VERIFYING PROBABILISTIC PROGRAMS USING A HOARE LIKE LOGIC

J.I. DEN HARTOG
Faculty of Exact Sciences, Free University, de Boelelaan 1081a,
1081 HV Amsterdam, The Netherlands

and

E.P. DE VINK
KPN Research, P.O. Box 421
2260 AK Leidschendam, The Netherlands

Received
Revised
Communicated by

ABSTRACT

Probability, be it inherent or explicitly introduced, has become an important issue in the verification of programs. In this paper we study a formalism which allows reasoning about programs which can act probabilistically. To describe probabilistic programs, a basic programming language with an operator for probabilistic choice is introduced and a denotational semantics is given for this language. To specify properties of probabilistic programs, standard first order logic predicates are insufficient, so a notion of probabilistic predicates is introduced. A Hoare-style proof system to check properties of probabilistic programs is given. The proof system for a sublanguage is shown to be sound and complete; the properties that can be derived are exactly the valid properties. Finally some typical examples illustrate the use of the probabilistic predicates and the proof system.

Keywords: Hoare logic, Probability, Randomized algorithms, Program verification, Probabilistic predicates

1. Introduction

Probability is introduced into the description of computer systems to model processes which behavior is inherently probabilistic or only approximately known. For example, of a faulty communication channel the error-rate provides stochastic information on the likelihood to have packages delivered without corruption along the channel. Probability may also explicitly enter the scene in randomized algorithms to solve problems which cannot be solved efficiently, or cannot be solved at all in a deterministic way [25, 2]. With the 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 precarious. Additional tools, such as formal verification techniques, are sought for to monitor and direct the software development process.

Quantitative approaches to formal methods for probabilistic systems, grafted upon the vast amount of work achieved in the area of performance modeling, start from a mathematical reconstruction of the system under consideration. Models that are often used are Markov chains and Markov decision processes (see, e.g., [16, 6]) and probabilistic input-output automata (cf. [28, 29], for example), sometimes augmented with notions of probabilistic bisimulation [22, 12]. In the probabilistic analyses of the model, results from probability theory are used to obtain, e.g. average performance or bounds on error probabilities. Model checking based techniques provide a logical language to characterize program properties and exploit automated tools to exhaustively search the state space (consult, e.g., [27, 19, 1, 15]).

For a wide range of programs the construction of the mathematical model can already be problematic. 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 and stochastic process algebra (see, for example, [5, 26, 3, 10, 9]) where equivalences of programs can be checked syntactically by equational reasoning.

Another approach is to introduce a calculus or a proof system as a vehicle to reason about the probabilistic programs directly. Earlier work on the proof theory for probabilistic programs that has inspired the present paper, can be found in e.g. [21, 20]. Kozen proposes in [21] a probabilistic dynamic logic in which arithmetical laws govern the program analysis. The thesis work of Jones [20], presents a proof system for probability in a state-less setting. An important strand of research in the syntactic approach (cf. [23, 24]) is focussed on predicate transformers. In [23], extending the predicate transformer work of [24], a calculus of greatest pre-expectations is given for a language with both probabilistic choice and nondeterminism. This calculus is illustrated by its application to the examples of an erratic 'sequence accumulator', an example recurring in this paper, and Rabin's 'choice coordination' algorithm.

The main contribution of the present paper is the proposal of a sound proof system for reasoning about a probabilistic extension of sequential programming. The assertion language, because of the introduction of probabilistic predicates, allows for expression of probabilities of deterministic predicates, which includes the possibility to state that a program variable has a certain standard probability distribution. For a fragment of the language a weakest precondition calculus and a proof of completeness are also provided. In some examples the relative strength of the probabilistic Hoare logic is illustrated.

The main difference between the work of Morgan et al. and ours is that we take Hoare logic as our starting point for which we introduce the notion of a probabilistic predicate. We are therefore in a position to justify the proof rules with respect to a denotational semantics. Moreover, from a correctness point of view, our probabilistic predicates seem intuitively more attractive. The expectation based calculus on the other hand provides a way to generate useful quantitative information, whereas the emphases in our approach is on the verification of given quantitative information. Further study should shed more light on the relative merits of the two methods.

Deterministic Hoare logic is a well-known system to derive (partial) correctness formulae, also known as Hoare triples (see [17, 7]). A formula $\{\,p\,\}\ s\ \{\,q\,\}$ states that the predicate p is a sufficient precondition for the program s to guarantee that predicate q holds after termination. What the values of the variables in a program, i.e. the (deterministic) state of the program, will be, cannot be fully determined if the program is probabilistic. Only the probability of being in a certain state can be given, thus yielding the notion of a probabilistic state. Probabilistic states are constructed using so called pseudo-distributions. In a pseudo-distribution, the total probability may be less than 1. Beside the technically convenience of working with pseudo-distributions instead of only with distributions, the possibility of states with a total probability of less than 1 is needed to describe non-termination and is needed in the conditioning of predicates. In a probabilistic state, a deterministic predicate will no longer be either true or false, but it is true with a certain probability. This can, for example, be dealt with by changing the interpretation of validity of a predicate to a function to $[0, 1]$ or $\mathbb{R}$ instead of a function to $\{ \mathit{true}, \mathit{false} \}$ as in [21, 23]. The approach chosen here is to extend the syntax of predicates to allow making claims about the probability that a certain deterministic predicate holds. We refer to the extended form of predicates as probabilistic predicates. A proof system for probabilistic programs should take these probabilistic predicates into account.

After mathematical preliminaries in Section 2, the language $\mathcal{L}_{\mathrm{pif}}$ with conditional and probabilistic choice is introduced in Section 3. Additionally, a denotational semantics for $\mathcal{L}_{\mathrm{pif}}$ is provided. Section 4 discusses probabilistic predicates and presents the Hoare-logic for $\mathcal{L}_{\mathrm{pif}}$. The relationship with weakest preconditions and the completeness of the Hoare-logic is subject of Section 5. Section 6 extends $\mathcal{L}_{\mathrm{pif}}$ with the construct of iteration, yielding $\mathcal{L}_{\mathrm{pw}}$. Furthermore a denotational semantics for $\mathcal{L}_{\mathrm{pw}}$ and a proof system are included. Section 7 discusses some examples that illustrate the Hoare-logic that is introduced in earlier sections. Section 8 finishes the paper with concluding remarks and indicates further work.

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 pointwise to functions from X to Y (for $f, g : X \to Y$ 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 those $x \in X$ for which $f(x) \neq 0$. The set of all functions from X to $[0, 1]$ with countable support is denoted by $X \to_{\mathcal{CS}} [0, 1]$. Given a function $f : X \to_{\mathcal{CS}} [0, 1]$ 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 distributions $\mathit{Dist}(X)$ over a set X is defined as the subset of functions in $X \to_{\mathcal{CS}} [0, 1]$ with sum at most 1, i.e.,

$$\mathit{Dist}(X) = \{ f \in X \to_{\mathcal{CS}} [0, 1] \mid \sum f[X] \leq 1 \} .$$

For a distribution $f \in \mathit{Dist}(X)$ and an element $x \in X$, $f(x)$ is interpreted as the probability that x occurs. The sum $\sum f[X]$ is called the total probability of the distribution f. The set $\mathit{Dist}(X)$ is a cpo, with minimal element $\underline{0}$, the function that assigns 0 to each element of X. For each ascending sequence in $\mathit{Dist}(X)$ the limit exists within $Dist(X)$ and corresponds to the least upper bound of the sequence. A distribution with total probability less than one, indicates a situation with partial information. Intuitively, part of the distribution is not known; either because it is calculated elsewhere or because it is not reached at all (due to non-termination).

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

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

3. Syntax and Semantics of $\mathcal{L}_{\text{pif}}$

In this section the language $\mathcal{L}_{\text{pif}}$ is introduced and a denotational semantics $\mathcal{D}$ for $\mathcal{L}_{\text{pif}}$ is given. The language $\mathcal{L}_{\text{pif}}$ is a basic programming language containing skip, assignment, sequential composition, conditional choice and probabilistic choice.

Programs in the language $\mathcal{L}_{\text{pif}}$ are interpreted as state transformers, where the state is comprised of the data stored in the variables. The value of a variable can be changed by executing an assignment. An assignment, written as x := e, can change the value belonging to the variable x to the value represented by the expression e. Each variable has a data type associated with it. Each data type has its own syntax for the expressions. Assignments are assumed to be type correct, i.e. only expressions of the right type are assigned to a variable. For readability, the definitions will be given only for a single type of data, the integers. In the examples other types, like arrays of integers, will also be used.

The conditional choice uses boolean conditions to choose between statements. Below the syntax for integer expressions and boolean expressions is given, followed by the syntax of the language $\mathcal{L}_{\text{pif}}$.

The variables used in the programs in $\mathcal{L}_{\text{pif}}$ are from a set $PVar$, called the set of program variables. As expressions are needed over several sets of variables, the definition of expressions uses a general set of variables. Given a set of variables, say Var, the set of integer expressions over Var is denoted by $Exp\langle Var\rangle$. The value of an expression can be found using the evaluation function $\mathcal{V}$.

Definition 1 *Let Var denote a set of variables and let v range over Var. The set integer expressions over Var, denoted by $Exp\langle Var\rangle$ and ranged over by e is given by*

$$e \quad = \quad v \mid n \mid e + e \mid e - e \mid e \cdot e \mid e \,\texttt{div}\, e \mid e \,\texttt{mod}\, e \mid \ldots ,$$

where n denotes any element of $Int_\perp$. The evaluation function $\mathcal{V} : Exp\langle Var\rangle \to (Var \to Int_\perp) \to Int_\perp$ that computes the value of an expression given the values of the variables is defined by:

$$\mathcal{V}(n)(f) = n, \qquad \mathcal{V}(v)(f) = f(v), \qquad \mathcal{V}(e \;\texttt{op}\; e')(f) = \mathcal{V}(e)(f) \;op\; \mathcal{V}(e')(f),$$

for $f \in Var \to Int_\perp$ and $op \in \{+, -, \cdot, \texttt{mod}, \texttt{div}\ \}$.

A basic integer expression is a constant n or a variable v. Integer expressions can be combined with the usual operators on integers. Basically any functions on integers that is deemed useful could also be added. The set $Int_\perp$ is the collection of integers to which $\perp$ is added. The symbol $\perp$ is used for undefinedness. The operations yield $\perp$ when the operation is not defined, e.g. for division by zero, and when any of the arguments is undefined. To avoid many technical complication associated with undefined expressions, the symbol $\perp$ is treated as an ordinary value which is smaller than any integer $n \in Int$. The choice to make $\perp$ the smallest element of $Int_\perp$ is an arbitrary one.

