \{\, p \,\}}{\{\, p \,\}\ \mathsf{while}\ c\ \mathsf{do}\ \mathsf{s}\ \mathsf{od}\ \{\, p \wedge \neg c \,\}}\ \text{(While)}$$

$$\frac{\{\, p \,\}\ \mathsf{s}\ \{\, p' \,\} \quad \{\, p' \,\}\ \mathsf{s}'\ \{\, q \,\}}{\{\, p \,\}\ \mathsf{s}; \mathsf{s}'\ \{\, q \,\}}\ \text{(Seq)} \qquad \frac{p' \Rightarrow p \quad \{\, p \,\}\ \mathsf{s}\ \{\, q \,\} \quad q \Rightarrow q'}{\{\, p' \,\}\ \mathsf{s}\ \{\, q' \,\}}\ \text{(Cons)}$$

$\blacksquare$

The rule (Skip) states that every predicate that holds before executing the program skip still holds after the execution. The rule (Assign) states that a predicate p is satisfied after an assignment of the form $x := e$ whenever the predicate $p[x/e]$ was satisfied before the assignation. Here, $p[x/e]$ is obtained from the predicate p where all occurrences of x are replaced by e. For a sequence of two statements $\mathsf{s}; \mathsf{s}'$ and a predicate p that is satisfied before the execution, the rule (Seq) states that q is satisfied after the execution, providing that there is a predicate p' which is satisfied after executing s and which is a sufficient precondition such that q holds after executing s'. The rule (If) handles if-statements by considering the two cases where the condition c is *true* and *false*. Providing that q is a valid postcondition in both cases, it can be obtained that q is also a valid post condition for the whole statement. To treat a statement of the form while c do s od, the rule (While) can be applied. Here, the precondition p has to be invariant; providing that the condition c is *true* and after executing the program s, p still has to hold. In this case, we know that c is *false* after the execution of the whole while-statement and the predicate p still holds. The consequence rule (Cons) can be used to strengthen a precondition or to weaken a postcondition.

Example 2.6 (Proof Using the Derivation System H)
Let s be the while program from Example 3.2. To verify that the Hoare triple

$$\{\, x \bmod y = 0 \land (y = 2 \lor y = 3) \,\}\ \mathsf{s}\ \{\, x \bmod 6 = 0 \,\}$$

holds, the following proof is provided.

```
⟨$x \bmod y = 0 \land (y = 2 \lor y = 3)$⟩
if ($y = 2$) then
   ⟨$x \bmod y = 0 \land (y = 2 \lor y = 3) \land y = 2$⟩   (†)
   ⟨$3 \cdot x \bmod 6 = 0$⟩
   $x := 3 \cdot x$
   ⟨$x \bmod 6 = 0$⟩                                        (†)
else
   ⟨$x \bmod y = 0 \land (y = 2 \lor y = 3) \land y \neq 2$⟩   (†)
   ⟨$2 \cdot x \bmod 6 = 0$⟩
   $x := 2 \cdot x$
   ⟨$x \bmod 6 = 0$⟩                                        (†)
fi
⟨$x \bmod 6 = 0$⟩
```

Note that a proof outline is provided instead of a proof tree to increase readability. Predicates are depicted in ⟨angle brackets⟩. Whenever a predicate is followed by another predicate, the rule (Cons) has been applied.

In the first step, we apply the rule (If) which yields the predicates marked with (†). The application of the rule (Cons) in both cases is valid, since

$$\begin{array}{ll} x \bmod y = 0 \wedge (y = 2 \vee y = 3) \wedge y = 2 & x \bmod y = 0 \wedge (y = 2 \vee y = 3) \wedge y \neq 2 \\ \Rightarrow 3 \cdot x \bmod 3 \cdot y = 0 \wedge y = 2 & \Rightarrow 2 \cdot x \bmod 2 \cdot y = 0 \wedge y = 3 \\ \Rightarrow 3 \cdot x \bmod 6 = 0 & \Rightarrow 2 \cdot x \bmod 6 = 0 \end{array}$$

Finally, we apply the rule (Assign) in both cases to obtain the desired post condition $x \bmod 6 = 0$. ■

3 Introducing Probabilistic Choice

For many applications it is necessary to allow probabilistic choices in a program. For this purpose we introduce the operator $\oplus_r$ to the language of while programs $\mathcal{L}_\mathrm{w}$ from Definition 2.1. We then obtain the language of *probabilistic programs.*

Definition 3.1 (Probabilistic Program)
A probabilistic program is an element of the language $\mathcal{L}_\mathrm{pw}$ which is given by:

$$\mathsf{s} ::= \mathsf{skip} \mid x := e \mid \mathsf{s}; \mathsf{s} \mid \mathsf{s} \oplus_r \mathsf{s} \mid \mathsf{if}\ c\ \mathsf{then}\ \mathsf{s}\ \mathsf{else}\ \mathsf{s}\ \mathsf{fi} \mid \mathsf{while}\ c\ \mathsf{do}\ \mathsf{s}\ \mathsf{od}$$

where $x \in \mathrm{Var}$ is a variable, $e \in \mathrm{Exp}$ an expression, $r \in (0,1) \subseteq \mathbb{R}$ a probability ratio, and $c \in \mathrm{BC}$ a Boolean condition. ■

Let s and s' be statements in $\mathcal{L}_\mathrm{pw}$ and $r \in (0,1) \subseteq \mathbb{R}$. A statement of the form $\mathsf{s} \oplus_r \mathsf{s}'$ can be interpreted as a (biased) coin flip. With probability r the coin shows “heads”, which means that the statement s is executed. Analogously, the coin shows tails with probability $1-r$ and in this case s' is executed. Beside the new operator for probabilistic choices, the ingredients of $\mathcal{L}_\mathrm{pw}$ are analogously defined to $\mathcal{L}_\mathrm{w}$ (including the sets Val, Var, Exp and BC).

Example 3.2 (Two Simple Probabilistic Programs)
Consider the two probabilistic programs s_1 and s_2 over $\mathrm{Var} = \{x, y\}$ and $\mathrm{Val} = \mathbb{N}$ depicted below.

```
s1:
    (x := 0 ⊕0.3 x := 1);
    if (x = 0) then
      y := 1
    else
      y := 0
    fi
```

```
s2:
    x := 0;
    y := 0;
    while (y < N) do
      y := y + 1;
      (x := x + 1 ⊕R skip)
    od
```

In $\mathtt{s}_1$ we start with flipping a biased coin: with probability 0.3 the value 0 is assigned to the variable x. Otherwise, x gets the value 1. The contrary value is then assigned to the variable y.

The probabilistic program $\mathtt{s}_2$ simulates N coin flips of a (biased) coin that shows "heads" with probability R. Note that N and R are constants and assumed to be specified before execution. The number of already performed coin flips is counted by the variable y. The variable x is used to count the number of times where the coin shows "heads", as x is incremented with probability R in every iteration. ■

In the next section, we will discuss predicates for probabilistic programs which evaluate to *true* or *false*, depending on the state of the program. In general, a deterministic state $\sigma\colon \mathrm{Var} \to \mathrm{Val}$ does not capture enough information to describe the state of a probabilistic program as the value of a variable may depend on former probabilistic choices. For instance, after executing the program $\mathtt{s}_1$ from Example 3.2, the value of the variable x is 1 only with a certain probability. The notion of *probabilistic states* is introduced to describe all possible variable assignments together with the corresponding probabilities. It is not sufficient to treat the possible assignments of every variable individually since their values may depend on each other. After executing the program $\mathtt{s}_1$ from Example 3.2, for instance, we have that $x = 1$ with probability 0.7 and $y = 1$ with probability 0.3. However, the probability that both, $x = 1$ and $y = 1$, hold is 0. To capture all information, we will consider the probability that an assignment of all variables (i.e., a deterministic state) holds.

