Backwards Abstract Interpretation of Probabilistic Programs

David Monniaux

LIENS, 45 rue d'Ulm
75230 Paris cedex 5, France
http://www.di.ens.fr/~monniaux

1 Introduction

In industrial contexts, safety regulations often mandate upper bounds on the probabilities of failure. Now that embedded computers are part of many industrial environments, it is often needed to analyze programs with non-deterministic and probabilistic behavior. We propose a general abstract interpretation based method for the static analysis of programs using random generators or random inputs. Our method also allows "ordinary" non-deterministic inputs, not necessarily following a random distribution.

1.1 Our Approach

Our method is set in the general framework of abstract interpretation. We first introduce an adjoint semantics for probabilistic programs using "weight functions", basing ourselves on the standard semantics of probabilistic programs as linear operators on measures [8,9,12] (see §1.4 for an explanation on measures). Similarly as it has been done for the standard semantics [12], we introduce a notion of abstract interpretation on weight functions. We then propose a highly generic construction of abstract lattices, lifting an "ordinary" abstract lattice used for the analysis of non-probabilistic programs to one suitable for probabilistic programs.

As salient point of this method is that it starts from the description of an output event (for instance, an anomalous condition) and computes back a description of areas in the input domain describing their probability of making the behavior happen. This allows finding what parts of the input domain are more likely to elicit anomalous behavior, as an extension to probabilistic programs of ordinary backwards analysis.

We shall give all our examples using a simple imperative language extended with nondeterministic and probabilistic inputs, for the sake of simplicity. This by no means indicates our method is restricted to imperative languages. There has been much work done, for instance, on the analysis of complex imperative languages [1], and our method can be applied to lift it to probabilistic cases as well.

1.2 Comparison with Other Approaches

There has been several propositions of weakest precondition semantics and associated sets of rules, similar to Dijkstra's for non-probabilistic programs [7,11,14, 15]. However, these methods, while adequate for computer-aided program design or verification, cannot be automated easily.

Abstract interpretation has already been applied to probabilistic semantics [12]. However, the method that we describe here, while similar, considers different semantics, hence leads to different informations about the programs. In [12], following the standard semantics proposed by Kozen [8,9] and used in most analysis schemes, the semantics of a program is a function mapping an input probability measure onto an output probability measure, taking into account the random generators and random inputs happening in the meantime. The goal is to derive knowledge on the output from knowledge of the input. Here, we derive weights on the input from an area in the output. Another notion of forward probabilistic abstract interpretation has been proposed by Di Pierro and Wiklicky [4], but it is unclear how it can handle problems except in simple, discrete cases; furthermore, their model does not support nondeterminism.

Statistical sampling methods are already used to test software, and they were improved to allow for both nondeterministic and probabilistic behavior [13] in a mathematically sound fashion. However, when dealing with rare behavior, these methods are greatly improved using additional knowledge on the system allowing for stratified sampling or importance sampling [16, chap. 4]. The analysis we describe in this paper could be used to supply data for importance sampling, improving the speed of precision of the Monte-Carlo method of [13].

1.3 Nondeterminism and Probabilities

We shall make clear what we call "nondeterministic" and "probabilistic". A nondeterministic choice allows for several independent outcomes of the choice. A probabilistic choice also allows for several outcomes, but constrains the frequency of those outcomes. For instance, let us consider an input $x \in [0, 1]$ to the program. If it is nondeterministic, then the only thing we know is that it is in $[0, 1]$. Simply supposing it is probabilistic, without any additional knowledge, already establishes that this variable has numerous properties such as an average and a standard deviate, and implies statistical properties on successive uses of this input.

With respect to program semantics, purely probabilistic programs are to be treated much like nondeterministic nonprobabilistic ones [7], except that the values that are manipulated are (sub)probability measures on the set of program environments instead of program environments. A notion of nondeterministic, probabilistic programs arises when nondeterministic choice between several measures is allowed. Our analysis takes care of that most complex case.

1.4 Notations, Measures, and Integrals

Standard probability theory is based on **measures** [17,5]. A probability measure μ on a set X is a function that assigns to each event (subset of X) its probability.

For instance, the **uniform** probability measure on the segment $[0, 1]$ assigns to each segment $[a, b]$ the probability $b - a$. The **Dirac** measure δ_{x_0} assigns 1 to any event containing x_0 and 0 otherwise; it modelizes a value that is "known for sure" to be x_0. For technical reasons, not all subsets are necessarily measurable — this is not a problem in our case. A **measurable space** is the couple of a set and an associated set of events (measurable subsets).

A function f is said to be **measurable** if and only if for any measurable set X, $f^{-1}(X)$ is measurable; we shall often use the vector space $M(X, \mathbb{R})$ of measurable functions from a measurable space to $\mathbb{R}$ (the real field).

For technical reasons, we shall also use **signed measures** in this paper. Signed measures over a measurable set X, using the norm of the total variation [5] $\| \cdot \|$, constitute a vector space $\mathcal{M}(X)$. $\mathcal{M}_+(X)$ will denote the positive measures. We shall consider continuous linear operators [10] over such spaces. As an extension to the usual notion of the **adjoint** of a linear operator with respect to a hermitian form [10, VIII, §4], we use adjoints of linear operators with respect to a bilinear form.

We shall often use **integrals**, in the sense of Lebesgue integration [17]. $\int f \, \mathrm{d}\mu$ denotes the integral of the function f with respect to the measure μ. For instance, if the integration set is $\mathbb{R}$ and μ is the usual Lebesgue measure on the segment $[a, b]$ (the measure that assigns to each segment its length), $\int f \, \mathrm{d}\mu$ is the usual integral $\int_a^b f(x) \, \mathrm{d}x$; if the measure μ is the Dirac measure at x_0, then $\int f \, \mathrm{d}\mu$ is $f(x_0)$.

We shall often use the vector space $\mathcal{B}(X, \mathbb{R})$ of bounded measurable functions from X to $\mathbb{R}$, with the norm $\|f\|_\infty = \sup_{x \in X} |f(x)|$.

$\mathcal{L}(X, Y)$ is the vector space of **linear functions** from X to Y. The phrase "linear function" shall always be taken in its linear algebra sense.

