Abstract interpretation of probabilistic semantics

David Monniaux

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

Abstract. Following earlier models, we lift standard deterministic and nondeterministic semantics of imperative programs to probabilistic semantics. This semantics allows for random external inputs of known or unknown probability and random number generators.
We then propose a method of analysis of programs according to this semantics, in the general framework of abstract interpretation. This method lifts an "ordinary" abstract lattice, for non-probabilistic programs, to one suitable for probabilistic programs.
Our construction is highly generic. We discuss the influence of certain parameters on the precision of the analysis, basing ourselves on experimental results.

1 Introduction

In this paper, we give both a theoretical framework for abstract semantics of probabilistic computer programs and practical methods of analysis. Our analyses are set in the general field of abstract interpretation.

1.1 Abstract Interpretation

A well-known fact of computer science is that properties of the denotational semantics of programs in Turing-complete languages cannot be decided mechanically. Automatic methods of analysis thus have to forget completeness, while still yielding interesting results on realistic programs.

The basic idea behind abstract interpretation [1,2] is to replace computations on sets that are non recursive or too complex to handle by computations on supersets of them (or subsets; what is important is we know whether we are handling a superset or a subset). For instance, instead of handling sets of integers, one might want to upper-approximate them using an interval. If all computations are done monotonically, the result interval is necessarily a superset of the exact set of possible values at the end of execution.

Intervals are just one of many possible abstract domains. For tuples of numerical values, polyhedra can be used [3]. Tuples of integers can be abstracted using congruences, interval congruences [13] or systems of linear congruential equations [6]. Appropriate domains can be used to discover data structure configurations [4].

1.2 Probabilistic Semantics

One of the drawbacks of such analysis methods is that they do not distinguish between what is possible (even with extremely low probability) and what is likely (possible with a non-negligible probability). This is especially important in the analysis of reactive systems.

Let us take a real-life example: the monitoring program of a copy machine system is a reactive program taking inputs for sensors and giving orders to servo-motors. Each sensor has a probability of failure, from mechanical or electric wearing, scraps of papers etc... Sensors are redundant, with the idea that if a moderate amount of sensors are failing, the system will diagnosis the sensor failure instead of getting a false idea of what is going on in the machine. It is possible that failure of several sensors can make the system err. The reliability of the system can be improved by increasing the number of sensors, which is not always economically and mechanically possible. It is interesting, given a description of the system, to get upper bounds on the probability of failure.

Another field of possible use of analysis is randomized algorithms. Randomized algorithms have enjoyed considerable interest [9]. While it is of course impossible to derive automatically the most advanced properties of some of these algorithms, it is still interesting to be able to deal with such programs in a more precise way than just considering them as nondeterministic.

It is required that the analysis method should not constrain the analyzed programs in a class of well-studied algorithms. Also, we must allow all usual flow-control constructs, including tests and loops.

1.3 Comparison to Other Works

The concrete semantics we consider is essentially equivalent to the one proposed by Kozen [10,11, second semantic]; we do not consider structures as complex as those proposed by Jones [9]. We extend this semantics to nondeterministic probabilistic cases [12,8].

Contrary to some other works [8,12,14,15], our goal is not to propose rules to reason on à la Dijkstra weakest precondition semantics and prove refinements. These methods are adequate for computer-aided program design and verification, but of course cannot deal automatically with loop invariants. We rather propose a natural extension of abstract interpretation [1,2] to probabilistic semantics. The analyses described here are meant to be fully automatic, even though some heuristics need some tuning guided by experience.

Some automatic program analysis techniques with a view on improving optimizing compilers have been developed [17]. These techniques are essentially ad hoc and imprecise; only the control flow is considered, the probability of tests being taken or not being estimated from crude syntactic criteria (such as: a branch whose condition is a conjunction is less likely to be taken than a branch whose condition is atomic, tests checking for null pointers are not likely to be taken...). While such techniques are interesting in heuristics for compilers, they are not suitable to get any precise result on programs.

1.4 Notations

$\mathcal{P}(X)$ shall be the set of parts of X. Y^C shall be the complement of X if there is no ambiguity as to the superset.

2 Probabilistic Concrete Semantics

Throughout this paper, we shall define compositionally several semantics and expose relationships between them. We shall use as an example some simple Pascal-like imperative language, but we do not mean that our analysis methods are restricted to such languages.

2.1 Summary of Non-probabilistic Concrete Semantics

We shall here consider denotational semantics for programs. (equivalent operational semantics could be easily defined, but we shall mostly deal with denotational ones for the sake of brevity).

The language is defined as follows: the compound program instructions are

instruction ::= *elementary*
instruction `;` *instruction*
`if` *boolean_expr* `then` *instruction* `else` *instruction* `endif`
`while` *boolean_expr* `do` *instruction* `done`

and the boolean expressions are defined as

boolean_expr ::= *boolean_atomic*
boolean_expr `and` *boolean_expr*
boolean_expr `or` *boolean_expr*
`not` *boolean_expr*

elementary instructions are deterministic, terminating basic program blocks like assignments and simple expression evaluations. *boolean_atomic* boolean expressions, such as comparisons, have semantics as sets of "acceptable" environments. For instance, a *boolean_atomic* expression can be `x < y + 4`; its semantics is the set of execution environments where variables `x` and `y` verify the above comparison. If we restrict ourselves to a finite number n of integer variables, an environment is just a n-tuple of integers.

The denotational semantics of a code fragment c is a mapping from the set X of possible execution environments before the instruction into the set Y of possible environments after the instruction. Let us take an example. If we take environments as elements of $\mathbb{Z}^3$, representing the values of three integer variables `x`, `y` and `z`, then $[\![\texttt{x:=y+z}]\!]$ is the function $\langle x, y, z\rangle \mapsto \langle y+z, y, z\rangle$. Semantics of basic constructs (assignments, arithmetic operators) can be easily dealt with this way; we shall now see how to deal with flow control.

The semantics of a sequence is expressed by simple composition

$$[\![e_1\texttt{;}\ e_2]\!] = [\![e_2]\!] \circ [\![e_1]\!]$$

Tests get expressed easily, using as the semantics $[\![c]\!]$ of a boolean expression c the set of environments it matches:

$$[\![\texttt{if}\ c\ \texttt{then}\ e_1\ \texttt{else}\ e_2]\!](x) = \text{if } x \in [\![c]\!] \text{ then } [\![e_1]\!](x) \text{ else } [\![e_2]\!](x)$$

and loops get the usual least-fixpoint semantics (considering the point-wise extension of the Scott flat ordering on partial functions)

$$[\![\texttt{while}\ c\ \texttt{do}\ f]\!] = \operatorname{lfp} \lambda\phi.\lambda x.\text{if } x \in [\![c]\!] \text{ then } \phi \circ [\![f]\!](x) \text{ else } x.$$

Non-termination shall be noted by $\perp$.

2.2 Our Framework for Probabilistic Concrete Semantics

We shall express probabilities using **measures** [16, §1.18]. We shall begin by a few classical mathematical definitions.

Measures The basic objects we shall operate on are measures.