Example 1 *The value of the expression* $(1 \texttt{ div } 0) + 1$ *is* $\mathcal{V}((1 \texttt{ div } 0) + 1)(f) = \perp + 1 = \perp$*. The value of the expression* $v - 2$ *depends on the value of* v*;* $\mathcal{V}(v-2)(f) = f(v) - 2$*.*

Conditional choices in the program are based on boolean conditions. As with integer expressions the boolean conditions over a set of variables is given for a general set of variables *Var*. The set of all boolean conditions over the set of variables *Var* is denoted by $BC\langle Var\rangle$ and a typical boolean condition is denoted by c. True, false and comparing expressions are the basic conditions. Conditions can be combined using the standard logical operators.

Definition 2 *The set of boolean conditions over* Var*, denoted by* $BC\langle \mathrm{Var}\rangle$ *and ranged over by* c*, is given by*

$$c \quad = \quad \texttt{true} \mid \texttt{false} \mid \mathrm{e} \texttt{=} \mathrm{e} \mid \mathrm{e} < \mathrm{e} \mid \ldots \mid c \wedge c \mid c \vee c \mid \neg c \mid c \rightarrow c,$$

where e *is an expression in* $Exp\langle \mathrm{Var}\rangle$*. The evaluation function* $\mathcal{B} : BC\langle \mathrm{Var}\rangle \rightarrow (\mathrm{Var} \rightarrow Int_\perp) \rightarrow Bool$*, that computes the value of a boolean condition given the values of the variables, is defined by:*

$$\begin{array}{rcl@{\qquad}rcl}
\mathcal{B}(\texttt{true})(f) & = & \mathit{true}, & \mathcal{B}(\mathrm{e} \texttt{=} \mathrm{e}')(f) & = & \mathcal{V}(\mathrm{e})(f) = \mathcal{V}(\mathrm{e}')(f), \\
\mathcal{B}(\texttt{false})(f) & = & \mathit{false}, & \mathcal{B}(\mathrm{e} < \mathrm{e}')(f) & = & \mathcal{V}(\mathrm{e})(f) < \mathcal{V}(\mathrm{e}')(f), \\
\mathcal{B}(\neg c)(f) & = & \neg \mathcal{B}(c)(f), & \mathcal{B}(c \texttt{ op } c')(f) & = & \mathcal{B}(c)(f) \mathbin{op} \mathcal{B}(c')(f),
\end{array}$$

where f *is a function in* $\mathrm{Var} \rightarrow Int_\perp$ *and* $\texttt{op}$ *is* $\wedge$*,* $\vee$ *or* $\rightarrow$*.*

The meaning of `true`, `false` and the logical connectives is clear. The value of the variables is needed to evaluate the expressions that occur in the boolean conditions. The boolean condition $\mathrm{e} \texttt{=} \mathrm{e}'$ is true when e and e' evaluate to the same value. The condition $\mathrm{e} < \mathrm{e}'$ is true when the value e evaluates to is smaller than the value that e' evaluates to.

Example 2 *For* $\mathrm{x}, y \in \mathrm{Var}$*, the boolean condition* $(\mathrm{x} > 0)$ *states that* x *is positive,* $(\mathrm{x} = 0 \texttt{ mod } y)$ *expresses that* x *can be divided by* y *and* $\neg\ (\mathrm{x} \texttt{=} y)$ *expresses that* x *and* y *are different.*

Note that $(\mathrm{x} \texttt{ div } 0) \leq y$ *is always true, as* $\perp$ *is the smallest element of* $Int_\perp$*.*

Having defined expressions and boolean conditions, the syntax of the language $\mathcal{L}_{\mathrm{pif}}$ can now be made precise. As mentioned, the language $\mathcal{L}_{\mathrm{pif}}$ contains skip, assignment, sequential composition, conditional choice and probabilistic choice.

Definition 3 *Let* $P\mathrm{Var}$ *be a set of program variables and let* x *range over* $P\mathrm{Var}$*. The statements in* $\mathcal{L}_{\mathrm{pif}}$*, ranged over by* s*, are given by:*

$$s ::= \texttt{skip} \mid \mathrm{x} \texttt{ := } \mathrm{e} \mid s \texttt{ ; } s \mid s \oplus_\rho s \mid \texttt{if } c \texttt{ then } s \texttt{ else } s \texttt{ fi},$$

where e is an expression in $Exp\langle PVar\rangle$, c is a condition in $BC\langle PVar\rangle$ and ρ is a ratio in the open interval $(0,1)$.

The statement `skip` does nothing. The statement x `:=` e assigns the value of expression e to the variable x. The statement s ; s' is executed by first executing s and then executing s'. The statement `if` c `then` s `else` s' `fi` is executed by evaluating the condition c and executing s if the condition is true and s' if the condition is false. The statement $s \oplus_\rho s'$ denotes the probabilistic choice between the statements s and s'. With probability ρ the statement s is chosen and the statement s' is chosen with probability $1-\rho$.

For a deterministic program that is interpreted as a state transformer, the state of the computation is given by the value of the variables. The state space $\mathcal{S}$ for deterministic programs consists of $\mathcal{S} = PVar \to Int_\perp$. 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 values should be given. A first idea may be to take as a state space $PVar \to Dist(Int_\perp)$. This does give, for each variable x, the chance that x takes a certain value but it does not describe the possible dependencies between the variables.

Let us, in order to clarify this matter, consider the following example. In the left situation (see below), 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	$\frac{1}{4}$	$\frac{1}{4}$

Clearly the dependencies between the variables must also be expressed in the probabilistic state. Therefore, the more general state space $\Theta = Dist(PVar \to Int_\perp)$ is required. In $\theta \in \Theta$, 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 x takes value n can be found by summing the probabilities of all deterministic states which assign n to x.

Definition 4

(a) The set of deterministic states $\mathcal{S}$, ranged over by σ, is given by $\mathcal{S} = PVar \to Int_\perp$.

(b) The set of probabilistic states Θ, ranged over by θ, is given by $\Theta = Dist(\mathcal{S})$.

As a program is interpreted as a state transformer, the meaning of a program is a function from states to states. This function returns the end state after execution of the program is return given a start state. The meaning of a program in $\mathcal{L}_{\text{pif}}$ is given by a denotational semantics $\mathcal{D}$.

A key step in the definition of the denotational semantics is giving the meaning of a basic program consisting of a single assignment, $\mathtt{x} := e$. For a deterministic state σ, the state resulting from executing the program $\mathtt{x} := e$ in state σ is a variant of the state σ where the value of e is assigned to the variable x: $\sigma[\mathrm{x}/\mathcal{V}(e)(\sigma)]$. In a deterministic state σ the value of an integer expression e can be found by using the evaluation function $\mathcal{V}$.

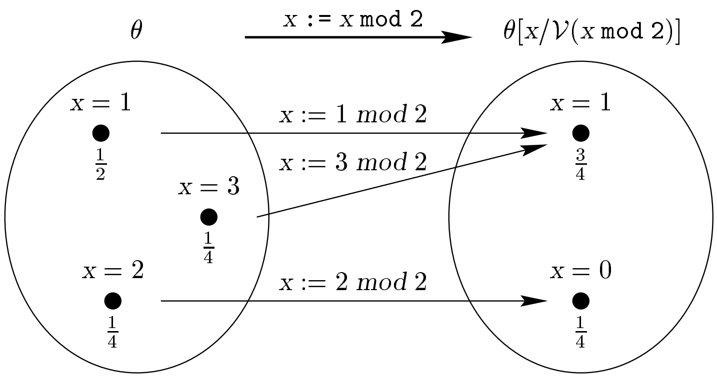
One would also like to define a notion of a variant of a probabilistic state which can be used to define the effect of assignment in a probabilistic state. Simply using a construction $[\mathrm{x}/v]$ where a (fixed) value is assigned to the a variable, however, is not sufficient. In a probabilistic state θ the values of the variables are, in general, not known and the value of integer expressions cannot be found.

The variant of a probabilistic state should be the probabilistic state after assignment of an expression e to a variable x. To find the probabilistic state after executing an assignment, the assignment has to be done in each deterministic state which has a positive probability. For each of these deterministic states, the assignment is done by taking a variant of the state. The value assigned to x depends on the deterministic state. This gives the form the variant of a probabilistic state should have: a variant of a probabilistic state should assign a new value to a variable, depending on the deterministic state being considered.

Definition 5 *Let $f : \mathcal{S} \to Int_\perp$ and let θ be a probabilistic state in Θ. The variant of θ, denoted by $\theta[\mathrm{x}/f]$, is given by: $\theta[\mathrm{x}/f](\sigma) = \sum_{\sigma' \in V} \theta(\sigma')$, with $V = \{\, \sigma' \in \mathcal{S} \mid \sigma'[\mathrm{x}/f(\sigma')] = \sigma \,\}$.*

The function f gives the value to use for a given deterministic state. To find the probability of ending up in a deterministic state σ, the probability of all deterministic states σ' that become σ after taking the variant using the value $f(\sigma')$, have to be added. Straightforward calculation shows that $\theta[\mathrm{x}/f]$ is again a pseudo-distribution in Θ. The following example shows how the variant of a probabilistic state can be used to calculate the effect of assignment on a probabilist state.

Example 3 *Assigning $\mathtt{x\,mod\,2}$ to x in the probabilistic state $\theta = \frac{1}{2}\langle \mathrm{x} = 1\rangle + \frac{1}{4}\langle \mathrm{x} = 2\rangle + \frac{1}{4}\langle \mathrm{x} = 3\rangle$ gives the probabilistic state $\theta[\mathrm{x}/\mathcal{V}(\mathtt{x\,mod\,2})] = \frac{3}{4}\langle \mathrm{x} = 1\rangle + \frac{1}{4}\langle \mathrm{x} = 0\rangle$.*



The evaluation function $\mathcal{V}$ for a fixed expression e is a function $\mathcal{V}(e)$ from $\mathcal{S}$ to $Int_\perp$; $\mathcal{V}(e)$ is a function of the right type for use in the variant of a probabilistic state.

Having described the meaning of a basic statement assignment, the next step in giving the denotational semantics for $\mathcal{L}_{\text{pif}}$ is defining the effect of the operations in $\mathcal{L}_{\text{pif}}$. Sequential composition is not difficult to deal with. The following operations on probabilistic states are used to define the effect of the operators probabilistic choice and conditional choice.

Definition 6 *The operations probabilistic choice* $\oplus_\rho : \Theta \times \Theta \to \Theta$ *and unscaled conditional* $c? : \Theta \to \Theta$ *are defined as follows:*

$$\begin{aligned}
\theta_1 \oplus_\rho \theta_2 &= \rho\theta_1 + (1-\rho)\theta_2, \\
c?\theta(\sigma) &= \begin{cases} \theta(\sigma) & \textit{if } c \textit{ is true in } \sigma \textit{ i.e. } \mathcal{B}(c)(\sigma) = \textit{true}, \\ 0 & \textit{otherwise}, \end{cases}
\end{aligned}$$

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

Recall that $\theta \in \Theta$ 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 operation $\oplus_\rho$ simply combines two probabilistic states with the appropriate probabilities. The probabilistic state $c?\theta$ is obtained from θ by removing any probability for deterministic states not satisfying c. As straightforward calculations show, for any probabilistic state θ the equations $\theta \oplus_\rho \theta = \theta$ and $c?\theta + \neg c?\theta = \theta$ hold.

