An Abstract Analysis of the Probabilistic Termination of Programs

David Monniaux
http://www.di.ens.fr/~monniaux

LIENS, 45 rue d'Ulm
75230 Paris cedex 5, France

Abstract. It is often useful to introduce probabilistic behavior in programs, either because of the use of internal random generators (probabilistic algorithms), either because of some external devices (networks, physical sensors) with known statistics of behavior. Previous works on probabilistic abstract interpretation have addressed safety properties, but somehow neglected probabilistic termination. In this paper, we propose a method to automatically prove the probabilistic termination of programs using exponential bounds on the tail of the distribution. We apply this method to an example and give some directions as to how to implement it. We also show that this method can also be applied to make unsound statistical methods on average running times sound.

1 Introduction

In this paper, we propose an analysis scheme for probabilistic programs, with the goal of proving probabilistic termination and other probabilistic properties.

1.1 Goals

It is in general difficult to automatically prove liveness properties of programs; nevertheless, the availability of probabilistic informations makes that task easier. In this paper, we shall address the cases of termination where the probability of taking a loop k times decreases at least exponentially with k. Such cases happen, for instance, in waiting loops on hardware peripherals where the probability that the busy peripheral will become ready over a period of time is a constant. However, we address cases where the constant in the exponential is unknown. Our analysis tries to derive automatically suitable constants for the exponential bound.

An obvious application of this analysis is to prove that the probability of non-termination is zero. Other applications include improvements on statistical methods to derive average execution times.

Our analysis is explained on block-structured programs. A reason for this is that the control-flow structure of a block-structured program is generally apparent from its syntactic structure, whereas converting the program to a probabilistic transition system would hide that structure. It is nevertheless possible to apply the method to a probabilistic transition system, considering it as a single `while` loop, although the results may not be very satisfactory.

As usual with abstract interpretation, our analysis is sound, that is, any result it gives is proven to be correct (provided that the implementation of the analysis is correct, of course). On the other hand, because of the essential undecidability of non-trivial program properties, our analyzer is forced to give non-optimal results. Careful experimentation and heuristics may then improve the analysis.

1.2 Related Works

Several analyses have been proposed to prove the termination of non-probabilistic programs. Generally speaking, proving the termination of a program is done by showing that some value taken in a well-founded ordered set decreases strictly as the program proceeds [7]. The problem is to detect the appropriate value and the appropriate order. We follow this general principle by showing that the probability that the loop count or execution time reaches k is bounded by a strictly decreasing function of k whose limit is 0; the choice of an exponential is natural since the distribution we wish to approximate is exactly exponential in some simple cases (where the control graph of the program is a Markov chain, as in §6.1).

We use the framework of abstract interpretation of probabilistic programs defined in our earlier work [8]. However, that work did not address the problem of probabilistic termination except perhaps in its most trivial cases (where there exists a constant number of iterations after which the program always terminate, regardless of the inputs). Here, we address a specific case (loops whose iteration count or total execution time follow a distribution bounded from above by a decreasing exponential) that was not addressed by our previous methods, yet is of common practical interest. Nevertheless, the analysis described in this paper can be "mixed" with the analysis described in [8]; both are likely to be implemented in the same system.

1.3 Overview of the Paper

In section 2, we explain the probabilistic concrete semantics that we analyze. In section 3, we explain how to apply abstract interpretation to such a semantics. In section 4, we give a first abstract domain able to express properties such as the exponential decrease of the tail of the distribution of an integer variable. In section 5, we explain how to build a more complex, but much more precise domain using elements of the former domain as building blocks. In section 6, we show on a simple example what kind of results the analysis achieves, and we explain how such an analysis can improve the mathematical soundness of experimental average time estimations.

2 Concrete Semantics

We take the same concrete semantics as in earlier papers [5, 6, 8]. For the sake of clarity and self-containedness, we shall summarize them here.

We shall express probabilities using *measures* [11, §1.18]. We shall begin by a few classical mathematical definitions.

2.1 Measures

The basic objects we shall operate on are measures.

- $\mathcal{P}(X)$ is the power-set of X.
- A *σ-algebra* is a subset of $\mathcal{P}(X)$ that contains $\emptyset$ and is stable by countable union and complementation (and thus contains X and is stable by countable intersection).
- A set X with a σ-algebra σ_X defined on it is called a *measurable space* and the elements of the σ-algebra are the *measurable subsets.* We shall often mention measurable spaces by their name, omitting the σ-algebra, if no confusion is possible.
- If X and Y are measurable spaces, $f : X \to Y$ is a *measurable function* if for all W measurable in Y, $f^{-1}(W)$ is measurable in X.
- A *positive measure* is a function μ defined on a σ-algebra σ_X whose range is in $[0, \infty]$ and which is countably additive. μ is countably additive if, taking $(A_n)_{n\in\mathbb{N}}$ a disjoint collection of elements of σ_X, then $\mu\left(\cup_{n=0}^{\infty} A_n\right) = \sum_{n=0}^{\infty} \mu(A_n)$. To avoid trivialities, we assume $\mu(A) < \infty$ for at least one A. The *total weight* of a measure μ is $\mu(X)$. μ is said to be *concentrated* on $A \subseteq X$ if for all B, $\mu(B) = \mu(B \cap A)$. We shall note $\mathcal{M}_+(X)$ the positive measures on X.
- A *probability measure* is a positive measure of total weight 1. A ***σ**-finite* measure is a measure μ so that there exists a countable partition $X = \sqcup_{k=1}^{\infty} X_k$ so that $\mu(X_k) < \infty$ for all k. This is a technical condition that will be met every time in our cases.
- Given two σ-finite measures measures μ and μ' on X and X' respectively, we note $\mu \otimes \mu'$ the product measure [11, definition 7.7], defined on the product σ-algebra $\sigma_X \times \sigma_{X'}$. The characterizing property of this product measure is that $\mu \otimes \mu'(A \times A') = \mu(A).\mu'(A')$ for all measurable sets A and A'.

Our semantics shall be expressed as continuous linear operators between measure spaces.

2.2 Semantics of Arithmetic Operators

Let us consider an elementary program statement c so that $[\![c]\!] : X \to Y$, X and Y being measurable spaces. We shall also suppose that $[\![c]\!]$ is measurable, which is a purely technical requirement.