- A ***σ*-algebra** is a set of subsets of a set X that contains $\emptyset$ and is stable by countable union and complementation (and thus contains X and is stable by countable intersection). For technical reasons, not all sets can be measured (that is, given a probability) and we have to restrict ourselves to some sufficiently large σ-algebras, such as the Borel or Lebesgue sets [16].
- 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 **sub-probability measure** has total weight less or equal to 1. We shall note $\mathcal{M}_{\leq 1}(X)$ the sub-probability measures on X.
- Given two sub-probability measures μ and μ' (or more generally, two σ-finite measures) on X and X' respectively, we note $\mu \otimes \mu'$ the product measure [16, 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, of norm less than 1, using the Banach norm of total variation on measures. This is necessary to ensure the mathematical well-formedness of certain definitions, such as the concrete semantics of loops. As the definitions for these concepts and some mathematical proofs for the definition of the concrete semantics are quite long and not relevant at all to the analysis, we shall omit them from this paper and refer the reader to an extended version. As a running example for the definitions of the semantics, we shall use a program with real variables `x`, `y` and `z`; the set of possible environments is then $\mathbb{R}^3$.

General form 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. Let us first remark that this condition happens, for instance, for any continuous function from X and Y if both are topological spaces and σ_Y is the Borel σ-algebra [16, §1.11]. $[\![\texttt{x := y+z}]\!] = \langle x, y, z\rangle \mapsto \langle y+z, y, z\rangle$ is continuous.

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

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

We shall see that all flow control constructs "preserve" measurability; i.e., if all sub-blocks of a complex construct have measurable semantics, then the construct shall have measurable semantics. We shall then extend the framework to programs containing `random`-like operators; their semantics will be expressed as linear operators of norm less than 1 on measure spaces.

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 equidistributed 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:[2]

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

[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.

[2] Another equivalent way, used by Kozen [10,11], is to consider random values as countable streams in the input environment of the program.

The second step is a simple assignment operator, addressed by the above method. The first step boils down to measure products:

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

Tests and Loops We restrict ourselves to test and loop conditions b so that $[\![\texttt{b}]\!]$ is measurable. This condition is fulfilled if all the *boolean_atomic* sets are measurable since the σ-algebra is closed by finite union and intersection. For instance, $[\![\texttt{x < y}]\!] = \{\langle x, y, z\rangle \mid x < y\}$ is measurable.

The deterministic semantics for tests are:

$$[\![\texttt{if } c \texttt{ then } e_1 \texttt{ else } e_2]\!](x) = \text{if } x \in [\![x]\!] \text{ then } [\![e_1]\!](x) \text{ else } [\![e_2]\!](x).$$

Let us first compute

$$[\![\texttt{if } c \texttt{ then } e_1 \texttt{ else } e_2]\!]^{-1}(W) = ([\![e_1]\!]^{-1}(W) \cap [\![c]\!]) \cup ([\![e_2]\!]^{-1}(W) \cap [\![c]\!]^C).$$

$[\![\texttt{c}]\!]$ is the set of environments matched by condition c. It is obtained inductively from the set of environment matched by the atomic tests (e.g. comparisons):

- $[\![c_1 \texttt{ or } c_2]\!] = [\![c_1]\!] \cup [\![c_2]\!]$
- $[\![c_1 \texttt{ and } c_2]\!] = [\![c_1]\!] \cap [\![c_2]\!]$
- $[\![\texttt{not } c]\!] = [\![c]\!]^C$

Using our above framework to lift deterministic semantics to probabilistic ones, we get

$$\begin{aligned} [\![\texttt{if } c \texttt{ then } e_1 \texttt{ else } e_2]\!]_p(\mu) &= X \mapsto \mu([\![\texttt{if } c \texttt{ then } e_1 \texttt{ else } e_2]\!]^{-1}(X)) \\ &= X \mapsto \mu(([\![e_1]\!]^{-1}(X) \cap [\![c]\!]) \cup ([\![e_2]\!]^{-1}(X) \cap [\![c]\!]^C) \\ &= X \mapsto \mu([\![e_1]\!]^{-1}(X) \cap [\![c]\!]) + \mu([\![e_2]\!]^{-1}(X) \cap [\![c]\!]^C) \\ &= [\![e_1]\!]_p \circ \phi_{[\![c]\!]}(\mu) + [\![e_2]\!]_p \circ \phi_{[\![c]\!]^C}(\mu) \end{aligned} \quad (1)$$

where $\phi_W(\mu) = \lambda X.\mu(X \cap W)$.

We lift in the same fashion the semantics of loops (we note $\sqcup$ an union of pairwise disjoint subsets of a set):

$$\begin{aligned} &[\![\texttt{while } c \texttt{ do } e]\!]^{-1}(X) \\ &\quad = (\text{lfp}\, \lambda\phi.\lambda x.\text{if } x \in [\![c]\!] \text{ then } \phi \circ [\![e]\!](x) \text{ else } x)^{-1}(X) \\ &\quad = \bigsqcup_{n \in \mathbb{N}} (\lambda Y.[\![e]\!]^{-1}(Y) \cap [\![c]\!])^n (X \cap [\![c]\!]^C) \end{aligned} \quad (2)$$

We therefore derive the form of the probabilistic semantics of the `while` loop:

$$
\begin{aligned}
[\![\texttt{while } c \texttt{ do } e]\!]_p(\mu) &= \lambda X.\mu\left(\bigsqcup_{n\in\mathbb{N}}(\lambda Y.[\![e]\!]^{-1}(Y)\cap[\![c]\!])^n(X\cap[\![c]\!]^C)\right)\\
&= \lambda X.\sum_{n\in\mathbb{N}}\mu\left((\lambda Y.[\![e]\!]^{-1}(Y)\cap[\![c]\!])^n(X\cap[\![c]\!]^C)\right)\\
&= \sum_{n=0}^{\infty}\phi_{[\![c]\!]^C}\circ([\![e]\!]_p\circ\phi_{[\![c]\!]})^n(\mu)\\
&= \phi_{[\![c]\!]^C}\left(\sum_{n=0}^{\infty}([\![e]\!]_p\circ\phi_{[\![c]\!]})^n(\mu)\right)\\
&= \phi_{[\![c]\!]^C}\left(\lim_{n\to\infty}(\lambda\mu'.\mu+[\![e]\!]_p\circ\phi_{[\![c]\!]}(\mu'))^n(\lambda X.0)\right) \qquad (3)
\end{aligned}
$$

Limits and infinite sums are taken according to the set-wise topology. We refer the reader to an extended version of this paper for the technical explanations on continuity and convergence.

2.3 Probabilities and Nondeterminism

It has been pointed out [12,8] that we must distinguish deterministic and nondeterministic probabilistic semantics. Deterministic, non-probabilistic semantics embed naturally into the above probabilistic semantics: instead of a value $x \in X$, we consider the Dirac measure $\delta_x \in \mathcal{M}_{\leq 1}(X)$ defined by $\delta_x(X) = \begin{cases} 1 & \text{if } x \in X \\ 0 & \text{otherwise.}\end{cases}$ How can we account for nondeterministic non-probabilistic semantics?

We move from deterministic to nondeterministic semantics by lifting to powersets. It is possible to consider nondeterministic probabilistic semantics: the result of a program is then a set of probability measures. Of course, nondeterministic non-probabilistic semantics get embedded naturally: to $A \in \mathcal{P}(X)$ we associate $\{\delta_a \mid a \in A\} \in \mathcal{P}(\mathcal{M}_+(X))$. We therefore consider four semantics:

determinism	nondeterminism
probabilistic	nondeterministic probabilistic

The advantage of probabilistic nondeterminism is that we can consider programs whose inputs are not all distributed according to a distribution, or whose distribution is not exactly known. Our analysis is based on probabilistic nondeterminism and thus handles all cases.