Example 4 *Using the notation of example 3 the effect of the operators* $\oplus_\rho$ *and* $c?$ *is easy to see:*

$$\begin{aligned}
1\langle x = 1\rangle \oplus_{\frac{1}{2}} (\tfrac{1}{2}\langle x = 2\rangle + \tfrac{1}{2}\langle x = 3\rangle) &= \tfrac{1}{2}\langle x = 1\rangle + \tfrac{1}{4}\langle x = 2\rangle + \tfrac{1}{4}\langle x = 3\rangle, \\
(x\, \texttt{mod 2 = 1})?(\tfrac{1}{2}\langle x = 1\rangle + \tfrac{1}{4}\langle x = 2\rangle + \tfrac{1}{4}\langle x = 3\rangle) &= \tfrac{1}{2}\langle x = 1\rangle + \tfrac{1}{4}\langle x = 3\rangle.
\end{aligned}$$

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

Definition 7 *The denotational semantics* $\mathcal{D} : \mathcal{L}_{\text{pif}} \to (\Theta \to \Theta)$ *is given by*

$$\begin{aligned}
\mathcal{D}(\texttt{skip})(\theta) &= \theta, \\
\mathcal{D}(x \texttt{ := } e)(\theta) &= \theta[x/\mathcal{V}(e)], \\
\mathcal{D}(s\ ;\ s')(\theta) &= \mathcal{D}(s')(\mathcal{D}(s)(\theta)), \\
\mathcal{D}(s \oplus_\rho s')(\theta) &= \mathcal{D}(s)(\theta) \oplus_\rho \mathcal{D}(s')(\theta), \\
\mathcal{D}(\texttt{if } c \texttt{ then } s \texttt{ else } s' \texttt{ fi})(\theta) &= \mathcal{D}(s)(c?\theta) + \mathcal{D}(s')(\neg c?\theta)\,.
\end{aligned}$$

The clause for `skip` is clear; the execution of `skip` leaves the state unchanged. The clause for assignment uses the notion of variant of a probabilistic state introduced above. The evaluation function for the expression e is assigned to the variable x. The clause for sequential composition is as usual. To find the state after executing $s\ ;\ s'$ in state θ, s' is executed in the state resulting from executing s, i.e. in the state $\mathcal{D}(s)(\theta)$. The clause for the probabilistic choice between s and s' uses the probabilistic choice between states as introduced in definition 6. To execute the statement `if` c `then` s `else` s' `fi`, the statement s is executed in the part of the state satisfying c and s' in the part of the state satisfying $\neg c$.

Example 5 *Using definition 7 and examples 3 and 4:*

$$\begin{aligned}\mathcal{D}((\mathtt{x := 1} \oplus_{\frac{1}{2}} \mathtt{skip}); \mathtt{x := x\,mod\,2})(\tfrac{1}{2}\langle x = 2\rangle + \tfrac{1}{2}\langle x = 3\rangle) \\ = \mathcal{D}(\mathtt{x := x\,mod\,2})(\tfrac{1}{2}\langle x = 1\rangle + \tfrac{1}{4}\langle x = 2\rangle + \tfrac{1}{4}\langle x = 3\rangle) \\ = \tfrac{3}{4}\langle x = 1\rangle + \tfrac{1}{4}\langle x = 0\rangle.\end{aligned}$$

The following property of the denotational semantics can easily be checked by structural induction.

Lemma 1 *The denotational semantics is linear in the probabilistic state:*

$$\mathcal{D}(s)(\theta + \theta') = \mathcal{D}(s)(\theta) + \mathcal{D}(s)(\theta') \text{ and } \mathcal{D}(s)(\rho\theta) = \rho\mathcal{D}(s)(\theta).$$

Note that in general $\mathcal{D}(s)(c?\theta) \neq c?\mathcal{D}(s)(\theta)$ as the execution of s may influence the value of the condition c.

4. Probabilistic Predicates and Hoare Logic

The denotational semantics gives the meaning of programs in $\mathcal{L}_{\text{pif}}$. One would like to be able to check claims about a program like "after executing the program property p will hold". In this section a probabilistic logic is given. The predicates in the logic specify properties of probabilistic states. The claim above becomes "the predicate p holds in the state resulting from execution the program". The state resulting from the execution of the program may depend on the state at the start of the program. A more precise claim about the behavior of a program is therefore "if the state before execution satisfies the predicate q then after executing the program, the state will satisfy the predicate p". Hoare triples, also known as correctness formulae, express exactly such claims.

To derive which claims, i.e. which Hoare triples, are valid, a proof system pH is introduced. The proof system pH is based on standard Hoare logic for non-probabilistic programs. The rules known from Hoare logic are adapted to fit the probabilistic setting. Also several new rules are added. The proof system is shown to be sound (with respect to the denotational semantics) by showing that any Hoare triple that can be derived from the proof system is valid.

For a deterministic state a basic property that one wants to check is e.g. "variable x has value n". For a probabilistic state, an example of a basic property is "variable x has value n with probability at least $\frac{1}{2}$". Below the exact syntax of deterministic predicates and Hoare triples are given after which the probabilistic predicates are introduced.

Definition 8 *Let LVar be as set of logical variables ranged over by i. The set of deterministic predicates, denoted by DPred and ranged over by dp, is given by:*

$$\begin{aligned} dp \quad ::= \quad & \mathtt{true} \mid \mathtt{false} \mid e = e \mid e < e \mid \ldots \mid dp \wedge dp \mid dp \vee dp \mid \neg\, dp \mid dp \rightarrow dp \mid \\ & \forall i : dp \mid \exists i : dp, \end{aligned}$$

where $e \in Exp\langle PVar \cup LVar\rangle$ is an expression over program variables and logical variables. The set of all interpretations of logical variables $\mathcal{I}$ is given by

$\mathcal{I} = LVar \to Int_\perp$. *A typical interpretation is denoted by I. The satisfaction relation for deterministic predicates, $\models$, works as expected. For example,*

$$
\begin{array}{lcllll}
(\sigma, I) & \models & \mathtt{true}, & & \\
(\sigma, I) & \models & e = e' & \textit{when} & \mathcal{V}(\sigma, I)(e) = \mathcal{V}(\sigma, I)(e'), \\
(\sigma, I) & \models & dp \wedge dp' & \textit{when} & (\sigma, I) \models dp \textit{ and } (\sigma, I) \models dp', \\
(\sigma, I) & \models & \forall i : dp & \textit{when} & (\sigma, I[i/n]) \models dp \textit{ for all } n \in \textit{Int}.
\end{array}
$$

Deterministic predicates are basically boolean conditions on program variables and logical variables with the addition of the quantifiers $\forall$ and $\exists$. The interpretation of deterministic predicates is as usual. Because both logical variables and program variables may be used, both a deterministic state σ and an interpretation of the logical variables I are required to evaluate a deterministic predicate. Only a few cases of the definition of the satisfaction relation $\models$ are given. The other cases are similar.

Example 6 *As $(i < 4) \wedge (i > x)$ holds in $(\langle x = 2\rangle, \langle i = 3\rangle)$, we have that for any interpretation I: $(\langle x = 2\rangle, I) \models \exists i : (i < 4) \wedge (i > x)$.*

Substituting an expression e for the variable x in the predicate is denoted by $[x/e]$. An important property of substitution on deterministic predicates is that $(\sigma, I) \models dp[x/e]$ holds exactly when $(\sigma[x/\mathcal{V}(e)(\sigma)], I) \models dp$. This property is essential for soundness of the assign rule in Hoare logic and for completeness of Hoare logic. Note that here $[x/e]$ is substitution in a predicate, while $[x/\mathcal{V}(e)(\sigma)]$ is taking a variant of a state.

A deterministic Hoare triple or correctness formula, $\{\, dp \,\}\ s\ \{\, dp' \,\}$, describes that dp is a precondition and dp' is a postcondition of program s. The Hoare triple is said to be correct or valid, denoted $\models\ \{\, dp \,\}\ s\ \{\, dp' \,\}$, if execution of s in any state that satisfies dp will lead to a state satisfying dp'. The execution of the program s can change the value of the program variables, i.e. can chance the state. The logical variables are not effected by the execution of the program, so their interpretation remains the same.

$$\models\ \{\, dp \,\}\ s\ \{\, dp' \,\} \quad \text{when} \quad (\sigma, I) \models dp \Rightarrow (\sigma', I) \models dp' \ (\text{for all } I \in \mathcal{I})\,.$$

where σ' is the state resulting from executing the program s in state σ.

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. [21, 24]). This approach is useful to describe the arithmetical aspects of the probabilities involved in a probabilistic program. There are however two drawbacks. The predicates cannot express claims about the probability, only the value of the predicate gives information about the probabilities. A property like "dp holds with probability $\frac{1}{2}$" cannot be expressed as a predicate. Secondly the normal logical operators like $\wedge$ have to be extended to work on $[0,1]$. There seems to be no way to do this which preserves the logical aspects of these operators and does not make assumptions about dependencies between the predicates.

The key observation that the chance that a deterministic predicate holds in a probabilistic state is a real number in $[0,1]$ is also used here, but only as the basis for building probabilistic predicates. Probabilistic predicates as used here are predicates in the usual sense and can only have a truth value i.e. true or false. The extension to probabilistic predicates is made in the syntax where constructs are added to express claims about probabilities.

Comparison of (integer) expressions forms the basis for deterministic predicates. For probabilistic predicates, comparison of real expressions is used as a starting point.

Definition 9 *Let RLVar be a set of a real logical variables, also called probabilistic variables, ranged over by r. The set of real expressions RealExp, ranged over by e_r is given by*

$$e_r \quad ::= \quad \rho \mid r \mid \mathbb{P}(dp) \mid e_r + e_r \mid e_r - e_r \mid e_r * e_r \mid e_r / e_r \mid {e_r}^e \mid \ldots$$

where e is an integer expression over logical variables, $e \in Exp\langle LVar\rangle$.

A basic real expression is a constant ρ, a real logical variable r or the probability of a deterministic predicate $\mathbb{P}(dp)$. The expressions can be combined using functions on real numbers like $+$, $-$, $*$ and $/$. Integer expressions over (integer) logical variables are introduced in the real expressions when using operators which take an integer argument as in ${e_r}^e$. Other operators can be added as needed. As the real expressions and real logical variables are used to denote probabilities, the real logical variables are also referred to as probabilistic variables.