2 Adjoint Semantics

In his seminal papers, Kozen proposed semantics of probabilistic programs as continuous linear operators on measure spaces. We shall see that operators representing the semantics of probabilistic programs have adjoints, in a certain sense that we shall define (§2.3). These adjoints are the basis of our analysis techniques; furthermore, their existence yields a proof of a theorem of Kozen's.

2.1 Intuition

Let us consider a very simple C program (Fig. 1) where `centered_uniform()` is a random generator returning a `double` uniformly distributed in $[-1, 1]$, independent of previous calls. We are interested in the relationship between the probability of executing `B` depending on the probability distribution generated in `x` by `A`. What we would therefore like is a (linear) function f mapping the probability measure μ generated at A onto the probability $f(\mu)$ that program point `B` is executed. It will be proved that there exists a "weight function" g such that $f(\mu) = \int g \, \mathrm{d}\mu$. We shall see how to compute such a g.

A plotting of g is featured in figure 2. Let us give a few examples of the use of this function:

```
double x, y;
... /* A */
y = centered_uniform()+centered_uniform();
x += y/2;
...
if (fabs(x) <= 1)
{
  ... /* B */
}
```

Fig. 1. A simple probabilistic program.

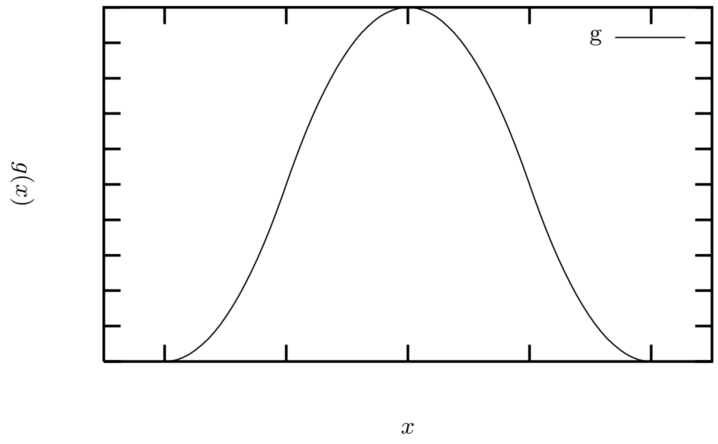

Fig. 2. Weight function g such that the probability of outcome of step B (see Fig. 1) given the probability measure μ at step A is $\int g\,\mathrm{d}\mu$. x is the value of variable x.

- The probability that B will happen if A drives x according to some uniform distribution in $[a,b]$ is $\displaystyle\int_a^b g(x)\,\mathrm{d}x$.
- The probability that B will happen if A sets x to a constant C is $g(C)$.

The set $g^{-1}(0)$ is the set of values at step A that have no chance of starting a program trace reaching step B. Please note that this is slightly different from the set of values that cannot start a program trace reaching step B. This is the difference between “impossible” and “happening with 0 probability”. For instance, if in the program of Fig 1 we put `x=2;` as statement A, then statement B is reachable; however $g(2) = 0$ thus there is zero probability that statement B is reached.

2.2 Summary of Semantics According to Kozen

The semantics of a probabilistic program c can be seen as a linear operator $[\![c]\!]_p$ mapping the input probability distribution (measure μ) onto an output

```
double x, y;
... /* A */
if (x+y >= -1)
{
  x += 2;
}
y = centered_uniform()+centered_uniform();
x += y/2;
...
if (fabs(x) <= 1)
{
  ... /* B */
}
```

Fig. 3. Another probabilistic program.

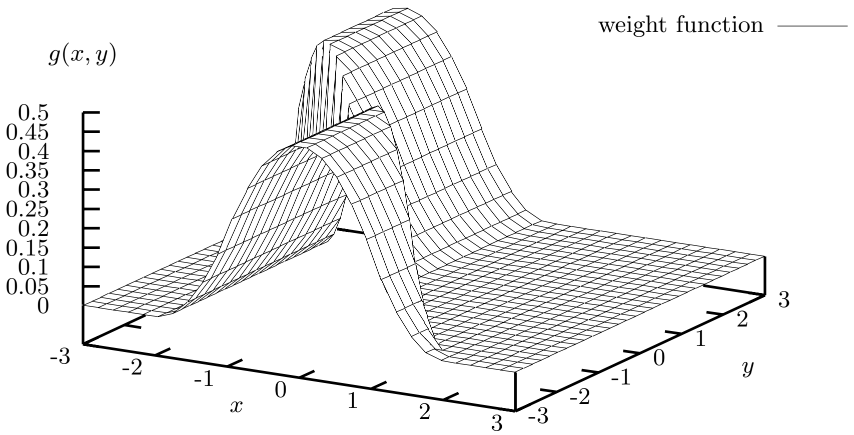

Fig. 4. Weight function g such that the probability of outcome of step `B` (see Fig. 3) given the probability measure μ at step `A` is $\int g \, \mathrm{d}\mu$. x and y are the respective values of variables `x` and `y`.

measure $[\![c]\!]_p.\mu$. Values given by random number generators can either be seen as successive reads from streams given as inputs or are handled internally in the semantics [8,9]; here we shall use the second approach, though both approaches are equivalent. We shall not discuss here the technical details necessary to ensure continuity of operators, convergences etc ... and we shall refer the reader to [12][extended version].

The semantics of a program c whose initial environment lies in the measurable set X and whose final environment lies in the measurable set Y shall be given as a linear operator (of norm less than 1 for the norm of total variation [5] on measures). If c contains no calls to random number generators, $[\![\mathtt{c}]\!]$ is just a measurable function.

We shall base ourselves on an ordinary denotational semantics: $[\![c]\!]$ is a function from set X to set Y if c has type $X \to Y$. For the sake of simplicity, we shall not deal with nontermination here so no $\perp$ value is needed. To make meaningful probabilities, X and Y are measurable sets (for instance, countable sets) and $[\![c]\!]$ is assumed to be a measurable function. These restrictions are of a technical nature and do not actually restrict the scope of the analysis in any way; the reader shall refer to [12] for more details.