3 Abstract Semantics

We shall first give the vocabulary and notations we use for abstractions in general. We shall then proceed by giving an example of an domain that abstracts probabilistic semantics as defined in the previous section. This domain is parametric in multiple ways, most importantly by the use of an abstract domain for the non-probabilistic semantics of the studied system.

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

$$\forall x^\sharp \in X^\sharp \; \forall x \in X \; x \in \gamma_X(x^\sharp) \Rightarrow f(x) \in \gamma_Y(f^\sharp(x^\sharp)) \tag{4}$$

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

$$\forall x^\flat \in X^\flat \; \forall x^\sharp \in X^\sharp \; x^\flat \sqsubseteq \gamma_X(x^\sharp) \Rightarrow f^\flat(x^\flat) \sqsubseteq \gamma_X(f^\sharp(x^\sharp)) \tag{5}$$

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 $T = \{(a,b) \in \mathbb{Z} \cup \{-\infty,+\infty\} \mid a \leq b\} \cup \{\bot\}$, $\gamma(a,b) = \{c \in \mathbb{Z} \mid a \leq c \leq b\}$ and γ induces a preorder $\sqsubseteq_T$ over T and, pointwise, 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 Probabilistic Abstraction

The concrete probabilistic domains given in 2.2 can be abstracted as in the previous definition. Interesting properties of such an abstraction would be, for instance, to give an upper bound on the probability of some subsets of possible environments at the end of a computation.

3.3 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 $[\![\mathtt{while}\ c\ \mathtt{do}\ e]\!]_p(\mu) = \phi_{[\![c]\!]^C}(\lim_{n\to\infty} \mu_n)$. We wish to approximate this limit in the measure space by an abstract element.

We shall use the following method: to get an approximation of the limit of a sequence $(u_n)_{n\in\mathbb{N}}$ defined recursively by $u_{n+1} = f(u_n)$, we can find a closed set

S stable by f so that $u_N \in S$ for some N; then $\lim_{n\to\infty} u_n \in S$. Let us note than finding such a set does not prove that the limit exists; we have to suppose that f is such that this limit exists. In our case, this condition is necessarily fulfilled.

Let us take $\mu^\sharp$ and $\mu^\sharp{}_0$ respective abstractions of μ. Let us call $\psi^\sharp(\nu^\sharp) = \mu^\sharp +^\sharp [\![e]\!]_p{}^\sharp \circ \phi^\sharp{}_{[\![c]\!]}(\nu^\sharp)$. Let us take a certain $N \in \mathbb{N}$ and call $L^\sharp = \mathrm{lfp}\,\lambda\nu^\sharp.\psi^{\sharp N}(\mu_0{}^\sharp) \sqcup \psi^\sharp(\nu^\sharp)$; then by induction, for all $n \geq N$, $\mu_n \in \gamma(L^\sharp)$. As $\gamma(L^\sharp)$ is topologically closed, $\lim n \to \infty \in \gamma(L^\sharp)$. Therefore $L^\sharp$ is an approximation of the requested limit.

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 $f^\sharp : X^\sharp \to X^\sharp$ is monotonic, then, noting $x_0^\sharp = \mathrm{lfp}^\sharp(f)$, $f^\sharp(x_0^\sharp) \sqsubseteq x_0^\sharp$. The justification of our appellation, and the interest of such a function, lies in the following well-known result:

Lemma 1. *If $f^\sharp : X^\sharp \to X^\sharp$ is an abstraction of $f^\flat : \mathcal{P}(X) \to \mathcal{P}(X)$ and $f^\sharp(x_0^\sharp) \sqsubseteq x_0^\sharp$, then $\mathrm{lfp}\, f^\flat \subseteq \gamma_X(x_0^\sharp)$.*

Of course, building such an operation is not easy. Trying the successive iterations of $f^{\sharp n}$ until reaching a fixpoint does not necessarily terminate. One has to use special tricks and widening operators to build such a function (see 5.3).

Provided we have such an operation, abstraction follows directly:

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

As usual in abstract interpretation, it might be wise to do some semantics-preserving transformations on the program, such as unrolling the first few turns of the loop, before applying this abstraction. This is likely to yield better results.

4 A Probabilistic Abstract Domain

As considerable effort has been put into the design and implementation of non-probabilistic abstract domains (see §1.1 for examples), it would be interesting to be able to create probabilistic abstract domains from these. In the this section, we shall give such a generic construction.

4.1 The Intuition Behind the Method

A finite sequence A_i of disjoint measurable subsets of X and corresponding coefficients $\alpha_i \in \mathbb{R}_+$, represent the set of measures μ so that:

- μ is concentrated on $\bigcup A_i$
- for all i, $\mu(A_i) \leq \alpha_i$.

For practical purposes, the A_i are concretizations of abstract elements, polyhedra for instance (Fig. 1).

This abstraction is intuitive, but lifting operations to it proves difficult: the constraint that sets must be disjoint is difficult to handle in the presence of non injective semantics $[\![c]\!]$. This is the reason why we rather consider the following

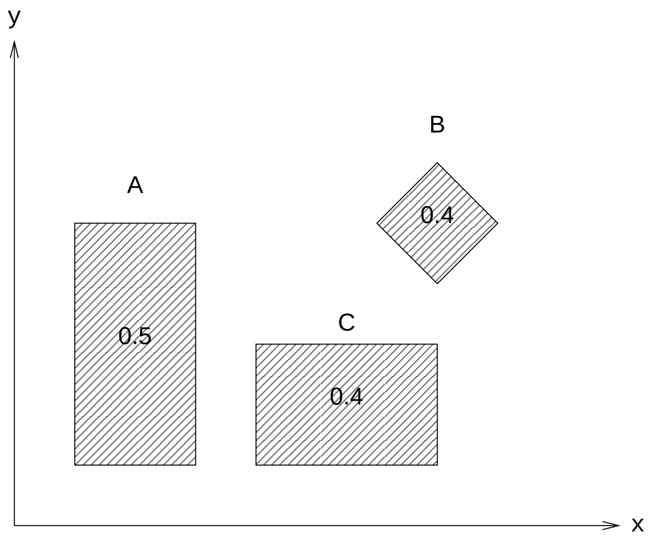

Fig. 1. An abstract value representing measures μ so that $\mu(A) \leq 0.5$, $\mu(B) \leq 0.4$ and $\mu(C) \leq 0.4$.

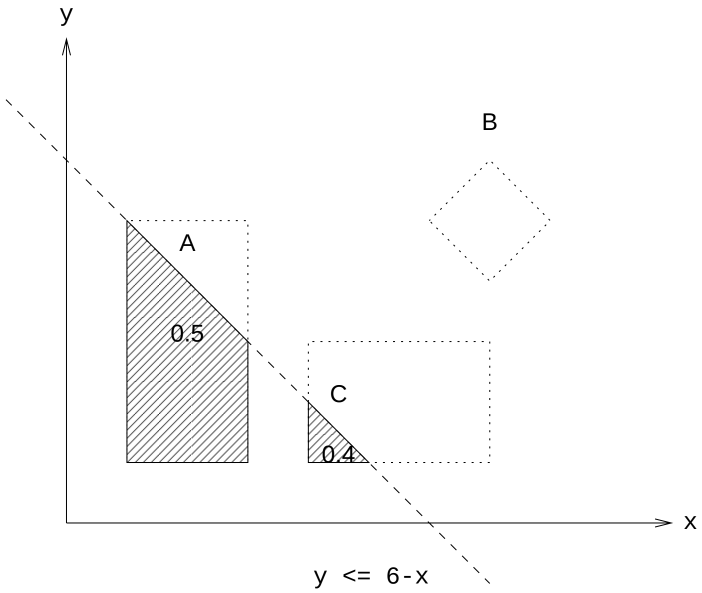

Fig. 2. The abstract value of Fig. 1 after going into the first branch of a `if y<=6-x...` test.

definition: a finite sequence A_i of (non-necessarily disjoint) measurable subsets of X and corresponding coefficients $\alpha_i \in \mathbb{R}_+$ represent the set of measures μ so that there exist measures μ_i so that:

- $\mu = \sum \mu_i$;
- for all i, μ_i is concentrated on A_i;
- for all i, $\mu(A_i) \leq \alpha_i$.