Example 7 *The expressions $\frac{1}{4}$, $\frac{1}{2} * \mathbb{P}(x = i)$ and $\mathbb{P}(x < 5) + r^i$ are all correct real expressions but ${\frac{1}{2}}^{x}$ is not, as program variables cannot be used in real expressions, except within a deterministic predicate in the $\mathbb{P}(dp)$ construct.*

To evaluate a real expression, like $\mathbb{P}(x < 5) + r^i$, one needs to know the value of the logical and probabilistic variables that are used in the expression. For a program variable, not the value but rather its distribution is required. The values of the logical and probabilistic variables are given by an interpretation functions J. The distribution of the program variables is given by a probabilistic state θ. This also explains why a program variable cannot be used outside of the $\mathbb{P}(dp)$ construct; the value of a program variable is not known, only its distribution.

Definition 10

(a) An interpretation J of probabilistic and logical variables is a function which assigns element of $\mathrm{Int}_\perp$ to each logical variable and an element of $\mathbb{R}_\perp$ to each probabilistic variable. The restriction of J to logical variables is denoted by $J_\mathcal{I}$. The set of all interpretations is denoted by $\mathcal{J}$.

(b) The evaluation function for real expressions $\mathcal{V}_r : RealExp \to (\Theta \times \mathcal{J}) \to \mathbb{R}_\perp$ is

given by:

$$\begin{array}{rcl}
\mathcal{V}_r(\rho)(\theta, J) & = & \rho, \\
\mathcal{V}_r(r)(\theta, J) & = & J(r), \\
\mathcal{V}_r(\mathbb{P}(dp))(\theta, J) & = & \sum_{\sigma \in V} \theta(\sigma), \\
\mathcal{V}_r(\mathrm{e}_r \; op \; \mathrm{e}'_r)(\theta, J) & = & \mathcal{V}_r(\mathrm{e}_r)(\theta, J) \; op \; \mathcal{V}_r(\mathrm{e}'_r)(\theta, J), \\
\mathcal{V}_r(\mathrm{e}_r^{\mathrm{e}'})(\theta, J) & = & \mathcal{V}_r(\mathrm{e}_r)(\theta, J)^{\mathcal{V}(\mathrm{e}')(J_{\mathcal{I}})},
\end{array}$$

with $V = \{ \sigma' \mid (\sigma', J_{\mathcal{I}}) \models \mathrm{dp} \}$ *and* $op \in \{ +, -, *, / \}$.

(c) *The meta variable* j *is used to range over the union of the set logical variables and the set of probabilistic variables:* $j ::= i \mid r$.

The value of the constant ρ is ρ. The value of the variable r is given by the interpretation J. The value of $\mathbb{P}(dp)$ is the probability that dp holds in the given state θ. This probability is found by summing the probabilities of all deterministic states which satisfy dp. The operations $+$, $-$, $*$ and $/$ on the normal operations on the reals where $\perp$ is used again to denote that the operation or one of its arguments is undefined, e.g. when dividing by zero.

The real expressions already show how chances are incorporated. The chance on a deterministic predicate holding is simply used as part of the expressions. Using the real expressions, real based conditions are build similar to the way integer expressions were used to built the boolean conditions.

Definition 11 *The set of real based conditions, denoted by RC and ranged over by* c_r*, is given by:*

$$c_r \quad = \quad c \mid \mathrm{e}_r = \mathrm{e}_r \mid \mathrm{e}_r < \mathrm{e}_r \mid \ldots \mid c_r \vee c_r \mid c_r \wedge c_r \mid \neg c_r \mid c_r \rightarrow c_r,$$

where c *is a condition over logical variables;* $c \in BC\langle LVar \rangle$*. The evaluation function* $\mathcal{B}_r : RC \rightarrow (\Theta \times \mathcal{J}) \rightarrow Bool$ *that computes the value of a real based condition given the values of the variables is given by:*

$$\begin{array}{rcl}
\mathcal{B}_r(c)(\theta, J) & = & \mathcal{B}(c)(J_{\mathcal{I}}), \\
\mathcal{B}_r(\neg c_r)(\theta, J) & = & \neg \mathcal{B}_r(c_r)(\theta, J), \\
\mathcal{B}_r(\mathrm{e}_r = \mathrm{e}'_r)(\theta, J) & = & \mathcal{V}_r(\mathrm{e}_r)(\theta, J) = \mathcal{V}_r(\mathrm{e}'_r)(\theta, J), \\
\mathcal{B}_r(\mathrm{e}_r < \mathrm{e}'_r)(\theta, J) & = & \mathcal{V}_r(\mathrm{e}_r)(\theta, J) < \mathcal{V}_r(\mathrm{e}'_r)(\theta, J), \\
\mathcal{B}_r(c_r \; \mathtt{op} \; c'_r)(\theta, J) & = & \mathcal{B}_r(c_r)(\theta, J) \; op \; \mathcal{B}_r(c'_r)(\theta, J),
\end{array}$$

where $\mathtt{op}$ *is* $\wedge$, $\vee$ *or* $\rightarrow$.

The evaluation of real based conditions is very similar to the evaluation of boolean conditions. The following example shows that real based conditions can express both claims about the distribution of variables as well as claims about the dependencies between variables.

Example 8 *With real based conditions it is possible to express claims like e.g.* $\mathbb{P}(\mathtt{x} = 1) = \frac{1}{2} \wedge \mathbb{P}(\mathtt{y} = 1) = \mathbb{P}(\mathtt{x} = 1)$ *and* $\mathbb{P}(\mathtt{x} = 1 \wedge \mathtt{y} = 1) = \frac{1}{2}$*. The first condition gives information about the distributions of* $\mathtt{x}$ *and* $\mathtt{y}$ *while the second condition also gives information about the dependency between* $\mathtt{x}$ *and* $\mathtt{y}$*.*

Real based conditions can be used to express claims about probabilities. The conditions also capture the logical combinations of such claims. Probabilities, however, also have an arithmetical aspect which prompts the interpretation of predicates as real numbers. If interpreted as real numbers, the predicates can be added and scaled, which are typical operations one wants to use when calculating probabilities. These operations are possible on real expressions, but not on the conditions. To allow arithmetical manipulation with conditions themselves, arithmetical operators are incorporated in the syntax of probabilistic predicates. Nevertheless, probabilistic predicates are still interpreted as functions to truth values.

Definition 12 *The set of probabilistic predicates* Pred*, ranged over by p and by q, is given by*

$$p \quad = \quad c_r \mid p \wedge p \mid p \vee p \mid \exists j : p \mid \forall j : p \mid \rho \cdot p \mid p + p \mid p \oplus_\rho p \mid c?p,$$

where c is a boolean condition over program variables; $c \in BC\langle P\mathrm{Var}\rangle$. The satisfaction relation for probabilistic predicates $\models$ is given by:

$$\begin{array}{rll}
(\theta, J) \models c_r & \textit{when} & \mathcal{B}_r(c_r)(\theta, J) = \textit{true}, \\
(\theta, J) \models \rho \cdot p & \textit{when} & \exists \theta' : \theta = \rho \cdot \theta', (\theta', J) \models p, \\
(\theta, J) \models p + p' & \textit{when} & \exists \theta_1, \theta_2 : \theta_1 + \theta_2 = \theta, (\theta_1, J) \models p, (\theta_2, J) \models p', \\
(\theta, J) \models p \oplus_\rho p' & \textit{when} & \exists \theta_1, \theta_2 : \theta_1 \oplus_\rho \theta_2 = \theta, (\theta_1, J) \models p, (\theta_2, J) \models p', \\
(\theta, J) \models c?p & \textit{when} & \exists \theta' : \theta = c?\theta', (\theta', J) \models p.
\end{array}$$

The other operators are as usual.

The operator $+$ is used for addition of predicates, the operator $\rho\cdot$ performs scaling. A weighted sum $\oplus_\rho$ combines scaling and addition. Finally $c?p$ gives conditional probabilities without normalizing. A state satisfies the predicate $\rho \cdot p$ if it is a scaled version of a state satisfying p. A state θ satisfies the predicate $p + p'$ if it can be split into parts satisfying p and p'. The operators $\oplus_\rho$ and $c?$ are similar.

As with deterministic predicates, substitution is denoted by $[x/e]$. As program variables can only occur within the deterministic predicate in the $\mathbb{P}(dp)$ construct, only the deterministic predicates within the probabilistic predicate are effected by the substitution.

Example 9 *The predicate $\forall i : (i \leq 0) \vee \mathbb{P}(x = i) = \frac{1}{2}^i$, states that the variable x is geometrically distributed. The third example in section 7 discusses a program that results in a variable having a geometric distribution.*

The predicate $(\mathbb{P}(\mathtt{x = 1}) = \frac{1}{2}) + (\mathbb{P}(\mathtt{x = 1}) = \frac{1}{2})$ is equivalent with the predicate $\mathbb{P}(\mathtt{x = 1}) = 1$, but addition is not always so straightforward: The predicate $(\mathbb{P}(\mathtt{x} = 1) = \frac{1}{2}) + (\mathbb{P}(y = 2) = \frac{1}{2})$ means that the state can be split into two parts. One part where x is one with probability $\frac{1}{2}$ and one part where y is two with probability $\frac{1}{2}$. This predicate is satisfied by $\frac{1}{2}\langle x = 1, y = 1\rangle + \frac{1}{2}\langle x = 2, y = 2\rangle$ and also by $1\langle x = 1, y = 2\rangle = \frac{1}{2}\langle x = 1, y = 2\rangle + \frac{1}{2}\langle x = 1, y = 2\rangle$. The state $\frac{1}{2}\langle x = 1, y = 2\rangle + \frac{1}{2}\langle x = 3, y = 3\rangle$ does not satisfy the predicate.

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 $p \oplus_\rho p$ does not imply p. Take for example $(\mathbb{P}(x = 1) = 1) \vee (\mathbb{P}(x = 2) = 1)$ for p. Then the state $\frac{1}{2}\langle x = 1\rangle + \frac{1}{2}\langle x = 2\rangle$ satisfies the predicate $p \oplus_{\frac{1}{2}} p$ but not the predicate p. Other examples are $p = \exists i : q$ and $p = \forall i : (q \vee q')$ for most predicates q and q'.

That $p \oplus_\rho p$ is not equivalent with p is not a short coming of the predicates, it is a general phenomenon when mixing nondeterminism and probability (see e.g. [14]). A predicate has an intrinsic nondeterminism, as it describes a set of states which satisfy a property. In the combination $p \oplus_\rho p$ selection of an element from the set of states that satisfy p acts as a nondeterministic choice. Different elements can be selected for the left and right side, possibly resulting in a combination which no longer satisfies p.