Definition 3.3 (Probabilistic State)
A probabilistic state is a mapping $\theta\colon \mathcal{S} \to [0,1]$ such that $\sum_{\sigma\in\mathcal{S}} \theta(\sigma) \leq 1$. The set of all probabilistic states is denoted by Π. ■

Recall that according to Definition 2.3, $\mathcal{S}$ is the set of all deterministic states. Hence, a probabilistic state assigns probabilities to deterministic states. $\theta(\sigma) = r$ means that the variable assignment corresponds to $\sigma \in \mathcal{S}$ with probability $r \in [0,1]$.

The semantics for $\mathcal{L}_{\mathrm{pw}}$ can now be defined as a mapping $\mathcal{D}\colon \mathcal{L}_{\mathrm{pw}} \to (\Pi \to \Pi)$ that states how a probabilistic state changes after a given program is executed. Assuming that $\theta \in \Pi$ is the current state, then $\theta' = \mathcal{D}(\mathtt{s})(\theta)$ is the new probabilistic state that is reached after the program $\mathtt{s}$ has been executed. A formal definition of $\mathcal{D}$ is omitted in this work since it is of less importance for the application of Hoare logic. For more information, we refer to [3].

A probabilistic state $\theta \in \Pi$ is substochastic, i.e., $\sum_{\sigma\in\mathcal{S}} \theta(\sigma) \leq 1$. The "missing" probability can intuitively be described as the probability of not reaching that part of the program. An example for this is a statement of the form $\mathsf{if}\ c\ \mathsf{then}\ \mathtt{s}\ \mathsf{else}\ \mathtt{s}'\ \mathsf{fi}$. For the if case, we only consider the deterministic states where the condition c is *true*. In this case, the probabilities for the deterministic states where c is *false* are missing. Another reason for missing probabilities can be a loop which does not terminate with probability 1. The difference between the sum of the probabilities before and after executing a statement of the form while c do s od corresponds to the probability of non-termination.

Example 3.4 (Probabilistic State)
Assume that for the current state $\theta \in \Pi$, the sum of the occurring probabilities is 1, i.e., $\sum_{\sigma \in \mathcal{S}} \theta(\sigma) = 1$. After executing the program $\mathtt{s_1}$ from Example 3.2, the values of the variables x and y correspond to one of the deterministic states $\sigma_1, \sigma_2 \in \mathcal{S}$, where

$$\sigma_1(x) = 1 \qquad \sigma_1(y) = 0 \quad \text{and} \quad \sigma_2(x) = 0 \qquad \sigma_2(y) = 1.$$

The corresponding probabilistic state $\theta' \in \Pi$ is given by

$$\theta'(\sigma_1) = 0.7 \quad \text{and} \quad \theta'(\sigma_2) = 0.3.$$

As a more compact notation, we can also depict a probabilistic state in form of a table:

$\theta'(\sigma_i)$	0.7	0.3
$\sigma_i(x)$	1	0
$\sigma_i(y)$	0	1

Deterministic states $\sigma \in \mathcal{S}$ with $\theta(\sigma) = 0$ are usually omitted whenever a probabilistic state θ is depicted. ■

We will now define some operations on probabilistic states in order to ease the definition of probabilistic predicates in the next section.

Definition 3.5 (Operations On Probabilistic States)
Restricting, scaling and merging of probabilistic states is defined as follows.

$$c?\theta(\sigma) = \begin{cases} \theta(\sigma) & \text{if } c \text{ is } \textit{true} \text{ with respect to } \sigma \\ 0 & \text{otherwise} \end{cases}$$

$$(r \cdot \theta)(\sigma) = r \cdot \theta(\sigma) \quad \text{if} \sum_{\sigma' \in \mathcal{S}} \big(r \cdot \theta(\sigma')\big) \leq 1$$

$$(\theta + \theta')(\sigma) = \theta(\sigma) + \theta'(\sigma) \quad \text{if} \sum_{\sigma' \in \mathcal{S}} \big(\theta(\sigma') + \theta'(\sigma')\big) \leq 1$$

where $\theta, \theta' \in \Pi$, $c \in \mathrm{BC}$, $r \in (0, 1)$, and σ ranges over all deterministic states in $\mathcal{S}$. ■

A state $c?\theta \in \Pi$ is a restriction of θ such that only the deterministic states $\sigma \in \mathcal{S}$ are considered, where c evaluates to *true*. The deterministic states where c is *false* are suppressed, i.e., their probability is set to 0. With $r \cdot \theta \in \Pi$ we can express a scaled version of θ in terms of the probabilities $\theta(\sigma)$. In $\theta + \theta' \in \Pi$, the probabilities occurring in both probabilistic states θ, θ' are added. This state can be considered if we want to merge two probabilistic states.

Example 3.6 (Operations on Probabilistic States)
Consider the two probabilistic states $\theta, \theta' \in \Pi$

$\theta(\sigma_i)$	0.3	0.2	0.1
$\sigma_i(x)$	1	0	0
$\sigma_i(y)$	0	1	0

$\theta'(\sigma_i)$	0.2	0.2
$\sigma_i(x)$	1	1
$\sigma_i(y)$	1	0

Let $c = (x = 1) \in \mathrm{BC}$. Applying the operations defined above yields:

$c?\theta(\sigma_i)$	0.3
$\sigma_i(x)$	1
$\sigma_i(y)$	0

$\neg c?\theta(\sigma_i)$	0.2	0.1
$\sigma_i(x)$	0	0
$\sigma_i(y)$	1	0

$(0.5 \cdot \theta')(\sigma_i)$	0.1	0.1
$\sigma_i(x)$	1	1
$\sigma_i(y)$	1	0

$(\theta + \theta')(\sigma_i)$	0.2	0.5	0.2	0.1
$\sigma_i(x)$	1	1	0	0
$\sigma_i(y)$	1	0	1	0

It is also easy to see that $c?\theta + \neg c?\theta = \theta$. Note that this equation holds for arbitrary $\theta \in \Pi$ and $c \in \mathrm{BC}$. ■

4 Probabilistic Predicates

We now discuss how predicates for probabilistic states can be formulated. We already mentioned that we want to formulate predicates that, given a probabilistic state, evaluate to either *true* or *false*. Another option is to consider deterministic predicates, i.e., first order predicate formulae like it is done for deterministic programs. Such a predicate can then be seen as a function which takes a probabilistic state θ and maps it to a probability ratio $r \in [0, 1]$. The interpretation is that the predicate is satisfied in θ with probability r. Such an approach is used in e.g. [7, 8]. However, we will later see that the idea of Hoare logic is almost directly applicable if we consider predicates that clearly evaluate to either *true* or *false*. Such predicates will be called *probabilistic predicates*. The basic idea is to use a construct of the form $\mathbb{P}(dp){\prec}e_{\text{const}}$ to express that the probability of the deterministic predicate dp to hold is $\prec e_{\text{const}}$, where $\prec \in \{<, \leq, =, \geq, >\}$.