Let us summarize the probabilistic semantics $[\![\mathtt{c}]\!]_p : \mathcal{L}(\mathcal{M}_+(X), \mathcal{M}_+(Y))$:

Elementary constructs (assignments etc...) get simple semantics: $[\![\mathtt{c}]\!]_p.\mu = \lambda X.\mu([\![\mathtt{c}]\!]^{-1}(X))$.

Random number generation. Let us suppose each invocation of `random` yields a value following distribution μ_R, each invocation being independent from another, and stores the value into a fresh variable. Then $[\![\mathtt{c}]\!]_p.\mu = \mu \otimes \mu_R$ where $\otimes$ is the product on measures.

Tests. Let us define $\phi_W(\mu) = \lambda X.\mu(X \cap W)$.
Then $[\![\mathtt{if}\ c\ \mathtt{then}\ e_1\ \mathtt{else}\ e_2]\!]_p(\mu) = [\![e_1]\!]_p \circ \phi_{[\![c]\!]}(\mu) + [\![e_2]\!]_p \circ \phi_{[\![c]\!]^C}(\mu)$ where X^C denotes the complement of the subset X.

Loops $[\![\mathtt{while}\ c\ \mathtt{do}\ e]\!]_p(\mu) = \sum_{n=0}^{\infty} \phi_{[\![c]\!]^C} \circ ([\![e]\!]_p \circ \phi_{[\![c]\!]})^n(\mu)$, the limit being taken set-wise [5, §III.10].

2.3 Adjoints and Pseudo-Adjoints

In this section, we shall recall the usual definition of an adjoint of a linear operator and give a definition of a pseudo-adjoint. We shall also give some easy properties, without proofs for the sake of brevity.

Let us consider two measurable sets (X, σ_X) and (Y, σ_Y). Let us first define, for f a measurable function and μ a measure, $\langle f, \mu \rangle = \int f \, \mathrm{d}\mu$.

Proposition 1. *Taking $f \in \mathcal{B}(X, \mathbb{R})$ and $\mu \in \mathcal{M}(X)$, this defines a continuous bilinear scalar form. Moreover, this form has the following properties:*

- *for all f and μ, $|\langle f, \mu \rangle| \leq \|f\|_\infty . \|\mu\|$;*
- *$\langle f, \cdot \rangle = \mu \mapsto \langle f, \mu \rangle$ has norm $\|f\|_\infty$;*
- *$\langle \cdot, \mu \rangle = f \mapsto \langle f, \mu \rangle$ has norm $\|\mu\|$.*

Corollary 1. *If $\langle f, \cdot \rangle = 0$ then $f = 0$. If $\langle \cdot, \mu \rangle = 0$ then $\mu = 0$.*

Let us consider a linear operator H from the signed measures on X to the signed measures on Y, and we can consider whether it admits an **adjoint operator** R:

$$\int (R.f) \, \mathrm{d}\mu = \int f \, \mathrm{d}(H.\mu) \tag{1}$$

or $\langle R.f, \mu \rangle = \langle f, H.\mu \rangle$.

Proposition 2. *If an operator has an adjoint, then this adjoint is unique.*

Proof. Follows from corollary 1.

Lemma 1. *If R is the adjoint of H:*

- *R is continuous if and only H is continuous;*
- *$\|R\| = \|H\|$.*

Corollary 2. *The operator mapping an operator onto its adjoint is therefore a linear isometry.*

The reason why we consider such adjoints is the following lemma:

Lemma 2. *If $H \in \mathcal{L}(\mathcal{M}(X), \mathcal{M}(Y))$ has an adjoint operator $R \in \mathcal{L}(\mathcal{B}(Y, \mathbb{R}), \mathcal{B}(X, \mathbb{R}))$ and H is zero on all the Dirac measures, then H is zero.*

The implications of this lemma on probabilistic program semantics is that if we can prove, which we shall do later, that linear operators representing program semantics have adjoints, then the semantics of two programs will be identical on all input measures if and only if they are identical on discrete measures.

Proposition 3. *In general, not all positive continuous linear operators on measures have adjoints in the above sense.*

For technical reasons, we shall also have to use a notion of pseudo-adjoint. Let H be a function from $\mathcal{M}(X)$ to $\mathcal{M}(Y)$. Let us suppose there exists a function R such that for any measurable function $f : Y \rightarrow [0, \infty]$ $R(f) : X \rightarrow [0, \infty]$, $\langle f, H.\mu \rangle = \langle R.f, \mu \rangle$. We shall then call R the **pseudo-adjoint** of H. As previously, we have:

Proposition 4. *An operator has a unique pseudo-adjoint.*

Adjoints and pseudo-adjoints are identical notions in well-behaved cases. A continuity condition ensures that we do not get undefined cases, e.g. $\infty - \infty$.

Lemma 3. *If H is a continuous positive linear operator on measures (positive meaning that $\mu \geq 0 \Rightarrow H.\mu \geq 0$) and R is a positive linear operator that is the pseudo-adjoint of H, then R is the adjoint of H and $\|R\| = \|H\|$.*

2.4 Program Semantics Have Adjoints

A few facts are easily proved:

Proposition 5. *Operators on measures that are lifted from functions (e.g. f_p where $f_p.\mu$ is the measure $X \mapsto \mu(f^{-1}(X))$) have (pseudo-)adjoints: the (pseudo-)adjoint of f_p is $g \mapsto g \circ f$.*

Proposition 6. *If H_1 and H_2 have (pseudo-)adjoints R_1 and R_2, then $R_2 \circ R_1$ is the adjoint of $H_1 \circ H_2$.*

Proposition 7. *ϕ_W has (pseudo-)adjoint $R_W = f \mapsto f.\chi_W$ where χ_W is the characteristic function of W.*

Proposition 8. *If H_1 and H_2 have respective (pseudo-)adjoint R_1 and R_2, then $H_1 + H_2$ has (pseudo-)adjoint $R_1 + R_2$.*

Proposition 9. *If μ_R is a σ-finite positive measure, $\mu \mapsto \mu \otimes \mu_R$ has pseudo-adjoint $f \mapsto \big(x \mapsto \int f(x,\cdot)\,\mathrm{d}\mu_R\big)$.*

This is an application of the Fubini-Tonelli theorem [5, VI.10].

