Static Analysis of Programs with Imprecise Probabilistic Inputs

Assale Adje[2], Olivier Bouissou[1], Jean Goubault-Larrecq[2], Eric Goubault[1], and Sylvie Putot[1]

[1] CEA LIST
CEA Saclay, 91191 Gif-sur-Yvette CEDEX, France
firstname.lastname@cea.fr
[2] LSV, ENS Cachan
61 avenue du président Wilson, 94230 Cachan, France
lastname@lsv.ens-cachan.fr

Abstract. Having a precise yet sound abstraction of the inputs of numerical programs is important to analyze their behavior. For many programs, these inputs are probabilistic, but the actual distribution used is only partially known. We present a static analysis framework for reasoning about programs with inputs given as imprecise probabilities: we define a collecting semantics based on the notion of previsions and an abstract semantics based on an extension of Dempster-Shafer structures. We prove the correctness of our approach and show on some realistic examples the kind of invariants we are able to infer.

1 Introduction

Static analysis of embedded softwares faces the difficulty of correctly and precisely handling the program inputs. These inputs are usually given by sensors that measure a physical value continuously evolving with time. The classical abstraction of such inputs is to assign them with the range of values that the sensor may measure: in this way, we obtain a non-deterministic over-approximation of the values of the inputs which is then propagated through the program.

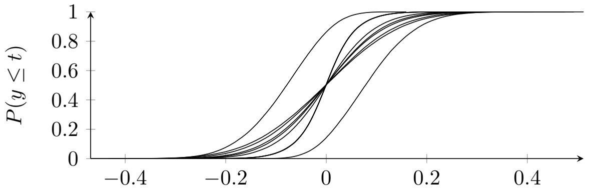
However, in addition to this non-deterministic abstraction of the values, we often have a probabilistic information on where the inputs lie in the range of possible values. This probabilistic information may be given by some knowledge on the physical environment with which the program interacts, or may be introduced as noise by the sensor. This noise can be very often modeled as a random variable with a Gaussian law; the value of the inputs is then given by $V + \varepsilon$ where V is a non-deterministically chosen value and ε is the probabilistic noise.

In this article, we present a framework to analyse *deterministic* programs with both probabilistic and non-deterministic inputs. In Section 2, we motivate our use of previsions and Probability-boxes. In Section 3, we define our concrete semantics based on previsions and in Section 4, we present our abstract semantics based on probabilistic affine forms. We prove its correctness in Section 5 and show in Section 6 the kind of invariants we are able to compute on realistic examples. Let us remark that to ease the understanding of this article, we have omitted various technical details such as the use of floating-point numbers or the impact of run-time errors on the semantics. We discuss these points in the course of the exposition.

Running example. In this article, we use a linear, order 2 filter to illustrate both our concrete and abstract semantics. This filter is given by the loop:

```
while(1) {
  y = 0.7*x - 1.3*x0 + 1.1*x1 + 1.4*y0 - 0.7*y1;
  x1 = x0; x0 = x; y1 = y0; y0 = y;
  x = input();
}
```

Numerical filters are very important for the analysis of embedded softwares as they are present in (almost) every software that must handle data coming from sensors. Computing the range of values reachable by the output variable `y` is a challenge adressed by many techniques [15]. However, all these methods assume that the inputs `x` (given by the function `input()` in the program) are bounded within a certain range and do not assume any distribution of the values within this range. Here, we assume that the input variables follow some probability distribution but we do not know which: we assume that `x` follows a uniform law on the range $[-A, A]$ for some $A \in [0, 0.2]$. Moreover, we assume that the distribution of the inputs may change during the execution of the filter, i.e. the distribution of input x read at iterate n (represented in the program by `x`) is not the same as the one of x read at iterate n-1 (represented in the program by `x1`). We however know that both are uniform distribution on some range $[-A, A]$. We ran 10 simulations of this filter and show below the 10 distributions in cumulative form (CDF) for the output variable `y`. Our goal is to compute guaranteed yet precise bounds on this set of distributions.



Contribution. In this paper, we present three main results. First, we define a semantics for imperative programs with inputs defined as imprecise probabilities. We define an operational and denotational semantics based on previsions and show their equivalence. Next, we define a new abstract domain based on probabilistic affine forms and especially new join and meet operators. Finally, we prove the correctness of the abstract semantics w.r.t the concrete ones and give some hints on how to adapt it to the analysis of hybrid systems by showing on one example how we can solve ODEs with our domain.

2 Related Work

One of our goals is to give a concrete semantics to an imperative language with access to imprecise inputs. Typically, these inputs will be numerical values given by physical sensors. Imagine a signal processing software that filters out thermal noise [29] from images given by a digital camera, for example with non-linear filtering techniques such as median filters [1]. Thermal noise is such that each pixel has an independent Gaussian noise, with zero mean and a standard deviation varying according to Nyquist law [29]. In particular, the standard deviation depends on the temperature, but not on pixels' values. In order to characterize the bounds on the noise after filtering, in all standard operational conditions (say, between -20 and 40 degrees Celsius), one has to consider all potential inputs, sum of a non-deterministic and bounded value (the range of the pixels, which is known) with a Gaussian noise of variance in a given interval, as computed by Nyquist law.

As exemplified above, one of our concerns will be to reason with so-called *imprecise probabilities.* There is a vast literature on the subject, and there are several mathematical notions that model imprecise probabilities, among which those based on *capacities* [8], and those based on *previsions* [41]. Capacities are simply monotone functions that map each measurable subset to its measure, such that the measure of the empty set is 0; but the measure of the disjoint union of two sets A and B does not necessarily coincide with the sum of their measures.

Previsions [41], on the other hand, are more abstract objects, but are better suited to giving semantics to programs [24], in a style akin to continuation-passing. Capacity-based semantics fail because we cannot even define sequential composition there [24]; sequential composition is defined by complex formulas in other models, such as convex powercones [40], where this involves unique extensions of maps to sets of non-empty closed convex subsets.