To $[\![\mathtt{c}]\!]$ we associate the following linear operator $[\![\mathtt{c}]\!]_p$:

$$[\![\mathtt{c}]\!]_p : \left| \begin{array}{ll} \mathcal{M}_+(X) & \to \mathcal{M}_+(Y) \\ \mu & \mapsto \lambda W.\mu([\![c]\!]^{-1}(W)) \end{array} \right. .$$

2.3 Random Inputs or Generators

An obvious interest of probabilistic semantics is to give an accurate semantics to assignment such as `x:=random();`, where `random()` is a function that, each time it is invoked, returns a real value uniformly distributed between 0 and 1, independently of previous calls.[1] We therefore have to give a semantics to constructs such as `x:=random();`, where `random` returns a value in a measured space R whose probability is given by the measure μ_R and is independent of all other calls and previous states.

We decompose this operation into two steps:

$$\begin{tikzcd} X_p \arrow[r, "{[\![\rho\texttt{:=random()}]\!]}"'] \arrow[rr, bend left, "{[\![\texttt{x:=random()}]\!]}"] & (X \times R)_p \arrow[r, "{[\![\texttt{x:=}\rho]\!]}"'] & X_p \end{tikzcd}$$

The second step is a simple assignment operator, addressed by the generic semantics for arithmetic operators. The first step is a product of measures:

$$[\![\rho\texttt{:=random()}]\!] : \left| \begin{array}{l} X_p \to (X \times R)_p \\ \mu \quad \mapsto \mu \otimes \mu_R \end{array} \right. . \tag{1}$$

In the rest of the paper, when dealing with random generators, we shall focus on this product operation.

2.4 Tests and Loops

We restrict ourselves to test and loop conditions b so that $[\![\texttt{b}]\!]$ is measurable. $[\![\texttt{b}]\!]$ is the set of environments matched by condition b. It is obtained inductively from the set of environment matched by the atomic tests (e.g. comparisons):

- $[\![b_1 \texttt{ or } b_2]\!] = [\![b_1]\!] \cup [\![b_2]\!]$
- $[\![b_1 \texttt{ and } b_2]\!] = [\![b_1]\!] \cap [\![b_2]\!]$
- $[\![\texttt{not } b]\!] = [\![b]\!]^C$

The semantics of tests is:

$$[\![\texttt{if } c \texttt{ then } e_1 \texttt{ else } e_2]\!]_p(\mu) = [\![e_1]\!]_p \circ \phi_{[\![c]\!]}(\mu) + [\![e_2]\!]_p \circ \phi_{[\![c]\!]^C}(\mu) \tag{2}$$

[1] Of course, functions such as the POSIX C function `drand48()` would not fulfill such requirements, since they are pseudo-random generators whose output depends on an internal state that changes each time the function is invoked, thus the probability laws of successive invocations are not independent. However, ideal random generators are quite an accurate approximation for most analyses.

where $\phi_W(\mu) = \lambda X.\mu(X \cap W)$. The semantics for the `while` loop is:

$$\begin{aligned}
[\![\texttt{while}\ c\ \texttt{do}\ e]\!]_p(\mu) &= \sum_{n=0}^{\infty} \phi_{[\![c]\!]^C} \circ ([\![e]\!]_p \circ \phi_{[\![c]\!]})^n(\mu) \\
&= \phi_{[\![c]\!]^C}\left(\sum_{n=0}^{\infty}([\![e]\!]_p \circ \phi_{[\![c]\!]})^n(\mu)\right) \\
&= \phi_{[\![c]\!]^C}\left(\lim_{n\to\infty}(\lambda\mu'.\mu + [\![e]\!]_p \circ \phi_{[\![c]\!]}(\mu'))^n(\lambda X.0)\right) \qquad (3)
\end{aligned}$$

Limits and infinite sums are taken according to the set-wise topology [3, §III.10]. We refer the reader to the extended version of [8] for the technical explanations on continuity and convergence.

3 Abstract Semantics

We first give the vocabulary and notations we use for abstractions in general (the reader is invited to consult [2] for further details). We then explain the particular treatment of probabilistic semantics.

3.1 Summary of Abstraction

Let us consider a preordered set $X^\sharp$ and a monotone function $\gamma_X : X^\sharp \to \mathcal{P}(X)$. $x^\sharp \in X^\sharp$ is said to be an *abstraction* of $x^\flat \subset X$ if $x^\flat \subseteq \gamma_X(x^\sharp)$. γ_X is called the *concretization function*. The triple $\langle \mathcal{P}(X), X^\sharp, \gamma_X \rangle$ is called an *abstraction*. $\mathcal{P}(X)$ is the *concrete domain* and $X^\sharp$ the *abstract domain*. Such definitions can be extended to any preordered set $X^\flat$ besides $\mathcal{P}(X)$.

Let us now consider two abstractions $\langle \mathcal{P}(X), X^\sharp, \gamma_X \rangle$ and $\langle \mathcal{P}(Y), Y^\sharp, \gamma_Y \rangle$ and a function $f : X \to Y$. $f^\sharp$ is said to be *an abstraction of f* if

$$\forall x^\sharp \in X^\sharp\ \forall x \in X\ x \in \gamma_X(x^\sharp) \Rightarrow f(x) \in \gamma_Y(f^\sharp(x^\sharp)) \qquad (4)$$

More generally, if $\langle X^\flat, X^\sharp, \gamma_X \rangle$ and $\langle Y^\flat, Y^\sharp, \gamma_Y \rangle$ are abstractions and $f^\flat : X^\flat \to Y^\flat$ is a monotone function, then $f^\sharp$ is said to be *an abstraction of $f^\flat$* if

$$\forall x^\flat \in X^\flat\ \forall x^\sharp \in X^\sharp\ x^\flat \sqsubseteq \gamma_X(x^\sharp) \Rightarrow f^\flat(x^\flat) \sqsubseteq \gamma_X(f^\sharp(x^\sharp)) \qquad (5)$$

Algorithmically, elements in $X^\sharp$ will have a machine representation. To any program construct c we shall attach an effectively computable function $[\![\mathtt{c}]\!]^\sharp$ so that $[\![\mathtt{c}]\!]^\sharp$ is an abstraction of $[\![\mathtt{c}]\!]$. Given a machine description of a superset of the inputs of the programs, the abstract version yields a superset of the outputs of the program. If a state is not in this superset, this means that, for sure, the program cannot reach this state.