Lemma 4. *If $f : X \to [0;\infty]$ is a positive measurable function and $(\mu_n)_{n\in\mathbb{N}}$ is a sequence of positive measures, then*

$$\int f\,\mathrm{d}\!\left(\sum_{n=0}^{\infty} \mu_n\right) = \sum_{n=0}^{\infty} \int f\,\mathrm{d}\mu_n. \tag{2}$$

The sum of measures is taken set-wise.

Corollary 3. *If $(H_n)_{n\in\mathbb{N}}$ are operators on measures with respective pseudo-adjoints $(R_n)_{n\in\mathbb{N}}$, then $\sum_{n=0}^{\infty} H_n$ has pseudo-adjoint $\sum_{n=0}^{\infty} R_n$ (these sum being taken as simple convergences).*

Theorem 1. *Let c be a probabilistic program. Then the linear operator $[\![\mathtt{c}]\!]_p$ has a pseudo-adjoint.*

Corollary 4. *Since program semantics operators are continuous, of norm less than 1, they have adjoints of norm less than 1.*

Kozen proved [8,9] the following theorem:

Theorem 2. *Semantics of probabilistic programs differ if and only if they differ on point masses.*

Proof. This theorem follows naturally from the preceding corollary and lemma 2.

We shall often note T^* the adjoint of T. We therefore have defined an adjoint semantics for programs: $[\![c]\!]_p^* \in L(M(Y,\mathbb{R}_+), M(X,\mathbb{R}_+))$.

3 Abstract Interpretation of Backwards Probabilistic Semantics

We wish to apply the usual framework of abstract interpretation to the above semantics; that is, for any program c, whose type (not considering probabilistic and nondeterministic effects) is $X \to Y$, we want:

- abstract domains $X_w^\sharp$ and $Y_w^\sharp$, representing sets of weight functions respectively on X and Y;
- a computable abstraction $[\![c]\!]_p^{*\sharp}$ of $[\![c]\!]_p^*$.

The construction of the abstract shall be made compositionnally. We shall first see briefly what we call "abstraction". The reader shall refer to the standard texts on abstract interpretation for more information [3].

3.1 Abstract Interpretation

Taking nondeterminism into consideration, our program semantics is defined as a function $[\![c]\!]^{*\flat}$ from the set $Y_w = \mathcal{P}(M(X, \mathbb{R}_+))$ of sets of weight functions on X to the set $X_w = \mathcal{P}(M(Y, \mathbb{R}_+))$ of sets of weight functions on Y. To simplify the treatment of such sets of weight functions and make operations effectively computable, we choose to over-approximate them by sets of a "simpler" form. Those sets are characterized by elements $x^\sharp_w$ of a preordered **abstract domain** $X^\sharp_w$ (resp. $Y^\sharp_w$). We also consider a $\gamma : X^\sharp_w \to X_w$ function, which maps an element of the abstract domain to what it represents: if $A \subseteq \gamma(A^\sharp)$, then $A^\sharp$ is said to be an **abstraction** of A and A a **concretization** of $A^\sharp$.

A function $H^\sharp_w : Y^\sharp_w \to X^\sharp_w$ is said to be an **abstraction** of an operator $H_w : Y_w \to X_w$ if for any weight function f_w in Y_w and any abstraction $f^\sharp_w$ of f_w, then $H^\sharp_w(y^\sharp_w)$ is an abstraction of $H_w(y_w)$.

Our idea is the following: as seen earlier, our objective is to give bounds on integrals of the form $V = \langle [\![c]\!]^*.\chi_P, \mu\rangle$ where P is a subset of Y; we take an abstraction $\chi^\sharp_P$ of its characteristic function, then compute $f^\sharp_w = [\![c]\!]^{*\sharp}(\chi^\sharp_P)$. We then compute a bound V' such that for any weight function f_w and any concretization f_w of $f^\sharp_w$, $\langle f_w, \mu\rangle \leq V'$; then $V \leq V'$. Of course, we choose the abstract domain so that computing such a V' from $f^\sharp_w$ is easy.

For technical reasons, we shall also require the concretizations to be topologically closed with respect to the topology of simple convergence on the weight functions. More precisely, we require that for any abstract element $f^\sharp$ and ascending sequence $(f_n)_{n\in\mathbb{N}}$ of concretizations of $f^\sharp$, then the point-wise limit $x \mapsto \lim_{n\to\infty} f_n(x)$ is also a concretization of $f^\sharp$.

3.2 Ordinary Backwards Abstract Interpretation

We shall suppose we have an abstraction of the normal, non-probabilistic, semantics, suitable for backwards analysis: for any elementary construct (assignments, arithmetic operations...) c such that $[\![c]\!] : X \to Y$, we must have a monotonic function $[\![c]\!]^{-1^\sharp} : Y^\sharp \to X^\sharp$ such that for all $A^\sharp$, $[\![c]\!]^{-1}(\gamma_Y(A^\sharp)) \subseteq \gamma_X([\![c]\!]^{-1^\sharp}(A))$. We also must have abstractions for the $\phi_{[\![\mathtt{c}]\!]}$ functions.

Let us note the following interesting property of the "usual" assignment operator: $[\![x\ \mathtt{:=}\ e]\!]^{-1} = \bar{\pi}_x \circ \phi_{[\![x\ \mathtt{=}\ e]\!]}$ where $\bar{\pi}_x$ is the "projection parallel to x": $\bar{\pi}_x(W) = \{\boldsymbol{v} \mid \exists \boldsymbol{v}'\ \forall x'\ x' \neq x \Rightarrow v_{x'} = v_x\}$. It is therefore quite easy to build reverse abstractions for the elementary constructs.

3.3 General Form of Abstract Computations

Let us now suppose we have an abstract domain with appropriate abstract operators for the elementary constructs of the language (we shall give an example of construction of such domains in the next section). We shall see in this section how to deal with the flow-control constructs: the sequence, the test and the loop. The abstract domain shall therefore also supply abstract operators $R^\sharp_{[\![c]\!]}$ and $+^\sharp$.