We shall see how to formalize this definition and how program constructs act on such abstract objects.

4.2 Theoretical Construction

Let us take an indexing set Λ, an abstraction (see §3.1) $\Gamma_X = \langle \sigma_X, X^\sharp, \gamma_X \rangle$ and an abstraction $\Gamma_W = \langle \mathcal{P}([0,1]^\Lambda), W^\sharp, \gamma_W \rangle$. We define an abstraction $\Gamma_{\Lambda,\Gamma_X,\Gamma_W} = \langle \mathcal{C}(X_p), S_{\Lambda,\Gamma_X,\Gamma_W}, \gamma_{\Lambda,\Gamma_X,\Gamma_W} \rangle$. $\mathcal{C}(X_p)$ is the set of closed sets of the topological space X_p for the set-wise topology [5, §III.10] — this is a technical requirement that is easy to fulfill. We wish to define compositionally abstract semantics for our language (defined in §2.1). We shall omit the $\Lambda, \Gamma_X, \Gamma_W$ subscript if there is no ambiguity.

Domain Let $S_{\Lambda,\Gamma_X,\Gamma_W} = {X^\sharp}^\Lambda \times W^\sharp$ be our abstract domain. We then define $\gamma_{\Lambda,\Gamma_X,\Gamma_W} : S_{\Lambda,\Gamma_X,\Gamma_W} \to \mathcal{P}(X_p)$ so that $((Z_\lambda)_{\lambda\in\Lambda}, w)$ maps to the set of measures $\mu \in \mathcal{M}_{\leq 1}(X)$ so that there exist measures $(\mu_\lambda)_{\lambda\in\Lambda}$ so that

- for each $\lambda \in \Lambda$, μ_λ is concentrated on $\gamma_X(Z_\lambda)$;
- the family $(\int d\mu_\lambda)_{\lambda\in\Lambda}$ of total weights of those measures is in $\gamma_W(w)$.

Regular constructs Given two such constructions $\Gamma_{\Lambda,\Gamma_X,\Gamma_W}$ and $\Gamma_{\Lambda,\Gamma_Y,\Gamma_W}$ and measurable function $f : X \to Y$ so that $f^\sharp$ is an abstraction of f (see formula 4), we define

$$ {f_p}^\sharp : \left| \begin{array}{ll} S_{\Lambda,\Gamma_X,\Gamma_W} & \to S_{\Lambda,\Gamma_Y,\Gamma_W} \\ ((Z_\lambda)_{\lambda\in\Lambda}, w) & \mapsto ((f^\sharp(Z_\lambda))_{\lambda\in\Lambda}, w). \end{array} \right. $$

Theorem 1. *${f_p}^\sharp$ is an abstraction of f_p.*

Random Inputs or Generators To accommodate calls to `random`-like instructions, we must be able to abstract a product of two independent random variables knowing an abstraction for each of the variables. More precisely, let us suppose we have two abstractions $S_{\Lambda,\Gamma_X,\Gamma_W}$ and $S_{\Lambda',\Gamma_{X'},\Gamma_{W'}}$. Let us also suppose we have an abstraction $\Gamma_{W_p} = \langle \mathcal{P}([0,1]^{\Lambda\times\Lambda'}), W_p, \gamma_{W_p} \rangle$ and an abstraction $p^\sharp : W \times W' \to W_p$ of

$$ p : \left| \begin{array}{ll} [0,1]^\Lambda \times [0,1]^{\Lambda'} & \to [0,1]^{\Lambda\times\Lambda'} \\ ((w_\lambda)_{\lambda\in\Lambda}, (w_{\lambda'})_{\lambda'\in\Lambda'}) & \mapsto (w_\lambda . w_{\lambda'})_{(\lambda,\lambda')\in\Lambda\times\Lambda'}. \end{array} \right. $$

Let us also suppose we have an abstraction $\Gamma_\Pi = \langle \mathcal{P}(X \times X'), \Pi^\sharp, \gamma_\Pi \rangle$ and an abstraction $\times^\sharp : X^\sharp \times X'^\sharp \to X_p$ of $\times : \mathcal{P}(X) \times \mathcal{P}(X') \to \mathcal{P}(X \times X')$ (see formula 5). Let us take abstract elements $A = ((Z_\lambda)_{\lambda \in \Lambda}, w) \in S_{\Lambda, \Gamma_X, \Gamma_W}$ and $A' = ((Z'_\lambda)_{\lambda' \in \Lambda'}, w') \in S_{\Lambda', \Gamma_{X'}, \Gamma_{W'}}$ then we define

$$A \otimes^\sharp A' = \left((Z_\lambda \times^\sharp Z'_{\lambda'})_{(\lambda,\lambda') \in \Lambda \times \Lambda'}, p^\sharp(W, W')\right)$$

Theorem 2. *$(A^\sharp, A'^\sharp) \mapsto A^\sharp \otimes^\sharp A'^\sharp$ is an abstraction of $(\mu, \mu') \mapsto \mu \otimes \mu'$. That is, if $\mu \in \gamma_{\lambda, \Gamma_X, \Gamma_W}(A^\sharp)$ and $\mu' \in \gamma_{\lambda, \Gamma_{X'}, \Gamma_{W'}}(A'^\sharp)$ then $\mu \otimes \mu' \in \gamma_p(A^\sharp \otimes^\sharp A'^\sharp)$.*

Tests Lifting equation 1 to powersets yields the concrete semantics:

$$[\![\texttt{if } c \texttt{ then } e_1 \texttt{ else } e_2]\!]_p^\flat(W) = [\![e_1]\!]_p^\flat \circ \phi^\flat{}_{[\![c]\!]}(W) +^\flat [\![e_2]\!]_p^\flat \circ \phi^\flat{}_{[\![c]\!]^C}(W)$$

which can be abstracted right away by replacing $\flat$'s by $\sharp$'s. All that is therefore needed are suitable $\phi^\sharp{}_{[\![c]\!]}(W)$ and $+^\sharp$.

We define

$$((Z_\lambda)_{\lambda \in \Lambda}, w) +^\sharp ((Z'_\lambda)_{\lambda' \in \Lambda'}, w') = ((Z_\lambda)_{\lambda \in \Lambda \amalg \Lambda'}, w \oplus^\sharp w')$$

where $\oplus^\sharp$ is an abstraction of the canonical bijection between $[0,1]^\Lambda \times [0,1]^{\Lambda'}$ and $[0,1]^{\Lambda \amalg \Lambda'}$ where $\Lambda \amalg \Lambda'$ is the disjoint union of Λ and Λ'. It is easy to see that such a $+^\sharp$ is an abstraction of $+^\flat$.