Let us take an example, the *domain of intervals*: if $X^\sharp = Y^\sharp = T^3$ where $T = \{(a,b) \in \mathbb{Z} \cup \{-\infty,+\infty\} \mid a \leq b\} \cup \{\perp\}$, $\gamma(a,b) = \{c \in \mathbb{Z} \mid a \leq c \leq b\}$ and γ induces a preorder $\sqsubseteq_T$ over T and, point-wise, over $X^\sharp$, then we can take $[\![\mathtt{x:=y+z}]\!]^\sharp((a_\mathtt{x}, b_\mathtt{x}), (a_\mathtt{y}, b_\mathtt{y}), (a_\mathtt{z}, b_\mathtt{z})) = ((a_\mathtt{y} + a_\mathtt{z}, b_\mathtt{y} + b_\mathtt{z}), (a_\mathtt{y}, b_\mathtt{y}), (a_\mathtt{z}, b_\mathtt{z}))$.

3.2 Turning Fixpoints of Affine Operators into Fixpoints of Monotone Set Operators

Equation 3 shows that the semantics of loops are given as infinite sums or, equivalently, as fixpoints of some affine operators. In non-probabilistic semantics, the semantics of loops is usually the fixpoint of some monotone operator on the concrete lattice, which get immediately abstracted as fixpoints on the abstract lattice. The approximation is not so evident in the case of this sum; we shall nevertheless see how to deal with it using fixpoints on the abstract lattice.

Defining μ_n recursively, as follows: $\mu_0 = \lambda X.0$ and $\mu_{n+1} = \psi\mu_n$, with $\psi(\nu) = \mu + [\![e]\!]_p \circ \phi_{[\![c]\!]}(\nu)$, we can rewrite equation 3 as $[\![\texttt{while}\ c\ \texttt{do}\ e]\!]_p(\mu) = \phi_{[\![c]\!]^C}(\lim_{n\to\infty}\mu_n)$. We wish to approximate this limit in the measure space by an abstract element.

We get this approximation by finding a subset L of $\mathcal{P}(\mathcal{M}_+(X))$ that:

- contains any element of the sequence (μ_n),
- is stable by ψ,
- is topologically closed.

Then $\lim_{n\to\infty}\mu_n \in L$.

Let us take $N \in \mathbb{N}$. Let us note $+^\sharp$ an abstraction of the sum operation on measures, $0^\sharp$ an abstraction of the null measure and $\psi^\sharp(\nu^\sharp) = W^\sharp + [\![e]\!]_p^\sharp \circ \phi^\sharp_{[\![c]\!]}(\nu^\sharp)$. Let us take $\mu \in W^\sharp$. By abstraction, $\mu_N \in \gamma({\psi^\sharp}^N(0^\sharp))$.

Let us suppose that we have an "approximate least fixpoint" operation $\mathrm{lfp}^\sharp : (X^\sharp \xrightarrow{\text{monotonic}} X^\sharp) \to X^\sharp$. By "approximate least fixpoint" we mean that if $\phi^\sharp$ is an abstraction of ϕ, then $\mathrm{lfp}^\sharp\,\phi^\sharp$ is an abstraction of $\mathrm{lfp}\,\phi$. The next sub-section will explain how to implement such an operation using widening operators.

Let us consider $L = \gamma\left(\mathrm{lfp}^\sharp\, X^\sharp \mapsto {\psi^\sharp}^N(0^\sharp) \sqcup (W^\sharp +^\sharp [\![e]\!]^\sharp(\phi^\sharp_{[\![\mathtt{c}]\!]}(X)))\right)$. L contains μ_N and is stable by ψ.

We can therefore take:

$$[\![\texttt{while}\ c\ \texttt{do}\ e]\!]^\sharp(W^\sharp) = \phi^\sharp_{[\![c]\!]^C}\left(\mathrm{lfp}^\sharp\, X^\sharp \mapsto {\psi^\sharp}^N(0^\sharp) \sqcup (W^\sharp +^\sharp [\![e]\!]^\sharp(\phi^\sharp_{[\![\mathtt{c}]\!]}(X)))\right), \tag{6}$$

3.3 Approximation of Least Fixpoints

As noted before, we need an "approximate least fixpoint" operation $\mathrm{lfp}^\sharp : (X^\sharp \xrightarrow{\text{monotonic}} X^\sharp) \to X^\sharp$. We shall see here that such an operation can be implemented using widening operators ([2]; see §4.4, 5.4 for the widening operators that we propose for our particular application).

A *widening operator* $\nabla : X^\sharp \times X^\sharp \to X^\sharp$ is such that

- for all $x^\sharp$, $y^\sharp$, $x^\sharp \sqcup y^\sharp \sqsubseteq x^\sharp \nabla y^\sharp$;
- for any ascending sequence $(y^\sharp_n)$ and any $x^\sharp_0$, the sequence $(x^\sharp_n)$ defined by $x^\sharp_{n+1} = x^\sharp_n \nabla y^\sharp_n$ is ultimately stationary.

Informally, a widening operator is a kind of “convergence accelerator” for ascending sequences of elements of the lattice; it allows obtaining in finite time an over-approximation of the limit of the sequence (which can be reached in infinite time).

Let us suppose we have a monotonic $\phi : X \to X$ and an abstraction of it $\phi^\sharp : X^\sharp \to X^\sharp$. Let us define $x_0^\sharp = \bot$ and $x_n^\sharp = x_n^\sharp \nabla \phi^\sharp(x_n^\sharp)$. Since ∇ is a widening operator, $x_n^\sharp$ is ultimately stationary. Let us call $L^\sharp$ its limit. We have $L^\sharp = L^\sharp \nabla \phi^\sharp(L^\sharp)$. Since ∇ is a widening operator, $\phi^\sharp(L^\sharp) \sqsubseteq L^\sharp \nabla \phi^\sharp(L^\sharp) = L^\sharp$. Let us apply γ (monotonic) to both sides of the inequality: $\gamma \circ \phi^\sharp(L^\sharp) \sqsubseteq \gamma(L^\sharp)$. Since $\phi^\sharp$ is an abstraction of ϕ, $\phi \circ \gamma(L^\sharp) \sqsubseteq \gamma \circ \phi^\sharp(L^\sharp)$. It follows that $\phi(\gamma(L^\sharp)) \sqsubseteq \gamma(L^\sharp)$. By Tarski's theorem [2, §4.1], $\operatorname{lfp} \phi = \bigsqcap_{\phi(x) \sqsubseteq x} x$ and thus $\operatorname{lfp} \phi \sqsubseteq \gamma(L^\sharp)$. $\gamma L^\sharp$ is thus an upper approximation of the least fixpoint of ϕ, obtained by a finite computation using the widening operator.

