Verifying Probabilistic Programs Using a Hoare like Logic

J.I. den Hartog

Faculty of Exact Sciences, Free Universiteit,
de Boelelaan 1081, 1081 HV Amsterdam, the Netherlands
jerry@cs.vu.nl

Abstract. Hoare logic can be used to verify properties of deterministic programs by deriving correctness formulae, also called Hoare triples. The goal of this paper is to extend the Hoare logic to be able to deal with probabilistic programs. To this end a generic non-uniform language $\mathcal{L}_{\mathrm{pw}}$ with a probabilistic choice operator is introduced and a denotational semantics $\mathcal{D}$ is given for the language. A notion of probabilistic predicate is defined to express claims about the state of a probabilistic program. To reason about the probabilistic predicates a derivation system pH, similar to that of standard Hoare logic, is given. The derivation system is shown to be correct with respect to the semantics $\mathcal{D}$. Some basic examples illustrate the use of the system.

1 Introduction

Probability is introduced into the description of computer systems to model the inherent probabilistic behaviour of processes like, for example, a faulty communication medium. Probability is also explicitly introduced to obtain randomized algorithms to solve problems which can not be solved efficiently, or can not be solved at all, by deterministic algorithms. With increasing complexity of computer programs and systems, formal verification has become an important tool in the design. The presence of probabilistic elements in a program usually makes understanding and testing of the program more difficult. A way of formally verifying programs becomes even more important.

To formally verify a probabilistic program, the semantics of the program is given. The mathematical model of the program that is obtained in this way can be used to directly check properties of the program. In the probabilistic analyses of the model, results from probability theory are used to obtain e.g. average performance or bounds on error probabilities [18, 15]. Models that are often used are Markov chains and Markov decision processes [11, 4] probabilistic input-output automata [19, 20] and probabilistic transition systems [9, 8], sometimes augmented with probabilistic bisimulation [14, 10, 2].

For some programs the construction of the mathematical model can already be difficult. A systematic approach to simplify the program, or obtain properties without having to actually calculate the semantics are useful. Approaches in this area are probabilistic process algebra [2] and stochastic process algebra [6] where

P.S. Thiagarajan, R. Yap (Eds.): ASIAN'99, LNCS 1742, pp. 113–125, 1999.

equivalences of programs can be checked syntactically by equational reasoning. Another approach is to introduce a logic to reason about the probabilistic programs. Here the later approach is followed by extending Hoare logic as known for deterministic programs to probabilistic programs. Other work on probabilistic logic can be found in e.g. [13, 4, 16, 17, 6]. In [13] an algebraic version of propositional dynamic logic is introduced. In [4] probabilities for temporal logic formulae are calculated. In [16] a weakest precondition calculus based on 'expectations' is defined and in [17] a notion of probabilistic predicate transformer, also used to find weakest preconditions, is given. Model checking is used in [1] to check formulae in a probabilistic temporal logic.

Deterministic Hoare logic is a system to derive correctness formulae, also called Hoare triples. A formula $\{\, p \,\}\ s\ \{\, q \,\}$ states that the predicate p is a sufficient precondition for the program s to guarantee that predicate q is true after termination. An extensive treatment of Hoare logic can be found in [5]. What the values of the variables in a program, i.e. the (deterministic) state of the program, will be, can not be fully determined if the program is probabilistic. Only the probability of being in a certain state can be given. This gives the notion of a probabilistic state. In a probabilistic state, a deterministic predicate will no longer be true or false, it is true with a certain probability. This can be dealt with by changing the interpretation of validity of a predicate to a function to $[0, 1]$ instead of to $\{\, \mathit{true}, \mathit{false} \,\}$ as in [13, 16]. The approach chosen here instead is to extend the syntax of predicates to allow making claims about the probability that a certain deterministic predicate holds. The extended form of predicates are called probabilistic predicates. A logic for probabilistic programs should reason with these probabilistic predicates.

In section 2 some mathematical definitions are given. The syntax of the non-uniform language $\mathcal{L}_{\mathrm{pw}}$ is given in section 3 together with its semantics. In section 4 probabilistic predicates are defined and a Hoare like logic is introduced to reason about probabilistic predicates. The logic is shown to be correct with respect to the denotational semantics. Some examples of the use of the logic are given in section 5 and some concluding remarks are given in section 6.

2 Mathematical Preliminaries

A complete partially ordered set (cpo) is a set with partial order $\leq$ that has a least element and for which each ascending chain has a least upper bound within the set. An order on Y is extended point wise to functions from X to Y ($f, g : X \to Y$ then $f \leq g$ if $f(x) \leq g(x)$ for all $x \in X$).

The support of a function $f : X \to [0, 1]$ is defined as the $x \in X$ for which $f(x) \neq 0$. The set of all functions from X to $[0, 1]$ with countable support is denoted by $X \to_{cs} [0, 1]$. Given a function $f : X \to_{cs} [0, 1]$ and a set $Y \subseteq X$ the sum $\sum f[Y] = \sum_{y \in Y} f(y)$ is well-defined (allowing the value ∞). The set of (pseudo) probabilistic measures $\mathcal{M}(X)$ on a set X is defined as the subset of functions in $X \to_{cs} [0, 1]$ with sum at most 1.

$$\mathcal{M}(X) = \{\, f \in X \to_{cs} [0, 1] \mid \textstyle\sum f[X] \leq 1 \,\}\,.$$

For a measure $f \in \mathcal{M}(X)$, $f(x)$ (for $x \in X$) is interpreted as the probability that x occurs. The set $\mathcal{M}(X)$ is a cpo, with minimal element $\underline{0}$, the function that assigns 0 to each element of X. For each ascending sequence in $\mathcal{M}(X)$ the limit exists within $\mathcal{M}(X)$ and corresponds to the least upper bound of the sequence.