Let us suppose we have a suitable abstraction $I^\sharp{}_{[\![c]\!]} : X^\sharp \to X^\sharp$ of the intersection function $W^\flat \mapsto W^\flat \cap [\![c]\!]$. We also require that $\forall x^\sharp \in X^\sharp\ I^\sharp{}_{[\![c]\!]}(x^\sharp) \sqsubseteq x^\sharp$.[3] Then we can define

$$\phi^\sharp{}_{[\![\mathrm{c}]\!]}((Z_\lambda)_{\lambda \in \Lambda}, w) = ((I^\sharp{}_{[\![\mathrm{c}]\!]}(Z_\lambda))_{\lambda \in \Lambda}, d^\sharp(w))$$

Theorem 3. *$\phi^\sharp{}_{[\![\mathrm{c}]\!]}$ is an abstraction of $\phi^\flat{}_{[\![\mathrm{c}]\!]}$.*

Loops Using the $\phi^\sharp{}_{[\![\mathrm{c}]\!]}$ functions defined in the preceding paragraph and the framework of §3.3, it is easy to build an abstract semantics for loops provided we have suitable widening operators. The heuristic design of such operators will be discussed in §5.3.

4.3 Multiplicity of Representations and Coalescing

The reader might have been surprised we consider a preorder on the abstract values, not an order. The reason is that we want to talk of algorithmic representations, and a same concrete set can be represented in several ways. For

[3] One possible construction for this function is $W^\sharp \mapsto W^\sharp \cap^\sharp [\![c]\!]^\sharp$ using an approximation $[\![c]\!]^\sharp$ of the set of environments matched by c and an approximation $\cap^\sharp$ of the greatest lower bound. This does not in general yield optimal results, and it is better to compute $I^\sharp{}_{[\![c]\!]}(W^\sharp)$ by induction on the structure of c if c is a boolean expression. An example of suboptimality is the domain of integer intervals on one variable, with $W = [0, +\infty[$ and boolean expression $(\mathtt{x} > 2) \vee (\mathtt{x} < -2)$. The abstraction of the domain matched by the expression is $\top$, which gives us the approximate intersection $[0, +\infty[$ while recursive evaluation yields $[2, +\infty[$. Further precision can be achieved by local iterations [7].

instance, rational languages can be represented by an infinity of finite automata. Of course, an interesting property is that there is a minimal automaton and thus a canonical form. Yet we point out that this minimal automaton is defined up to state renaming, thus it has several representations.

We propose two **coalescing operations** to simplify representations without loss of precision:

1. If there is a certain Z_0 so that several λ are so that $Z_\lambda = Z_0$, and our numerical lattice enables us to represent exactly a sum, then one could replace all the entries for all these λ's by a single one.
2. Similarly, if Z_{λ_1} and Z_{λ_2} are so that there exists W so that $\gamma_X(Z_{\lambda_1}) \cup \gamma_X(Z_{\lambda_2}) = \gamma_X(W)$, then one can coalesce Z_{λ_1} and Z_{λ_2} into W, with probability $\min(w_{\lambda_1} + w_{\lambda_2}, 1)$.

5 Practical Constructions

In the previous section, we have given a very parametric construction, depending on parameters and assuming the existence of some operators. In this section we shall give examples of instances of suitable parameters and experimental results on a simple example.

5.1 Abstract Domain

We shall first define a narrower class of abstract domains for probabilistic applications, for which we shall give algorithms for some operations.

Finite Sequences Let us suppose that $X^\sharp$ has a minimum element $\bot_{X^\sharp}$. We then take $\Lambda = \mathbb{N}$. We note ${X^\sharp}^{(\mathbb{N})}$ the set of sequences with finite support; that is, those that are stationary on the value $\bot_{X^\sharp}$. We shall restrict ourselves to such abstract sequences.

As for the set of numeric constraints, one can for example use polyhedric constraints. Such constraints have the following nice property:

Theorem 4. *Let us suppose that:*

- *the numeric constraints are expressed as convex polyhedra, and the inclusion of two such polyhedra is decidable;*
- *the intersection test over $X^\sharp$ is computable.*

Then the preorder test on $S(X^\sharp)$ (i.e. the function that, taking $(a, b) \in S(X^\sharp)^2$ as parameter, returns 1 if and only if $a \sqsubseteq_{S(X^\sharp)} b$ and 0 otherwise) is computable.

Proof. Let $a = ((Z_i)_{i<N}, w)$ and $b = ((Z'_i)_{i<N'}, w')$. Let us call $\alpha_i = \mu_i(Z_i)$ and $\alpha'_i = \mu_i(Z'_i)$. Let $(\Xi_i)_{i<M}$ be the set of all nonempty intersections of elements of the sequences Z and Z_i. Let E be the system of equations of the form $\alpha_i = \sum \xi_j$ taking only the ξ_j so that $\Xi_j \subseteq Z_i$, E' the system of equations of the form

$\alpha'_i = \sum \xi_j$ taking only the xi_j so that $\Xi_j \subseteq Z_i$. F the system of linear inequations yielded by $(\alpha_i)_{i<N} \in w$ and F' the system of linear inequations yielded by $(\alpha'_i)_{i<N} \in w'$. Given a system of (in)equations σ, we call $\mathcal{S}(\sigma)$ the set of solutions of σ. We claim that

$$a \sqsubseteq_{S(X^\sharp)} b \iff \mathcal{S}(E \cup E' \cup F) \subseteq \mathcal{S}(E \cup E' \cup F').$$

The right-hand side of the equivalence is decidable by hypothesis.

Our claim is the consequence of the following lemma:

Lemma 2. *If Z is a nonempty measurable set and $c \in \mathbb{R}_+$, then there exists $\mu \in \mathcal{M}_+(Z)$ so that $\mu(Z) = c$.*

Proof. Let $z_0 \in Z$. Then we define $\mu(A)$ to be equal to c if $z_0 \in A$ and to 0 otherwise.

Simple Constraints We propose a very restricted class of polyhedric constraints, given by finite sequences $(c_n)_{n\in\mathbb{N}} \in [0,1]^{(\mathbb{N})}$, so that

$$(\alpha_n)_{n\in\mathbb{N}} \in \gamma_W((c_n)_{n\in\mathbb{N}}) \iff \forall n \in \mathbb{N}\ \alpha_n \leq c_n.$$

An abstract element is thus stored a finite sequence (Z_n, c_n) of pairs in $X^\sharp \times [0,1]$. Similar **convex hulls** have already been proposed for rules operating on concrete semantics [8].

It is very easy in such a framework to get an upper approximation of the probability of a set W, if we have a function $\tau_W : X^\sharp \to \{\mathsf{true}, \mathsf{false}\}$ so that $\tau_W(X^\sharp) = \mathsf{true} \iff W \cap \gamma_X(X^\sharp) \neq \emptyset$: just take $\sum_{n\in\{n\in\mathbb{N} \mid \tau_W(Z_n)=\mathsf{true}\}} c_n$.

5.2 Experiments

Using our framework, we analyzed the following C program:

```
double x=0.0;
int i;
for (i=0; i<4; i++)
  x += drand48()*2.0-1.0;
```

The `drand48()` function returns a `double` number equidistributed in $[0,1[$. We chose such a simple program so as to have an easy exact computation.

As an accurate representation of the `double` type would be complex and dependent on the particular C implementation, we rather chose to use an idealized version of this type as the real numbers. Figures 3, 4 and 5 show results of the experiments with different parameters, comparing abstract samples and exact computations.