4 Basic Abstract Domain and Abstract Operations

We wish to represent sets of (sub)probability measures symbolically. In this section, we give a basic abstract domain expressing exponentially decreasing tails of probabilistic distributions on integers. This abstract domain is not very satisfactory by itself (it is very crude when it comes to loops), but is a basis for the definition of a more complex domain (section 5).

4.1 Abstract Domain

Let V be the set of variables. Each element of the abstract domain E is a tuple of coefficients. Those coefficients will represent three kinds of known facts on the measures:

- an upper bound W on their total weight;
- upper bounds on the intervals of probable variation for the integer variables: for any variable v, the probability that v is outside $[a_v, b_v]$ is 0; of course, a_b and/or b_v can be infinite ($a_v \leq b_v$);
- for each integer variable v, some data C_v on its exponential decreasing: either none or a pair $(\alpha_v, \beta_v) \in \mathbb{R}_+ \times [0, 1[$ meaning that the probability that variable v is k is bounded by $\alpha_v \beta_v^k$.

$\gamma_E(W, (a_v, b_v)_{v \in V}, (C_v)_{v \in V})$ is the set of all measures matching the above conditions. We shall note $\mu(\textit{condition})$ for the application of the measure μ to the set of environments matching condition *condition*. The three conditions above then get written as:

$$\mu \in \gamma_E(W, (a_v, b_v)_{v \in V}, (C_v)_{v \in V}) \iff \begin{cases} \mu(\mathsf{true}) \leq W \\ \forall v \in V \ \mu(v \notin [a_v, b_v]) = 0 \\ \forall v \in V \ C_v = (\alpha_v, \beta_v) \Rightarrow \forall k \in \mathbb{Z} \ \mu(v = k) \leq \alpha_v \beta_v^k \end{cases} \quad (7)$$

4.2 Arithmetic Operations

We do not provide an abstract operator for each of the basic operations that a program may encounter; for instance, we say nothing of multiplication. In cases that are not described, we just apply interval propagation [1] and set $C_v = \mathsf{none}$ for every modified variable v. In some case, we shall provide only for some cases, while some others can be handled by using the symmetry of the operation and reverting to a described case.

We focus on the operations that will be most useful for our analysis goals (number of iterations taken in a loop, number of used CPU cycles). For instance, we consider the case of the arithmetic plus since it will be used to count loop iterations or program cycles, whereas multiplication is of little use for such tasks.

Arithmetic Plus We define here the abstract operation $(W, (a_v, b_v)_{v \in V}, (C_v)_{v \in V}) \mapsto (W', (a'_v, b'_v)_{v \in V}, (C'_v)_{v \in V}) = \texttt{z := x+y}^\sharp_p.(W, (a_v, b_v)_{v \in V}, (C_v)_{v \in V})$.

The distribution after applying an arithmetic plus obeys the following convolution equation:

$$(\llbracket \texttt{x+y} \rrbracket_p.\mu)(\texttt{z} = t) = \sum_{k \in \mathbb{Z}} \mu(\texttt{x} = k \wedge \texttt{y} = t - k). \tag{8}$$

Let us suppose that $\mu \in \gamma_E(W, (a_v, b_v)_{v \in V}, (C_v)_{v \in V})$; we want to produce $(W', (a'_v, b'_v)_{v \in V}, (C'_v)_{v \in V})$ so that $(\llbracket \texttt{x+y} \rrbracket_p.\mu \in \gamma_E(W', (a'_v, b'_v)_{v \in V}, (C'_v)_{v \in V})$.

Obviously we can take $W' = W$, $a'_\texttt{z} = a_\texttt{x} + a_\texttt{y}$, $b'_\texttt{z} = b_\texttt{x} + b_\texttt{y}$, and $b'_v = b_v$ and $C'_v = C_v$ for all $v \neq \texttt{z}$.

We therefore have four cases:

- $C_x = \mathsf{none}$ and $C_y = \mathsf{none}$, then $C'_z = \mathsf{none}$;
- $C_x = \mathsf{none}$ and $C_y = (\alpha_y, \beta_y)$. We then have $\mu(x = k \wedge y = t - k) \leq \alpha_y \beta_y^{t-k}$ if $k \in [a_x, b_x]$ $\mu(x = k \wedge y = t - k) = 0$ otherwise. Inequality 8 then yields $\mathbb{P}\,(x + y = t) \leq \alpha_y \sum_{k=a_x}^{b_x} \beta_y^{t-k}$.
  Let $\alpha'_z = \alpha_y \beta_y^{-b_x} \frac{\beta_y^{b_x - a_x + 1} - 1}{\beta_y - 1}$ and $\beta'_z = \beta_y$. In particular, if $b_x = a_x$ (variable x is actually a constant), then $\alpha'_z = \alpha_y \beta_y^{-b_x}$.
  Then $(\llbracket \texttt{x+y} \rrbracket_p.\mu)(\texttt{z} = t) \leq \alpha'_z {\beta'_z}^t$. If $\alpha'_z = \infty$, we take $C'_z = \mathsf{none}$ else we take $C'_z = (\alpha'_z, \beta'_z)$.
- $C_x = (\alpha_x, \beta_x)$ and $C_y = \mathsf{none}$; this is *mutatis mutandis* the previous case.
- $C_x = (\alpha_x, \beta_x)$ and $C_y = (\alpha_y, \beta_y)$; we then apply the previous cases and take the greatest lower bound of both.

4.3 Random Generation

We define here the abstract operation $(W, (a_v, b_v)_{v \in V}, (C_v)_{v \in V}) \mapsto (W', (a'_v, b'_v)_{v \in V}, (C'_v)_{v \in V}) = \rho \texttt{ := random}^\sharp_p.(W, (a_v, b_v)_{v \in V}, (C_v)_{v \in V})$.

Let us recall that $\rho\texttt{:=random}_p.\mu = \mu \otimes \mu_R$ where μ_R is the distribution of the generator. Let us note W_R the total weight of μ_R (it can be less than 1, see §5.3). We take $W' = W_R.W$, and for any variable v except ρ, $a'_v = a_v$ and $b'_v = b_v$; if $C_v = (\alpha_v, \beta_v)$ then $C'_v = (W_R.\alpha_v, \beta_v)$, else $C'_v = \mathsf{none}$. If the generator has an integer output and is bounded in $[a_R, b_R]$, then $a'_\rho = a_R$ and $b'_\rho = b_R$. $C'_\rho = \mathsf{none}$.