For element $x \in X$ and $y \in Y$ and a function $f : X \to Y$, $f[x/y]$, called a variant of f, is defined by

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

3 Syntax and Semantics of $\mathcal{L}_{\mathbf{pw}}$

The language $\mathcal{L}_{\mathrm{pw}}$ is a basic programming language with an extra operator used to denote probabilistic choice. A typical variable is denoted by v, the set of all variables by *Var*. The types of the variables are not made explicit. Instead a set of values *Val* is fixed as the range for all variables. (The examples deviate from this assumption and use integer and boolean variables.) Types can easily be added at the cost of complicating notation with less important details.

Definition 1. *The statements in $\mathcal{L}_{pw}$, ranged over by s, are given by:*

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

where $c \in BC$ is a boolean condition, $e \in Exp$ is an expression over values in Var and variables in Var and r is a ratio in the open interval $(0, 1)$.

The statements are interpreted as follows. The statement $s \oplus_r s'$ makes a probabilistic choice. With probability r the statement s will be executed, and s' will be executed with probability $1 - r$. The other constructs of $\mathcal{L}_{\mathrm{pw}}$ are well known. The *skip* statement does nothing. Assignment $v := e$ assigns the value of the expression e to the variable v. Sequential composition $s; s'$ is executed by first executing s, then executing s'. The *if c then s else s' fi* statement executes s if the condition c holds, and otherwise s'. Finally, *while c do s od* repeatedly executes s until condition c no longer holds.

The internal details of the boolean conditions (*BC*) and expressions (*Exp*) are abstracted away from. Instead of defining an explicit syntax for the boolean conditions and the expressions, it is assumed that given the value of the variables, they can be evaluated. This is made more precise below.

For a deterministic program, the state of the computation is given by the value of the variables. The state space $\mathcal{S}$ for a deterministic program consists of $\mathcal{S} = \mathit{Var} \to \mathit{Val}$. For a probabilistic program, the values of the variables are no longer determined. For example, after executing $x := 0 \oplus_{\frac{1}{2}} x := 1$, the value of x could be zero but it could also be one. Instead of giving the value of a variable, a distribution over possible variables should be given. A first idea may be to take as a state space $\mathit{Var} \to \mathcal{M}(\mathit{Val})$. This does give, for each variable v, the chance that v takes a certain value but it does not describe the possible dependencies between the variables.

Consider the following example. In the left situation, a fair coin is thrown and a second coin is put beside it with the same side up. In the right situation, two fair coins are thrown. The two situations are indistinguishable if the dependency between the two coins is not known; the probability of heads or tails is $\frac{1}{2}$ for both coins in both situations. The difference between the situations is important e.g. if the next step is comparing the coins. In the first situation the coins are always equal, In the second situation they are equal with probability $\frac{1}{2}$ only.

		coin 1	
		heads	tails
coin 2	heads	$\frac{1}{2}$	0
	tails	0	$\frac{1}{2}$

		coin 1	
		heads	tails
coin 2	heads	$\frac{1}{4}$	$\frac{1}{4}$
	tails	$\frac{1}{4}$	$\frac{1}{4}$

The more general state space $\Pi = \mathcal{M}(Var \to Val)$ is required. In $\theta \in \Pi$, instead of giving the distributions for the variables separately, the probability of being in a certain deterministic state is given. The chance that a variable v takes value w can be found by summing the probabilities of all states which assign w to v.

Definition 2.

(a) The set of deterministic states $\mathcal{S}$, ranged over by σ, is given by $\mathcal{S} = Var \to Val$.
(b) The evaluation functions $\mathcal{V}: Exp \to \mathcal{S} \to Val$ and $\mathcal{B}: BC \to \mathcal{S} \to \{true, false\}$ are the functions that compute the value of expressions and boolean conditions.
(c) The set of (pseudo) probabilistic states Π, ranged over by θ, is given by $\Pi = \mathcal{M}(\mathcal{S})$.
(d) On Π the following operations are defined:

$$\theta_1 \oplus_r \theta_2 = r \cdot \theta_1 + (1 - r) \cdot \theta_2,$$
$$c?\theta(\sigma) = \begin{cases} \theta(\sigma) & \text{if } c \text{ true in } \sigma \text{ i.e. } \mathcal{B}(c)(\sigma) = true, \\ 0 & \text{otherwise,} \end{cases}$$
$$\theta[v/\mathcal{V}(e)](\sigma) = \sum \theta[\{\, \sigma' \mid \sigma'[v/\mathcal{V}(e)(\sigma')] = \sigma \,\}] \,.$$

where $+$ is standard addition of functions and $r\cdot$ is scalar multiplication.

Note that $\theta \in \Pi$ is a function from $\mathcal{S}$ to $[0, 1]$. The value $\theta(\sigma)$ returned by θ is the probability of being in the deterministic state σ.