Sequence. Since $[\![e_1\,;e_2]\!]^*_p = [\![e_1]\!]^*_p \circ [\![e_2]\!]^*_p$ then $[\![e_1\,;e_2]\!]^{*\sharp}_p = [\![e_1]\!]^{*\sharp}_p \circ [\![e_2]\!]^{*\sharp}_p$.

Tests. Let us recall the reverse semantics of the `if` construct:

$$[\![\texttt{if}\ c\ \texttt{then}\ e_1\ \texttt{else}\ e_2]\!]^*_p = R_{[\![\mathtt{c}]\!]} \circ [\![e_1]\!]^*_p + R_{[\![\mathtt{c}]\!]^C} \circ [\![e_2]\!]^*_p \tag{3}$$

This equation gets straightforwardly lifted to the abstract domain:

$$[\![\texttt{if}\ c\ \texttt{then}\ e_1\ \texttt{else}\ e_2]\!]^{*\sharp}_p = R^\sharp_{[\![\mathtt{c}]\!]} \circ [\![e_1]\!]^{*\sharp}_p +^\sharp R^\sharp_{[\![\mathtt{c}]\!]^C} \circ [\![e_2]\!]^{*\sharp}_p \tag{4}$$

is a valid abstraction.

Loops. Let us recall the reverse semantics of the `while` construct:

$$[\![\texttt{while}\ c\ \texttt{do}\ e]\!]^*_p = \sum_{n=0}^{\infty} \left(R_{[\![c]\!]} \circ [\![e]\!]^*_p\right)^n \circ R_{[\![\mathtt{c}]\!]^C} \tag{5}$$

Defining f_n recursively, as follows: $f_0 = \lambda x.0$ and $f_{n+1} = \psi f_n$, with

$$\psi(g) = R_{[\![\mathtt{c}]\!]^C}.f + R_{[\![c]\!]} \circ [\![e]\!]^*_p.g,$$

we can rewrite equation 5 as $[\![\texttt{while}\ c\ \texttt{do}\ e]\!]^*_p.f = \lim_{n\to\infty} f_n$. We wish to approximate this limit in the measure space by an abstract element.

ψ gets lifted straightforwardly to an abstract operator:

$$\psi^\sharp(g^\sharp) = R^\sharp_{[\![\mathtt{c}]\!]^C}.f^\sharp +^\sharp R^\sharp_{[\![c]\!]} \circ [\![e]\!]^{*\sharp}_p.g^\sharp. \tag{6}$$

Let us define $f_0^\sharp$ to be an abstraction of the set $\{f_0\}$ and $f^\sharp_{n+1} = \psi^\sharp(f^\sharp_n)$. Obviously, for all n, $f_n \in \gamma(f^\sharp_n)$. If there exists an N such that $\forall n \geq N$, $f_n \in \gamma(f^\sharp_N)$ then $\lim_{n\to\infty} f_n \in \gamma(f^\sharp_N)$ since $\gamma(f^\sharp_N)$ is topologically closed. We have therefore found an abstraction of $[\![\texttt{while}\ c\ \texttt{do}\ e]\!]^*_p.f$.

If the lattice $T^\sharp$ is such that all ascending sequences are stationary, then such a N will necessarily exist. In general, such a N does not exist and we are forced to use so-called **widening operators** [3, §4.3], as follows: we replace the sequence f_n by the sequence defined by $\hat{f}^\sharp_0 = f^\sharp_0$ and $\hat{f}^\sharp_{n+1} = \hat{f}^\sharp_n \nabla_n \psi^\sharp(\hat{f}^\sharp_n)$ where ∇_n are widening operators:

- for all a and b, $a \sqsubseteq a\nabla b$ and $b \sqsubseteq a\nabla b$;
- for all sequence $(u_n)_{n\in\mathbb{N}}$ and any sequence $(v_n)_{n\in\mathbb{N}}$ defined recursively as $v_{n+1} = v_n \nabla u_n$, then (v_n) is stationary.

Obviously, for all n, $f_n \sqsubseteq \gamma(\hat{f}^\sharp_n)$. Since $(\hat{f}^\sharp_n)_{n\in\mathbb{N}}$ is stationary after rank N, and $\gamma(\hat{f}^\sharp_N)$ is topologically closed, this implies that $\lim_{n\to\infty} f_n \in \gamma(f^\sharp_N)$ as above.

This proof extends to the cases where the body of the loop contains nondeterministic constructs in addition to probabilistic ones — we then consider a set of ascending sequences, each having a limit in $\gamma(\hat{f}^\sharp_N)$.

4 A Generic Construction of Abstract Domains

Our goal now is to have an efficient way of representing sets of weight functions. In this section we propose an abstract lattice based on **step functions**. As usual in Lebesgue integration theory, a step function is a finite linear combination of characteristic functions of (measurable) subsets of the domain (see Fig. 5); this generalizes the usual definition when the domain is $\mathbb{R}$. χ_M will denote the characteristic function of subset M — that is, the function mapping x onto 1 if $x \in M$ and 0 elsewhere.

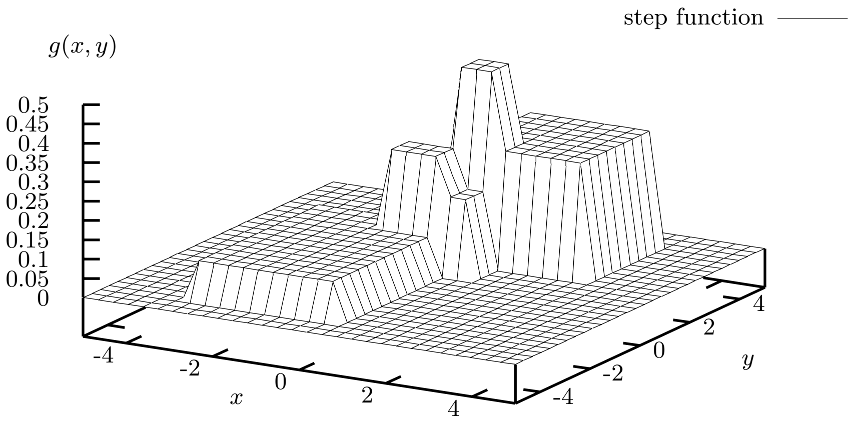

Fig. 5. An example of a step function: $0.2\chi_{[-1,1]\times[0,2]} + 0.3\chi_{[0,3]\times[1,4]} + 0.1\chi_{[-3,0]\times[-4,1]}$. The nonvertical slopes are of course an artefact from the plotting software.