Definition 4.1 (Probabilistic Predicate)
The syntax of probabilistic predicates is given by

$$p ::= \mathbb{P}(dp){\prec}e_{\text{const}} \mid p + p \mid r \cdot p \mid c?p \mid \neg p \mid p \vee p \mid p \wedge p \mid p \to p \mid \exists i\colon p \mid \forall i\colon p$$

where dp is a deterministic predicate (i.e., a first order predicate formula), $\prec \in \{<, \leq, =, \geq, >\}$, e_{const} is an expression without program variables that evaluates to a number in $[0, 1]$, $r \in (0, 1) \subseteq \mathbb{R}$, and $c \in \mathrm{BC}$. Let $\theta \in \Pi$ be a probabilistic state and p and q two probabilistic predicates. The satisfaction relation "$\models$" is defined as follows.

$$\begin{aligned}
&\theta \models \mathbb{P}(dp){\prec}e_{\text{const}} \text{ iff } \sum_{\sigma\models dp} \theta(\sigma) \prec e_{\text{const}}\\
&\theta \models p + q \text{ iff there are } \theta_1, \theta_2 \in \Pi \text{ with } \theta = \theta_1 + \theta_2,\ \theta_1 \models p, \text{ and } \theta_2 \models q\\
&\theta \models r \cdot p \text{ iff there is a } \theta' \in \Pi \text{ with } \theta = r \cdot \theta' \text{ and } \theta' \models p\\
&\theta \models c?p \text{ iff there is a } \theta' \in \Pi \text{ with } \theta = c?\theta' \text{ and } \theta' \models p.
\end{aligned}$$

The remaining logical constructs are defined as usual. ■

Substitution of a variable x by an expression e on a probabilistic predicate p (written $p[x/e]$) is passed down until a deterministic predicate is reached, e.g.:

$$\begin{aligned}
(\mathbb{P}(dp){\prec}e_{\text{const}})[x/e] &= \mathbb{P}(dp[x/e]){\prec}e_{\text{const}}\\
(p + q)[x/e] &= p[x/e] + q[x/e]\\
(c?p)[x/e] &= c[x/e]?p[x/e]
\end{aligned}$$

We make use of the following shorthand notations:

$$\begin{aligned}
p \oplus_r q &= r \cdot p + (1 - r) \cdot q\\
[dp] &= \mathbb{P}(dp){=}1\\
\text{not}\, c &= \mathbb{P}(c){=}0\\
\mathit{true} &= \mathbb{P}(\mathit{true}){\geq}0
\end{aligned}$$

Intuitively, a state satisfies a predicate of the form $p+q$ if it is a merged states obtained from two states that satisfy p and q, respectively. The predicate $r \cdot p$ is satisfied by a state, whenever the state is a scaled version of another state satisfying p. The two ideas are combined in the predicate $p \oplus_r q$, which is satisfied if the considered state is a merged state obtained from two parts that are scaled versions of states satisfying p and q, respectively. A predicate of the form $c?p$ is satisfied by a state, whenever that state is the restricted "c-part" of another state that satisfies p.,

Example 4.2 (Probabilistic Predicates)
Consider the following probabilistic states $\theta \in \Pi$.

$\theta(\sigma_i)$	0.2	0.3	0.1	0.4
$\sigma_i(x)$	1	1	0	0
$\sigma_i(y)$	1	0	1	0

It holds that $\theta \models \big(\mathbb{P}(x = 1){=}0.5\big) \wedge \big(\mathbb{P}(x = 0){=}0.5\big)$. For $p_1 = \big(\mathbb{P}(x = 0 \wedge y = 1){=}0.1\big)$ and $p_2 = \big(\mathbb{P}(x = 0){=}0.2\big)$ it holds that $\theta \models p_1 + p_2$ as we can write θ as $\theta_1 + \theta_2$ such that $\theta_1 \models p_1$ and $\theta_2 \models p_2$:

$\theta_1(\sigma_i)$	0.2	0.1	0.2
$\sigma_i(x)$	1	0	0
$\sigma_i(y)$	1	1	0

$\theta_2(\sigma_i)$	0.3	0.2
$\sigma_i(x)$	1	0
$\sigma_i(y)$	0	0

It is easy to see that the state θ_1 satisfies $0.5\cdot\Big(\big(\mathbb{P}(y=1){=}0.6\big)\wedge\big(\mathbb{P}(x=0\wedge y=0){=}0.4\big)\Big)$. Let us now consider the predicate q given by $[y=1]\vee[y=0]$. We can see that $\theta\not\models q$ since the probability for $y=1$ sums up to $0.3<1$ and for $y=0$ we obtain the probability $0.7<1$. However, $q\oplus_{0.3}q$ is satisfied by θ as there are $\theta'_1,\theta'_2\in\Pi$ such that $\theta'_1,\theta'_2\models q$ and θ can be split into $0.3\cdot\theta'_1+0.7\cdot\theta'_2$:

$\theta'_1(\sigma_i)$	2/3	1/3
$\sigma_i(x)$	1	0
$\sigma_i(y)$	1	1

$\theta'_2(\sigma_i)$	3/7	4/7
$\sigma_i(x)$	1	0
$\sigma_i(y)$	0	0

$0.3\cdot\theta'_1(\sigma_i)$	0.2	0.1
$\sigma_i(x)$	1	0
$\sigma_i(y)$	1	1

$0.7\cdot\theta'_2(\sigma_i)$	0.3	0.4
$\sigma_i(x)$	1	0
$\sigma_i(y)$	0	0

Let θ_c be the state $0.3\cdot\theta'_1$ depicted above. For $c=(y=1)\in\mathrm{BC}$ it holds that $\theta_c=c?\theta$ and therefore the predicate $c?p$ is satisfied by θ_c for all p with $\theta\models p$. The reverse direction, however, does not hold since there are other states θ' for which $\theta_c=c?\theta'$. ■

Up to our knowledge, it is not clear whether $\theta\models p$ is decidable for arbitrary probabilistic states θ and probabilistic predicates p. Consider, for example, a predicate of the form $p+q$. As there are infinitely many possibilities to write a state $\theta\in\Pi$ as a merged state $\theta=\theta_1+\theta_2$, it is not trivial to prove that there are no θ_1,θ_2 for which $\theta_1\models p$ and $\theta_2\models q$. Note that [3] does not deal with this problem.

Working with probabilistic predicates frequently requires to show that a predicate p implies another predicate q, i.e., $\forall\theta\in\Pi\colon\theta\models p\implies\theta\models q$. Such implications are not always trivial, although they may seem clear at first sight. In Example 4.2, for instance, we have already seen that, in general, $p\oplus_r p$ does not imply p. It should also be noted that not every probabilistic predicate implies $[true]$ as we allow $\sum_{\sigma\models true}\theta(\sigma)=\sum_{\sigma\in\mathcal{S}}<1$ and thus $\theta\models\mathbb{P}(true){\neq}1$. Similar to the problem of deciding $\theta\models p$, it is also not clear if it is decidable whether p implies q for two probabilistic predicates p and q. Some correct implications are presented in the following example.