The functions $\mathcal{V}$ and $\mathcal{B}$ are assumed given. The syntactic details of expressions and conditions as well as the precise definitions of these functions are abstracted away from. In a probabilistic state the values of the variables are, in general, not known and the value of expressions and conditions can not be found. Evaluation of expressions and conditions can only be done in a deterministic state. To find the probability of being in a state σ if in θ the expression e is assigned to variable v, the probabilities of all states σ' that yield σ after changing the value for v to that of e (evaluated in σ') have to be added.

The denotational semantics $\mathcal{D}$ for $\mathcal{L}_{\text{pw}}$ gives, for each statement s, and state θ, the state $\mathcal{D}(s)(\theta)$ resulting from executing s starting in state θ.

Definition 3.

(a) The higher-order operator $\Psi_{\langle c,s\rangle} : (\Pi \to \Pi) \to (\Pi \to \Pi)$ *is given by*

$$\Psi_{\langle c,s\rangle}(\psi)(\theta) = \psi(\mathcal{D}(s)(c?\theta)) + \neg c?\theta.$$

(b) The denotational semantics $\mathcal{D} : \mathcal{L}_{pw} \to (\Pi \to \Pi)$ *is given by*

$$\begin{aligned}
\mathcal{D}(skip)(\theta) &= \theta \\
\mathcal{D}(v := e)(\theta) &= \theta[v/\mathcal{V}(e)] \\
\mathcal{D}(s; s')(\theta) &= \mathcal{D}(s')(\mathcal{D}(s)(\theta)) \\
\mathcal{D}(s \oplus_r s')(\theta) &= \mathcal{D}(s)(\theta) \oplus_r \mathcal{D}(s')(\theta) \\
\mathcal{D}(\textit{if } c \textit{ then } s \textit{ else } s' \textit{ fi})(\theta) &= \mathcal{D}(s)(c?\theta) + \mathcal{D}(s')(\neg c?\theta) \\
\mathcal{D}(\textit{while } c \textit{ do } s \textit{ od}) \quad &= \textit{the least fixed point of } \Psi_{\langle c,s\rangle} .
\end{aligned}$$

For a while statement *while c do s od*, one would like to use the familiar unfolding to *if c then s*; *while c do s od else skip fi*. This can not be done directly, as the second statement is more complex than the first. Instead we can use the fact that $\mathcal{D}(\textit{while } c \textit{ do } s \textit{ od})$ is a fixed point of the higher-order operator $\Psi_{\langle c,s\rangle}$ to show that

$$\begin{aligned}
\mathcal{D}(\textit{while } c \textit{ do } s \textit{ od})(\theta) &= \Psi_{\langle c,s\rangle}(\mathcal{D}(\textit{while } c \textit{ do } s \textit{ od}))(\theta) \\
&= \mathcal{D}(\textit{while } c \textit{ do } s \textit{ od})(\mathcal{D}(s)(c?\theta)) + \neg c?\theta \\
&= \mathcal{D}(\textit{if } c \textit{ then } s; \textit{while } c \textit{ do } s \textit{ od else skip fi})(\theta) .
\end{aligned}$$

Note that the total probability of $\mathcal{D}(\textit{while } c \textit{ do } s \textit{ od})(\theta)$ may be less than that of θ. The 'missing' probability is the probability of non-termination.

The least fixed point of $\Psi_{\langle c,s\rangle}$ can be contructed explicitly.

Definition 4. *For a statement s define* $s^0 = s$ *and* $s^{n+1} = s; s^n$. *The functions* $if^n_{\langle c,s\rangle}$ *and* $L_{\langle c,s\rangle}$ *from probabilistic states to probabilistic states are given by*

$$\begin{aligned}
if^n_{\langle c,s\rangle}(\theta) &= \mathcal{D}((\textit{if } c \textit{ then } s \textit{ else skip fi})^n)(\theta) \\
L_{\langle c,s\rangle}(\theta) &= \lim_{n\to\infty} \neg c?if^n_{\langle c,s\rangle}(\theta).
\end{aligned}$$

Lemma 1. *The least fixed point of* $\Psi_{\langle c,s\rangle}$ *is given by* $L_{\langle c,s\rangle}$.

The function $\text{if}^n_{\langle c,s\rangle}$ is merely a shorthand notation. The function $L_{\langle c,s\rangle}$ characterizes the least fixed point of $\Psi_{\langle c,s\rangle}$ and is thus equal to $\mathcal{D}(\textit{while } c \textit{ do } s \textit{ od})$.

4 Probabilistic Predicates and Hoare Logic

The deterministic predicates used with deterministic Hoare logic are first order predicate formulae. Here *dp* is used to denote a deterministic predicate. The usual notions of fulfillment, $\sigma \models dp$ i.e. *dp* holds in σ, and substitution, $dp[v/e]$,

for deterministic predicates are assumed to be known. An important property of substitution is that $\sigma \models dp[v/e]$ exactly when $\sigma[v/\mathcal{V}(e)(\sigma)] \models dp$. (Replacing the variable by an expression in the predicate is the opposite of assigning the value of the expression to the variable in the state.)