4.4 Flow Control

As with preceding operations, we only define the cases that will be actually used for our analysis goals. Other cases are handled by simple interval propagation and setting C_v to none for every modified variable v.

Least Upper Bound We define here the abstract operation $((W, (a_v, b_v)_{v\in V}, (C_v)_{v\in V}), (W', (a'_v, b'_v)_{v\in V}, (C'_v)_{v\in V})) \mapsto (W'', (a''_v, b''_v)_{v\in V}, (C''_v)_{v\in V})$ noted as $(W, (a_v, b_v)_{v\in V}, (C_v)_{v\in V}) \sqcup^\sharp (W', (a'_v, b'_v)_{v\in V}, (C'_v)_{v\in V})$.

Given $(W, (a_v, b_v)_{v\in V}, (C_v)_{v\in V})$ and $(W', (a'_v, b'_v)_{v\in V}, (C'_v)_{v\in V})$, we want $(W'', (a''_v, b''_v)_{v\in V}, (C''_v)_{v\in V})$ so that $\gamma_E(W, (a_v, b_v)_{v\in V}, (C_v)_{v\in V}) \cup \gamma_E(W', (a'_v, b'_v)_{v\in V}, (C'_v)_{v\in V}) \subseteq \gamma_E(W'', (a''_v, b''_v)_{v\in V}, (C''_v)_{v\in V})$ with little loss of precision. Let us take $W'' = \max(W, W')$ and for all v:

- $a''_v = \min(a_v, a'_v)$ and $b''_v = \max(b_v, b'_v)$
- if $C_v = (\alpha_v, \beta_v)$ and $C'_v = (\alpha'_v, \beta'_v)$ then we take $\beta''_v = \max(\beta_v, \beta'_v)$ and $\alpha''_v = \max(\alpha_v \beta_v^{a''_v}, \alpha'_v {\beta'_v}^{a''_v}).{\beta''_v}^{-a''_v}$.
- if $C_v =$ none or $C'_v =$ none then $C''_v =$ none.

Widening As noted before (§3.3), we need some widening operators to approximate least fixpoints.

We shall define here the abstract operation $((W, (a_v, b_v)_{v\in V}, (C_v)_{v\in V}), (W', (a'_v, b'_v)_{v\in V}, (C'_v)_{v\in V})) \mapsto (W'', (a''_v, b''_v)_{v\in V}, (C''_v)_{v\in V})$ noted as $(W, (a_v, b_v)_{v\in V}, (C_v)_{v\in V}) \nabla_E (W', (a'_v, b'_v)_{v\in V}, (C'_v)_{v\in V})$.

$(W, (a_v, b_v)_{v\in V}, (C_v)_{v\in V}) \nabla_E (W', (a'_v, b'_v)_{v\in V}, (C'_v)_{v\in V})$ should be higher than $(W, (a_v, b_v)_{v\in V}, (C_v)_{v\in V}) \sqcup_E (W', (a'_v, b'_v)_{v\in V}, (C'_v)_{v\in V})$. We also require the sequence defined inductively by $\mu'_{n+1} = \mu'_n \nabla \mu^\sharp_n$ to be stationary, for any sequence $(\mu^\sharp_n)_{n\in\mathbb{N}}$. The main interest of such an operator is the computation of least fixpoints (see §3.3.

We shall use a widening operator $\nabla_\mathbb{R}$ on the reals:

- $x \nabla_\mathbb{R} y = \infty$ if $x < y$;
- $x \nabla_\mathbb{R} y = y$ otherwise.

Intuitively, using this operator means that if a sequence of real coefficients keeps of ascending, we get an upper approximation of it using $+\infty$.

Let us take $W'' = \max(W, W')$ and for all v:

- if $a_v > a'_v$, $a''_v = -\infty$ else $a''_v = a_v$;
- if $b_v < b'_v$, $b''_v = +\infty$ else $b''_v = a_v$;
- two cases:
  - If $a''_v = +\infty$ or $C_v =$ none or $C'_v =$ none then $C''_v =$ none.
  - Otherwise, if $C_v = (\alpha_v, \beta_v)$ and $C'_v = (\alpha'_v, \beta'_v)$ then we take $\beta''_v = \exp(-(-\ln \beta_v) \nabla_\mathbb{R} (-\ln \beta'_v))$ and $\alpha''_v = (\alpha_v \beta_v^{a''_v} \nabla_\mathbb{R} \alpha'_v {\beta'_v}^{a''_v})).{\beta''_v}^{-a''_v}$. If $\alpha''_v < \infty$ then $C''_v = (\alpha''_v, \beta''_v)$, otherwise $C''_v =$ none.

Addition We shall define here the abstract operation $((W,(a_v,b_v)_{v\in V},(C_v)_{v\in V}),(W',(a'_v,b'_v)_{v\in V},(C'_v)_{v\in V})) \mapsto (W'',(a''_v,b''_v)_{v\in V},(C''_v)_{v\in V})$ noted as $(W,(a_v,b_v)_{v\in V},(C_v)_{v\in V}) +^\sharp (W',(a'_v,b'_v)_{v\in V},(C'_v)_{v\in V})$.

Two cases are of particular interest:

– For all $t \in \mathbb{Z}$, $\mu(v=t) \leq \alpha_v \beta_v^t$, $t \notin [a_v,b_v] \Rightarrow \mu(v=t)$, $t \notin [a'_v,b'_v] \Rightarrow \mu'(v=t)$ and $\mu'(\text{true}) \leq W'$.
  Two cases:
  - $b'_v < a'_v$; then let us take $\alpha''_v = \max(\alpha_v, W'.\beta_v^{-b'_v})$ and $\beta''_v = \beta_v$, then $\mu + \mu(t=v) \leq \alpha''_v {\beta''_v}^t$ (see Fig. 1 for an example);
  - otherwise, let us take $\alpha''_v = \alpha_v, W' + \beta_v^{b'_v}$ and $\beta''_v = \beta_v$, then $\mu + \mu(t=v) \leq \alpha''_v {\beta''_v}^t$.