There are variations in what a prevision on a space X of values is. To us, a *prevision* on X will be any map $F\colon \langle X \to [0,1]\rangle \to [0,1]$ such that $F(ah) = aF(h)$ for all $a \in [0,1]$, where $\langle X \to [0,1]\rangle$ is the set of all measurable maps from X to $[0,1]$. We say that F is *ω-continuous* if and only if $F(\sup_{n\in\mathbb{N}} h_n) = \sup_{n\in\mathbb{N}} F(h_n)$ for every countable chain $(h_n)_{n\in\mathbb{N}}$, and F is *ω-cocontinuous* iff $F(\inf_{n\in\mathbb{N}} h_n) = \inf_{n\in\mathbb{N}} F(h_n)$, where the sups and infs are taken pointwise. Compared to [24], h is allowed to range over measurable maps, not just the bounded continuous maps, we drop the requirement of Scott-continuity, and the target space is $[0,1]$, not $\mathbb{R}^+$.

In this form, previsions are *generalized integrals*: we write $F(h)$ for the integral of h. Ordinary integrals $F(h) = \int_{x\in X} h(x)d\mu$ along an (additive) measure μ define particular cases of previsions F; distinctively, such previsions are linear, in the sense that $F(h+h') = F(h) + F(h')$, and are ω-continuous and ω-cocontinuous. Previsions do not demand linearity. Dropping linearity allows us to encode imprecise probabilities. For one, every capacity ν gives rise to a prevision, by the formula $F(h) = \int_{x\in X} h(x)d\nu$, where the integral with respect to the non-additive measure ν is the so-called Choquet integral [8]. E.g., if $\nu = \frac{1}{3}\mathfrak{e}_{A_1} + \frac{2}{3}\mathfrak{e}_{A_2}$ (the *example capacity* $\mathfrak{e}_A$ gives measure 1 to any set that meets A, and 0 to the others), then $F(h) = \int_{x\in X} h(x)d\nu = \frac{1}{3}\sup_{x\in A_1} h(x) + \frac{2}{3}\sup_{x\in A_2} h(x)$. Not all previsions are obtained as integrals from capacities, and this is the key ingredient used in [24] to provide a monad-based semantics to languages with non-deterministic and probabilistic choice. The basic intuition is that while capacities encode measures over sets, previsions encode sets of measures. Precisely, an *upper* prevision F (i.e. $\forall h, h',\ F(h)+F(h') \geq F(h+h')$, and F is ω-cocontinuous), encodes the set of all linear previsions G that are pointwise below F. The single functional F therefore encodes the set of all those measures μ such that $\int_{x\in X} h(x)d\mu \leq F(h)$ for every h. There is also a dual notion of *lower* prevision F (namely, $F(h) + F(h') \leq F(h + h')$, and F is ω-continuous), which encodes the set of all the measures μ such that $\int_{x\in X} h(x)d\mu \geq F(h)$ for every h.

Implementations of imprecise probabilities. P-boxes [17] and Dempster-Shafer structures [36], which are both related to capacities, are used to propagate both probabilistic and non-deterministic information in numerical simulation for instance. Arithmetic rules on P-boxes were defined in e.g. [42], and implementations are available, for instance the DSI Toolbox [2] based on Matlab and INTLAB [34], Statool [4] implementing the arithmetic of [3] and RiskCalc [16]. They were not designed to be used for static analysis of programs (there is no consideration on semantics of programs nor join operators defined, typically) as we do in this paper but are rather designed for making numerical simulations or optimizations [19] for instance for risk assessment [18]. Several recent papers proposed extensions of these arithmetics that either increase the precision or the efficiency of this arithmetic, as in e.g. [7], [37], [38] and [5]. Let us mention as well Neumaier's clouds [33] as another way to formalize imprecise probabilities (used in [19]). A unification of the different uncertainty representations was proposed in [11, 12] with comparisons between P-boxes and clouds.

The domain theoretic foundations of imprecise probabilities were studied by several authors, among which one of the authors of this paper [24, 23, 26, 25]. In particular, the *convex powerdomains* of spaces of measures on X was studied by Mislove [31], by Tix *et al.* [39, 40], and by Morgan and McIver [30].

Static analysis of probabilistic systems. There is a large literature in static analysis of probabilistic systems, some in abstract interpretation but most notably in model-checking. Our work is orthogonal to the one in probabilistic model-checking (as implemented in e.g. PRISM [27]) where probability distributions (but not imprecise probabilities) are considered on transitions of a transition model (and not on data, as we do here). The models used are mostly based on discrete time Markov chains [14].

In static analysis of programs by abstract interpretation, which is the subject of this paper, several abstract semantics have been considered. Monniaux [32] automatically constructs a probabilistic abstract domain as a collection of abstract elements with an associated weight. This is very similar to Dempster-Shafer structures where focal elements are elements of the underlying abstract domain. Our main advantage with respect to Monniaux' framework is that our arithmetic is efficient, precise and keeps some dependencies between variables, while the construction in [32] is not optimized for a specific abstract domain.

Another choice that has been sometimes made in abstract interpretation is to model imprecise probabilistic choice by *random variables* instead of probability distributions. The distinction is tenuous but real. Instead of giving a probability distribution over the intended value, one defines a probability distribution π on another, fixed space Ω (of so-called events), and describe the probability over the intended value v as the image measure of π by some measurable map

f from Ω to the space of values. This is the approach taken by Cousot and Monereau [10], where Ω is the space of infinite sequences of coin flips, each coin flip being independent and unbiased. A probability distribution on a space X is then encoded by a measurable map $f\colon \Omega \to X$, and the (image) measure of $A \subseteq X$ is $\pi(f^{-1}(A))$. One can then encode imprecise probabilities as well, as sets of measurable maps f.

The difficulty with this approach is that every probability law has to be described through some measurable map $f\colon \Omega \to X$, and must be implemented by a program $[f]$ for the analysis to proceed. E.g., to describe the Gaussian distribution on $X = \mathbb{R}$ with mean μ and standard deviation σ, one would implement a function $[f]$ that takes a sequence of independent, unbiased random booleans, and returns a (μ, σ)-normal random real. This is certainly possible, but the static analyzer will have to be sufficiently precise to derive meaningful, precise semantic invariants from the code of $[f]$. Our approach, based on P-box approximants to actual sets of distribution laws, is more direct.

Another approach, which is very promising, is taken in [35] that presents an approach for finding interval bounds on the probability of assertions over program variables by examining finitely many paths and performing a standard symbolic execution along each path. The goal of this approach is to use polyhedral volume bounding techniques for summing up the probability of assertion being satisfied along each path. The probability of unexplored paths is computed and added to the overall interval bound. Unlike our work, Sankaranarayanan et al. deal with precisely specified probability distributions whereas our work can handle imprecise probabilities. Furthermore, our approach here can represent the joint distributions of program variables at intermittent program points to potentially answer a larger set of questions about the program behavior. Whereas, their work focuses on queries posed at the end of the program execution. Since their work is unpublished at the time of writing, we provide an indirect comparison by demonstrating our technique on some of the benchmarks used in their paper.