Example 4.3 (Implications for Probabilistic Predicates)
For a deterministic predicate dp, $r,r'\in(0,1)$ and $c\in BC$ it holds that

$$r\cdot\mathbb{P}(dp){=}r' \text{ implies } \mathbb{P}(dp){=}r\cdot r', \tag{1}$$

$$[dp]\oplus_r[\neg dp] \text{ implies } \big(\mathbb{P}(dp){=}r\big)\wedge\big(\mathbb{P}(\neg dp){=}1-r\big), \text{ and} \tag{2}$$

$$c?p \text{ implies } \mathbb{P}(\neg c){=}0. \tag{3}$$

Let $\theta \in \Pi$ be an arbitrary probabilistic state. Implication (1) holds since:

$$\begin{aligned}
&\theta \models r \cdot \mathbb{P}(dp)=r' \\
\Longrightarrow\ &\text{there is a } \theta' \text{ such that } \theta = r \cdot \theta' \text{ and } \theta' \models \mathbb{P}(dp)=r' \\
\Longrightarrow\ &\text{there is a } \theta' \text{ such that } \sum_{\sigma \models dp} \theta(\sigma) = \sum_{\sigma \models dp} r \cdot \theta'(\sigma) = r \cdot \underbrace{\sum_{\sigma \models dp} \theta'(\sigma)}_{=r'} = r \cdot r' \\
\Longrightarrow\ &\theta \models \mathbb{P}(dp)=r \cdot r'
\end{aligned}$$

The implication (2) can be proven as follows:

$$\begin{aligned}
&\theta \models [dp] \oplus_r [\neg dp] \\
\Longrightarrow\ &\text{there are } \theta_1, \theta_2 \text{ such that } \theta = \theta_1 + \theta_2,\ \theta_1 \models r \cdot [dp] \text{ and } \theta_2 \models (1-r) \cdot [\neg dp] \\
\overset{(1)}{\Longrightarrow}\ &\text{there are } \theta_1, \theta_2 \text{ such that } \theta = \theta_1 + \theta_2,\ \theta_1 \models \mathbb{P}(dp)=r \text{ and } \theta_2 \models \mathbb{P}(\neg dp)=(1-r)
\end{aligned}$$

We also know that $\theta_1 \models \mathbb{P}(\neg dp)=0$ and $\theta_2 \models \mathbb{P}(dp)=0$ because otherwise

$$\begin{aligned}
\sum_{\sigma \models true} \theta(\sigma) = \sum_{\sigma \models (dp \vee \neg dp)} \theta(\sigma) &= \sum_{\sigma \models (dp \vee \neg dp)} \theta_1(\sigma) + \sum_{\sigma \models (dp \vee \neg dp)} \theta_2(\sigma) \\
&= \Big(r + \sum_{\sigma \models \neg dp} \theta_1(\sigma)\Big) + \Big((1-r) + \sum_{\sigma \models dp} \theta_2(\sigma)\Big) > 1
\end{aligned}$$

would hold. We can conclude that $\theta \models \big(\mathbb{P}(dp)=r\big) \wedge \big(\mathbb{P}(\neg dp)=1-r\big)$.

Finally, implication (3) is correct since:

$$\begin{aligned}
&\theta \models c?p \\
\Longrightarrow\ &\text{there is a } \theta' \text{ such that } \theta = c?\theta' \\
\Longrightarrow\ &\theta(\sigma) = 0 \text{ if } c \text{ is } \textit{false} \text{ with respect to } \sigma \in \mathcal{S} \\
\Longrightarrow\ &\sum_{\sigma \models \neg c} \theta(\sigma) = 0 \ \Longrightarrow\ \theta \models \mathbb{P}(\neg c)=0
\end{aligned}$$

■

5 The Derivation System pH

In the previous section we presented probabilistic predicates which, given a probabilistic state, evaluate to *true* or *false*. With this notion the idea of Hoare triples as introduced for deterministic programs is directly applicable for the probabilistic case. A Hoare triple for probabilistic programs is of the form $\{\, p \,\}\ \mathtt{s}\ \{\, q \,\}$ where p and q are probabilistic predicates (called pre and postcondition) and $\mathtt{s} \in \mathcal{L}_{\mathrm{pw}}$ is a probabilistic program. Analogous to the deterministic case we write $\models \{\, p \,\}\ \mathtt{s}\ \{\, q \,\}$ to denote that a Hoare triple holds, i.e.,

$$\models \{\, p \,\}\ \mathtt{s}\ \{\, q \,\} \text{ iff } \forall \theta \in \Pi\colon\ \theta \models p \Rightarrow \mathcal{D}(\mathtt{s})(\theta) \models q.$$

Recall that $\mathcal{D}(\mathsf{s})(\theta)$ is the resulting probabilistic state after executing the program s and starting in θ. To make assertions for (parts of) probabilistic programs, Hoare triples can be formulated. It then remains to prove that a Hoare triple holds.

Example 5.1 (Hoare Triples for Probabilistic Programs)
Consider again the programs s_1 and s_2 from Example 3.2:

```
s1:
    ($x := 0 \oplus_{0.3} x := 1$);
    if ($x = 0$) then
      $y := 1$
    else
      $y := 0$
    fi
```

```
s2:
    $x := 0$;
    $y := 0$;
    while ($y < N$) do
      $y := y + 1$;
      ($x := x + 1 \oplus_R$ skip)
    od
```

Here $N \in \mathbb{N}$ and $R \in [0, 1]$ are constants. The Hoare triples

$$\models \{\, [\mathit{true}] \,\}\ \mathsf{s}_1\ \{\, \mathbb{P}(y = 0){=}0.7 \,\} \text{ and}$$
$$\models \{\, [\mathit{true}] \,\}\ \mathsf{s}_2\ \{\, \mathbb{P}(x = N){\geq}R^N \,\}$$

are valid. In both cases we start with a state where the probabilities sum up to 1. After executing the probabilistic choice at the first line of the program s_1, we know that x is 1 with probability 0.7. Therefore, the else case (i.e., $y := 0$) is executed with probability 0.7 and the program ends in a state which satisfies the stated postcondition. For the program s_2, we consider the case where x will get the value N. Such a variable assignment is only obtained if we execute $x := x + 1$ in all of the N iterations of the while-loop. Basic probability theory yields that this will occur with probability R^N and therefore the given postcondition holds. We will later formally proof the correctness of both Hoare triples. Two examples for Hoare triples that do not hold are given by:

$$\not\models \{\, [\mathit{true}] \,\}\ \mathsf{s}_1\ \{\, \mathbb{P}(x = 0 \land y = 0){>}0 \,\} \text{ and}$$
$$\not\models \{\, \mathit{true} \,\}\ \mathsf{s}_2\ \{\, \mathbb{P}(x = N){\geq}R^N \,\}.$$

There are two possible program flows for the program s_1. Depending on the outcome of the initial "coin flip" we either execute $x := 1$ or $y := 1$. Hence, it is not possible to end with a variable assignment where $x = y = 0$. The precondition of the last Hoare triple is satisfied for all states, including states θ with $\theta \models \mathbb{P}(\mathit{true}) = 0$. Starting from such a state we can not "grow" the required probability to satisfy the postcondition. ■

In the example above, it was more or less easy to see whether a Hoare triple holds or not, as the considered programs were simple. In general, this task is not trivial. The derivation system pH can be used to ease this task.