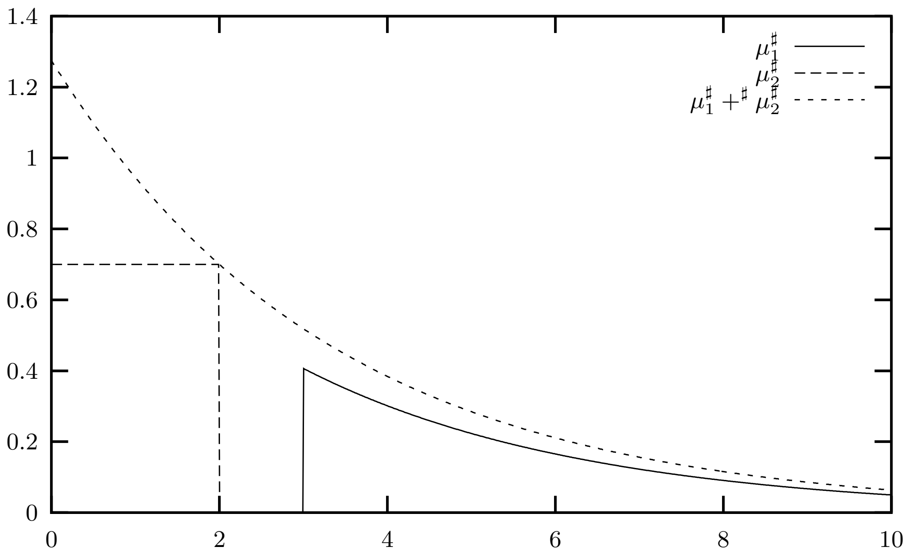

Fig. 1. Abstract addition of measures. $C_x = (1, 0.3)$, $a_x = 3$, $b_x = +\infty$, $W' = 0.7$, $C'_x = \mathsf{none}$, $a'_x = 0$, $b'_x = 2$. For the sake of readability, the discrete distributions are extended to continuous ones. The near-vertical slopes are artefacts of the plotting software replacing vertical slopes.

– For all $t \in \mathbb{Z}$, $t \notin [a_v,b_v] \Rightarrow \mu(v=t)$, $t \notin [a'_v,b'_v] \Rightarrow \mu'(v=t)$, where $a'_v > b_v$. Let us take $\beta''_v = (W/W')^{\frac{1}{b'v-b_v}}$ and $\alpha''_v = W.{\beta''_v}^{b_v}$, then $\mu + \mu(t=v) \leq \alpha''_v {\beta''_v}^t$ (see Fig. 2 for an example).

4.5 Machine Reals in our Abstract Domain

Our abstract domain makes ample use of real numbers: each coefficient α_v or β_v is *a priori* a real number. A possible implementation of these coefficients is

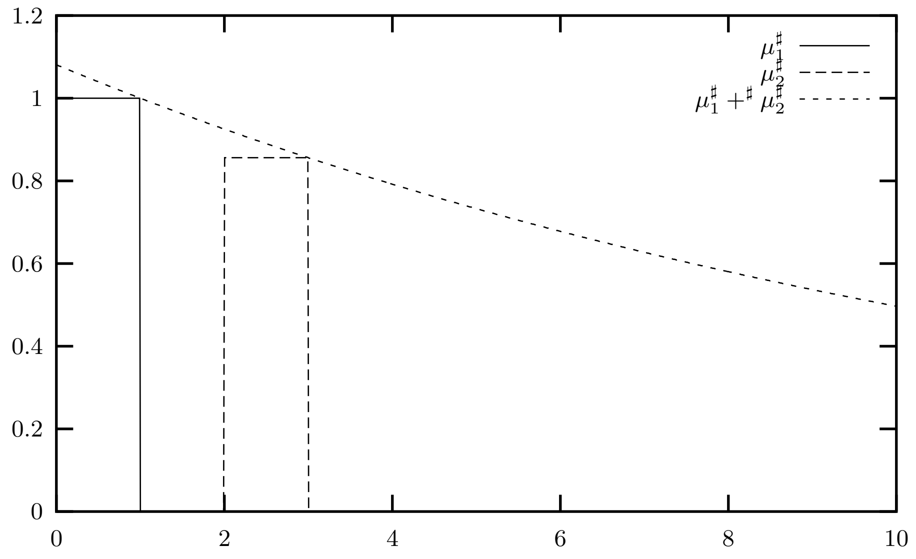

Fig. 2. Abstract addition of measures. $C_x = \mathsf{none}$, $a_x = 0$, $b_x = 1$, $W = 1$, $a'_x = 2$, $b'_x = 3$, $W' = 0.856$.

machine reals (IEEE 754 or similar); another is rational numbers as quotients of arbitrary-precision integers. We discuss here the possible consequences of the use of machine reals for the implementation of those coefficients.

The use of machine reals leads to two implementation problems. The first problem is that it is difficult to ascertain the loss of precision induced by machine reals throughout computations: how can the user be convinced that the output of the analyzer is sound? This can be worked around by using directed rounding modes.

A more annoying problem, relevant in implementation, is the fact that accrued imprecisions may lead to "drift", especially when using directed rounding modes. By "drift", we mean that a sequence of real numbers that, mathematically speaking, should appear to be stationary may be strictly ascending in the floating-point approximation. In that case, our widening operator on the reals $\nabla_{\mathbb{R}}$ (§4.4) may jump prematurely to $+\infty$.

It may be desirable to consider that small changes in some real coefficients do not indicate that the coefficient is actually changing but rather indicate some loss of precision. We expect the current work on abstract domains for real numbers [10] to provide better solutions to that problem.

5 Abstract Domain of Finite Sums

The preceding domain is not yet very suitable to handle random generation. In this section, we lift it to the domain of its finite sums. This is similar to [8, section 4].

5.1 Definition

We shall define another domain S and another concretization function γ_S. S consists of finite tuples of elements of E or, more interestingly, of a reduced product [2, §4.2.3.3] P of E and another domain D. For our subsequent examples, we shall use for P the product of E and the lattice of real intervals (for real variables). γ_P is the concretization function for this reduced product: $\gamma_P(e^\sharp, d^\sharp) = \gamma_E(e^\sharp) \cap \gamma_D(d^\sharp)$.

As a running example we shall use for D the lifted domain of real intervals: to each real variable v we attach an interval $[a_v, b_v]$, with possibly infinite bounds. The probability that variable v is outside $[a_v, b_v]$ is zero.