In those tests, `drand48()` is supposed to return an uniformly distributed real number in $[0,1]$. It is abstracted as $([n/N, (n+1)/N], 1/N)_{0\leq n<N}$ where N is a parameter. The "samples" are segments $[\alpha, \beta]$; for each sample segment W, both an upper bound and an exact results of $\mu(W)$ are computed, where μ is the distribution of the final value of `x` in the above program. The upper bound

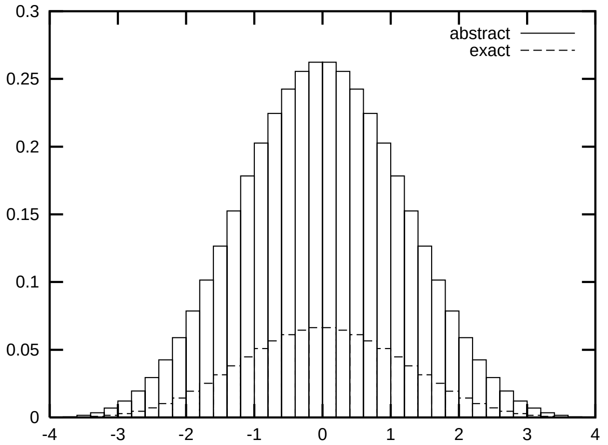

Fig. 3. Experimental results: $X_1 + X_2 + X_3 + X_4$ where the X_i are independent random variables equidistributed in $[-1, 1]$. The approximate simulation divided $[-1, 1]$ into 10 sub-segments each of maximal probability 0.1. Estimates on segments of length 0.2.

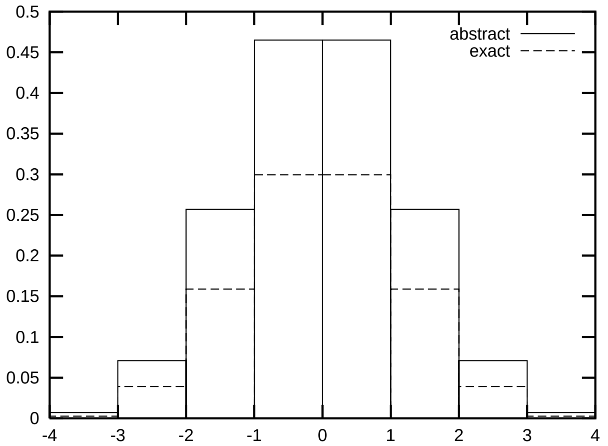

Fig. 4. Same computations as Fig. 3. Approximations on segments of length 1 give more accurate results than approximations on smaller segments.

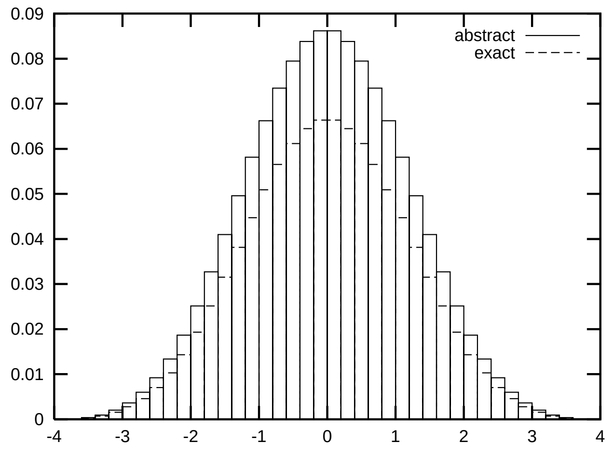

Fig. 5. Same computation as Fig. 3, but the approximate simulation divided $[-1, 1]$ into 100 sub-segments each of maximal probability 0.01. The sampled segments are of length 0.2. As with Fig. 4, precision is improved is sampling segments are bigger than the segments used in the computation.

is is computed from the abstract result, by the method described in 5.1. The bars displayed in the figure are the chosen segments $[\alpha, \beta]$ in x and their exact and approximate probabilities in y.

Those figures illustrate the following phenomenon: as computations go, the abstract areas Z_n grow bigger. If the samples are not enough bigger than those areas, the approximation are bad (Fig. 3). Results improve if N is increased (Fig. 5) or the sample size is increased (Fig. 4). An intuitive vision of this somewhat paradoxical behavior is that our abstract domain represents masses quite exactly, but loses precision on their exact location. If we ask our analysis to provide information on the probability of a very small area of the output domain (small compared to the precision of the input distribution and of the complexity of the transfer function), it tends to overestimate it (Fig. 3) because lots of masses could be located at that point. If we ask on a wider area, the error on the locations of the masses compared to the area becomes small and thus the error on the result becomes acceptable (Fig. 4).

5.3 Widenings

The crucial problem of the abstract domains not satisfying the ascending chain condition is the "widening" to choose. By widening, we mean some kind of overapproximation that jumps higher in the abstract domain to allow for convergence in finite time even if the abstract domain does not enjoy the property that every ascending sequence is stationary. Let us take a simple example on a nonprobabilistic program, with the domain of intervals: if successive abstract values are

$[1,1]$, $[1,2]$, $[1,3]$, $[1,4]$, the system might try jumping to $[1,+\infty[$ for the next iteration. As this overestimates the set, it is safe.

The design of widenings is experimental in order to find a satisfying balance between cost and precision. While it is always possible to give a widening in all abstract domains with a maximum element (just jump to $\top$), it is quite difficult to design widenings giving interesting results. Here, we shall propose a few ideas:

– Let us suppose we have a widening operator in $X^\sharp$. When successive abstract values in an iteration are $(Z_n, c_n)_{n\leq N}$ so that both Z_n and c_n increase, then try the next iteration with (Z, c_N) where Z is the result of the widening in $X^\sharp$.
– We can also apply widenings on the numerical coefficients c_n. For instance, if we have an increasing sequence $(Z, c_n)_{n\leq N}$, we can jump to (Z, c) where c is slightly above c_N, possibly 1.

Both approaches can be combined.

Another important area is simplification. Each call to `random`-like functions yields a product of measures and multiplies the number of length of the sequence making up the abstract environment by the length of the sequence approximating the measure of the function. This of course can mean fast explosion. While coalescing (see 4.3) can help, we might have to consider more energic steps. A possibility is to coalesce several abstract sets that have high probability (let us say, > 0.8) and are "close enough", such as $[0, 2]$ and $[1, 3]$.

We are currently working on designing on implementing such strategies and testing them on realistic examples.

6 Conclusions and Prospects

We have given simple probabilistic semantics to a deterministic language supplementing the usual constructions by functions returning random values of known distributions. We have a generic construct to lift usual (that is, non-probabilistic) abstract analyses to probabilistic analyses. The analysis we propose can be used to get upper bounds on the probability of certain events at certain points of a program. We tested it on some simple examples where an exact computation of the probabilities was possible, so as to have early experimental results of the influence of certain parameters over the quality of approximation.

We have proposed heuristics for some operators needed to handle large programs or loops. We expect to be able soon to propose results as to efficient heuristics on certain classes of problems.

References

1. Patrick Cousot. *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes.* Thèse d'état ès sciences mathématiques, Université scientifique et médicale de Grenoble, Grenoble, France, 21 mars 1978.