4.1 Representation

Let us take an "ordinary" abstract interpretation lattice $X^\sharp$ for the domain X. This means we have a nondecreasing function $\gamma : (X^\sharp, \sqsubseteq) \to (\mathcal{P}(X), \subseteq)$. We shall only consider the set $X^\sharp_w$ of step functions of the form $\sum_k \alpha_k . \chi_{\gamma A^\sharp_k}$ where $A^\sharp_k \in X^\sharp$. This function can be represented in machine by a finite list of couples $(A^\sharp_k, \alpha_k)_{1 \leq k \leq n}$.

The set $X^\sharp_w$ is pre-ordered by the usual pointwise ordering: $(A^\sharp_k, \alpha_k) \sqsubseteq (B^\sharp_k, \beta_k)$ if and only if for all $x \in X$ then $\sum_k \alpha_k . \chi_{\gamma A^\sharp_k}(x) \leq \sum_k \beta_k . \chi_{\gamma B^\sharp_k}(x)$. Please note that while the pointwise ordering $\leq$ on step function is indeed antisymmetric, $\sqsubseteq$ is only a preorder since representation is not unique: $(([0,1],1), (]1,2],1))$ and $(([0,2],1))$ both represent $\chi_{[0,2]}$. Let us define

$$\gamma_w : \left| \begin{array}{l} (X^\sharp_w, \sqsubseteq) \to (\mathcal{P}(M(X, \mathbb{R}_+)), \subseteq) \\ (A^\sharp_k, \alpha_k) \mapsto \{ f \in M(X, \mathbb{R}_+) \mid f \leq \sum_k \alpha_k . \chi_{\gamma A^\sharp_k} \} \end{array} \right. \tag{7}$$

$(X^\sharp_w, \sqsubseteq)$ is therefore a suitable abstract domain for weight functions.

4.2 Comparison

Our abstract lattice does not have unicity of representation, as noted above. Yet comparisons and equivalence testings are easily computable, provided the underlying abstract domain provides an intersection test — a computable function $(A^\sharp, B^\sharp) \mapsto \begin{cases} 1 & \text{if } \gamma(A) \cap \gamma(B) \neq \emptyset \\ 0 & \text{otherwise.} \end{cases}$

Let us take two abstract values $A^\sharp_w = ((A^\sharp_i, \alpha_i)_{1 \leq i \leq m})$ and $B^\sharp_w = ((B^\sharp_j, \alpha_j)_{1 \leq j \leq n})$. Let us consider the set C of nonempty intersections $\bigcap \gamma(A^\sharp_i)_{i \in I} \cap \bigcap \gamma(B^\sharp_j)_{j \in J} \neq \emptyset$ where I is a subset of the indices $1..m$ and J is a subset of the indices $1..n$: each element of C is denoted by a couple (I, J).

Let us define $w : \left| \begin{array}{ll} C & \to \mathbb{R} \\ (I,J) & \mapsto \sum_{i \in I} \alpha_i - \sum_{j \in J} \beta_i. \end{array} \right.$ Then $A^\sharp_w \sqsubseteq B^\sharp_w \iff \forall c \in C \; w(c) \leq 0$.

4.3 Abstract Operations

We must provide abstract operators for each construction of the language.

Basic Constructs. Let us now suppose we have an abstraction $[\![c]\!]^{-1^\sharp}$ of the function $[\![c]\!]^{-1} : \mathcal{P}(Y) \to \mathcal{P}(X)$. Then an abstraction of $[\![c]\!]^*_p$ is

$$[\![c]\!]^{*\sharp}_p = (X^\sharp_\lambda, \alpha_\lambda)_{\lambda \in \Lambda} \mapsto ([\![c]\!]^{-1^\sharp}(X^\sharp_\lambda), \alpha_\lambda)_{\lambda \in \Lambda} \tag{8}$$

Random Number Generation. We shall obtain here an abstraction of `r:=random` where `r` is a new variable and `random` follows probability measure μ_R. Let us suppose the random variable lies in a set R. $[\![\texttt{r:=random}]\!]^*$ is therefore a linear operator from $M(X \times R, \mathbb{R}_+)$ to $M(X, \mathbb{R}_+)$.

Let us suppose that $\mu_R = \sum_{k=1}^n \mu_k$ where each μ_k is concentrated on a subset M_k or R. For instance, taking $R = \mathbb{R}$, the uniform probability measure on $[0,1]$ can be split into n measures μ_k, the Lebesgue measure on $[k/n, (k+1)/n]$. Let us call π_X and π_R the projections of $X \times R$ onto X and R respectively.

Using prop. 9,

$$[\![\texttt{r:=random}]\!]^*.\chi_A = x \mapsto \sum_{k=1}^n \int \chi_A(x,y)\,\mathrm{d}\mu_k(y). \tag{9}$$

But $\int \chi_A(x,y)\,\mathrm{d}\mu_k(y) \leq \mu_k(\pi_R(A))$, and $\int \chi_A(x,y)\,\mathrm{d}\mu_k(y) = 0$ if $x \notin \pi_X(A)$. Therefore

$$[\![\texttt{r:=random}]\!]^*.\chi_A \leq \mu_k(\pi_R(A)).\chi_{\pi_X(A)}. \tag{10}$$

Lifting to abstract semantics is then easy: $[\![\texttt{r:=random}]\!]^{*\sharp}(A^\sharp_i, \alpha_i)_{1 \leq i \leq m}$ maps to $(A^\sharp_{i,k}, \alpha_i.\beta_{i,k})_{1 \leq i \leq m, 1 \leq k \leq n}$ where $A^\sharp_{i,k}$ is an abstraction of $\pi_X(\gamma(A^\sharp_i) \cap (X \times$