Definition 5.2 (Derivation System pH)
The derivation system pH is given by the following set of rules.

$$\{\,p\,\}\ \mathsf{skip}\ \{\,p\,\} \quad \text{(Skip)} \qquad \frac{\{\,p\,\}\ \mathsf{s}\ \{\,q\,\} \quad \{\,p'\,\}\ \mathsf{s}\ \{\,q\,\}}{\{\,p \vee p'\,\}\ \mathsf{s}\ \{\,q\,\}} \quad \text{(Or)}$$

$$\{\,p[x/e]\,\}\ x \coloneqq e\ \{\,p\,\} \quad \text{(Assign)} \qquad \frac{\{\,p[j]\,\}\ \mathsf{s}\ \{\,q\,\} \quad j \notin p, q}{\{\,\exists i\colon p[i]\,\}\ \mathsf{s}\ \{\,q\,\}} \quad \text{(Exists)}$$

$$\frac{\{\,p\,\}\ \mathsf{s}\ \{\,p'\,\} \quad \{\,p'\,\}\ \mathsf{s}'\ \{\,q\,\}}{\{\,p\,\}\ \mathsf{s};\mathsf{s}'\ \{\,q\,\}} \quad \text{(Seq)} \qquad \frac{\{\,p\,\}\ \mathsf{s}\ \{\,q[j]\,\} \quad j \notin p, q}{\{\,p\,\}\ \mathsf{s}\ \{\,\forall i\colon q[i]\,\}} \quad \text{(Forall)}$$

$$\frac{p' \Rightarrow p \quad \{\,p\,\}\ \mathsf{s}\ \{\,q\,\} \quad q \Rightarrow q'}{\{\,p'\,\}\ \mathsf{s}\ \{\,q'\,\}} \quad \text{(Cons)} \qquad \frac{\{\,c?p\,\}\ \mathsf{s}\ \{\,q\,\} \quad \{\,\neg c?p\,\}\ \mathsf{s}'\ \{\,q'\,\}}{\{\,p\,\}\ \mathsf{if}\ c\ \mathsf{then}\ \mathsf{s}\ \mathsf{else}\ \mathsf{s}'\ \mathsf{fi}\ \{\,q + q'\,\}} \quad \text{(If)}$$

$$\frac{\{\,p\,\}\ \mathsf{s}\ \{\,q\,\} \quad \{\,p\,\}\ \mathsf{s}'\ \{\,q'\,\}}{\{\,p\,\}\ \mathsf{s} \oplus_r \mathsf{s}'\ \{\,q \oplus_r q'\,\}} \quad \text{(Prob)} \qquad \frac{p \text{ invariant for } \langle c, \mathsf{s}\rangle}{\{\,p\,\}\ \mathsf{while}\ c\ \mathsf{do}\ \mathsf{s}\ \mathsf{od}\ \{\,p \wedge \text{not}\, c\,\}} \quad \text{(While)}$$

■

The rules (Skip), (Assign), (Seq), and (Cons) are similar to the rules of the derivation system H for non-probabilistic Hoare logic given in Definition 2.5. It should, however, be noted that we now consider probabilistic predicates instead of deterministic predicates.

The rules (Prob), (Or), (Exists) and (Forall) are new. Statements of the form $\mathsf{s} \oplus_r \mathsf{s}'$ can be handled with the help of the (Prob) rule. Here the postconditions q and q' for the statements s and s' are combined to the predicate $q \oplus_r q'$ which considers the probability of executing s or s'. The rules (Or), (Exists) and (Forall) can be used to handle the logical constructs that occur in a predicate.

The rules (If) and (While) are different to the rules for non-probabilistic Hoare logic. The intuition is that, given a probabilistic state, a condition $c \in \mathrm{BC}$ only evaluates to *true* (or *false*) with a certain probability. For an if statement and a state that satisfies a predicate p, we split the state into a part where c is *true* and a part where c is *false*. Note that these parts satisfy $c?p$ and $\neg c?p$, respectively. The (If) rule then states that these parts are the initial situation before executing s and s'. The postconditions q and q' can then be combined to $q + q'$.

For the rule (While) we need to define when a predicate p is *invariant* for $\langle c, \mathsf{s}\rangle$. For now let us assume that the considered program terminates, i.e., the program terminates for all possible outcomes of the probabilistic choices. In this case, the probabilistic predicate

p is invariant for $\langle c, \mathtt{s} \rangle$ if

$$\models \{\, p \,\} \ \mathsf{if}\ c\ \mathsf{then}\ \mathtt{s}\ \mathsf{else}\ \mathsf{skip}\ \mathsf{fi}\ \{\, p \,\}.$$

Providing that p is invariant and satisfied before executing a while loop, the rule (While) states that p is also satisfied after the loop and that the condition c is *true* with probability 0.

Example 5.3 (Verification of a Program without Loops)
Consider the program $\mathtt{s}_1$ from Example 3.2 as well as the Hoare triple

$$\{\, [\mathit{true}] \,\}\ \mathtt{s}_1\ \{\, \mathbb{P}(y = 0){=}0.7 \,\}$$

from Example 5.1. We want to use the derivation system pH in order to prove that the Hoare triple holds. As an auxiliary statement we start with showing that

$$\models \{\, [\mathit{true}] \,\}\ x \coloneqq 0 \oplus_{0.3} x \coloneqq 1\ \{\, \big(\mathbb{P}(x = 0){=}0.3\big) \wedge \big(\mathbb{P}(x \neq 0){=}0.7\big) \,\}.$$

The following proof tree depicts how this statement can be derivated with using pH.

$$\cfrac{\cfrac{\cfrac{}{\{\, [0 = 0] \,\}\ x \coloneqq 0\ \{\, [x = 0] \,\}}\,\text{(Assign)}}{\{\, [\mathit{true}] \,\}\ x \coloneqq 0\ \{\, [x = 0] \,\}}\,\text{(Cons)} \qquad \cfrac{\cfrac{}{\{\, [1 \neq 0] \,\}\ x \coloneqq 1\ \{\, [x \neq 0] \,\}}\,\text{(Assign)}}{\{\, [\mathit{true}] \,\}\ x \coloneqq 1\ \{\, [x \neq 0] \,\}}\,\text{(Cons)}}{\cfrac{\{\, [\mathit{true}] \,\}\ x \coloneqq 0 \oplus_{0.3} x \coloneqq 1\ \{\, [x = 0] \oplus_{0.3} [x \neq 0] \,\}}{\{\, [\mathit{true}] \,\}\ x \coloneqq 0 \oplus_{0.3} x \coloneqq 1\ \{\, \big(\mathbb{P}(x = 0){=}0.3\big) \wedge \big(\mathbb{P}(x \neq 0){=}0.7\big) \,\}}\,\text{(Cons)}}\,\text{(Prob)}$$

In the last application of the rule (Cons), we makes use of the fact that

$$[x = 0] \oplus_{0.3} [x \neq 0] \text{ implies } \big(\mathbb{P}(x = 0){=}0.3\big) \wedge \big(\mathbb{P}(x \neq 0){=}0.7\big)$$

which is a special case of the implication (2) which we already proved in Example 4.3. The remaining steps are simple applications of the rules from pH. For the rest of the proof we use an outline notation. Similar to the non-probabilistic case, we will use $\langle$angle brackets$\rangle$ to depict predicates. Whenever a predicate is directly followed by another predicate, the rule (Cons) has been applied.

```
$\langle [\mathit{true}] \rangle$
$(x \coloneqq 0 \oplus_{0.3} x \coloneqq 1)$;
$\langle \big(\mathbb{P}(x = 0){=}0.3\big) \wedge \big(\mathbb{P}(x \neq 0){=}0.7\big) \rangle$
$\langle \mathbb{P}(x \neq 0){=}0.7 \rangle$
if $(x = 0)$ then
  $\langle (x = 0)?\big(\mathbb{P}(x \neq 0){=}0.7\big) \rangle$
  $\langle \mathbb{P}(1 = 0){=}0 \rangle$
  $y \coloneqq 1$
  $\langle \mathbb{P}(y = 0){=}0 \rangle$
                ⋮
```

```
                ⋮
else
  $\langle (x \neq 0)?\big(\mathbb{P}(x \neq 0){=}0.7\big) \rangle$
  $\langle \mathbb{P}(0 = 0){=}0.7 \rangle$
  $y \coloneqq 0$
  $\langle \mathbb{P}(y = 0){=}0.7 \rangle$
fi
$\langle \big(\mathbb{P}(y = 0){=}0\big) + \big(\mathbb{P}(y = 0){=}0.7\big) \rangle$
$\langle \mathbb{P}(y = 0){=}0.7 \rangle$
```