3 Concrete Semantics

We consider the toy imperative language shown in Figure 1 as the core of languages such as C, to which we add a specific assignment instruction $x_1, \ldots, x_k :=$

$$\begin{array}{lll}
e ::= & v \mid c \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 \times e_2 \mid e_1 \div e_2 & \\
b ::= & \texttt{true} \mid \texttt{false} \mid x \leq c \mid x < c \mid x \geq c \mid x > c \mid \neg b & \\
s ::= & x := e & \text{assignment} \\
\mid & \ell\colon & \text{program point} \\
\mid & x_1, \ldots, x_k := \texttt{input} & \text{sensor input} \\
\mid & s_1 ; s_2 & \text{sequence} \\
\mid & \texttt{if}\ (b)\ \{\, s_1 \,\}\ \texttt{else}\ \{\, s_2 \,\} & \text{conditional} \\
\mid & \texttt{while}(b)\ \{\, s_1 \,\} & \text{loop}
\end{array}$$

Fig. 1. Syntax

`input`, which we explain below, and program points $\ell\colon$, where ℓ is taken from a countably infinite set $\mathcal{L}$ of so-called *labels*. The latter are used to name program points, not as targets of goto statements.

All instructions except one are standard. The `input` construction is meant to read the value of some sensors. Semantically, theses values will be probability distributions, so it may be helpful to visualize them as returning some actual value plus some random noise. We assume k noisy sensors, which may be correlated. If they were uncorrelated, i.e., independently distributed, then a language with k sensor-reading expressions $\texttt{input}_i$, each one returning the distribution of sensor number i, would have been sufficient. Instead, we use one `input` construction that returns a *joint* distribution $\nu_{\texttt{inp}}$ over all the values of the k sensors. To make the semantics simpler, we reserve k variables $x_1, \ldots, x_k$ as destinations of the `input` instruction, and call them the *sensor variables.* If one wishes to read, say, the first and the fourth sensor variables only, and in variables x and z instead of the fixed variables x_1 and x_4, one should write $x := x_1; z := x_4$ after the instruction $x_1, \ldots, x_k := \texttt{input}$. Our concrete semantics is parameterized by the joint sensor distribution $\nu_{\texttt{inp}}$, which we do not necessarily know. Our abstract semantics will abstract the latter by so-called Dempster-Shafer structures.

The variable x in assignments $x := e$ denotes any non-sensor variable; these form a finite set $\mathcal{V}ar$. We shall write $\mathcal{V}ar^+$ for $\mathcal{V}ar \cup \{x_1, \ldots, x_k\}$. The set Σ of environments is $\mathbb{R}^{\mathcal{V}ar^+}$, the set of all maps, denoted ρ, from $\mathcal{V}ar^+$ to $\mathbb{R}$. Expressions e and boolean tests b have deterministic semantics $[\![e]\!]\rho \in \mathbb{R}$ and $[\![b]\!]\rho \in \mathbb{B}$. We shall not describe it in detail, as it is mostly obvious, e.g., $[\![e_1 + e_2]\!]\rho = [\![e_1]\!]\rho + [\![e_2]\!]\rho$. The maps $\lambda\rho \cdot [\![b]\!]\rho$ and $\lambda\rho \cdot [\![e]\!]\rho$ are measurable maps of $\rho \in \Sigma$, where Σ is equated with $\mathbb{R}^{|\mathcal{V}ar^+|}$, equipped with its standard σ-algebra. We do not consider runtime errors here, and therefore assume division by 0 to return some arbitrary real number. We discuss this choice (and the choice of real numbers) at the end of this section.

We write a sequence of statements, i.e. a program, as Λ. We define an operational semantics of this language that infers judgments of the form $\ell \vdash \left(\Lambda, \rho\right) \downarrow_\kappa^\pm a$, for $\pm \in \{+, -\}$, where $\ell \in \mathcal{L}$, $a \in [0, 1]$ and $\kappa\colon \Sigma \to [0, 1]$ is a bounded measurable map. When κ is the indicator map χ_E of a measurable subset E of Σ, the intended meaning will be: starting from Λ in environment ρ, the probability of reaching label ℓ with some environment in E is at least (resp. at most) a if $\pm$ is $-$ (resp. $+$). In general, it is practical to use general measurable maps κ instead of indicator maps χ_E, and the meaning of $\ell \vdash \left(\Lambda, \rho\right) \downarrow_\kappa^- a$ will be: the average value of $\kappa(\rho')$ over all environments ρ' reached when at label ℓ is at least a (at most a for $\ell \vdash \left(\Lambda, \rho\right) \downarrow_\kappa^+ a$). For lack of space, we won't describe it here, but see Appendix B. We directly proceed to a prevision-based, denotational semantics that will play a role similar to collecting semantics in non-probabilistic settings, and will get us closer to an abstract semantics. For a proof that the prevision-based semantics matches the operational semantics, see Appendix C.

Our prevision-based denotational semantics is in the spirit of [24], except for the use of measure theory in place of domain theory. This is given in Figure 2: $[\![s]\!]^\pm_{\ell,\kappa} h\rho$ is meant to give the sup (if $\pm$ is $+$), resp. the inf (if $\pm$ is $-$) over all possible non-deterministic choices (when to stop and observe κ, which probability distribution $\nu_{\texttt{inp}}$ satisfying (6) to choose) of the average values that κ takes when

$$\llbracket x := e \rrbracket^{\pm}_{\ell,\kappa} h\rho = h(\rho[x \mapsto \llbracket e \rrbracket\rho]) \qquad \llbracket s_1; s_2 \rrbracket^{\pm}_{\ell,\kappa} h\rho = \llbracket s_1 \rrbracket^{\pm}_{\ell,\kappa} \left(\llbracket s_2 \rrbracket^{\pm}_{\ell,\kappa} h \right) \rho$$

$$\llbracket \ell \colon \rrbracket^{\pm}_{\ell,\kappa} h\rho = \kappa(\rho) \vee^{\pm} h(\rho) \qquad \llbracket \ell' \colon \rrbracket^{\pm}_{\ell,\kappa} h\rho = h(\rho) \text{ if } \ell' \neq \ell$$