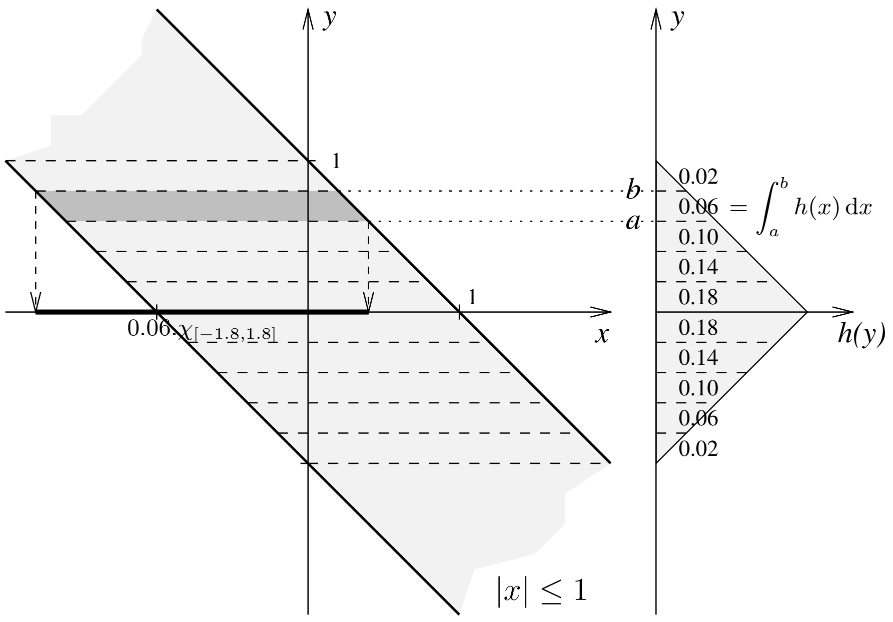

Fig. 6. Construction of the output value of Fig. 7 for $n = 10$. The $|x + y| \leq 1$ abstract area is sliced along divisions on the y axis. Each slice S_k is projected onto the y axis and the integral of the distribution function h of `(centered_uniform()+centered_uniform())/2` is taken on this projection, yielding a coefficient α_k. The slice is then projected on the x axis and this projection B_k, with the α_k coefficient is an element of the abstract value $\sum_{k=i}^{n} \alpha_k.\chi_{B_k}$. The approximations plotted in Fig. 7 are those sums, with various numbers of slices.

M_k)) and $\beta_{i,k} \geq \mu_k(\pi_R(A))$ (Fig. 6 explains how we built the approximations in Fig. 7). Both the $A^\sharp_{i,k}$ and the $\beta_{i,k}$ can be computed easily for many underlying abstract domains, such as the nondependent product of intervals [2].

Of course, there is some amount of choice in the choice of how to cut μ into μ_k. We suggest to cut into measures of equal weights. Of course, the higher the number of μ_k's, the better the immediate results (Fig. 7), nevertheless a high number of elements in abstract values may necessitate an early use of widenings (see 4.3). We hope the forthcoming implementation will help adjust such heuristic parameters.

Tests. The semantics for tests gets straightforwardly lifted to abstract semantics, provided we have abstract operators for R_W and $+$:

$$[\![\texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2]\!]^{*\sharp}_p.f^\sharp = R^\sharp_{[\![b]\!]} \circ [\![c_1]\!]^{*\sharp}_p.f^\sharp +^\sharp R^\sharp_{[\![b]\!]^C} \circ [\![c_2]\!]^{*\sharp}_p.f^\sharp \quad (11)$$

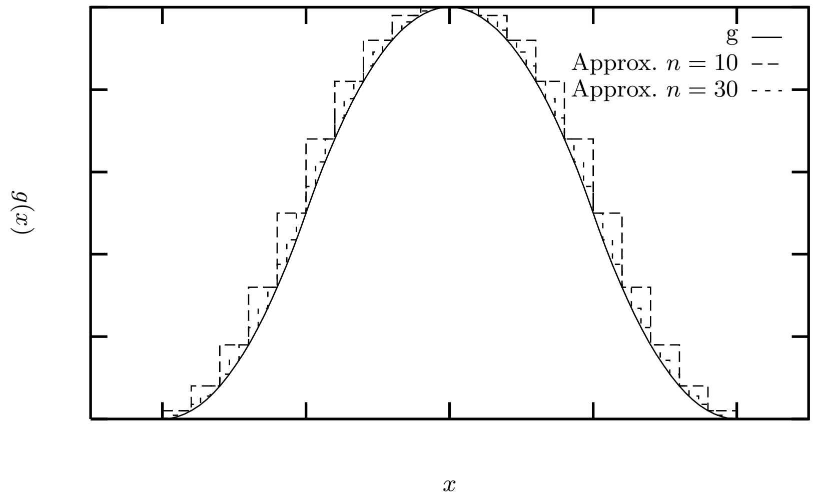

Fig. 7. Two approximations of the actual distribution of Fig. 2, resulting from the abstract interpretation of the program of Fig. 1. Our generic lattice is parameterized by the product lattice of intervals. Different discretizations of the probability measure $h(x)\,\mathrm{d}x$ of `(centered_uniform()+centered_uniform())/2` yield more or less precise abstractions. Here, the interval $[-1,1]$ where h is nonzero is divided into n segments of equal size, yielding n elements in the output abstract value.

Abstracting $+$ is easy: $+^\sharp$ is the concatenation operator on sequences (or families); as for R_W:

$$R^\sharp_{W^\sharp} = (X^\sharp_\lambda, \alpha_\lambda)_{\lambda\in\Lambda} \mapsto (X^\sharp_\lambda \cap^\sharp W^\sharp, \alpha_\lambda)_{\lambda\in\Lambda} \tag{12}$$

Widening Operators. Using the above abstractions for R_W and $+^\sharp$, it is easy (3.3) to find an approximation of the semantics of the loop, provided we have suitable widening operators. We shall here propose a few heuristics for widenings. Widenings are also useful since they provide a way to "simplify" abstract elements, to save memory, even if such a simplification loses precision.