The validity of the Hoare triple for the first statement has already been shown above. The implication after entering the if case relies on the fact that $\theta \models \mathbb{P}(\mathit{false}) = 0$ for all $\theta \in \Pi$. After entering the else case, we use the following statement. Whenever $\theta \models c?p$ it holds that $\theta(\sigma) = 0$ for all $\sigma \models \neg c$. Hence, the probability for *true* can not be larger then the probability for c. Finally, the probabilities for y to equal 0 for the if and the else cases are added. ■

Example 5.4 (Verification of a Program with a Terminating Loop)
Consider the program s_2 from Example 3.2 as well as the Hoare triple

$$\models \{\, [\mathit{true}] \,\}\ \mathsf{s}_2\ \{\, \mathbb{P}(x = N) {\geq} R^N \,\}$$

from Example 5.1. It is assumed that $N \in \mathbb{N}$ and $R \in [0,1]$ are constants and fixed before executing the program. We prove the validity of the Hoare triple using the derivation system pH:

```
$\langle [\mathit{true}] \rangle$
$\langle [0 = 0 \wedge 0 = 0] \rangle$
$x := 0;$
$\langle [x = 0 \wedge 0 = 0] \rangle$
$y := 0;$
$\langle [x = 0 \wedge y = 0] \rangle$
$\big\langle \big(\mathbb{P}(y = x = N){\geq}R^N\big) \vee \big(\exists i < N\colon \mathbb{P}(y = x = i){\geq}R^i\big) \big\rangle$
while $(y < N)$ do
  $y := y + 1;$
  $x := x + 1 \oplus_R$ skip
od
$\Big\langle \Big( \big(\mathbb{P}(y = x = N){\geq}R^N\big) \vee \big(\exists i < N\colon \mathbb{P}(y = x = i){\geq}R^i\big) \Big) \wedge \mathrm{not}\,(y < N) \Big\rangle$
$\big\langle \mathbb{P}(x = N){\geq}R^N \big\rangle$
```

Here it is assumed that the predicate p given by

$$\big(\mathbb{P}(y = x = N){\geq}R^N\big) \vee \big(\exists i < N\colon \mathbb{P}(y = x = i){\geq}R^i\big)$$

is invariant for $\langle c, \mathsf{s} \rangle$, where $c = (y < N)$ and s is the program inside the while-loop. The predicate $p \wedge \mathrm{not}\,(y < N)$ implies the requested postcondition $\mathbb{P}(x = N){\geq}R^N$ since

$$\text{for all } \theta \in \Pi\colon \quad \theta \not\models \exists i < N\colon \mathbb{P}(y = x = i){\geq}R^i \wedge \mathrm{not}\,(y < N)$$

and $\mathbb{P}(y = x = N){\geq}R^N$ implies $\mathbb{P}(x = N){\geq}R^N$. (Note that proving the postcondition $\mathbb{P}(x = N){=}R^N$ would require a stronger invariant).

To prove that p is invariant for $\langle c, \mathsf{s} \rangle$, we first observe that s_2 always terminates. Hence, we only need to show that $\{\, p \,\}$ if c then s else skip fi $\{\, p \,\}$ holds. Applying the rule (Or) yields that proving the validity of

$$\{\, \mathbb{P}(y = x = N) {\geq} R^N \,\} \text{ if } c \text{ then s else skip fi } \{\, p \,\} \text{ and} \tag{$\star$}$$

$$\{\, \exists i < N\colon \mathbb{P}(y = x = i) {\geq} R^i \,\} \text{ if } c \text{ then s else skip fi } \{\, p \,\} \tag{$\star\star$}$$

suffices. This can be done the following way:

proof for ($\star$):

```
⟨$\mathbb{P}(y = x = N){\geq}R^N$⟩
if $(y < N)$ then
    ⟨$(y < N)?(\mathbb{P}(y = x = N){\geq}R^N)$⟩
    ⟨$true$⟩
    $y := y + 1;$
    ⟨$true$⟩
    $x := x + 1 \oplus_R$ skip
    ⟨$true$⟩

else
    ⟨$(y \geq N)?(\mathbb{P}(y = x = N){\geq}R^N)$⟩
    ⟨$\mathbb{P}(y = x = N){\geq}R^N$⟩
    skip
    ⟨$\mathbb{P}(y = x = N){\geq}R^N$⟩
fi
⟨$(true) + (\mathbb{P}(y = x = N){\geq}R^N)$⟩
⟨$\mathbb{P}(y = x = N){\geq}R^N$⟩
⟨$p$⟩
```

proof for ($\star\star$):

```
⟨$\exists i < N\colon \mathbb{P}(y = x = i){\geq}R^i$⟩
if $(y < N)$ then
    ⟨$(y < N)?(\exists i < N\colon \mathbb{P}(y = x = i){\geq}R^i)$⟩
    ⟨$\exists i \leq N\colon \mathbb{P}(y + 1 = x + 1 = i){\geq}R^{i-1}$⟩
    $y := y + 1;$
    ⟨$\exists i \leq N\colon \mathbb{P}(y = x + 1 = i){\geq}R^{i-1}$⟩
    $x := x + 1 \oplus_R$ skip
    ⟨$(\exists i \leq N\colon \mathbb{P}(y = x = i){\geq}R^{i-1}) \oplus_R (true)$⟩
    ⟨$\exists i \leq N\colon \mathbb{P}(y = x = i){\geq}R^i$⟩
else
    ⟨$(y \geq N)?(\exists i < N\colon \mathbb{P}(y = x = i){\geq}R^i)$⟩
    ⟨$true$⟩
    skip
    ⟨$true$⟩
fi
⟨$(\exists i \leq N\colon \mathbb{P}(y = x = i){\geq}R^i) + (true)$⟩
⟨$\exists i \leq N\colon \mathbb{P}(y = x = i){\geq}R^i$⟩
⟨$p$⟩
```

In the proof for ($\star$) we use that a predicate of the form $c?q$ implies q, providing that q only considers deterministic states for which the condition c is *true*. For the implication after the if-statement, we use that a state $\theta \in \Pi$ that satisfies a predicate of the form $true + \mathbb{P}(dp){\geq}r$ also satisfies $\mathbb{P}(dp){\geq}r$, since θ can be split into $\theta_1 + \theta_2$ such that

$$\sum_{\sigma \models dp} \theta(\sigma) = \underbrace{\sum_{\sigma \models dp} \theta_1(\sigma)}_{\geq 0} + \underbrace{\sum_{\sigma \models dp} \theta_2(\sigma)}_{\geq r} \geq r.$$

The proof for ($\star\star$) uses similar ideas. The probabilistic choice ($x := x + 1 \oplus_R$ skip) is handled by using that the Hoare triples

$$\{\, \exists i \leq N\colon \mathbb{P}(y = x + 1 = i){\geq}R^{i-1} \,\}\ x := x + 1\ \{\, \exists i \leq N\colon \mathbb{P}(y = x = i){\geq}R^{i-1} \,\} \text{ and}$$

$$\{\, \exists i \leq N\colon \mathbb{P}(y = x + 1 = i){\geq}R^{i-1} \,\} \text{ skip } \{\, true \,\}$$

are valid. This can easily be seen by applying the rules (Assign), (Skip) and (Cons) using that every predicate implies *true*. ■

Let $c \in \mathrm{BC}$ and $\mathsf{s} \in \mathcal{L}_{\mathrm{pw}}$. To apply the rule (While) from pH on a program of the form while c do s od, we need to show that a predicate p is invariant for $\langle c, \mathsf{s}\rangle$. In order to do so, we have already seen that the Hoare triple $\{\, p\,\}$ if c then s else skip fi $\{\, p\,\}$ has to be valid. If the considered while-loop terminates for all possible outcomes of the probabilistic choices, this condition suffices. Otherwise, the loop only terminates with a certain probability. It is then additionally required to show that p is $\langle c, \mathsf{s}\rangle$-closed.