A deterministic Hoare triple, or correctness formula, $\{\, dp \,\}\ s\ \{\, dp' \,\}$, describes that dp is a pre condition and dp' is a post condition of program s. The Hoare triple is said to be correct if execution s in any state that satisfies dp will lead to a state satisfying dp'. To extend the Hoare triples to probabilistic programs, a notion of probabilistic predicate has to be introduced. One option is to use the same predicates as for deterministic programs but to change the interpretation of a predicate. A deterministic predicate can be seen as a function from states to $\{\, 0, 1 \,\}$, returning 1 if the state satisfies the predicate and 0 otherwise. The predicates can be made probabilistic by making them into functions to $[0, 1]$, returning the probability that the predicate is satisfied in a probabilistic state (See e.g. [13, 17]). This approach, however, does not allow making claims about the probability within the predicate itself, only the value of the predicate gives information about the probabilities. A property like "dp holds with probability ce" can not be expressed as a predicate. Also the normal logical operators like $\wedge$ have to be extended to work on $[0, 1]$.

In this paper probabilistic predicates can only have a truth value i.e. true or false. Probabilistic predicates are predicates in the usual sense, but with an extended syntax to express claims about probabilities. The construct $\mathbb{P}(dp) \prec ce$, for $\prec \in \{\, <, \leq, =, \geq, > \,\}$, is the basis for probabilistic predicates. Here dp is any deterministic predicate and ce is an expression, not using program variables, evaluating to a number in $[0, 1]$. The predicate $\mathbb{P}(dp) = ce$ holds in a state θ if the chance in θ of being in a deterministic state that satisfies dp is equal to ce. Similar for the other choices for $\prec$. Probabilistic predicates can be combined by the logical operators from predicate logic. For example, assuming that $Val = \{\, 1, 2, \ldots \,\}$, $\forall i : \mathbb{P}(v = i) = \frac{1}{2}^i$ is a valid predicate stating that v has a geometric distribution. The expression $\frac{1}{2}^i$ uses the logical variable i, but does not depend on program variables like v. Furthermore for probabilitic predicates p and p', $p + p'$, $r \cdot p$ and $c?p$ are also probabilistic predicates. Their interpretation is given below.

Definition 5.

(a) A probabilistic predicate is a basic probabilistic predicate of the form $\mathbb{P}(dp) \prec ce$ *(*$\prec \in \{\, <, \leq, =, \geq, > \,\}$*) or a composition of probabilistic predicates with one of the logic operators* $\neg$, $\vee$, $\wedge$, $\Rightarrow$, $\exists$, $\forall$, *or one of the operators* $+$, $r\cdot$, $c?$. *Probabilistic predicates are ranged over by* p *and* q. *The following shorthand notations are also used*

$$\begin{aligned} p \oplus_r p' &= r \cdot p + (1 - r) \cdot p', \\ [dp] &= \mathbb{P}(dp) = 1, \\ not\ c &= \mathbb{P}(c) = 0\,. \end{aligned}$$

(b) The probabilistic predicates are interpreted as follows.
$\theta \models \mathbb{P}(dp) \prec ce$ *when* $\sum_{\sigma \models dp} \theta(\sigma) \prec ce$,
$\theta \models p_1 + p_2$ *when there exists* θ_1, θ_2 *with* $\theta = \theta_1 + \theta_2$, $\theta_1 \models p_1$ *and* $\theta_2 \models p_2$,
$\theta \models r \cdot p$ *when there exists* θ' *such that* $\theta = r \cdot \theta'$ *and* $\theta' \models p$,
$\theta \models c?p$ *when there exists* θ' *for which* $\theta = c?\theta'$ *and* $\theta' \models p$.
For the logical connectives the interpretation is as usual.

(c) Substitution on probabilistic predicates $[v/e]$ *is passed down through all constructs until a deterministic predicate is reached.*

$$\begin{aligned}
(\mathbb{P}(dp) \prec ce)[v/e] &= \mathbb{P}(dp[v/e]) \prec ce,\\
(p \; op \; p')[v/e] &= p[v/e] \; op \; p'[v/e] \quad op \in \{ \wedge, \vee, \Rightarrow, +, \oplus_r \},\\
(op \; p)[v/e] &= op \; (p[v/e]) \quad op \in \{ \neg, \exists, \forall, r \cdot \}\\
(c?p)[v/e] &= c[v/e]?(p[v/e]) \; .
\end{aligned}$$

Note that the extension of deterministic predicates to [0,1]-valued functions is more or less incorporated within the probabilistic predicates as used in this paper. To check the probability of a certain deterministic predicate dp in state θ, look for which r the predicate $\mathbb{P}(dp) = r$ is true in θ instead of checking the value of dp in θ is r.

When reasoning about probabilistic predicates, caution is advised. Some equivalences which may seem true at first sight do not hold. The most important of these is that in general $p \oplus_r p \nleftrightarrow p$. Take for example $\mathbb{P}(x = 1) = 1 \vee \mathbb{P}(x = 2) = 1$ for p and a state satisfying $\mathbb{P}(x = 1) = \frac{1}{2} + \mathbb{P}(x = 2) = \frac{1}{2}$ will satisfy $p \oplus_{\frac{1}{2}} p$ but not p. Other examples are $p = \exists i : q[i]$ and $p = \forall i : (q[i] \vee q'[i])$. The equivalence does hold for the basic predicates $\mathbb{P}(dp) \prec r$ and if the equivalence holds when $p = q$ and when $p = q'$ then it also holds for $p = q \wedge q'$.