Let us suppose we have a widening sequence ∇_k on $X^\sharp$. We shall now give heuristics for computing $x\nabla_k y$ where $x = (X^\sharp_i, \alpha_i)_{1\leq i\leq a}$ and $y = (Y^\sharp_j, \beta_j)_{1\leq j\leq b}$. For the sake of simplicity, we shall suppose we aim at keeping Λ index sets less than n elements for a certain fixed n. We shall suppose that $a \leq n$.

The main idea is that of coalescing. For each element $(X^\sharp_i)$, find "close" elements $(Y^\sharp_{j_{i,1}}), \dots, (Y^\sharp_{j_{i,m}})$, the closeness criterion being largely heuristic and dependent on the chosen lattice; this criterion does not influence the correctness of the method, only its precision and efficiency. We then pose

$$x\nabla_k y = (X^\sharp_i \nabla_k \left(Y^\sharp_{j_{i,1}} \cup \cdots \cup Y^\sharp_{j_{i,m}}), \max(\alpha_i, \beta_{j_{i,1}} + \ldots + \beta_{j_{i,m}})\right). \tag{13}$$

Let us now take a sequence $(x^{(k)})_{k\in\mathbb{N}}$ such that $x^(k+1) = x^{(k)}\nabla y^(k)$. Then for all $1 \leq i \leq n$, $X^(k+1)_i = X_i^{(k)}\nabla v_i^{(k)}$ for some $v_i^{(k)}$, so the sequence $(X_i^{(k)})_{k\in\mathbb{N}}$ is stationary. Since this holds for all i, this means that $(x^{(k)})$ is stationary.

The design of widenings is an inherently heuristic process, and we thus expect to have better widenings as implementation progresses and experiments are possible.

5 Comparison with Other Methods

Let us first remark that our method is a natural extension of conventional backwards abstract interpretation. Indeed, let us consider only programs containing no `random`-like operations; any program, even including `random`-like operations, can be transformed into a program containing none by moving the streams of random numbers into the environment of the program (this corresponds to the first semantics proposed by Kozen [8,9]).

With such programs, our framework is equivalent to computing reverse images of sets: $[\![c]\!]_p^*.\mu.\chi_W = \mu([\![c]\!]^{-1}(W))$ and our proposed abstract domain just expresses that $[\![c]\!]_p^*.\mu.\chi_W \leq \mu \circ \gamma \circ [\![c]\!]^{-1^\sharp}(W^\sharp)$. There are nevertheless two differences that makes our abstract domain more interesting:

- in the presence of streams of random numbers, our abstract domain just makes use of an ordinary abstract domain, while computing approximate reverse images in the presence of infinite streams requires an abstract domain capable of abstracting infinite sequences so that the results remain interesting;
- we can "simplify" abstract values representing weight functions heuristically so that we do not waste time giving too much precision to negligible parts of the domain.

6 Conclusions

We have proposed a general scheme for the backwards abstract interpretation of nondeterministic, probabilistic programs. This scheme allows the effective computation of upper bounds on the probability of outcomes of the program. It is based on abstract interpretation, which it extends to an adequate "adjoint semantics" of probabilistic programs. We propose a parametric abstract domain for this analysis; this domain is based on an underlying "ordinary" abstract domain, which can be any domain proposed for abstract interpretation of non-probabilistic programs.

References

1. François Bourdoncle. Interprocedural abstract interpretation of block structured languages with nested procedures, aliasing and recursivity. In *PLILP '90*, volume 456 of *LNCS*, pages 307–323. Springer-Verlag, 1990.

2. P. Cousot and R. Cousot. Static determination of dynamic properties of programs. In *Proceedings of the Second International Symposium on Programming*, pages 106–130. Dunod, Paris, France, 1976.
3. Patrick Cousot and Radhia Cousot. Abstract interpretation and application to logic programs. *J. Logic Prog.*, 2-3(13):103–179, 1992.
4. Alessandra Di Pierro and Herbert Wiklicky. Concurrent constraint programming: Towards probabilistic abstract interpretation. In *2nd International Conference on Principles and Practice of Declarative Programming (PPDP 2000)*, 2000.
5. J.L. Doob. *Measure Theory*, volume 143 of *Graduate Texts in Mathematics.* Springer-Verlag, 1994.
6. Paul R. Halmos. *Measure theory.* The University series in higher mathematics. Van Nostrand, 1950.
7. Jifeng He, K. Seidel, and A. McIver. Probabilistic models for the guarded command language. *Science of Computer Programming*, 28(2–3):171–192, April 1997. Formal specifications: foundations, methods, tools and applications (Konstancin, 1995).
8. D. Kozen. Semantics of probabilistic programs. In *20th Annual Symposium on Foundations of Computer Science*, pages 101–114, Long Beach, Ca., USA, October 1979. IEEE Computer Society Press.
9. D. Kozen. Semantics of probabilistic programs. *Journal of Computer and System Sciences*, 22(3):328–350, 1981.
10. Serge Lang. *Linear algebra.* Springer-Verlag, New York, 1989.
11. Gavin Lowe. Representing nondeterminism and probabilistic behaviour in reactive processes. Technical Report TR-11-93, Oxford University, 1993.
12. David Monniaux. Abstract interpretation of probabilistic semantics. In *Seventh International Static Analysis Symposium (SAS'00)*, number 1824 in Lecture Notes in Computer Science. Springer-Verlag, 2000. © Springer-Verlag.
13. David Monniaux. An abstract Monte-Carlo method for the analysis of probabilistic programs (extended abstract). In *28th Symposium on Principles of Programming Languages (POPL '01).* Association for Computer Machinery, 2001. To appear.
14. Carroll Morgan, Annabelle McIver, Karen Seidel, and J. W. Sanders. Probabilistic predicate transformers. Technical Report TR-4-95, Oxford University, February 1995.
15. Carroll Morgan, Annabelle McIver, Karen Seidel, and J. W. Sanders. Refinement-oriented probability for CSP. *Formal Aspects of Computing*, 8(6):617–647, 1996.
16. Reuven Y. Rubinstein. *Simulation and the Monte-Carlo Method.* Wiley series in probabilities and statistics. John Wiley & Sons, 1981.
17. Walter Rudin. *Real and Complex Analysis.* McGraw-Hill, 1966.