To clarify the idea of $\langle c, \mathsf{s}\rangle$-closeness, let us first consider sequences of probabilistic states that we will call *strong* $\langle c, \mathsf{s}\rangle$*-sequences within* p.

Definition 5.5 (Strong $\langle c, \mathsf{s}\rangle$-sequence)
A strong $\langle c, \mathsf{s}\rangle$-sequence within a probabilistic predicate p is a sequence of probabilistic states $(\theta_n)_{n\in\mathbb{N}}$ where

- every θ_n satisfies p and
- θ_n corresponds to the state that is reached after n iterations of the while-loop. ■

Thus, the successor θ_{n+1} of some state θ_n can be obtained by executing s on the c-part of θ_n (the $\neg c$-part remains untouched), i.e., $\theta_{n+1} = \mathcal{D}(\mathsf{s})(c?\theta_n) + \neg c?\theta_n$. We can directly derive that $\neg c?\theta_{n+1}(\sigma) \geq \neg c?\theta_n(\sigma)$ for all $n \in \mathbb{N}$ and $\sigma \in \mathcal{S}$. The sequence $(\neg c?\theta_n)_{n\in\mathbb{N}}$ can therefore be seen as an ascending chain. The predicate p is $\langle c, \mathsf{s}\rangle$-closed if the least upper bound of this chain satisfies p for all considered sequences.

Example 5.6 (Strong $\langle c, \mathsf{s}\rangle$-sequence)
Let us consider the following probabilistic program:

$$\mathsf{while}\ \underbrace{(x = 0)}_{=:c}\ \mathsf{do}\ \underbrace{x := 1 \oplus_{0.5} \mathsf{skip}}_{=:\mathsf{s}}\ \mathsf{od}$$

As well as the predicate $p = \big([x \neq 0]\big) \vee \exists i\colon \Big(\big(\mathbb{P}(x = 0)=0.5^i\big) \wedge \big(\mathbb{P}(x \neq 0)=1 - 0.5^i\big)\Big)$.

The sequence of states $(\theta_n)_{n\in\mathbb{N}}$ depicted below is a strong $\langle c, \mathsf{s}\rangle$-sequence within p.

$\theta_n(\sigma_i)$	0.5^n	$1 - 0.5^n$
$\sigma_i(x)$	0	1

$\neg c?\theta_n(\sigma_i)$	0	$1 - 0.5^n$
$\sigma_i(x)$	0	1

$\theta(\sigma_i)$	0	1
$\sigma_i(x)$	0	1

The state $\theta \in \Pi$ is the least upper bound of the ascending chain $(\neg c?\theta_n)_{n\in\mathbb{N}}$. As $\theta \models p$, the condition for $\langle c, \mathsf{s}\rangle$-closeness is satisfied for this particular sequence. ■

However, for $\langle c, \mathsf{s}\rangle$-closeness, we do not only consider strong $\langle c, \mathsf{s}\rangle$-sequences. Doing so would be challenging as finding exactly these sequences can be hard. Instead of that we weaken the constraints and consider $\langle c, \mathsf{s}\rangle$-sequences.

Definition 5.7 ($\langle c, \mathsf{s}\rangle$-sequence)
A $\langle c, \mathsf{s}\rangle$-sequence within a predicate p is a sequence of probabilistic states $(\theta_n)_{n\in\mathbb{N}}$ such that

- every θ_n satisfies p,
- $(\neg c?\theta_n)_{n\in\mathbb{N}}$ is an ascending chain, and
- the probability for $\neg c$ in the state θ_n is at least the n-step termination ratio, i.e.,

$$\sum_{\sigma\models\neg c} \theta_n(\sigma) \geq r^n_{\langle c,\mathsf{s}\rangle}.$$

■

The n-step termination ratio $r^n_{\langle c,\mathsf{s}\rangle}$ is the minimum probability that, starting from a state satisfying p, the loop terminates within n steps, formally

$$r^n_{\langle c,\mathsf{s}\rangle} = \inf\left\{ \textstyle\sum_{\sigma\models\neg c} \theta'_n(\sigma) \;\middle|\; (\theta'_n)_{n\in\mathbb{N}} \text{ is a strong } \langle c,\mathsf{s}\rangle\text{-sequence within } p \right\}.$$

The predicate p is $\langle c,\mathsf{s}\rangle$-closed if for all $\langle c,\mathsf{s}\rangle$-sequences $(\theta_n)_{n\in\mathbb{N}}$ within p the least upper bound of the chain $(\neg c?\theta_n)_{n\in\mathbb{N}}$ satisfies p.

Example 5.8 (Verification of a Program Requiring $\langle c,\mathsf{s}\rangle$-Closeness)
Consider again the program while c do s od as well as the predicate p from Example 5.6. Observe that $[x=0]$ implies p and $p \wedge \text{not}\, c$ implies $[x\neq 0]$. Hence, it is easy to prove that the Hoare triple $\{\,[x=0]\,\}$ while c do s od $\{\,[x\neq 0]\,\}$ holds, assuming that p is invariant for $\langle c,\mathsf{s}\rangle$. To prove $\models \{\,p\,\}$ if c then s else skip fi $\{\,p\,\}$ we apply the rules (Or) and (Exists) and derive:

```
⟨$[x\neq 0]$⟩
if $(x=0)$ then
    ⟨$(x=0)?[x\neq 0]$⟩
    ⟨$\mathbb{P}(\mathit{true})=0$⟩
    $x := 1 \oplus_{0.5}$ skip
    ⟨$\mathbb{P}(\mathit{true})=0$⟩
else
    ⟨$(x\neq 0)?[x\neq 0]$⟩
    ⟨$[x\neq 0]$⟩
    skip
    ⟨$[x\neq 0]$⟩
fi
⟨$\big(\mathbb{P}(\mathit{true})=0\big) + \big([x\neq 0]\big)$⟩

⟨$[x\neq 0]$⟩
⟨$p$⟩
```

```
⟨$\big(\mathbb{P}(x=0)=0.5^i\big) \wedge \big(\mathbb{P}(x\neq 0)=1-0.5^i\big)$⟩
if $(x=0)$ then
    ⟨$(x=0)?\Big(\big(\mathbb{P}(x=0)=0.5^i\big) \wedge \big(\mathbb{P}(x\neq 0)=1-0.5^i\big)\Big)$⟩
    ⟨$\big(\mathbb{P}(x=0)=0.5^i\big) \wedge \big(\mathbb{P}(x\neq 0)=0\big)$⟩
    $x := 1 \oplus_{0.5}$ skip
    ⟨$\big(\mathbb{P}(x\neq 0)=0.5^{i+1}\big) \wedge \big(\mathbb{P}(x=0)=0.5^{i+1}\big)$⟩
else
    ⟨$(x\neq 0)?\Big(\big(\mathbb{P}(x=0)=0.5^i\big) \wedge \big(\mathbb{P}(x\neq 0)=1-0.5^i\big)\Big)$⟩
    ⟨$\big(\mathbb{P}(x=0)=0\big) \wedge \big(\mathbb{P}(x\neq 0)=1-0.5^i\big)$⟩
    skip
    ⟨$\big(\mathbb{P}(x=0)=0\big) \wedge \big(\mathbb{P}(x\neq 0)=1-0.5^i\big)$⟩
fi
⟨$\Big(\big(\mathbb{P}(x\neq 0)=0.5^{i+1}\big) \wedge \big(\mathbb{P}(x=0)=0.5^{i+1}\big)\Big)$
      $+\Big(\big(\mathbb{P}(x=0)=0\big) \wedge \big(\mathbb{P}(x\neq 0)=1-0.5^i\big)\Big)$⟩
⟨$\big(\mathbb{P}(x=0)=0.5^{i+1}\big) \wedge \big(\mathbb{P}(x\neq 0)=1-0.5^{i+1}\big)$⟩
⟨$p$⟩
```