Using probabilistic predicates the Hoare-triples as introduced for deterministic programs can be extended to probabilistic programs. Hoare triple $\{ p \} \; s \; \{ q \}$ indicates that p is a pre condition and q is a post condition for the probabilistic program s. The Hoare triple is said to hold, denoted by $\models \{ p \} \; s \; \{ q \}$, if the pre condition p guarantees that post condition q holds after execution of s.

$$\models \{ p \} \; s \; \{ q \} \text{ if } \forall \theta \in \Pi : \theta \models p \Rightarrow \mathcal{D}(s)(\theta) \models q.$$

For example $\models \{ p \} \; skip \; \{ p \}$ and $\models \{\mathbb{P}(x = 0) = 1\} \; x := x + 1 \; \{\mathbb{P}(x = 1) = 1\}$.

To prove the validity of Hoare triples, a derivation system called pH is introduced. The derivation system consists of the axioms and rules as given below.

$$\{ p \} \; skip \; \{ p \} \qquad \text{(Skip)} \qquad \frac{\{ p \} \; s \; \{ q \} \quad \{ p \} \; s' \; \{ q' \}}{\{ p \} \; s \oplus_c s' \; \{ q \oplus_c q' \}} \qquad \text{(Prob)}$$

$$\{ p[v/e] \} \; v := e \; \{ p \} \qquad \text{(Assign)} \qquad \frac{\{ c?p \} \; s \; \{ q \} \quad \{ \neg c?p \} \; s' \; \{ q' \}}{\{ p \} \; \textit{if } c \textit{ then } s \textit{ else } s' \textit{ fi} \; \{ q + q' \}} \qquad \text{(If)}$$

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

$$\frac{\{\, p[j] \,\}\ s\ \{\, q \,\} \quad j \notin p, q}{\{\, \exists i : p[i] \,\}\ s\ \{\, q \,\}} \text{ (Exists)} \qquad \frac{p' \Rightarrow p \quad \{\, p \,\}\ s\ \{\, q \,\} \quad q \Rightarrow q'}{\{\, p' \,\}\ s\ \{\, q' \,\}} \text{ (Imp)}$$

$$\frac{\{\, p \,\}\ s\ \{\, q[j] \,\} \quad j \notin p, q}{\{\, p \,\}\ s\ \{\, \forall i : q[i] \,\}} \text{ (Forall)} \qquad \frac{\{\, p \,\}\ s\ \{\, q \,\} \quad \{\, p' \,\}\ s\ \{\, q \,\}}{\{\, p \vee p' \,\}\ s\ \{\, q \,\}} \text{ (Or)}$$

The rules (Skip), (Assign), (Seq) and (Cons) are as within standard Hoare logic but now dealing with probabilistic predicates. The rules (If) and (While) have changed and the rules (Prob), (Or), (Exists) and (Forall) are new.

For p to hold after the execution of *skip*, it should hold before the execution since *skip* does nothing. The predicate p holds after an assignment $v := e$ exactly when p with e substituted for v holds before the assignment, as the effect of the assignment is exactly replacing v with the value of e. The rule (Seq) states that p is a sufficient pre condition for q to hold after execution of $s; s'$ if there exists an intermediate predicate p' which holds after the execution of s and which implies that q holds after the execution of s'. The rule (Cons) states that the pre condition may be strengthened and the post condition may be weakened.

The rule (Prob) states that the result of executing $s \oplus_r s'$ is obtained by combining the results obtained by executing s and s' with the appropriate probabilities. The necessity for the (Or), (Exists) and (Forall) rules becomes clear when one recalls that $p \oplus_r p \not\rightarrow p$. Proving correctness of $\{\, p \vee q \,\}\ \mathit{skip} \oplus_r \mathit{skip}\ \{\, p \vee q \,\}$ is, in general, not possible without the (Or)-rule. Similar examples show the need for the (Exists) and (Forall) rule. Note the similarity with the natural deduction rules for $\vee$ and $\exists$ elimination and $\forall$ introduction.

The rule (If) has changed with respect to the (If) rule of standard Hoare logic. In a probabilistic state the value of the boolean condition c is not determined. Therefore the probabilistic state is split into two parts, a part in which c is true and a part in which c is false. After splitting the state, the effect of the corresponding statement, either s or s', can be found after which the parts are recombined using the + operator.

To use the (While) rule, an invariant p should be found. For p to be an invariant, it should satisfy $\{\, p \,\}\ \mathit{if}\ c\ \mathit{then}\ s\ \mathit{else}\ \mathit{skip}\ \mathit{fi}\ \{\, p \,\}$. This condition is sufficient to obtain partial correctness. If the program s terminates and $\{\, p \,\}\ s\ \{\, q \,\}$ can be derived from *pH*, then $\models \{\, p \,\}\ s\ \{\, q \,\}$. A probabilistic program is said to terminate, if the program is sure to terminate when all probabilistic choices are interpreted as non-deterministic choices, i.e if the program terminates for all possible outcomes of the probabilistic choices. Partial correctness, however, is not sufficient for probabilistic programs. Many probabilistic programs do not satisfy the termination condition, they may for instance only terminate with a certain probability. (Note that, even if that probability is one, the termination condition need not be satisfied.) To derive valid Hoare triples for programs that need not terminate, a form of total correctness is required. This requires somehow adding termination conditions to the rules. To obtain total correctness we strengthen the notion of invariant by imposing the extra condition of $\langle c, s \rangle$-closedness.

Definition 6.