$$\llbracket^{\ell} x_1, \ldots, x_k := \texttt{input} \rrbracket^{\pm}_{\ell,\kappa} h\rho = F^{\pm}_{\texttt{inp}}(\lambda v_1, \ldots, v_k \cdot h(\rho[x_1 \mapsto v_1, \ldots, x_k \mapsto v_k]))$$

$$\llbracket \texttt{if } (b) \ \{ \, s_1 \, \} \texttt{ else } \{ \, s_2 \, \} \rrbracket^{\pm}_{\ell,\kappa} h\rho = \begin{cases} \llbracket s_1 \rrbracket^{\pm}_{\ell,\kappa} h\rho & \text{if } \llbracket b \rrbracket\rho = 1 \\ \llbracket s_2 \rrbracket^{\pm}_{\ell,\kappa} h\rho & \text{if } \llbracket b \rrbracket\rho = 0 \end{cases}$$

$$\begin{aligned} \llbracket \texttt{while}(b) \ \{ \, s_1 \, \} \rrbracket^{\pm}_{\ell,\kappa} &= \textstyle\bigvee^{\pm}_{i \in \mathbb{N}} H^{i}_{b,s_1}(\bot^{\pm}) \\ \text{where } H_{b,s_1} &= \lambda\varphi \colon \langle \Sigma \to [0,1] \rangle \to \langle \Sigma \to [0,1] \rangle \cdot \\ & \qquad \lambda h \in \langle \Sigma \to [0,1] \rangle, \rho \in \Sigma \cdot \begin{cases} \llbracket s_1 \rrbracket^{\pm}_{\ell,\kappa}(\varphi(h))\rho & \text{if } \llbracket b \rrbracket\rho = 1 \\ h(\rho) & \text{if} \llbracket b \rrbracket\rho = 0 \end{cases} \end{aligned}$$

Fig. 2. Concrete semantics

we reach ℓ, running statement s starting from environment ρ. It is helpful to think of h as a continuation, as in [24], and of κ as another continuation, triggered at certain times where we reach label $\ell\colon$. Write $a \vee^{\pm} b$ for $\min(a, b)$ if $\pm$ is $-$, $\max(a, b)$ if $\pm$ is $+$. We define $\llbracket \ell' \colon \rrbracket^{\pm}_{\ell,\kappa} h\rho$, when $\ell' = \ell$, as the result of a nondeterministic choice ($\vee^{\pm}$, i.e., max or min) of what happens if we choose to end computation right here (giving an expected value of $\kappa(\rho)$), and of what happens ($h(\rho)$) if we decide to proceed. The semantics is defined for continuations h that are measurable maps from Σ to $[0, 1]$, just like for κ, and produces functionals $\llbracket s \rrbracket^{\pm}_{\ell,\kappa}$ mapping continuations h and environments $\rho \in \Sigma$ to elements of $[0, 1]$. The *bottom* functional $\bot^{-}$ maps all h, ρ to 0, and the *top* functional $\bot^{+}$ maps all h, ρ to 1. This is used in the rule for while loops, where we also agree to write $\bigvee^{+}$ for sup and $\bigvee^{-}$ for inf.

The semantics of Figure 2 is only defined provided the integral used in the case of noisy (random) inputs is defined. This is ensured by checking that the semantics of any program is measurable. In the case of while loops, this follows from the fact that the pointwise sup of a countable chain of measurable maps is measurable. To prove this in the case of sequential composition, we need to prove the following more general result.

Theorem 1. *For every measurable maps $\kappa, h\colon \Sigma \to [0, 1]$, $\llbracket s \rrbracket^{\pm}_{\ell,\kappa} h\rho$ is a well-defined number in $[0, 1]$. For fixed κ, h, $\llbracket s \rrbracket^{\pm}_{\ell,\kappa} h\colon \Sigma \to [0, 1]$ is a measurable map. For fixed κ, the map $\lambda h \cdot \llbracket s \rrbracket^{\pm}_{\ell,\kappa} h\rho$ is a prevision, which is upper if $\pm$ is $+$, lower if $\pm$ is $-$. For fixed h, the map $\lambda\kappa \cdot \llbracket s \rrbracket^{+}_{\ell,\kappa} h\rho$ is ω-continuous, and $\lambda\kappa \cdot \llbracket s \rrbracket^{-}_{\ell,\kappa} h\rho$ is ω-cocontinuous.*

Proof. (Sketch.) By structural induction on s. For assignments $x := e$, the measurability of $\llbracket x := e \rrbracket^{\pm}_{\ell,\kappa} h = \lambda\rho \cdot h(\rho[x \mapsto \llbracket e \rrbracket\rho])$ follows from the fact that $\lambda\rho \cdot \llbracket e \rrbracket\rho$ is measurable, and that h is measurable. For sequences $s_1; s_2$, we use the fact that composition preserves measurability, monotonicity, being upper, being lower, ω-continuity, and ω-cocontinuity. In the case of while loops, we use the fact that the sup of a countable chain of measurable, resp., ω-continuous maps, is again measurable, resp., ω-continuous. Dually, the inf of a countable chain of measurable, resp., ω-cocontinuous maps, is again measurable, resp., ω-cocontinuous. $\Box$

We extend the denotational semantics to lists Λ by $[\![\epsilon]\!]^{\pm}_{\ell,\kappa} = \mathrm{id}$ (i.e., $[\![\epsilon]\!]^{\pm}_{\ell,\kappa} h\rho = h(\rho)$), and $[\![s \bullet \Lambda]\!]^{\pm}_{\ell,\kappa} = [\![s]\!]^{\pm}_{\ell,\kappa} \circ [\![\Lambda]\!]^{\pm}_{\ell,\kappa}$; so the semantics of $\Lambda = s_1 \bullet s_2 \bullet \ldots \bullet s_n$ coincides with that of the sequence $s_1; s_2; \ldots; s_n$. Now, given a program Λ, an intial state ρ, a label ℓ and a measurable map κ, the denotational semantics computes two values $[\![\Lambda]\!]^{-}_{\ell,\kappa} h_1\rho$ and $[\![\Lambda]\!]^{+}_{\ell,\kappa} h_0\rho$ that enclose the possible value of κ on the program variables when we reach ℓ. Here, h_0 (resp. h_1) is the constant map associating to each environment ρ h_0 (resp. h_1).