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

$$\models\ \{\, p \,\}\ s\ \{\, q \,\} \quad \text{if} \quad \forall \theta \in \Theta, J \in \mathcal{J} : (\theta, J) \models p \Rightarrow (\mathcal{D}(s)(\theta), J) \models q.$$

For example $\models\ \{\, p \,\}\ \texttt{skip}\ \{\, p \,\}$ and $\models\ \{\, \mathbb{P}(x = 0) = 1 \,\}\ x \texttt{ := } x + 1\ \{\, \mathbb{P}(x = 1) = 1 \,\}$. Also, we have $\models\ \{\, i = 5 \,\}\ s\ \{\, i = 5 \,\}$ for any program s, as i is a logical variable remaining uneffected by any program s.

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

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

$$\{\, p[x/e] \,\}\ x \texttt{ := } e\ \{\, p \,\} \quad \text{(Assign)} \qquad \frac{\{\, p \,\}\ s\ \{\, q \,\} \quad \{\, p \,\}\ s'\ \{\, q' \,\}}{\{\, p \,\}\ s \oplus_\rho s'\ \{\, q \oplus_\rho q' \,\}} \ \text{(Prob)}$$

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

$$\frac{\{\, p \,\}\ s\ \{\, q \,\} \quad j \text{ not free in } q}{\{\, \exists j : p \,\}\ s\ \{\, q \,\}} \ \text{(Exists)} \qquad \frac{\{\, p \,\}\ s\ \{\, q \,\} \quad \{\, p' \,\}\ s\ \{\, q \,\}}{\{\, p \vee p' \,\}\ s\ \{\, q \,\}} \ \text{(Or)}$$

$$\frac{\{\, p \,\}\ s\ \{\, q \,\} \quad j \text{ not free in } p}{\{\, p \,\}\ s\ \{\, \forall j : q \,\}} \ \text{(Forall)} \qquad \frac{\{\, p \,\}\ s\ \{\, q \,\} \quad \{\, p \,\}\ s\ \{\, q' \,\}}{\{\, p \,\}\ s\ \{\, q \wedge q' \,\}} \ \text{(And)}$$

$$\frac{\{\, p \,\}\ s\ \{\, q \,\}}{\{\, r \cdot p \,\}\ s\ \{\, r \cdot q \,\}} \ \text{(Lin }\cdot\text{)} \qquad \frac{\{\, p \,\}\ s\ \{\, q \,\} \quad \{\, p' \,\}\ s\ \{\, q' \,\}}{\{\, p + p' \,\}\ s\ \{\, q + q' \,\}} \ \text{(Lin +)}$$

In section 7 an example of a proof tree in the proof system *pH* is given.

The rules (Skip), (Assign), (Seq) and (Cons) are as for standard Hoare logic but now dealing with probabilistic predicates. The rule (If) has changed and the rules (Prob), (Or), (And), (Exists), (Forall), (Lin +) and (Lin $\cdot$) are new.

For a predicate 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 x `:=` e exactly when p with e substituted for x holds before the assignment, as the effect of the assignment is exactly replacing x with the value of e. The rule (Seq) states that p is a sufficient precondition 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 precondition may be strengthened and the postcondition may be weakened. For this any tautologie $p' \Rightarrow p$ is implicitly assumed as an axiom in the proof system. Note that the completeness result presented in the next section relies on this assumption. (See [8, 7] for a further discussion of this topic.) One would like to have a derivation system for probabilistic predicates to obtain these tautologies. Several rules that could be included in such a system are clear. A complete derivation system, however, is not available and may not even exist.

The rule (Prob) states that the result of executing $s \oplus_\rho s'$ is obtained by combining the results obtained by executing s and s' with the appropriate probabilities. The necessity for the (Or), (And), (Exists), (Forall) and (Lin) rules becomes clear when one recalls that $p \oplus_\rho p$ does not imply p. Proving correctness of e.g. $\{\, p \vee q \,\}$ $\texttt{skip} \oplus_\rho \texttt{skip}$ $\{\, p \vee q \,\}$ is, in general, not possible without the (Or)-rule. Similar examples show the need for the other rules. Note the similarity with the natural deduction rules for $\vee$ and $\exists$ elimination and $\wedge$ 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 may not be 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.

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 proof system is sound, i.e. only valid Hoare triples can be derived from *pH*.

Theorem 10 *The proof system pH is sound, 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* $(\theta, J) \models p$ *and* $\vdash \{\, p \,\}\; s \;\{\, q \,\}$ *then* $(\mathcal{D}(s)(\theta), J) \models q$*, for all predicates p, q, states θ, and interpretations J. This is shown by induction on the depth of the proof tree for* $\{\, p \,\}\; s \;\{\, q \,\}$*, by looking at the last rule used. We only present two typical cases:*

- *As in non-probabilistic Hoare logic one can show that* $(\sigma[\mathrm{x}/\mathcal{V}(\mathrm{e})(\sigma)], I) \models \mathrm{dp}$ *exactly when* $(\sigma, I) \models \mathrm{dp}[\mathrm{x}/\mathrm{e}]$. *By induction on the structure of the probabilistic predicate p this extends to* $(\theta[\mathrm{x}/\mathcal{V}(\mathrm{e})], J) \models p$ *exactly when* $(\theta, J) \models p[\mathrm{x}/\mathrm{e}]$. *Soundness of the (Assign) rule follows directly.*
- *If rule (Prob) is used to derive* $\vdash \{\, p \,\}\; s \oplus_\rho s' \;\{\, q \oplus_\rho q' \,\}$ *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* $(\theta, J) \models p$ *then* $(\mathcal{D}(s)(\theta), J) \models q$ *and* $(\mathcal{D}(s')(\theta), J) \models q'$. *But then* $(\mathcal{D}(s \oplus_\rho s')(\theta), J) = (\mathcal{D}(s)(\theta) \oplus_\rho \mathcal{D}(s')(\theta), J) \models q \oplus_\rho q'$. □

5. Weakest Preconditions and Completeness

Theorem 10 shows that the proof system pH is sound. This means that any Hoare triple derived with the proof system is valid. The next question is whether the proof system is complete: Can any Hoare triple that is valid be derived with the proof system? In this section it is shown that all valid Hoare triple with a slight restriction on the postcondition can be derived with the proof system. (This section deviates from the style of presentation so far in that it is technically involved. The remainder of the paper does not rely on this section.)

Completeness of the proof system is shown by using weakest preconditions. For a postcondition q and a statement s the weakest precondition p that yields a valid Hoare triple $\{\,p\,\}\;s\;\{\,q\,\}$ is found. Next it is shown that the Hoare triple $\{\,p\,\}\;s\;\{\,q\,\}$ can be derived in the proof system. Using the rule of consequence (Cons) this gives that any valid Hoare triple with q as a postcondition can be derived.

Before finding the weakest precondition, the form of the postconditions is restricted in two ways. The first restriction does not influence the expressiveness of the predicates. For the second restriction it is not clear whether this restricts the expressiveness of the predicates. The restricted probabilistic predicates used in the remainder of this section are defined by:

$$\begin{array}{rcl} p & ::= & \mathtt{true} \mid \mathtt{false} \mid c \mid \mathbb{P}(dp) = r \mid e_r = e_r \mid e_r < e_r \mid \ldots \mid p \wedge p \mid p \vee p \\ & & \mid \forall j : p \mid \exists j : p \mid \rho \cdot p \mid p + p \mid p \oplus_\rho p, \\ e_r & ::= & \rho \mid r \mid e_r + e_r \mid e_r - e_r \mid e_r * e_r \mid e_r * e_r \mid {e_r}^e, \end{array}$$

where c is an boolean condition on logical variables in $BC\langle LVar\rangle$ and e is an (integer) expression on logical variables in $Exp\langle LVar\rangle$.

Compared to the predicates used in the previous section, this definition restricts the real expressions by disallowing the use of $\mathbb{P}(dp)$ as part of a real expression. Instead $\mathbb{P}(dp)$ may only be used in the construct $\mathbb{P}(dp) = r$. The predicates are further restricted by disallowing the use of the operator $c?p$.

The restriction of the real expression implies that the comparison of unrestricted real expressions using $\mathbb{P}(dp)$ is no longer possible in the restricted probabilistic predicates. The construct $\mathbb{P}(dp)$ may only be used in the form $\mathbb{P}(dp) = r$ where r is a real valued logical variable. This restriction does not apply for real expression not containing the construct $\mathbb{P}(dp)$. It is easy to addapting a predicate so that it satisfy this restriction by introducing additional real valued variables (see the example below). The restriction of the real expressions, therefore, does not influence the expressiveness of the probabilistic predicates.

Example 11 *The predicate* $({\frac{1}{2}}^i + \mathbb{P}(\mathrm{x} = 1)) > \frac{1}{2}$ *is not of the restricted form used in this section, however the equivalent predicate* $\exists r : \mathbb{P}(\mathrm{x} = 1) = r \wedge ({\frac{1}{2}}^i + r) > \frac{1}{2}$ *is.*

The second restriction that applies to the predicates allowed as postconditions is that the operation $c?$ may not be used. It is not clear whether for every predicate of the form $c?p$, an equivalent predicate without use of the $c?$ operator exists. The omission of the $c?$ operator may restrict the postconditions which can be checked.

The operator $c?$ is more an auxiliary operator for use in the proof system than something that has a clear use in specification of properties. It is therefore not considered a severe restriction that the completeness result obtained in this section does not consider specifications of the postcondition that use the operator $c?$.

The restrictions on the probabilistic predicates allow for finding the weakest precondition. Formally a weakest precondition is defined as follows: A predicate p is a weakest precondition for postcondition q after statement s when

1. p is a sufficient precondition, i.e. $(\theta, J) \models p$ implies that $(\mathcal{D}(s)(\theta), J) \models q$, and
2. if p' satisfies condition 1 then p' implies p.

If a predicate p can be found which satisfies $\theta \models p \iff \mathcal{D}(s)(\theta) \models q$, then the predicate p must clearly be a weakest precondition, not only for this logic, but also for any logic that extends it. (In particular, for the logic using predicates that are not restricted.)

Below a predicate $\wp(s)(q)$ is defined for each statement s and predicate q. The proof system is shown to be general enough to be able to derive the Hoare triple $\{\, \wp(s)(q) \,\}\ s\ \{\, q \,\}$. By soundness of the proof system this also means that $\{\, \wp(s)(q) \,\}\ s\ \{\, q \,\}$ is a valid Hoare triple. Next it is shown that $\wp(s)(q)$ satisfies the stronger property mentioned above: $\theta \models \wp(s)(q) \iff \mathcal{D}(s)(\theta) \models q$.