(a) For a predicate p the n-step termination ratio, denoted by $r^n_{\langle c,s\rangle}$, is the probability that, starting from a state satisfying p, the while loop "while c do s od" terminates within n steps.

$$r\theta^n_{\langle c,s\rangle} = \sum \neg c?if^n_{\langle c,s\rangle}(\theta)[\mathcal{S}]$$
$$r^n_{\langle c,s\rangle} = \inf\{\, r\theta^n_{\langle c,s\rangle} \mid \theta \models p \,\}\,.$$

(b) A sequence of states $(\theta_n)_{n\in\mathbb{N}}$ is called a $\langle c,s\rangle$-sequence if $(\neg c?\theta_n)_{n\in N}$ is an ascending sequence with $\sum \neg c?\theta_n[\mathcal{S}] \geq r^n_{\langle c,s\rangle}$.

(c) A predicate p is called $\langle c,s\rangle$-closed if each $\langle c,s\rangle$-sequence within p has a limit (least upper bound) within p.

p invariant for $\langle c,s\rangle$ when
$\{\, p \,\}$ if c then s else skip fi $\{\, p \,\}$ and p is $\langle c,s\rangle$-closed.

Note that for a loop *while c do s od* that terminates every p automatically satisfies $\langle c,s\rangle$-closedness. Therefore, for a terminating program, there is no need to check any $\langle c,s\rangle$-closedness conditions.

A Hoare triple $\{\, p \,\}\ s\ \{\, q \,\}$ is said to be derivable from the system pH, denoted by $\vdash \{\, p \,\}\ s\ \{\, q \,\}$, if there exists a proof tree for $\{\, p \,\}\ s\ \{\, q \,\}$ in pH. The derivation system is correct, i.e. only valid Hoare triples can be derived from pH.

Lemma 2. *The derivation system pH is correct, i.e. for all predicates p and q and statements s, $\vdash \{\, p \,\}\ s\ \{\, q \,\}$ implies $\models \{\, p \,\}\ s\ \{\, q \,\}$.*

Proof. It is sufficient to show that if $\theta \models p$ and $\vdash \{\, p \,\}\ s\ \{\, q \,\}$ then $\mathcal{D}(s)(\theta) \in q$. This is shown by induction on the depth of the derivation tree for $\{\, p \,\}\ s\ \{\, q \,\}$, by looking at the last rule used. A few cases are given below.

- *If the rule (Exists) was used and $\theta \models \exists i : p[i]$ then there is an i_0 for which $\theta \models p[i_0]$. By induction $\models \{\, p[j] \,\}\ s\ \{\, q \,\}$ which gives, by substituting the value i_0 for the free variable j $\{\, p[i_0] \,\}\ s\ \{\, q \,\}$. But then $\mathcal{D}(s)(\theta) \models q$.*
- *Known from the non-probabilistic case is that $\sigma[v/\mathcal{V}(e)(\sigma)] \models \mathrm{dp}$ exactly when $\sigma \models \mathrm{dp}[v/e]$. By induction on the structure of the probabilistic predicate p this extends to $\theta[v/\mathcal{V}(e)] \models p$ exactly when $\theta \models p[v/e]$. Correctness of the (Assign) rule follows directly.*
- *If rule (Prob) is used to derive $\vdash \{\, p \,\}\ s \oplus_r s'\ \{\, q \oplus q' \,\}$ from $\vdash \{\, p \,\}\ s\ \{\, q \,\}$ and $\vdash \{\, p \,\}\ s'\ \{\, q' \,\}$ then by induction $\models \{\, p \,\}\ s\ \{\, q \,\}$ and $\models \{\, p \,\}\ s'\ \{\, q' \,\}$. This means that if $\theta \models p$ then $\mathcal{D}(s)(\theta) \models q$ and $\mathcal{D}(s')(\theta) \models q'$. But then $\mathcal{D}(s \oplus s')(\theta) = \mathcal{D}(s)(\theta) \oplus_r \mathcal{D}(s')(\theta) \models q \oplus_r q'$. The case for rule (If) is similar.*
- *Assume rule (While) is used with statement s, condition c and invariant p. Clearly $\mathcal{D}(\text{while } c \text{ do } s \text{ od})(\theta) \models \text{not } c$ and $\models \{\, p \,\}$ if c then s else skip fi $\{\, p \,\}$ can be used repeatedly to gives that if $\theta \models p$ then $(if^n_{\langle c,s\rangle}(\theta))_{n\in\mathbb{N}}$ is a $\langle c,s\rangle$ sequence. By $\langle c,s\rangle$-closedness $\mathcal{D}(\text{while } c \text{ do } s \text{ od})(\theta) = L_{\langle c,s\rangle} \models p$.*

5 Examples

The picture below gives an example of a proof tree in the system *pH*. For larger programs, instead of giving the proof tree, a proof outline is used. In a proof outline the rules (Imp) and (Seq) are implicitly used by writing predicates between the statements and some basic steps are skipped. A predicates between the statements give conditions that the intermediate states in the computation must satisfy.