We finish this section with a remark on real vs. floating-point numbers. This semantics is what we shall call the *ideal semantics* of expressions and tests. Actual programs will handle floating-point numbers, not real numbers. At the level of the concrete semantics, that would be easily repaired, as follows. First, there is a finite subset $\mathbb{F} \subseteq \mathbb{R}$ of so-called floating-point numbers, and a rounding function $\pi_{\mathbb{F}} \colon \mathbb{R} \to \mathbb{F}$. Mathematically, $\pi_{\mathbb{F}}$ is a projection, namely $\pi_{\mathbb{F}}(v) = v$ for every $v \in \mathbb{F}$. The floating-point semantics $[\![e]\!]'\rho$ is obtained by rounding back results, as in, e.g., $[\![e_1 + e_2]\!]'\rho = \pi_{\mathbb{F}}([\![e_1]\!]'\rho + [\![e_2]\!]'\rho)$. Considering a floating-point semantics instead of the ideal semantics would make the statement of our semantics complicated. We would have to take rounding modes into account, and the fact that they can change over the course of a program running; we would need to extend $\mathbb{R}$ and $\mathbb{F}$ to include non-numerical values such as infinites and NaNs; and we would have to make several cases to define the results of tests such as $e_1 < e_2$ whenever $[\![e_1]\!]'\rho$ or $[\![e_2]\!]'\rho$ is non-numerical. Errors would also handle the case of division by zero, which we mentioned earlier.

We consider these issues orthogonal to the present paper, whose main purpose is to give a semantics to numerical programs *with uncertain probabilities*. Errors incurred by the fact that actual programs use floating-point values instead of real numbers can be handled at the level of the abstract semantics. One can extend probabilistic affine forms to handle rounding errors, as quickly described in [5], in the same way as for affine forms [22], and we intend to invest in that direction in the future.

4 Abstract semantics

We now formally define our abstract semantics. It is based on an abstract domain that extends the probability affine forms of [5] as it introduces a join operator and an order relation. We first recall the notion of Dempster-Shafer structures.

4.1 Dempster-Shafer structures

An interval based Dempster-Shafer structure [36] (DSI in short) is a finite set of closed intervals (named focal elements), each associated with a weight (in a more general setting [36], focal elements are not necessarily closed intervals). DSI structures thus represent real variables whose value can be obtained by first probabilistically picking up an interval, and then non-deterministically picking up a value within this interval. In this article, we write a DSI structure d as

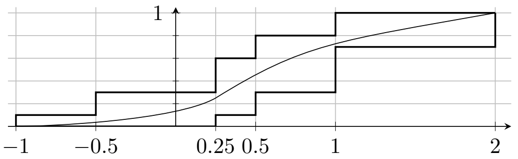
$d = \big\{\langle \boldsymbol{x_1}, w_1\rangle, \langle \boldsymbol{x_2}, w_2\rangle, \dots, \langle \boldsymbol{x_n}, w_n\rangle\big\}$, where $\boldsymbol{x_i} \in \mathbb{I}$ is a closed interval and $w_i \in (0,1]$ is the associated probability. For example, the DSI

$$d_1 = \big\{\langle[-1, 0.25], 0.3\rangle \,;\, \langle[-0.5, 0.5], 0.2\rangle \,;\, \langle[0.25, 1], 0.5\rangle\big\}$$

represents a real-valued random variable X whose value is in $[-1, 0.25]$ with probability 0.3, in $[-0.5, 0.5]$ with probability 0.2, and in $[0.25, 1]$ with probability 0.5. We require that all intervals are non-empty, and that $\sum_{k=1}^{n} w_k \leq 1$: when the inequality is strict, this means that the variable X is with probability $1 - \sum_{k=1}^{n} w_k > 0$ in $\mathbb{R}\backslash \bigcup_{k=1}^{n} \boldsymbol{x_k}$. We write $\mathbb{DS}$ for the set of all DSI structures over closed intervals.

Remark that there exists another popular model for imprecise probabilities, namely Probability-boxes [17] (P-box in short). We already showed in [5] that finite DSI structures and discrete P-box are equivalent. Intuitively, a P-box is a couple of two increasing functions $\left[\underline{P}, \overline{P}\right]$ that pointwise enclose a set of cumulative distribution functions. From a DSI d, we define the P-box $\left[\underline{P_d}, \overline{P_d}\right]$ by $\underline{P_d}(u) = \sum_{\overline{\boldsymbol{x_i}}<u} w_i$ and $\overline{P_d}(u) = \sum_{\underline{\boldsymbol{x_i}}\leq u} w_i$. We then graphically represent d by the graphs of the two functions $\left[\underline{P_d}, \overline{P_d}\right]$.

Example 1. Let $d_1 = \big\{\langle[-1, 0.25], 0.1\rangle \,;\, \langle[-0.5, 0.5], 0.2\rangle \,;\, \langle[0.25, 1], 0.3\rangle \,;\, \langle[0.5, 1], 0.1\rangle \,;\, \langle[0.5, 2], 0.1\rangle \,;\, \langle[1, 2], 0.2\rangle\big\}$. Then $\left[\underline{P_2}, \overline{P_2}\right] = \zeta(d_1)$ is plotted on the graph below.



Join and meet on DS structures The *join* of two DSI d_X and d_Y is defined as the union of all focal elements from d_X and d_Y, with the same probabilities, followed potentially by a normalization if the sum of all probabilities is greater than 1. For example, the join of the DSI $d_x = \{\langle[-1, 0], 0.5\rangle \,;\, \langle[0, 1], 0.4\rangle\}$ and $d_y = \{\langle[0.5, 1.5], 0.2\rangle\}$ is $\{\langle[-1, 0], 0.46\rangle \,;\, \langle[0, 1], 0.36\rangle \,;\, \langle[0.5, 1.5], 0.18\rangle\}$.

We do not define a *meet* operator on DSI but rather we define the operator $\mathtt{lt}_d(d_x, d_y)$ that reduces a DSI on a variable X to enforce the constraint $X \leq Y$. Intuitively, the resulting DSI contains all the focal elements of the form $\mathtt{lt}_{\mathbb{I}}(\boldsymbol{x_i}, \boldsymbol{y_j})$, when $\langle \boldsymbol{x_i}, a_i\rangle$ is a focal element of d_x and $\langle \boldsymbol{y_j}, b_j\rangle$ is a focal element of d_y, with:

$$\mathtt{lt}_{\mathbb{I}}([a, b], [c, d]) = \begin{cases} \emptyset & \text{if } a > d \\ [a, \min(b, d)] & \text{otherwise} \end{cases}$$

and the associated probability is then $w_i \times w_j$. For example, if d_x is a DSI over-approximating a uniform distribution on $[-1, 1]$ and d_y is the DSI with one focal element $[-0.05, 0.05]$ (i.e. mimicking a Dirac at 0), then $\mathtt{lt}_d(d_x, d_y)$ is depicted on Figure 3. Remark that even if d_x and d_y are normalized (with the sum of probabilities equal to 1), $\mathtt{lt}_d(d_x, d_y)$ may be denormalized. We use this operator to give the semantics of conditional statements of the form `if (X<=Y) s1 else s2`, and define equivalently a $\mathtt{gt}_d(d_x, d_y)$ operator that enforces that $X \geq Y$.