Definition 13 *For a statement s and predicate q, the predicate $\wp(s)(q)$ is defined by induction on the structure of the statement s as follows:*

$$\begin{array}{rcl} \wp(s)(q) &=& q, \\ \wp(x \mathrel{\texttt{:=}} e)(q) &=& q[x/e], \\ \wp(s\,;\,s')(q) &=& \wp(s)(\wp(s')(q)) . \end{array}$$

For the statements $s \oplus_\rho s'$ and `if` c `then` s `else` s' `fi` *subinduction on the structure of the predicate q is used. For a basic predicate of the from $\mathbb{P}(dp) = r$, $\wp$ is given by:*

$$\begin{array}{l} \wp(s \oplus_\rho s')(\mathbb{P}(dp) = r) = \\ \quad \exists r_1, r_2 : (\rho * r_1 + (1-\rho) * r_2 = r) \wedge \wp(s)(\mathbb{P}(dp) = r_1) \wedge \wp(s')(\mathbb{P}(dp) = r_2), \\ \wp(\texttt{if}\ c\ \texttt{then}\ s\ \texttt{else}\ s'\ \texttt{fi})(\mathbb{P}(dp) = r) = \\ \quad \exists r_1, r_2 : (r_1 + r_2 = r) \wedge \\ \qquad ((\wp(s)(\mathbb{P}(dp) = r_1) \wedge \mathbb{P}(\neg c) = 0) + (\wp(s')(\mathbb{P}(dp) = r_2) \wedge \mathbb{P}(c) = 0)). \end{array}$$

For basic predicates which do not use the $\mathbb{P}(dp)$ construct and for combined predicates, $\wp$ is given by:

$$\begin{array}{rcl} \wp(s)(e_r = e_r) &=& e_r = e_r \\ \wp(s)(e_r < e_r) &=& e_r < e_r \\ \wp(s)(q\ op\ q') &=& \wp(s)(q)\ op\ \wp(s)(q') \qquad op \in \{\, \wedge, \vee, +, \oplus_{\rho'} \,\}, \\ \wp(s)(op\ q) &=& op\ \wp(s)(q) \qquad op \in \{\, \exists j, \forall j, \rho\, \cdot \,\}, \end{array}$$

where s is either a probabilistic choice or a conditional choice.

The predicate $\wp(s)(q)$ is meant to be the weakest precondition for postcondition q after statement s. For skip, the weakest precondition is the postcondition itself. The weakest precondition for assignment can be found by substituting the expression for the variable in the postcondition. To find the weakest precondition for a sequential composition $s\ ;\ s'$, the weakest precondition for s' is found and used as postcondition to be satisfied after s.

The weakest precondition for the probabilistic choice is harder to find. The rule (Prob) does not give a clear precondition for every postcondition, only for postconditions which are of the form $q \oplus_\rho q'$ and even for this postcondition, $\wp(s)(q) \wedge \wp(s')(q')$ is a precondition for $s \oplus_\rho s'$, but not the weakest precondition. A different approach is needed. Instead of trying to give the weakest precondition for general postconditions directly, the weakest precondition is built by combining the weakest preconditions for the parts of the predicate.

For a simple predicate of the form $\mathbb{P}(dp) = r$ the weakest precondition can be found. If the chance that dp holds must be r after executing $q \oplus_\rho q'$, then combining the chance that dp holds after executing s, say r_1, and the chance that dp holds after executing s', say r_2, should yield exactly r, thus $r = \rho * r_1 + (1-\rho) * r_2$. Basic predicates that do not use the $\mathbb{P}(dp)$ construct are not effected by the execution of a program; the weakest precondition is the predicate itself. The weakest precondition for combined predicates can be found by combining the weakest preconditions for the basic predicates.

To find the weakest precondition for postcondition $\mathbb{P}(dp) = r$ after a conditional choice, one can reason as follows: To execute $\texttt{if}\ c\ \texttt{then}\ s\ \texttt{else}\ s'\ \texttt{fi}$ the state is split into a part satisfying c and a part satisfying $\neg c$. That a state can be split into two parts can be described in a predicate by using the operator $+$. A predicate $(p \wedge \mathbb{P}(\neg c) = 0) + (q \wedge \mathbb{P}(c) = 0)$ holds in a state θ exactly when p holds in $c?\theta$ and q holds in $\neg c?\theta$. Using this similar reasoning as for the probabilistic choice leads to the weakest precondition as given in definition 13.

Example 12 *We have that* $\wp(\mathrm{x} := \mathrm{x} + 1)(\mathbb{P}(\mathrm{x} = 3) = r_1)$ *is equal to* $\mathbb{P}(\mathrm{x} + 1 = 3) = r_1$ *which is equivalent with* $\mathbb{P}(\mathrm{x} = 2) = r_1$ *so,*

$$\begin{aligned}
&\wp(\mathrm{x} := \mathrm{x} + 1 \oplus_\rho \mathrm{x} := \mathrm{x} + 2)(\mathbb{P}(\mathrm{x} = 3) = \tfrac{1}{2}) \quad = \\
&\qquad \exists r_1, r_2 : (\rho * r_1 + (1-\rho) * r_2 = \tfrac{1}{2}) \wedge \wp(\mathrm{x} := \mathrm{x} + 1)(\mathbb{P}(\mathrm{x} = 3) = r_1) \wedge \\
&\qquad\qquad \wp(\mathrm{x} := \mathrm{x} + 2)(\mathbb{P}(\mathrm{x} = 3) = r_2) \quad = \\
&\qquad \exists r_1, r_2 : (\rho * r_1 + (1-\rho) * r_2 = \tfrac{1}{2}) \wedge \mathbb{P}(\mathrm{x} = 2) = r_1 \wedge \mathbb{P}(\mathrm{x} = 1) = r_2
\end{aligned}$$

This predicate can be simplified to $(\rho * \mathbb{P}(\mathrm{x} = 2) + (1-\rho) * \mathbb{P}(\mathrm{x} = 1)) = \frac{1}{2}$. *For the simplified predicate the restrictions imposed in this section are no longer satisfied so if further calculation of weakest preconditions is required, as e.g. for the sequential composition* $s\ ;\ (\mathrm{x} := \mathrm{x} + 1 \oplus_\rho \mathrm{x} := \mathrm{x} + 2)$, *the simplification cannot be used.*

The first part in proving that $\wp$ does indeed give the weakest precondition is to shown that the postcondition q can be derived from $\wp(s)(q)$ by the proof system pH. By soundness of the of the proof system (theorem 10) this gives that $\wp(s)(q)$ is a sufficient precondition.

Lemma 2 *For any statement* s *and predicate* q *(of the restricted form used in this section),* $\vdash \{\, \wp(s)(q) \,\}\ s\ \{\, q \,\}$.

Proof. *As with the definition of $\wp$ this proof uses induction on the structure of the statement s and subinduction on the structure of the predicate p if the statement is a probabilistic choice or conditional choice. Each case uses the corresponding rule from the proof system pH. Only the case for probabilistic choice with the predicate* $\mathbb{P}(dp) = r$ *is given.*

- *Using the induction assumption* $\vdash \{\, \wp(s)(p) \,\}\ s\ \{\, p \,\}$ *for* $p = (\mathbb{P}(dp) = r_1)$ *and* $\vdash \{\, \wp(s')(p) \,\}\ s'\ \{\, p \,\}$ *for* $p = (\mathbb{P}(dp) = r_2)$ *and rules (Cons) and (Prob) gives that* $\vdash \{\, \wp(s)(\mathbb{P}(dp) = r_1) \wedge \wp(s')(\mathbb{P}(dp) = r_2) \,\}\ s \oplus_\rho s'\ \{\, (\mathbb{P}(dp) = r_1) \oplus_\rho (\mathbb{P}(dp) = r_2) \,\}$. *The postcondition implies* $\mathbb{P}(dp) = \rho * r_1 + (1-\rho) * r_2$.
  As $c_r \oplus_\rho c_r$ *is equivalent with* c_r *for any condition that does not use the* $\mathbb{P}(dp)$ *construct, we have* $\vdash \{\, r = \rho * r_1 + (1-\rho) * r_2 \,\}\ s \oplus_\rho s'\ \{\, r = \rho * r_1 + (1-\rho) * r_2 \,\}$. *Using rules (And) and (Cons) gives* $\vdash \{\, \wp(s)(\mathbb{P}(dp) = r_1) \wedge \wp(s')(\mathbb{P}(dp) = r_2) \wedge r = \rho * r_1 + (1-\rho) * r_2 \,\}\ s \oplus_\rho s'\ \{\, \mathbb{P}(dp) = r \,\}$. *Applying rule (Exists) for* r_1 *and* r_2 *yields the desired result.* □

Having shown that $\wp$ gives a sufficient precondition, it remains to be shown that $\wp$ gives the weakest precondition. To show that $\wp$ yields the weakest precondition the following stronger property is shown: For any statement s and predicate q, $(\mathcal{D}(s)(\theta), J) \models q \iff (\theta, J) \models \wp(s)(q)$. The implication $(\theta, J) \models \wp(s)(q) \Rightarrow (\mathcal{D}(s)(\theta), J) \models q$ is direct from $\models \{\, \wp(s)(q) \,\}\ s\ \{\, q \,\}$ shown above. The following lemma shows the reverse implication.

Lemma 3 *For any statement* $s \in \mathcal{L}_{\mathrm{pif}}$ *and predicate* q *of the restricted form used in this section,* $(\mathcal{D}(s)(\theta), J) \models q$ *implies* $(\theta, J) \models \wp(s)(q)$.

Proof. *The proof again uses induction on the structure of s and subinduction on the structure of q if s is a probabilistic choice or conditional choice. Only the case for probabilistic choice with the predicate* $\mathbb{P}(dp) = r$ *is given.*

- *Assume that* $(\mathcal{D}(s \oplus_\rho s')(\theta), J) \models \mathbb{P}(dp) = r$. *By writing out the definition, it is easy to check that this is exactly the case when there exist real numbers* ρ_1 *and* ρ_2 *such that:* $\rho\rho_1 + (1-\rho)\rho_2$ *is equal to* $J(r)$, $(\mathcal{D}(s)(\theta), J[r_1/\rho_1]) \models \mathbb{P}(dp) = r_1$ *and* $(\mathcal{D}(s')(\theta), J[r_2/\rho_2]) \models \mathbb{P}(dp) = r_2$. *By induction this gives that* $(\theta, J[r_1/\rho_1]) \models \wp(s)(\mathbb{P}(dp) = r_1)$ *and* $(\theta, J[r_2/\rho_2]) \models \wp(s')(\mathbb{P}(dp) = r_2)$. *Combining this with* $\rho\rho_1 + (1-\rho)\rho_2 = J(r)$, *immediately gives* $(\theta, J) \models \exists r_1, r_2 : \rho * r_1 + (1-\rho) * r_2 = r \wedge \wp(s)(\mathbb{P}(dp) = r_1) \wedge \wp(s')(\mathbb{P}(dp) = r_2)$. □

This lemma, combined with lemma 2 above give all the ingredients needed for the completeness result.