As the program does not terminate for all probabilistic choices, it remains to show that p is $\langle c, \mathsf{s}\rangle$-closed. We first inspect the n-step termination ratio $r^n_{\langle c,\mathsf{s}\rangle}$. To this end, let $(\theta'_n)_{n\in\mathbb{N}}$ be a strong $\langle c, \mathsf{s}\rangle$-sequence within p. The first case is that θ'_0 satisfies $[x \neq 0]$. The prove above can be used to see that then $\theta'_n \models [x \neq 0]$ and therefore $\sum_{\sigma\models\neg c} \theta'_n(\sigma) = 1$ for every $n \in \mathbb{N}$. The second case is that θ'_0 satisfies $\big(\mathbb{P}(x = 0){=}0.5^i\big) \wedge \big(\mathbb{P}(x \neq 0){=}1 - 0.5^i\big)$ for some $i \geq 0$. Again, the prove above can be used to see that $\theta'_n \models \big(\mathbb{P}(x = 0){=}0.5^{i+n}\big) \wedge \big(\mathbb{P}(x \neq 0){=}1 - 0.5^{i+n}\big)$ for every $n \in \mathbb{N}$. Hence, $\sum_{\sigma\models\neg c} \theta'_n(\sigma) = 1 - 0.5^{i+n} \geq 1 - 0.5^n$. Taking the minimum from both cases, we conclude that $r^n_{\langle c,\mathsf{s}\rangle} \geq 1 - 0.5^n$.

Let now $(\theta_n)_{n\in\mathbb{N}}$ be an arbitrary $\langle c, \mathsf{s}\rangle$-sequence within p. We have to show that the least upper bound of the ascending chain $(\neg c?\theta_n)_{n\in\mathbb{N}}$ satisfies p. For a state θ_n we know that $\sum_{\sigma\models\neg c} \theta_n(\sigma) \geq r^n_{\langle c,\mathsf{s}\rangle} \geq 1 - 0.5^n$. This implies that $\sum_{\sigma\models\neg c} \neg c?\theta_n(\sigma) \geq 1 - 0.5^n$. For the least upper bound $\theta \in \Pi$ of the chain $(\neg c?\theta_n)_{n\in\mathbb{N}}$ we obtain $\sum_{\sigma\models\neg c} \theta(\sigma) \geq 1$. With $\neg c = (x \neq 0)$, this yields $\theta \models [x \neq 0]$ and therefore $\theta \models p$ holds. ■

It should be noted that whenever the termination of a loop `while` c `do s od` is independent of the probabilistic choices, a predicate p is always $\langle c, \mathsf{s}\rangle$-closed. This justifies that we do not have to check for $\langle c, \mathsf{s}\rangle$-closeness in this case. To see this, let us consider an arbitrary $\langle c, \mathsf{s}\rangle$-sequence $(\theta_n)_{n\in\mathbb{N}}$. We can observe that there is always an $m \in \mathbb{N}$ such that the loop has terminated after at most m iterations (m might depend on the state θ_0). Therefore, the least upper bound of the chain $(\neg c?\theta_n)_{n\in\mathbb{N}}$ always satisfies p as it is equal to θ_m.

6 Conclusion

In this work, we have seen how a Hoare like logic can be used to verify probabilistic programs. For this purpose, a language for probabilistic programs, called $\mathcal{L}_{\mathrm{pw}}$, has been introduced. While executing such a program, the assignment of the occurring variables depends on probabilistic choices. Probabilistic states have been considered in order to reflect the probability that the variables adhere to a certain assignment. To make claims about these states, one could use deterministic predicates (similar to the non-probabilistic case) which evaluate to a probability ratio. This has been done in, e.g., [7, 8]. For our purposes, however, we desired a notion of predicates which, given a probabilistic state, evaluate to either *true* or *false*. Therefore, the idea of probabilistic predicates has been introduced.

The presented notions allow us to adapt the idea of Hoare triples as known for non-probabilistic programs. To verify claims regarding a probabilistic program s, a Hoare triple $\{\,p\,\}\ \mathsf{s}\ \{\,q\,\}$ can be formulated. Here, p and q are probabilistic predicates called precondition and postcondition. The aim is to prove that whenever the precondition p is satisfied by the current probabilistic state, the postcondition q holds after executing the program s. To this end, a derivation system called pH can be used.

Our examples have shown that the application of this procedure can be challenging. One reason for this is that implications of probabilistic predicates are rarely trivial. Another problem is the treatment of loops since a sufficiently strong invariant has to be found. However, the basic idea of the discussed notions is very similar to the well-known Hoare logic for non-probabilistic programs. Hence, this work presented a technique to verify probabilistic programs by extending already existing approaches. This increases the potential of other approaches and tools for standard Hoare logic to be adapted to work on probabilistic programs.

References

[1] K. R. Apt. Ten Years of Hoare's Logic: A Survey – Part I. *ACM Trans. Program. Lang. Syst.*, 3(4):431–483, Oct. 1981.

[2] C. Baier and J.-P. Katoen. *Principles of Model Checking.* The MIT Press, 2008.

[3] J. I. den Hartog. Verifying probabilistic programs using a hoare like logic. In P. Thiagarajan and R. Yap, editors, *Advances in Computing Science — ASIAN'99*, volume 1742 of *Lecture Notes in Computer Science*, pages 113–125. Springer Berlin Heidelberg, 1999.

[4] J. I. den Hartog. *Probabilistic Extensions of Semantical Models.* PhD thesis, Vrije Universiteit Amsterdam, 2002.

[5] A. D. Gordon, T. A. Henzinger, A. V. Nori, and S. K. Rajamani. Probabilistic programming. In *International Conference on Software Engineering (ICSE Future of Software Engineering).* IEEE, May 2014.

[6] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning.* The MIT Press, 2009.

[7] D. Kozen. A probabilistic {PDL}. *Journal of Computer and System Sciences*, 30(2):162 – 178, 1985.

[8] C. Morgan, A. McIver, and K. Seidel. Probabilistic predicate transformers. *ACM Transactions on Programming Languages and Systems*, 18:325–353, 1996.