Items of S are therefore finite tuples of elements of P. The concretization function γ_S is defined as follows: $\mu \in \gamma_S(p_1^\sharp, ..., p_n^\sharp)$ if and only if there exist $\mu_1 \in \gamma_P(p_1)$, ..., $\mu_n \in \gamma_P(p_n)$ so that $\mu = \sum_{k=1}^{n} \mu_k$.

5.2 Arithmetic Operations

Deterministic Operations Basic operations are handled by linearity: since for any program construct P its semantics $[\![P]\!]_p$ is a linear operator, it follows that if $\mu \in \gamma_S(p_1^\sharp, ..., p_n^\sharp)$ then $[\![P]\!]_p.\mu \in \gamma_S([\![P]\!]_p^\sharp.p_1^\sharp, \ldots, [\![P]\!]_p^\sharp.p_n^\sharp)$.

The abstract operator is therefore constructed:

$$[\![P]\!]_p^\sharp(p_1^\sharp, ..., p_n^\sharp) = ([\![P]\!]_p^\sharp.p_1^\sharp, \ldots, [\![P]\!]_p^\sharp.p_n^\sharp).$$

5.3 Random Operations

Let us consider a random generator G operating according to the distribution μ_R. The semantics of the random operation is $[\![\texttt{random}]\!]_p.\mu = \mu \otimes \mu_R$ (Equ. 1). We shall suppose that our underlying domain P has an abstract operation $\otimes^\sharp$.

Let us suppose that $\mu_R = \sum_{i=1}^{N} \mu_i$. Then $\mu \otimes \mu_R = \sum_{i=1}^{N} \mu \otimes \mu_i$ thus if $\mu \in \gamma_S([\![P]\!]_p^\sharp.p_1^\sharp, \ldots, [\![P]\!]_p^\sharp.p_n^\sharp)$ then $\mu \otimes \mu_R \in \gamma_S([\![P]\!]_p^\sharp.p_1^\sharp \otimes^\sharp \mu_1^\sharp, \ldots, [\![P]\!]_p^\sharp.p_1^\sharp \otimes^\sharp \mu_N^\sharp, \ldots, [\![P]\!]_p^\sharp.p_n^\sharp \otimes^\sharp \mu_1^\sharp, \ldots, [\![P]\!]_p^\sharp.p_n^\sharp \otimes^\sharp \mu_N^\sharp)$.

As an example, let us consider the case of an uniform random generator in $[0, 1]$. Let us "cut" it into N equal sub-segments: let us note μ_R the uniform measure on $[0, 1]$. $\mu_R = \sum_{i=1}^{N} \mu_i$ with $\mu_i(X) = \mu_R(X \cap [(i-1)/N; i/N])$. For the abstraction of $\mu \mapsto \mu \otimes \mu_i$ over our example for the domain P: $(e, (d_1, \ldots, d_n)) \mapsto (e, (d_1, \ldots, d_n, [(i-1)/N; i/N])$.

5.4 Flow Control

$[\![\texttt{while } c \texttt{ do } e]\!]^\sharp(W^\sharp) = \phi^\sharp_{[\![c]\!]^C}(\text{lfp}^\sharp\, X^\sharp \mapsto W^\sharp \sqcup [\![e]\!]^\sharp(\phi^\sharp_{[\![c]\!]}(X^\sharp)))$.

Addition Addition is very simple:

$$(p_1, \ldots, p_n) +^\sharp (p'_1, \ldots, p'_{n'}) = (p_1, \ldots, p_n, p'_1, \ldots, p'_{n'}) \tag{9}$$

Loops For accuracy reasons, we wish $[\![\texttt{while } b \texttt{ do } e]\!]^\sharp.(p^\sharp_1, \ldots, p^\sharp_n)$ to operate separately on each component $p^\sharp_1, \ldots, p^\sharp_n$. We thus define an auxiliary function $H^\sharp$ so that $[\![\texttt{while } b \texttt{ do } e]\!]^\sharp.(p^\sharp_1, \ldots, p^\sharp_n) = (H^\sharp.p^\sharp_1, \ldots, H^\sharp.p^\sharp_n)$. $H^\sharp$ is defined following equation 6:

$$H^\sharp(W^\sharp) = \phi^\sharp_{[\![c]\!]^C}(\text{lfp}^\sharp\, X^\sharp \mapsto \mathsf{merge}(W^\sharp \sqcup W^\sharp +^\sharp [\![e]\!]^\sharp(\phi^\sharp_{[\![c]\!]}(X)))).$$

where $\mathsf{merge}(p_1, \ldots, p_n) = (p_1 +^\sharp \cdots +^\sharp p_n)$.

We construct the "approximate least fixpoint operation" $\text{lfp}^\sharp(f^\sharp)$ as follows: we consider the sequence $u^\sharp_0 = (0)$, $u^\sharp_{n+1} = u^\sharp_n \nabla_P f^\sharp(u^\sharp_n)$. All elements of the sequences are 1-tuples of elements of P. If ∇_P is a widening operator for the domain P, then this sequence is stationary, and its limit is an approximation of the least fixpoint of $f^\sharp$.

Such a ∇_P operator can be defined as follows: $(e, d)\nabla_P(e', d') = (e\nabla_E e', d\nabla_D d')$ where ∇_E is the widening operator defined at §4.4.

6 Examples and Applications

6.1 Proof of Probabilistic Termination

Let us consider the following program:

```
double x, y;
int k = 0;
do
{
  x = uniform()+uniform();
  y = uniform();
  k++;
}
while(x < 1.4 || y < 0.2)
```

`uniform()` is assumed to be a random generator uniformly distributed in $[0, 1]$. In this simple case, the probability that the loop is taken is independent of the input data and of the number of iterations done so far. Furthermore, we can compute it mathematically (0.856), with our knowledge of the distribution of the random variables and the computation of an integral.

Let us examine here what happens if we apply the above method, dividing the random generator into $N = 20$ sub-segments of equal length (§5.3). At the end of the first iteration, after the merge, the analyzer establishes that the probability of reaching 1 iteration is less than 0.8805.[2] Applying the last case of §4.4, we obtain $a_{\mathtt{k}} = 1$, $b_{\mathtt{k}} = 2$, $\beta_{\mathtt{k}} = 0.8805$, $\alpha_{\mathtt{k}} \approx 1.1357$. After another iteration, we obtain $a_{\mathtt{k}} = 1$, $b_{\mathtt{k}} = 3$, $\beta_{\mathtt{k}} = 0.8805$, $\alpha_{\mathtt{k}} \approx 1.1357$. The analyzer widens to $a_{\mathtt{k}} = 1$, $b_{\mathtt{k}} = \infty$, $\beta_{\mathtt{k}} = 0.8805$, $\alpha_{\mathtt{k}} \approx 1.1357$, which is stable.