$$\dfrac{\dfrac{\dfrac{}{\{\,[x+1=2]\,\}\ x := x+1\ \{\,[x=2]\,\}}(\text{Assign})}{\{\,[x=1]\,\}\ x := x+1\ \{\,[x=2]\,\}}(\text{Imp}) \quad \dfrac{\dfrac{}{\{\,[x+2=3]\,\}\ x := x+2\ \{\,[x=3]\,\}}(\text{Assign})}{\{\,[x=1]\,\}\ x := x+2\ \{\,[x=3]\,\}}(\text{Imp})}{\{\,[x=1]\,\}\ x := x+1 \oplus_{\frac{1}{2}} x := x+2\ \{\,[x=2] \oplus_{\frac{1}{2}} [x:=3]\,\}}(\text{Prob})}{\{\,[x=1]\,\}\ x := x+1 \oplus_{\frac{1}{2}} x := x+2\ \{\,\mathbb{P}(x=2)=\tfrac{1}{2} \wedge \mathbb{P}(x=3)=\tfrac{1}{2}\,\}}(\text{Imp})$$

The following program adds an array of numbers, but some elements may inadvertently get skipped. A lower bound on the probability that the answer will still be correct is derived. An n-ary version of $\vee$ is used as a shorthand.

int $ss[1 \ldots N],\ r,\ k$;
$\{\,[true]\,\} \Rightarrow \{\,\mathbb{P}(0=0, 1=1)=1\,\}$
$t := 0; \qquad k := 1;$
$\{\,\mathbb{P}(t=0, k=1)=1\,\} \Rightarrow$
$\{\,\mathbb{P}(k=N+1, t=\sum_{i=1}^{N} ss[i]) \geq r^N \ \vee\ \vee_{n=0}^{N} \mathbb{P}(k=n, t=\sum_{i=1}^{k-1} ss[i]) \geq r^{n-1}\,\}$
while $(k \leq N)$ do
  $\{\,\vee_{n=0}^{N} \mathbb{P}(k=n, t=\sum_{i=1}^{k-1} ss[i]) \geq r^{n-1}\,\} \Rightarrow$
  $\{\,\vee_{n=0}^{N} \mathbb{P}(k=n, t+ss[k]=\sum_{i=1}^{k} ss[i]) \geq r^{n-1}\,\}$
  $t := t + ss[k] \oplus_r skip;$
  $\{\,\vee_{n=0}^{N} \mathbb{P}(k=n, t=\sum_{i=1}^{k} ss[i]) \geq r^{n-1} \oplus_r true\,\} \Rightarrow$
  $\{\,\vee_{m=1}^{N+1} \mathbb{P}(k+1=m, t=\sum_{i=1}^{k} ss[i]) \geq r^{m-1}\,\}$
  $k := k+1$
  $\{\,\vee_{m=1}^{N+1} \mathbb{P}(k=m, t=\sum_{i=1}^{k-1} ss[i]) \geq r^{m-1}\,\}$
od
$\{\,\vee_{n=0}^{N+1} \mathbb{P}(k=n, t=\sum_{i=1}^{k-1} ss[i]) \geq r^{n-1}\,\} \wedge \text{not } (k \leq N) \ \Rightarrow$
$\{\,\mathbb{P}(t=\sum_{i=1}^{N} ss[i]) \geq r^N\,\}$

In the following example, a coin is tossed until heads is thrown. The number of required throws is shown to be geometrically distributed. For ease of notation the following shorthand notations are used.

$$p = q_\infty \vee \exists i : q[i]$$
$$q_\infty = \forall j > 0 : \mathbb{P}(x=j, done=true) = \tfrac{1}{2}^j$$
$$q[i] = \mathbb{P}(x=i, done=false) = \tfrac{1}{2}^i \wedge \forall j \in \{1, \ldots, i\} : \mathbb{P}(x=j, done=true) = \tfrac{1}{2}^j.$$

Then, assuming p is an invariant:

$$\{\,[\mathit{true}]\,\}$$
$$\text{bool } \mathit{done} = \mathit{false};\quad \text{int } n = 0;$$
$$\{\,\mathbb{P}(n = 1, \mathit{done} = \mathit{false}) = 1\,\} \Rightarrow \{\,p\,\}$$
$$\mathit{while\ not\ done\ do}\ x := x + 1; \mathit{done} = \mathit{true} \oplus_{\frac{1}{2}} \mathit{skip}\ \mathit{od}$$
$$\{\,p \wedge \mathit{done}\,\} \Rightarrow \{\,\forall n > 0 : \mathbb{P}(x = n) = \tfrac{1}{2}^n\,\}$$

To show that p is an invariant proof the rule (Or) is used to split the proof into two parts, the first of which is trivial. For the second part the rule (Exists) is used to give:

$$\{\,q[k]\,\}$$
$$\mathit{while\ not\ done\ do}$$
$$\quad \{\,(\mathit{not\ done})?q[k]\,\} \Rightarrow \{\,\mathbb{P}(x = i, \mathit{done} = \mathit{false}) = \tfrac{1}{2}^i\,\}$$
$$\quad x := x + 1;$$
$$\quad \{\,\mathbb{P}(x = i + 1, \mathit{done} = \mathit{false}) = \tfrac{1}{2}^i\,\}$$
$$\quad \mathit{done} = \mathit{true} \oplus_{\frac{1}{2}} \mathit{skip}$$
$$\quad \{\,\mathbb{P}(x = i + 1, \mathit{done} = \mathit{false}) = \tfrac{1}{2}^{i+1} + \mathbb{P}(x = i + 1, \mathit{done} = \mathit{true}) = \tfrac{1}{2}^{i+1}\,\}$$
$$\mathit{od}$$
$$\{\,\mathbb{P}(x = i + 1, \mathit{done} = \mathit{false}) = \tfrac{1}{2}^{i+1} + \mathbb{P}(x = i + 1, \mathit{done} = \mathit{true}) = \tfrac{1}{2}^{i+1} +$$
$$\forall j \in \{\,1, \ldots, i\,\}: \mathbb{P}(x = j, \mathit{done} = \mathit{true}) = \tfrac{1}{2}^j\,\} \Rightarrow \{\,q[j + 1]\,\} \Rightarrow \{\,p\,\}.$$