2. Patrick Cousot and Radhia Cousot. Abstract interpretation and application to logic programs. *J. Logic Prog.*, 2-3(13):103–179, 1992.
3. Patrick Cousot and Nicolas Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Proceedings of the Fifth Conference on Principles of Programming Languages*. ACM Press, 1978.
4. Alain Deutsch. Semantic models and abstract interpretation techniques for inductive data structures and pointers. In *Proceedings of the ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation*, pages 226–229, La Jolla, California, June 21–23, 1995.
5. J.L. Doob. *Measure Theory*, volume 143 of *Graduate Texts in Mathematics*. Springer-Verlag, 1994.
6. Philippe Granger. Static analysis of linear congruence equalities among variables of a program. In *Proceedings of TAPSOFT '91*, volume 493 of *Lecture Notes in Computer Science*, pages I. 169–172. Springer-Verlag, 1991.
7. Philippe Granger. Improving the results of static analyses programs by local decreasing iteration. In R. K. Shyamasundar, editor, *Foundations of Software Technology and Theoretical Computer Science, 12th Conference, New Delhi, India*, volume 652 of *Lecture Notes in Computer Science*, pages 68–79. Springer-Verlag, December 1992.
8. 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).
9. Claire Jones. *Probabilistic Non-Determinism*. PhD thesis, University of Edinburgh, 1990.
10. 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.
11. D. Kozen. Semantics of probabilistic programs. *Journal of Computer and System Sciences*, 22(3):328–350, 1981. A novel attempt at defining the semantics of probabilistic programs. Two equivalent semantics are presented.
12. Gavin Lowe. Representing nondeterminism and probabilistic behaviour in reactive processes. Technical Report TR-11-93, Oxford University, 1993.
13. François Masdupuy. Semantic analysis of interval congruences. In *Formal methods in programming and their applications*, volume 735 of *Lecture Notes in Computer Science*, Novosibirsk, Russia, June/july 1993. Springer-Verlag.
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. Walter Rudin. *Real and Complex Analysis*. McGraw-Hill, 1966.
17. Tim A. Wagner, Vance Maverick, Susan L. Graham, and Michael A. Harrison. Accurate static estimators for program optimization. *ACM SIGPLAN Notices*, 29(6):85–96, June 1994.

A Proofs of Convergence and Norm of the Concrete Semantics

This appendix is meant only for the purpose of refereeing. It contains technical proofs necessary for the construction of the concrete semantics. They are not relevant to the main topic of the paper, which is the analysis of this semantics.

A.1 Normal and random basic operations

Let us first remark that if H is a continuous operator on measures, using the norm of total variation, then $\|H\| = \sup\{\|H.\mu\| \mid \|\mu\| \leq 1\}$ can be computed on finite positive measures only: let $\|H\|_+ = \sup\{\|H.\mu\| \mid \mu \geq 0 \wedge \|\mu\| \leq 1\}$: if $\mu = \mu^+ - \mu^-$, $\|H.\mu\| = \|H.\mu^+ - H.\mu^-\| \leq \underbrace{\|H.\mu^+\|}_{\leq \|H\|_+ . \|\mu^+\|} + \underbrace{\|H.\mu^-\|}_{\leq \|H\|_+ . \|\mu^-\|} \leq \|H\|_+ . \underbrace{(|\mu^+\| + |\mu^-\|)}_{\|\mu\|}$, so $\|H\| \leq \|H\|_+$. But finite positive measures are signed measures, so $\|H\| \geq \|H\|_+$ and thus $\|H\| = \|H\|_+$.

If $f : X \to Y$ is an application, then $\|f_p.\mu\| = \|\mu\|$: on positive measures, $\|f_p.\mu\| = (f_p.\mu)(Y) = \mu(f^{-1}(Y)) = \mu(X) = \|\mu\|$ and the result extends to signed measures.

If μ_R is a probability measure, then for any μ, $\|\mu \otimes \mu_R\| = \|\mu\|$. This, combined with the norm of assignment, implies that $\left\|[\![\texttt{x:=random}]\!]_p\right\| = 1$.

A.2 Flow control

Let us first remark that if W is measurable, then for all μ, $\|\phi_W.\mu\| + \|\phi_{W^C}.\mu\| = \|\mu\|$.

Equation 1 defines linear operator $H = [\![\texttt{if } c \texttt{ then } e_1 \texttt{ else } e_2]\!]_p$ as $[\![e_1]\!]_p \circ \phi_{[\![c]\!]} + [\![e_2]\!]_p \circ \phi_{[\![c]\!]^C}$. Let us suppose that $\left\|[\![e_1]\!]_p\right\| \leq 1$ and $\left\|[\![e_2]\!]_p\right\| \leq 1$. Then

$$\begin{aligned} \|H.\mu\| &\leq \underbrace{\left\|[\![e_1]\!]_p\right\|}_{\leq 1} . \|\phi_{[\![\texttt{c}]\!]}.\mu\| + \underbrace{\left\|[\![e_2]\!]_p\right\|}_{\leq 1} . \left\|\phi_{[\![\texttt{c}]\!]^C}.\mu\right\| \\ &\leq \|\phi_{[\![\texttt{c}]\!]}.\mu\| + \left\|\phi_{[\![\texttt{c}]\!]^C}.\mu\right\| = \|\mu\|, \end{aligned}$$

thus $\|H\| \leq 1$.

In equation 3, we define a measure as the limit of a sequence of measures:

$$[\![\texttt{while } c \texttt{ do } e]\!]_p.\mu = \sum_{n=0}^{\infty} \phi_{[\![c]\!]^C} \circ ([\![f]\!]_p \circ \phi_{[\![c]\!]})^n(\mu) \tag{6}$$

This limit is taken set-wise: ie, we mean that for any measurable set X, $([\![\texttt{while } c \texttt{ do } e]\!]_p.\mu)(X) = \sum_{n=0}^{\infty}(\phi_{[\![c]\!]^C} \circ ([\![e]\!]_p \circ \phi_{[\![c]\!]})^n.\mu)(X)$. If μ is a positive measure, sum 6 indeed defines a positive measure: the partial sums are measures and the set-wise limit of an increasing sequence of positive measures is a positive measure [5, §III.10]. The result extends to signed measures by splitting the measure into its positive and negative parts. It is also quite evident that equation 6 defines a linear operator H, so we have shown that this linear operator maps measures onto measures.

Let us define the partial sums $H_n = \sum_{k=0}^{n} \phi_{[\![c]\!]^C} \circ ([\![f]\!]_p \circ \phi_{[\![c]\!]})^k$. H_n can be equivalently defined by the following recurrence:

$$\begin{cases} H_0 = \phi_{[\![c]\!]^C} \\ H_{n+1} = \phi_{[\![c]\!]^C} + H_n.[\![e]\!].\phi_{[\![c]\!]}. \end{cases}$$

By induction, we prove that for all n, $\|H_n\| \leq 1$, similarly as for the `if` construct.

Let us now consider the sequence of measures $H_n.\mu$. It converges set-wise to $H.\mu$; also, for all n, $\|H_n.\mu\| \leq \|\mu\|$. For all measurable X, $|(H_n.\mu)(X)| \leq \|\mu\|$ and thus, using the set-wise limit, $|(H.\mu)(X)| \leq \|\mu\|$. It follows that $\|H.\mu\| \leq \mu$. This achieves proving $\|H\| \leq 1$.

Remark 1. In general, H_n does not converge norm-wise to H.

Proof. Let us consider the following C program:

```
/* x is in ]0, 1] */
while (x<=0.5)
  x=x*2;
```

It is easy to see that $\int \mathrm{d}(H.\mu) = \int \mathrm{d}\mu$. Let us consider the operator π that maps the 1-tuple containing `x` to the 0-tuple and the operators H_n of the loops. $\pi \circ H_n(\mu)$ is then the probability of termination on input measure μ. If $H_n \to H$ in norm, we would have $\pi \circ H_n \to \pi \circ H$ in norm too. Nevertheless, if μ_n is a measure of weight 1 lying in $]0, 2^{-n-1}]$, $(\pi \circ H_n).\mu_n = 0$ while $(\pi \circ H).\mu_n = 1$ so $\|\pi \circ H - \pi \circ H_n\| = 1$.