We have therefore established that the probability that $\mathtt{k} = x$ at the beginning of the body of the loop is less than 0.8805^{k-1}. That is of course not as good as the exact computation, but still offers a reasonable bound.

6.2 Statistical Inference of Average Running Times

It is often needed to derive statistical facts such as average running times for real-time systems. Results achieved by the preceding method can be too rough to give precise estimates; nevertheless, they can help in making mathematically accurate some experimental statistical results.

Intuitively, to get the average running time of a system, we should take a large number n of random samples $(x_k)_{1 \leq k \leq n}$ for the input (distributed according to the supposed or inferred distribution on the inputs of the system). Let us call $R(x)$ the running time on input x; we then hope to have an approximation of the means $\bar{R}$ by taking $\frac{1}{n} \sum_{k=1}^{n} R(x_k)$.

Unfortunately, this method is not mathematically sound, essentially because R is not bounded, or rather because we have no information as to the very large values of R (the "tail" of the distribution). For instance, let us suppose that R is 0 for 99.99% of the input values and V for 0.01% of the input values. With 90% probability, the experimental average obtained by the Monte-Carlo method using 1000 samples will be 0, whereas the correct value is $V/10000$. As V can be arbitrarily large, this means we cannot gain any confidence interval on the result.

On the other hand, if, using the analysis described in this paper, we can show that the tail of the distribution decreases exponentially, we can get a confidence bound. Let us note $\mathbb{P}(A)$ the probability of an event A. Let us suppose that our analysis has established that the probability of running a loop at least k times is less than $\alpha.\beta^k$. Let $N \geq 1$. Let us run the program n times, stopping each time when either the loop terminates or it has been executed N times. Let us consider the random variable R where $R(x)$, the number of iterations the program takes when run on input x. We wish to determine $\bar{R} = \sum_{k=0}^{\infty} k.\mathbb{P}(R = k)$. To achieve this, we split the sum in two parts: $\bar{R} = \bar{R}_{\leq N} + \bar{R}_{>N}$ where $\bar{R}_{\leq N} = \sum_{k=0}^{N} k.\mathbb{P}(R = k)$ and $\bar{R}_{>N} = \sum_{k=N+1}^{\infty} k.\mathbb{P}(R = k)$.

[2] This result has been computed automatically by the Absinthe abstract interpreter `http://cgi.dmi.ens.fr/cgi-bin/monniaux/absinthe`.

We wish to obtain an upper bound on $\bar{R}_{>N}$.

$$\sum_{k=a}^{b} k.\beta^k = \frac{1}{(\beta-1)^2}.\left[\beta^{b+1}\left((b+1)(\beta-1)-\beta\right)-\beta^a\left(a(\beta-1)-\beta\right)\right] \quad (10)$$

and thus

$$\sum_{k=a}^{\infty} k\beta^k = \frac{\beta^a}{(\beta-1)^2}.\left(\beta - a(\beta-1)\right) \quad (11)$$

Therefore $\bar{R}_{>N} \leq \alpha.\frac{\beta^{N+1}}{(\beta-1)^2}.\left(\beta-(N+1)(\beta-1)\right)$.

On the other hand, $\bar{R}_{<N}$ can be estimated experimentally [9]. Let us consider the random variable R^*: $R^*(x) = 0$ if $R(x) > N$ and $R^*(x) = R(x)/N$ if $R(x) < N$; this is a random variable whose range is a subset of $[0, 1]$. It is therefore possible to apply to R^* the various techniques known for estimating the expectation of such random variables (such as Chernoff-Hoeffding bounds [4] or simple Gaussian confidence intervals, using the Central Limit Theorem). We then obtain $\bar{R}_{<N} = N.\bar{R}^*$.

We therefore obtain a confidence upper bound on $\bar{R}$ by adding together the confidence intervals obtained on $\bar{R}_{\leq N}$ and $\bar{R}_{>N}$.

7 Conclusions and Future Work

We have proposed a method to analyze automatically the termination of certain kinds of computer programs. This analysis outputs probabilistic invariants of loops, especially on the distribution of the number of loop iterations done or the number of program cycles used. The invariants obtained can be used in several ways, including:

– proofs of probabilistic termination,
– proofs of soundness of experimental statistical methods.

The analysis is proved to be sound, as it is set in a framework of abstract interpretation suitable for probabilistic programs. It has been partially implemented in an automatic analyzer for a subset of the C programming language. We plan to implement it fully in an analyzer for industrial embedded systems, where it could serve as a quick method to validate active waiting loops and similar constructs.

This analysis deals with a particular kind of probabilistic distributions, for example typical of the running time of a program waiting for data with probabilistic timings on a network. Other kinds of distributions, such as the normal (Gaussian) one, are of particular interest when analyzing embedded systems — for instance, to model error propagation. We plan to give analyses suitable for such distributions.

References

1. P. Cousot and R. Cousot. Abstract intrepretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the 4th ACM Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, CA, January 1977.
2. Patrick Cousot and Radhia Cousot. Abstract interpretation and application to logic programs. *J. Logic Prog.*, 2-3(13):103–179, 1992.
3. J.L. Doob. *Measure Theory*, volume 143 of *Graduate Texts in Mathematics*. Springer-Verlag, 1994.
4. Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *J. Amer. Statist. Assoc.*, 58(301):13–30, 1963.
5. 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.
6. D. Kozen. Semantics of probabilistic programs. *Journal of Computer and System Sciences*, 22(3):328–350, 1981.
7. Chin Soon Lee, Neil D. Jones, and Amir M. Ben-Amram. The size-change principle for program termination. In *ACM Symposium on Principles of Programming Languages*, volume 28, pages 81–92. ACM press, January 2001.
8. 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. Extended version on the author's web site.
9. 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)*, pages 93–101. Association for Computer Machinery, 2001.
10. Éric Goubault. Static analyses of floating-point operations. In *Static Analysis (SAS '01)*, Lecture Notes in Computer Science. Springer-Verlag, July 2001.
11. Walter Rudin. *Real and Complex Analysis*. McGraw-Hill, 1966.