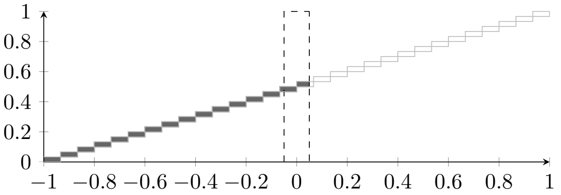


Fig. 3. Intersection between two DSI structures d_1 (in gray) and d_2 (dotted). The result is the filled DSI.

4.2 Abstract domain: probabilistic affine forms

Clearly, P-boxes or DSI could be chosen as an abstraction of a set of probability distributions, arithmetic rules and an order relation can also be easily defined. However, P-boxes alone cannot be used efficiently for the analysis of programs as the arithmetic between them must differentiate the case when variables are independent or not [3]. And in the case when two variables x and y are not independent, the interpretation of arithmetic operations creates a large over-approximation as any dependency relation between x and y must be assumed. To increase the precision, we present the abstract domain of probabilistic affine forms, for which arithmetic operations were defined in [5].

Intuitively, a probabilistic affine form encodes both the linear dependency between every program variable and the input (as with classical affine forms, see [21]), and an abstraction of the inputs as a DSI. We can thus compute the DSI associated with each variable (it is a linear transformation of the inputs), and we use the linear correlations between variables to compute the arithmetic operations. The potential non-linear relations (due to non-linear operations in the program) are over-approximated by an additional linear term.

More formally, perturbed affine forms [20, 21] are an extension of affine forms [9] in which each variable x is over-approximated by an expression of the form $\hat{x} = \alpha_0^x + \sum_{i=1}^n \alpha_i^x \varepsilon_i + \sum_{j=1}^m \beta_j^x \eta_j$ where the noise symbols ε_i or η_j are formal variables ranging over $[-1, 1]$ just as in affine forms, but where we differenciate the symbols ε_i that are directly related to an (uncertain) input value or parameter, and the symbols η_j that express an uncertainty in the analysis (loss of relation due to non linear computations for instance). For probabilistic affine forms, we will also use two kind of symbols, which will be random variables: the ε_i that are considered independent from one another, the η_j have unknown dependencies to the others.

Affine forms are closed under affine transformations: for $\lambda \in \mathbb{R}$,

$$\hat{x} + \lambda\hat{y} = \alpha_0^x + \lambda\alpha_0^y + \sum_{i=1}^n (\alpha_i^x + \lambda\alpha_i^y)\varepsilon_i + \sum_{j=1}^m (\beta_j^x + \beta_j^y)\eta_j \ .$$

Multiplication creates a new symbol η_{m+1} associated with the non-linear part:

$$\hat{x} \times \hat{y} = \alpha_0^x\alpha_0^y + \frac{R}{2} + \sum_{i=1}^n (\alpha_0^x\alpha_i^y + \alpha_i^x\alpha_0^y)\varepsilon_i + \sum_{j=1}^m (\alpha_0^x\beta_j^y + \beta_j^x\alpha_0^y)\eta_j + T\eta_{m+1}$$

with $R = \sum_{i=1}^{n} |\alpha_i^x \alpha_i^y| + \sum_{j=1}^{m} |\beta_j^x \beta_j^y|$ and

$$T = \sum_{i=1}^{n} \sum_{j=1}^{m} |\alpha_i^x \beta_j^y + \beta_j^x \alpha_i^y| + \sum_{i=1}^{n} \sum_{j>i}^{n} |\alpha_i^x \alpha_j^y + \alpha_j^x \alpha_i^y| + \sum_{i=1}^{m} \sum_{j>1}^{m} |\beta_i^x \beta_j^y + \beta_j^x \beta_i^y| + \frac{1}{2} R \ .$$

We next define probabilistic affine forms, an extension of affine forms, and formally define our abstract semantics.

Definition 1 (Probabilistic affine form). *We define a probabilistic affine form for variable x, on n independent noise symbols $(\varepsilon_1, \ldots, \varepsilon_n)$ and m noise symbols $(\eta_1, \ldots, \eta_m)$ with unknown dependency to the others, by a form*

$$\hat{x} = \alpha_0^x + \sum_{i=1}^{n} \alpha_i^x \varepsilon_i + \sum_{j=1}^{m} \beta_j^x \eta_j$$

together with n DSI $(d_{\varepsilon_1}, \ldots, d_{\varepsilon_n})$ and m DSI $(d_{\eta_1}, \ldots, d_{\eta_m})$ describing the possible random variables (of support $[-1, 1]$) for the noise symbols.

The interest of affine forms is to be able to represent affine relations that hold between uncertain quantities. We still have this representation, except only *imprecise* affine relations hold, as can be shown in the example below.

Example 2. Let $\hat{x}_1 = 1 + \varepsilon_1 - \eta_1$, $\hat{x}_2 = -\frac{1}{2}\varepsilon_1 + \frac{1}{4}\eta_1$, $d_{\varepsilon_1} = \{\langle[-1, 0], \frac{1}{2}\rangle, \langle[0, 1], \frac{1}{2}\rangle\}$, $d_{\eta_1} = \{\langle[-\frac{1}{10}, 0], \frac{1}{2}\rangle, \langle[0, \frac{1}{10}], \frac{1}{2}\rangle\}$, Then $\hat{x}_1 + 2\hat{x}_2 = 1 - \frac{1}{2}\eta_1$, with $d = d_{x_1+2x_2} = \{\langle[\frac{19}{20}, 1], \frac{1}{2}\rangle, \langle[1, \frac{21}{20}], \frac{1}{2}\rangle\}$. Thus the lower probability that $x_1 + 2x_2 \leq \frac{21}{20}$ is 1; and the upper probability that $x_1 + 2x_2 < \frac{19}{20}$ is 0. But for instance, $x_2 + 2x_2 \leq 1$ has upper probability $\frac{1}{2}$ and lower probability 0 and is thus an imprecise relation.