Theorem 13 *Let s be any statement in* $\mathcal{L}_{\mathrm{pif}}$, *p any predicate and q a predicate that does not contain the* $c?$ *operator then*

$$\models \{\, p \,\}\ s\ \{\, q \,\} \text{ if and only if } \vdash \{\, p \,\}\ s\ \{\, q \,\}\ .$$

Proof. *Soundness of the proof system (Theorem 10) already gives the 'if' part. For the 'only if' part assume that* $\models \{\, p \,\}\ s\ \{\, q \,\}$. *For a predicate q which does not contain any* $c?$ *operator, an equivalent predicate* q' *of the restricted form used in this section can be found. Lemma 2 shows that* $\vdash \{\, \wp(s)(q') \,\}\ s\ \{\, q' \,\}$. *It is sufficient to show that p implies* $\wp(s)(q')$ *because then* $\vdash \{\, p \,\}\ s\ \{\, q \,\}$ *by rule (Cons).*

For any state θ satisfying p i.e. $(\theta, J) \models p$, $\mathcal{D}(s)(\theta)$ satisfies q' because $\models \{\, p \,\}\ s\ \{\, q \,\}$ and because q' is equivalent with q. But then by lemma 3 $\theta \models \wp(s)(q')$. So any state satisfying p also satisfies $\wp(s)(q')$, i.e. p implies $\wp(s)(q')$. $\square$

6. Extending $\mathcal{L}_{\mathrm{pif}}$: Adding Iteration

The language $\mathcal{L}_{\mathrm{pif}}$, discussed above, lacks a construct for iteration. In this section we extend $\mathcal{L}_{\mathrm{pif}}$, obtaining a new language $\mathcal{L}_{\mathrm{pw}}$ in which a `while`-construct is present. We add a proofrule for `while` to our Hoare logic and establish its soundness. In contrast to the setting of the language $\mathcal{L}_{\mathrm{pif}}$, the completeness of the Hoare logic for $\mathcal{L}_{\mathrm{pw}}$ is an open issue. In section 7 the usage of the proofrule is illustrated in the context of the example of an erratic sequence summer and of geometrically distributed program variable.

The syntax of the language $\mathcal{L}_{\mathrm{pw}}$ is a straightforward extension of the syntax of the language $\mathcal{L}_{\mathrm{pif}}$. The statements remain the same except for the addition of a clause for the `while` construct.

Definition 14 *Let PVar be a set of program variables and let* x *range over PVar. The statements in $\mathcal{L}_{\mathrm{pw}}$, ranged over by s, are given by:*

$$s ::= \mathtt{skip} \mid \mathrm{x} \mathbin{\mathtt{:=}} \mathrm{e} \mid s \mathbin{\mathtt{;}} s \mid s \oplus_\rho s \mid \mathtt{if}\ c\ \mathtt{then}\ s\ \mathtt{else}\ s\ \mathtt{fi} \mid \mathtt{while}\ c\ \mathtt{do}\ s\ \mathtt{od}\,,$$

with $\mathrm{e} \in \mathit{Exp}\langle \mathit{PVar}\rangle$, $c \in \mathit{BC}\langle \mathit{PVar}\rangle$ *and* $\rho \in (0,1)$.

The statement `while` c `do` s `od` is interpreted as "repeatedly execute s as long as the condition c holds". The `while` construct introduces the possibility of arbitrarily many repetitions as well as the possibility of non-termination; the program `while true do` s `od`, for example, will never finish.

The denotational semantics $\mathcal{D}$ for $\mathcal{L}_{\mathrm{pw}}$ gives, for each statement s, and state θ, the state $\mathcal{D}(s)(\theta)$ resulting from executing s starting in state θ. The denotational semantics is also an extension of the denotational semantics for the language $\mathcal{L}_{\mathrm{pif}}$ given in section 3. A clause is added for statements built with `while`. This clause uses a least fixed point construction. The cpo structure on Θ guarantees that this least fixed point exists.

Definition 15

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

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

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

$$\begin{array}{rcl}
\mathcal{D}(\mathtt{skip})(\theta) & = & \theta \\
\mathcal{D}(\mathrm{x} \mathbin{\mathtt{:=}} \mathrm{e})(\theta) & = & \theta[\mathrm{x}/\mathcal{V}(\mathrm{e})] \\
\mathcal{D}(s \mathbin{\mathtt{;}} s'\,)(\theta) & = & \mathcal{D}(s'\,)(\mathcal{D}(s)(\theta)) \\
\mathcal{D}(s \oplus_\rho s'\,)(\theta) & = & \mathcal{D}(s)(\theta) \oplus_\rho \mathcal{D}(s'\,)(\theta) \\
\mathcal{D}(\mathtt{if}\ c\ \mathtt{then}\ s\ \mathtt{else}\ s'\ \mathtt{fi})(\theta) & = & \mathcal{D}(s)(c?\theta) + \mathcal{D}(s'\,)(\neg c?\theta) \\
\mathcal{D}(\mathtt{while}\ c\ \mathtt{do}\ s\ \mathtt{od}) & = & \textit{the least fixed point of } \Psi_{\langle c,s\rangle}\,.
\end{array}$$

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

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

The total probability of $\mathcal{D}(\texttt{while}\ c\ \texttt{do}\ s\ \texttt{od})(\theta)$ may be less than that of θ. The 'missing' probability is caused by non-termination; if a possible calculation does not terminate, it does not contribute to the probabilities in the final state.

The semantics of the while construction is given as the least fixed point of $\Psi_{\langle c,s\rangle}$. The set of probabilistic states Θ are the distributions over the set of deterministic states and as such form a cpo. With that it is not difficult to check that $\Psi_{\langle c,s\rangle}$ indeed has a least fixed point. A more constructive description of the state $\mathcal{D}(\texttt{while}\ c\ \texttt{do}\ s\ \texttt{od})(\theta)$, however, is also useful. Below the least fixed point of $\Psi_{\langle c,s\rangle}$ is constructed by giving a sequence which converges to $\Psi_{\langle c,s\rangle}$.

Definition 16 *For a statement s define $s^0 = \texttt{skip}$ 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*

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

The function $\mathrm{if}^n_{\langle c,s\rangle}$ 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 $\mathcal{D}(\texttt{while}\ c\ \texttt{do}\ s\ \texttt{od})$.

Lemma 4 *The least fixed point of $\Psi_{\langle c,s\rangle}$ is given by $L_{\langle c,s\rangle}$.*

Proof. *It is easy to check that $L_{\langle c,s\rangle}$ is a fixed point. Any fixed point, say ξ, not larger or equal to $L_{\langle c,s\rangle}$ would have to be smaller on a certain θ, $\xi(\theta) < L_{\langle c,s\rangle}(\theta)$. But then $\xi(\theta) < if^n_{\langle c,s\rangle}(\theta)$ for some n. By using that ξ is a fixed point and working out $\Psi^n_{\langle c,s\rangle}(\xi)(\theta)$ one gets: $\xi(\theta) = \Psi^n_{\langle c,s\rangle}(\xi)(\theta) = if^n_{\langle c,s\rangle}(\theta) + \xi(\ldots) \geq if^n_{\langle c,s\rangle}(\theta)$ which gives a contradiction.* $\square$

To reason about programs in $\mathcal{L}_{\mathrm{pw}}$ an extension of the proof system *pH* given in section 4 is used. The semantics of $\mathcal{L}_{\mathrm{pw}}$ is a conservative extension of the semantics of $\mathcal{L}_{\mathrm{pif}}$. It is, therefore, not surprising that the existing rules of the proof system *pH* still remain valid. To deal with statements containing the while construct, the following rule is added.

$$
\frac{p \text{ invariant for } \langle c, s\rangle}{\{\,p\,\}\ \texttt{while}\ c\ \texttt{do}\ s\ \texttt{od}\ \{\,p \wedge \mathbb{P}(c) = 0\,\}}\ \text{(While)}
$$

The extended system, including the rule (While), is also referred to as *pH*. The rule (While) has the same form as the rule used in (non-probabilistic) Hoare logic, however the notion of invariant is more complicated.

To use the (While) rule, an invariant p should be found. For an assertion p to be an invariant, it should satisfy $\{p\}$ `if` c `then` s `else skip fi` $\{p\}$. As in Hoare logic, this condition is sufficient to obtain partial correctness. If the program s is terminating and $\{p\}\ s\ \{q\}$ can be derived from pH, then $\models\ \{p\}\ s\ \{q\}$ holds. A probabilistic program is terminating, if the program terminates for all possible outcomes of the probabilistic choices. Formally, a while loop is said to terminate for start state θ if `while` c `do` s `od`$(\theta) = \mathrm{if}^n_{\langle c,s\rangle}(\theta)$ for some n. The loop is said to be terminating if it terminates for all state states θ. A statement s is said to be terminating if all while loops in s are terminating.

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. To derive valid Hoare triples for programs that need not terminate, a form of unconditional correctness is required. This requires somehow adding termination conditions to the rules. To obtain this unconditional correctness the notion of invariant is strengthened by imposing the extra condition of $\langle c, s\rangle$-closedness.

Definition 17

(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.*

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

(b) A sequence of states $(\theta_n)_n$ is called a $\langle c, s\rangle$-sequence if $(\neg c?\theta_n)_n$ 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 least upper bound within p.

(d) A predicate p is called an invariant for $\langle c, s\rangle$ when p is $\langle c, s\rangle$-closed and $\{p\}$ `if` c `then` s `else skip fi` $\{p\}$.

Note that, for a loop `while` c `do` s `od` that is terminating, 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. In section 7 two examples of programs using the while construct are given. The first program is terminating and no closedness conditions need to be checked. The second program shows a situation where the closedness condition is essential to guarantee soundness.

With addition of the rule (While) the proof system still remains sound, i.e. only valid Hoare triples can be derived from pH.

Theorem 14 *The proof system pH is sound, 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 proof tree for $\{p\}\ s\ \{q\}$, by looking* *at the last rule used. The only new case, compared to Theorem 10, is that of rule (While).*

- *Assuming rule (While) is used with statement s, condition c and invariant p then clearly $\mathcal{D}(\texttt{while}\ c\ \texttt{do}\ s\ \texttt{od})(\theta) \models \mathbb{P}(c) = 0$ holds. Also, by induction $\models \{p\}\ \texttt{if}\ c\ \texttt{then}\ s\ \texttt{else skip fi}\ \{p\}$ holds, which 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 $\mathcal{D}(\texttt{while}\ c\ \texttt{do}\ s\ \texttt{od})(\theta) = L_{\langle c,s\rangle} \models p$.* □

In this section a way has been given to check properties of programs containing `while` loops by using invariants. The following section gives two examples of this technique. In these examples the invariant is clear but as for non-probabilistic programs, and maybe even more so, finding an invariant for a loop can, in general, be far from trivial. It is not clear whether the probabistic predicates are sufficiently expressive to be able to always specify the invariant for a given `while` loop. Already in the non-probabilistic setting , see e.g. [7], the issue of the finding a weakest precondition formula for the `while`-construct and showing completeness of the extended proof system is significantly more involved. Although some preliminary results are available this issue remains untouched in this paper. Further study of the expressiveness issue meantioned above is required.