Continuity of the various operators enables us to swap them with infinite sums, which is important to get some of the results in equation 3.

B Proofs of Abstraction

Theorem 1. *$S(f)$ is an abstraction of ${f_p}^\sharp$.*

Proof. This amounts to:

$$\forall x \in S(X^\sharp)\ \forall \mu \in S(\gamma_X)(x)\ f_p(\mu) \in S(\gamma_Y)(S(f)).$$

Let $((Z_\lambda)_{\lambda \in \Lambda}, w) \in S(X^\sharp)$ and $\mu \in \mathcal{M}_+(X)$. Let us suppose that there exists a family of measures μ_λ each respectively in $\mathcal{M}_+(Z_\lambda)$ so that $\mu = \sum_{\lambda \in \Lambda} \mu_\lambda$ and $(\mu_\lambda(\gamma_X(Z_\lambda)))_{\lambda \in \Lambda} \in \gamma_W(w)$. We want to prove that there exists a family of measures μ'_λ each respectively in $\mathcal{M}_+(f^\sharp(Z_\lambda))$ so that

1. $f_p(\mu) = \sum_{\lambda \in \Lambda} \mu'_\lambda$ and
2. $(\mu'_\lambda(\gamma_Y(f^\sharp(Z_\lambda)))_{\lambda \in \Lambda} \in w$.

Let us take candidates $\mu'_\lambda(U) = \mu_\lambda(f^{-1}(U))$. The first condition is then obviously met. As for the second, $\mu'_\lambda(f^\sharp(Z_\lambda)) = \mu_\lambda(f^{-1}(\gamma_Y \circ f^\sharp(Z_\lambda)))$. As $f^\sharp$ abstracts $f^\flat$, $f^\flat \circ \gamma_X(Z_\lambda) \subseteq \gamma_Y \circ f^\sharp(Z_\lambda)$; this implies that $f^{-1}(f^\flat \circ \gamma_X(Z_\lambda)) \subseteq f^{-1}(\gamma_Y \circ f^\sharp(Z_\lambda))$. As $\gamma_X(Z_\lambda) \subseteq f^{-1}(f^\flat \circ \gamma_X(Z_\lambda))$, by transitivity $\gamma_X(Z_\lambda) \subseteq f^{-1}(\gamma_Y \circ f^\sharp(Z_\lambda))$. This implies that $\mu'_\lambda(\gamma_Y \circ f^\sharp(Z_\lambda)) = \mu_\lambda(f^{-1}(\gamma_Y \circ f^\sharp(Z_\lambda))) = \mu_\lambda(\gamma_X(Z_\lambda))$, which in turn implies that $(\mu'_\lambda(\gamma_Y \circ f^\sharp(Z_\lambda)))_{\lambda \in \Lambda} \in \gamma_W(w)$.

Theorem 2. *$(A^\sharp, A'^\sharp) \mapsto A^\sharp \otimes^\sharp A'^\sharp$ is an abstraction of $(A^\flat, A'^\flat) \mapsto A^\flat \otimes^\flat A'^\flat$ with respect to $\gamma_{S_{\Lambda\times\Lambda'},\Gamma_{X_p},\Gamma_{W_p}}$. That is, if $\mu \in \gamma_X(A^\sharp)$ and $\mu' \in \gamma_X(A'^\sharp)$ then $\mu \otimes \mu' \in \gamma_p(A^\sharp \otimes^\sharp A'^\sharp)$.*

Proof. We have to prove that there exists a family of measures $(\mu^*_{(\lambda,\lambda')})_{(\lambda,\lambda')\in\Lambda\times\Lambda'}$ defined respectively over $(Z_\lambda \times Z_{\lambda'})_{(\lambda,\lambda')\in\Lambda\times\Lambda'}$ so that

1. $\mu^* = \sum_{(\lambda,\lambda')\in\Lambda\times\Lambda'} \mu^*_{(\lambda,\lambda')}$ and
2. $(\mu^*_{(\lambda,\lambda')}(Z_\lambda, Z'_{\lambda'}))_{(\lambda,\lambda')\in\Lambda\times\Lambda'} \in p^\sharp(W, W')$.

We take $\mu^*_{(\lambda,\lambda')} = \mu_\lambda \otimes \mu'_{\lambda'}$ as a candidate. The first point is trivial since $\otimes$ is bilinear. Since $\times^\sharp$ is an abstraction of $\times$, $\gamma_X(Z_\lambda) \times \gamma_{X'}(Z'_{\lambda'}) \subseteq \gamma_P(Z_\lambda \times^\sharp Z'_{\lambda'})$. By the definition of the product of measures, since μ_λ is concentrated on $\gamma_X(Z_\lambda)$ and $\mu'_{\lambda'}$ on $\gamma_{X'}(Z'_{\lambda'})$, then $\mu_\lambda \otimes \mu'_{\lambda'}$ is concentrated on $\gamma_X(Z_\lambda) \times \gamma_{X'}(Z'_{\lambda'})$, which, using the above inclusion, yields

$$\mu^*_{(\lambda,\lambda')}(\gamma_P(Z_\lambda \times^\sharp Z'_{\lambda'}) = \mu^*_{(\lambda,\lambda')}(\gamma_X(Z_\lambda) \times \gamma_{X'}(Z'_{\lambda'})).$$

By the definition of the product of measures,

$$\mu^*_{(\lambda,\lambda')}(\gamma_X(Z_\lambda) \times \gamma_{X'}(Z'_{\lambda'})) = \mu_\lambda(Z_\lambda).\mu'_{\lambda'}(Z_{\lambda'}).$$

The second point then follows from the fact that $p^\sharp$ is an abstraction of $p^\flat$.

Theorem 3. *${\phi^\sharp}_{[\![c]\!]}$ is an abstraction of ${\phi^\flat}_{[\![c]\!]}$.*

Proof. Let us take $((Z_\lambda)_{\lambda\in\Lambda}, w) \in S(X^\sharp)$ and μ one of its concretizations: $\mu = \sum_{\lambda\in\Lambda} \mu_\lambda$ where μ_λ is concentrated on Z_λ and $\int d\mu_\lambda = \alpha_\lambda$ so that $(c_\lambda)_{\lambda\in\Lambda} \in w$. We have to show that $\phi_{[\![c]\!]}(\mu) \in \gamma_{X^\sharp}({\phi^\sharp}_{[\![c]\!]})$; that is, that there exists a family $(\mu^*_\lambda)_{\lambda\in\Lambda}$ so that

1. $\lambda \in \Lambda$, μ^*_λ is concentrated on $\gamma_{X^\sharp}({I^\sharp}_{[\![c]\!]}(Z_\lambda))$ and
2. $(\mu^*_\lambda)_{\lambda\in\Lambda} \in \gamma_W(d^\sharp(w))$.

We take $\mu^*_\lambda(X) = \mu_\lambda(X \cap \gamma_X({I^\sharp}_{[\![c]\!]}))$.

The first condition is trivial. ${I^\sharp}_{[\![c]\!]}(Z_\lambda) \sqsubseteq Z_\lambda$ so by monotonicity, $\gamma_X({I^\sharp}_{[\![c]\!]}(Z_\lambda) \subseteq \gamma_X(Z_\lambda)$. Therefore $\int d\mu^*_\lambda \leq \alpha_\lambda$, which proves the second point.