Given a probabilistic affine form $\hat{x}$, we define its enclosing DS structure, denoted $\gamma_d(\hat{x})$, by: $\gamma_d(\hat{x}) = \alpha_0^x + \Sigma_{i=1}^{n} \alpha_i^x d_{\varepsilon_i} \oplus \bigoplus_{j=1}^{m} \beta_j^x d_{\eta_j}$,, where $+$ and Σ represents the sum of DSI using the independent arithmetic, and $\oplus$ is the sum of DSI using the unknown dependency arithmetic [5]. In other worlds, $\gamma_d(\hat{x})$ computes a DSI by summing the DSI associated with each noise symbol of $\hat{x}$.

Definition 2 (Abstract environment). *Given variables $\mathcal{V}ar^+$, an abstract environment $\rho^\sharp$ is a function mapping each variable $x \in \mathcal{V}ar^+$ to a probabilistic affine form over the same set of noise symbols. Let $\Sigma^\sharp$ be the set of all abstract environments. For $x \in \mathcal{V}ar^+$, we shall write:*

$$\rho^\sharp(x) = \alpha_0^x + \sum_{i=1}^{n} \alpha_i^x \varepsilon_i + \sum_{j=1}^{m} \beta_j^x \eta_j \ .$$

Our abstract semantics is a classical collecting operational semantics on abstract environments. Given a program, equivalently a statement list Λ, and an initial environment ρ_0, the abstract semantics associates an abstract environment with each label ℓ such that ℓ : is in Λ. We thus define a function $[\![\Lambda]\!]_\ell^\sharp : \Sigma \to \Sigma^\sharp$. Its value on a variable x is an affine P-box which encodes an upper set of probability distributions on the value that x can take at control point ℓ. They are potentially denormalized since this is not describing the conditional probability distributions of x knowing that we are in ℓ but rather the probability distributions of reaching ℓ with certain values. We will prove in Section 5 that $[\![\Lambda]\!]^{\sharp}_{\ell}(\rho_0)$ is a correct abstraction of $[\![\Lambda]\!]^{\pm}_{\ell,\kappa}\rho_0$. The abstract semantics depends on various elementary operations on abstract environnments: input, join, meet and expression evaluation. We present them in the rest of this section.

Inputs The concrete semantics is parametrized by probability distribution $\nu_{\mathtt{inp}}$ for the possible values of the input variables. Equivalently, our abstract semantics is parametrized by a set of DSI $d_1, \ldots, d_k$ such that all marginal distributions of $\nu_{\mathtt{inp}}$ for x_i are contained in d_i, for $i \in [1, k]$. Then, the abstract semantics $[\![x_1, \ldots, x_k := \mathtt{input}]\!]^{\sharp}$ assigns each input variable x_i to a new noise symbol (either ε_k or η_k depending on the assumed dependency between inputs and other variables) and we associate the DSI d_i with this noise symbol. For example, if we assume we have one input variable x_1 independent from other variables and uniformely distributed on $[a, b]$, the probabilistic affine form $\hat{x}_1$ after the instruction $x_1 := \mathtt{input}$ will be $\hat{x}_1 = \frac{a+b}{2} + \frac{b-a}{2}\varepsilon_k$ where ε_k is a fresh noise symbol associated with a DSI enclosing the uniform distribution on $[-1, 1]$.

Join Let $\rho_1^{\sharp}$ and $\rho_2^{\sharp}$ be two abstract environments. We define the join $\rho_1^{\sharp} \sqcup \rho_2^{\sharp}$ pointwise, i.e. $\forall x \in \mathcal{V}ar,\ \rho_1^{\sharp} \sqcup \rho_2^{\sharp}(x) = \rho_1^{\sharp}(x) \dot{\sqcup} \rho_2^{\sharp}(x)$ where $\dot{\sqcup}$ is the join operator between two affine forms defined below.

Let now $x \in \mathcal{V}ar$ and let us write $\rho_1^{\sharp}(x) = \alpha_0^1 + \sum_{i=1}^{n} \alpha_i^1 \varepsilon_i + \sum_{j=1}^{m} \beta_j^1 \eta_j$ and $\rho_2^{\sharp}(x) = \alpha_0^2 + \sum_{i=1}^{n} \alpha_i^2 \varepsilon_i + \sum_{j=1}^{m} \beta_j^2 \eta_j$. As in [21], the join $\hat{x} = \rho_1^{\sharp}(x) \dot{\sqcup} \rho_2^{\sharp}(x)$ is computed as an affine transformation of the existing noise symbols plus a new noise symbol η_{m+1} that is used to over-approximate the error made by the linearization, i.e. $\hat{x} = \hat{x}_l + \eta_{m+1}$ with $\hat{x}_l = \alpha^0 + \sum_{i=1}^{n} \alpha^i \varepsilon_i + \sum_{j=1}^{m} \beta^j \eta_j$ where the values of the coefficients are given by Equations (1) to (3).

$$\alpha^0 = m\big(\gamma_d(\alpha_1^0) \curlyvee \gamma_d(\alpha_1^0)\big) \tag{1}$$

$$\forall i \in [1, n],\ \alpha^i = \operatorname{argmin}(\alpha_1^i, \alpha_2^i) \tag{2}$$

$$\forall j \in [1, m],\ \beta^j = \operatorname{argmin}(\beta_1^j, \beta_2^j) \tag{3}$$

Intuitively, the central term of $\hat{x}$ is the middle of the support of the DSI concretization of both affine forms (we note this $m(d)$ for a DSI d). The argmin function, defined by $\operatorname{argmin}(x, y) = z$ with $z \in [\min(x, y), \max(x, y)]$ and $|z|$ is minimal, keeps the best possible relation between on each noise symbol between the affine forms $\rho_1^{\sharp}(x)$ and $\rho_2^{\sharp}(x)$.

The new noise symbol η_{m+1} is defined to compensate this linearization. So, we define the DSI associated with η_{m+1} by $d_{\eta_{m+1}} = \gamma_d(\hat{x}_l - x) \curlyvee \gamma_d(\hat{x}_l - y)$. Recall that $\curlyvee$ is the join operator on DSI.