7. 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 (Cons) 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.

$$\dfrac{\dfrac{\dfrac{}{\{[x+1=2]\}\ \texttt{x := x + 1}\ \{[x=2]\}}\text{(Assign)}}{\{[x=1]\}\ \texttt{x := x + 1}\ \{[x=2]\}}\text{(Cons)} \qquad \dfrac{\dfrac{}{\{[x+2=3]\}\ \texttt{x := x + 2}\ \{[x=3]\}}\text{(Assign)}}{\{[x=1]\}\ \texttt{x := x + 2}\ \{[x=3]\}}\text{(Cons)}}{\dfrac{\{[x=1]\}\ \texttt{x := x + 1} \oplus_{\frac{1}{2}} \texttt{x := x + 2}\ \{[x=2] \oplus_{\frac{1}{2}} [x := 3]\}}{\{[x=1]\}\ \texttt{x := x + 1} \oplus_{\frac{1}{2}} \texttt{x := x + 2}\ \{\mathbb{P}(x=2) = \frac{1}{2} \wedge \mathbb{P}(x=3) = \frac{1}{2}\}}\text{(Cons)}}\text{(Prob)}$$

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. The shorthand $\exists i \le e : p$ is used for $\exists i : (i \le e) \wedge p$. Similarly $\forall i \le e : p$ is short for $\forall i : (i > e) \vee p$.

$\texttt{Int}\ ss[1\ldots N],\ k,\ t;$

$\{\mathbb{P}(\texttt{true}) = 1\} \Rightarrow \{\mathbb{P}(0 = 0 \wedge 1 = 1) = 1\}$

t `:= 0;` $\quad k$ `:= 1;`

$\{\mathbb{P}(t = 0, k = 1) = 1\} \Rightarrow$

$\{\mathbb{P}(k = N+1 \wedge t = \sum_{i=1}^{N} ss[i]) \ge \rho^N \ \vee\ \exists n \le N : \mathbb{P}(k = n, t = \sum_{i=1}^{k-1} ss[i]) \ge \rho^{n-1}\}$

$\texttt{while } (k \leq N) \texttt{ do}$

$\{ \exists n \leq N : \mathbb{P}(k = n, t = \sum_{i=1}^{\mathrm{k}-1} ss[i]) \geq \rho^{n-1} \} \Rightarrow$

$\{ \exists n \leq N : \mathbb{P}(k = n, t + ss[k] = \sum_{i=1}^{\mathrm{k}} ss[i]) \geq \rho^{n-1} \}$

$t \texttt{ := } t + ss[k] \oplus_\rho \texttt{skip};$

$\{ \exists n \leq N : \mathbb{P}(k = n, t + ss[k] = \sum_{i=1}^{\mathrm{k}} ss[i]) \geq \rho^{n-1} \oplus_\rho \texttt{true} \} \Rightarrow$

$\{ \exists m \leq N+1 : \mathbb{P}(k+1 = m, t = \sum_{i=1}^{\mathrm{k}-1} ss[i]) \geq \rho^{m-1} \}$

$k := k + 1$

$\{ \exists m \leq N+1 : \mathbb{P}(k = m, t = \sum_{i=1}^{\mathrm{k}-1} ss[i]) \geq \rho^{m-1} \}$

$\texttt{od}$

$\{ \mathbb{P}(k \leq N) = 0 \wedge \exists n \leq N+1 : \mathbb{P}(k = n, t = \sum_{i=1}^{\mathrm{k}-1} ss[i]) \geq \rho^{n-1} \} \Rightarrow$

$\{ \mathbb{P}(t = \sum_{i=1}^{N} ss[i]) \geq \rho^N \}$

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 shorthands are used.

$$\begin{array}{rcl} p & = & q_\infty \vee \exists i : q \\ q & = & \mathbb{P}(x = i, done = false) = \frac{1}{2}^i \wedge \forall j \in \{1, \ldots, i\} : \mathbb{P}(x = j, done = true) = \frac{1}{2}^j, \\ q_\infty & = & \forall j > 0 : \mathbb{P}(x = j, done = true) = \frac{1}{2}^j . \end{array}$$

Then, assuming p is an invariant:

$\{ \mathbb{P}(\texttt{true}) = 1 \}$

$\texttt{Bool } done \texttt{ := false;} \quad \texttt{Int } x \texttt{ := 0;}$

$\{ \mathbb{P}(x = 0, done = false) = 1 \} \Rightarrow \{ p \}$

$\texttt{while } \neg done \texttt{ do } x \texttt{ := } x + 1; done \texttt{ := true} \oplus_{\frac{1}{2}} \texttt{skip od}$

$\{ p \wedge \mathbb{P}(\neg done) = 0 \} \Rightarrow \{ \forall n > 0 : \mathbb{P}(x = n) = \frac{1}{2}^n \}$

To show that p is an invariant 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:

$\{ q \}$

$\texttt{while } \neg done \texttt{ do}$

$\{ (\neg done)?q \} \Rightarrow \{ \mathbb{P}(x = i, done = false) = \frac{1}{2}^i \}$

$x \texttt{ := } x + 1;$

$\{ \mathbb{P}(x = i+1, done = false) = \frac{1}{2}^i \}$

$done \texttt{ := true } \oplus_{\frac{1}{2}} \texttt{ skip}$

$\{ \mathbb{P}(x = i+1, done = false) = \frac{1}{2}^{i+1} \wedge \mathbb{P}(x = i+1, done = true) = \frac{1}{2}^{i+1} \}$

$\texttt{od}$

$\{ \mathbb{P}(x = i+1, done = false) = \frac{1}{2}^{i+1} \wedge \mathbb{P}(x = i+1, done = true) = \frac{1}{2}^{i+1} \wedge$

$\forall j \in \{1, \ldots, i\} : \mathbb{P}(x = j, done = true) = \frac{1}{2}^j \} \Rightarrow \{ q[i/i+1] \} \Rightarrow \{ p \}.$

The requirement that p is $\langle \neg done, x \texttt{ := } x + 1; done \texttt{ := true} \oplus_{\frac{1}{2}} \texttt{skip} \rangle$-closed is easy to check but requires the presence of the q_∞ term.

8. 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 either hold or not. Deterministic predicates can be extended to arithmetical functions yielding the probability that the predicate holds as done in e.g. [21] and [24]. This extension is incorporated by using the notation $\mathbb{P}(dp)$ to refer to 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 shown to be sound with respect to the semantics $\mathcal{D}$. For an unpublished 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. (A treatment of nondeterminism will be included in the PhD thesis of the first author [13].)

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. [4, 11]) 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 tool supported. Some work has been done to embed the logic in the proof verification system PVS. (See e.g. [18] 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 (Cons) rule. By modeling probabilistic states, PVS could perhaps also be used to verify the soundness of the logic, however it is expected that a substantial effort is required for the modeling of infinite sums as an important step towards achieving this goal.

9. References

1. L. de Alfaro. *Formal Verification of Probabilistic Systems.* PhD thesis, Stanford University, 1997.
2. N. Alon, J.H. Spencer, and P. Erdös. *The Probabilistic Method.* Series in Discrete Mathematics and Optimization. Wiley, 1992.
3. S. Andova. Process algebra with probabilistic choice. In J.-P. Katoen, editor, *Proceedings ARTS'99*, pages 111–129, Bamberg, 1999. LNCS 1601.
4. K.R. Apt and E.-R. Olderog. *Verification of Sequential and Concurrent Programs.* Texts and Monographs in Computer Science. Springer, 1991.
5. J.C.M. Baeten and J.A. Bergstra. Process algebra with partial choice. In B. Jonsson and J. Parrow, editors, *Proc. CONCUR'94*, pages 465–480. LNCS 836, 1994.
6. C. Baier, M.Z. Kwiatkowska, and G. Norman. Computing probability bounds for linear time formulas over concurrent Markov chains. In *Proc. PROBMIV'98.*

ENTCS 22, 1999.

7. J.W. de Bakker. *Mathematical Theory of Program Correctness.* Series in Computer Science. Prentice-Hall International, 1980.
8. S.A. Cook. Soundness and completeness of an axiom system for program verification. *SIAM Journal of Computing*, 7(1):70–90, 1978.
9. P.R. D'Argenio. *Algebras and Automata for Timed and Stochastic Systems.* PhD thesis, University of Twente, 1999.
10. P.R. D'Argenio, J.-P. Katoen, and E. Brinksma. A compositional approach to generalised semi-Markov processes. In *Proc. WODES'98,* Caligari. IEE, 1998.
11. N. Francez. *Program Verification.* Addison-Wesley, 1992.
12. R.J. van Glabbeek, S.A. Smolka, and B. Steffen. Reactive, generative and stratified models of probabilistic processes. *Information and Computation*, 121:59–80, 1995.
13. J.I. den Hartog. *Probabilistic Extensions of Semantical Models.* PhD thesis, Vrije Universiteit Amsterdam, 2002. To appear.
14. J.I. den Hartog and E.P. de Vink. Mixing up nondeterminism and probability: A preliminary report. In *Proc. PROBMIV'98.* ENTCS 22, 1999.
15. V. Hartonas-Garmhausen. *Probabilistic Symbolic Modelchecking with Engineering Models and Applications.* PhD thesis, Carnegie Mellon University, 1998.
16. H. Hermanns. *Interactive Markov Chains.* PhD thesis, University of Erlangen-Nurnberg, July 1998.
17. C.A.R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12:576–580, 1969.
18. J. Hooman. Program design in PVS. In *Proc. Workshop on Tool Support for System Development and Verification*, June 1996.
19. M. Huth and M.Z. Kwiatkowska. Quantitive analysis and model checking. In *Proc. LICS'97*, pages 111–122, Warzaw, 1997.
20. C. Jones. *Probabilistic Nondeterminism.* PhD thesis, ECS–LFCS–90–105, University of Edinburgh, 1990.
21. D. Kozen. A probabilistic PDL. *Journal of Computer and System Sciences*, 30:162–178, 1985.
22. K.G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94:1–28, 1991.
23. C. Morgan and A. McIver. pGCL: formal reasoning for random algorithms. *Proc. WOFACS 1998, Special Issue of the South African Computer Journal*, 22, 1999.
24. C. Morgan, A. McIver, and K. Seidel. Probabilistic predicate transformers. *ACM Transactions on Programming Languages and Systems*, 18:325–353, May 1996.
25. R. Motwani and P. Raghavan. *Randomized Algorithms.* CUP, 1995.
26. M. Nuñez, D. de Frutos, and L. Llana. Acceptance trees for probabilistic processes. In I. Lee and S.A. Smolka, editors, *Proc. CONCUR'95*, pages 249–263. LNCS 962, 1995.
27. A. Pnueli and L.D. Zuck. Probabilistic verification. *Information and Computation*, 103:1–29, 1993.
28. R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems.* PhD thesis, Massachusetts Institute of Technology, 1995.
29. 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.