The requirement that p is $\langle \mathit{not\ done}, x := x + 1; \mathit{done} = \mathit{true} \oplus_{\frac{1}{2}} \mathit{skip}\rangle$-closed is easy to check but requires the presence of the q_∞ term.

6 Conclusions and Further Work

The main result of this paper is the introduction of a Hoare like logic, called *pH*, for reasoning about probabilistic programs. The programs are written in a language $\mathcal{L}_{\mathrm{pw}}$ and their meaning is given by the denotational semantics $\mathcal{D}$.

The probabilistic predicates used in the logic retain their usual truth value interpretation, i.e. they can be interpreted as true or false. Deterministic predicates can be extended to arithmetical functions yielding the probability that the predicate holds as done in e.g. [13] and [17]. This extension is incorporated by using the notation $\mathbb{P}(dp)$ to refer to exactly that, the chance that deterministic predicate dp holds. The chance of dp holding can then be exactly expressed or lower and/or upper bounds can be given within a probabilistic predicate. The main advantage of keeping the interpretation as truth values is that the logical operators do not have to be extended.

The logic *pH* is show correct with respect to the semantics $\mathcal{D}$. For an (earlier) infinite version of the logic a completeness result exists. For the current logic the question of completeness is still open. Especially the expressiveness of the probabilistic predicates has to studied further.

To be able to describe distributed randomized algorithms, it would also be interesting to extend the language and the logic with parallelism. However, verification of concurrent systems in general and extending Hoare logic to concurrent

systems in specific (see e.g. [3, 7]) is already difficult in the non-probabilistic case.

To make the logic practically useful, the process of checking the derivation of a Hoare-triple should be automated. Some work has been done to embed the logic in the proof verification system PVS. (See e.g. [12] on non-probabilistic Hoare logic in PVS.) The system PVS can then be used both to check the applications of the rules and to check the derivation of the implications between predicates required for the (Imp) rule. By modeling probabilistic states, PVS could perhaps also be used to verify the correctness of the logic, however this would require a lot of work on modeling infinite sums.

References

[1] L. de Alfaro. *Formal Verification of Probabilistic Systems.* PhD thesis, Stanford University, 1997.
[2] S. Andova. Probabilistic process algebra. In *Proceedings of ARTS'99*, Bamberg, Germany, 1999.
[3] K.R. Apt and E.-R. Olderog. *Verification of Sequential and Concurrent Programs.* Texts and Monographs in Computer Science. Springer-Verlag, 1991.
[4] C. Baier, M. Kwiatkowska, and G. Norman. Computing probabilistic lower and upper bounds for ltl formulae over sequential and concurrent markov chains. Technical Report CSR-98-4, University of Birmingham, June 1998.
[5] J.W. de Bakker. *Mathematical Theory of Program Correctness.* Series in Computer Science. Prentice-Hall Internationall, 1980.
[6] P.R. D'Argenio, J.-P. Katoen, and E. Brinksma. A stochastic process algebra for discrete event simulation. Technical report, University of Twente, 1998.
[7] N. Francez. *Program Verification.* International Computer Science Series. Addison-Wesley, 1992.
[8] J.I. den Hartog. Comparative semantics for a process language with probabilistic choice and non-determinism. Technical Report IR-445, Vrije Universiteit, Amsterdam, February 1998.
[9] J.I. den Hartog and E.P. de Vink. Mixing up nondeterminism and probability: A preliminary report. *ENTCS*, 1999. to appear.
[10] J.I. den Hartog and E.P. de Vink. Taking chances on merge and fail: Extending strong and probabilistic bisimulation. Technical Report IR-454, Vrije Universiteit, Amsterdam, March 1999.
[11] H. Hermanns. *Interactive Markov Chains.* PhD thesis, University of Erlangen-Nurnberg, July 1998.
[12] J. Hooman. Program design in PVS. In *Proceedings Workshop on Tool Support for System Development and Verification*, June 1996.
[13] D. Kozen. A probabilistic PDL. *Journal of Computer and System Sciences*, 30:162–178, 1985.
[14] K.G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94:1–28, 1991.
[15] N. Lynch. *Distributed Algorithms.* The Morgan Kaufmann series in data management systems. Kaufmann, 1996.
[16] C. Morgan and A. McIver. pGCL: formal reasoning for random algorithms. *South African Computer Journal*, 1999. to appear.

[17] C. Morgan, A. McIver, and K. Seidel. Probabilistic predicate transformers. *ACM Transactions on Programming Languages and Systems*, 18(3):325–353, May 1996.
[18] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.
[19] R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, Massachusetts Institute of Technology, June 1995.
[20] S.-H. Wu, S.A. Smolka, and E.W. Stark. Composition and behavior of probabilistic I/O automata. *Theoretical Computer Science*, 176:1–38, 1999.