Meet As for DSI, we do not define formally the meet over probabilistic affine forms but rather give an operator $\mathtt{lt}_d(\hat{x}, \hat{y})$ that reduces the DSI of each symbol in $\hat{x}$ to enforce that the variables x and y verify $x \leq y$. It will be used in our abstract semantics to handle boolean expressions of the form $X \leq Y$. We here use an approach equivalent to the one over deterministic affine forms as introduced in [20], in which the meet was interpreted in an abstract domain over noise symbols. To simplify the presentation, let us consider two probabilistic affine forms $\hat{x}$ and $\hat{y}$ over two noise symbols ε_1 and ε_2. The generalization to arbitrary many noise symbols is straightforward. Intuitively, we want to enforce that $\alpha_0^x + \alpha_1^x d_{\varepsilon_1} + \alpha_2^x d_{\varepsilon_2} \le \alpha_0^y + \alpha_1^y d_{\varepsilon_1^y} + \alpha_2^y d_{\varepsilon_2^y}$, which leads to the following reduction:

$$\begin{aligned} d_{\varepsilon_1} &= \mathtt{lt}_d\Big(d_{\varepsilon_1}, \frac{\alpha_0^x - \alpha_0^y + (\alpha_2^x - \alpha_2^y)d_{\varepsilon_2}}{\alpha_1^x - \alpha_1^y}\Big) \\ d_{\varepsilon_2} &= \mathtt{lt}_d\Big(d_{\varepsilon_2}, \frac{\alpha_0^x - \alpha_0^y + (\alpha_1^x - \alpha_1^y)d_{\varepsilon_1}}{\alpha_2^x - \alpha_2^y}\Big) \end{aligned}$$

These equations can be iterated to reduce the DSI associated with ε_1 and ε_2, and we define $\mathtt{lt}(\hat{x}, \hat{y})$ as the greatest fixpoint of the iteration of these two equations.

Arithmetic operations We defined the arithmetic operations on probabilistic affine forms in [5]. For affine arithmetic operations, there is nothing new compared to the deterministic case and the DSI structures attached to symbols are not modified. For non-linear operations, we can rely on the affine form calculus, but instead of only bounding the non-linear part of the multiplication of the affine forms, we use the available calculus on DSI to form a correct DSI representing this non-linear part. This makes the calculus correct even for denormalized DSI. We carefully alternate between the independent and unknown dependency arithmetic on DSI to have a sound over-approximation.

5 Correctness proofs

In this section, we relate our notion of probabilistic affine forms with the semantics defined in Section 3. Intuitively, for each label ℓ that appears in the program, both semantics compute a set of probability distributions on the program variables when the program reaches ℓ. The concrete semantics computes bounds on the probability distributions at each label ℓ, they are denoted $[\![A]\!]^-_{\ell,\kappa}\rho$ and $[\![A]\!]^+_{\ell,\kappa}\rho$. Our abstract semantics associates a probabilistic affine form with each label of the program, from which we can define a set $\mathcal{P}$ of *compatible* probability distributions, see Section 5.1. We will prove that the bounds one can infer on the probabilities over program variables using $\mathcal{P}$ over-approximate the bounds computed by the concrete semantics. We first define the concretization function and then state and prove our soundness theorem.

5.1 Concretization

We assume that we have k variables $x_1, \ldots, x_k$, to each of them being attached an affine form $\hat{x_i}$ on n central and m perturbed noise symbols:

$$\hat{x_i} = \alpha_0^{x_i} + \sum_{i=1}^{n} \alpha_i^{x_i} \varepsilon_i + \sum_{j=1}^{m} \beta_j^{x_i} \eta_j \enspace .$$

Each noise symbol is associated with a DS structure denoted d_{ε_i} or d_{η_j}. We may represent this abstract element as a matrix $M \in \mathcal{M}^{d \times n+m}$ defined by

$M_{i,j} = \alpha_j^{x_i}$ if $j \leq n$ and $M_{i,j} = \beta_j^{x_i}$ for $n < j \leq n+m$ and a $\mathbb{R}^d$-vector A where $A_i = \alpha_0^{x_i}$. In the purely non-deterministic case, i.e. when the DS structures are the interval $[-1,1]$, the concretization of such an affine form is the set of points obtained by the linear transformation M of some point in $[-1,1]^{m+n}$. In the case of probabilistic affine forms, we proceed in a similar way: the concretization of the abstract element is the set of all probabilities on $\mathbb{R}^d$ that are obtained as the image under the linear transformation M of some probability on $[-1,1]^{n+m}$ which is *compatible* with the DS structures on the noise symbols. In the rest of the section, we formally define these notions of compatibility for probabilities.

We say that a probability P on $[-1,1]$ is compatible with a DS structure d, denoted $P \backsim d$, if and only if, for all $u \in [-1,1]$, $\underline{P_d}(u) \leq \int_{[-1,u]} dP \leq \overline{P_d}(u)$. This means that the cumulative distribution function (CDF, [14]) associated with the probability P belongs to the P-box constructed from the DSI d.

The collecting semantics that we use relies on sets of probability distributions P on $\mathbb{R}^d$. Up to a linear transformation, we must ensure that all probabilities marginals P_i are compatible with the DSI d_{ϵ_i} (or d_{η_i}). We recall that the marginal probability $P_{1,\ldots,k}$ on $[-1,1]^k$ of a probability distribution P on $[-1,1]^{n+m}$ is defined as follows, for all Borel sets B on $[-1,1]^k$:

$$P_{1,\ldots,k}(B) = \int_{\{x=(y,z)\in[-1,1]^k\times[-1,1]^{n+m-k} | y \in B\}} dP(x)$$

When $k = 1$, we get the probability marginal on a fixed coordinate. We can thus define the CDF marginal by the simple formula, for all $u \in [-1,1]$:

$$F_i(u) = \int_{\{x\in[-1,1]^{n+m} | x_i \in [-1,u]\}} dP_i(x)$$

Given a probability P on $[-1,1]^{n+m}$ and $n+m$ DSI on $[-1,1]$ denoted $ds_1, \ldots, ds_{n+m}$, we denote by $P \backsimeq (ds_1, \ldots, ds_{n+m})$ if and only if $\forall i \in [1, n+m]$, $P_i \backsim ds_i$.

Finally, we need to recall how we construct a probability distribution $[-1,1]^k$ from a probability on $[-1,1]^i$ and a probability on $[-1,1]^j$ where $i+j=k$. This will be needed as we will construct i marginal probabilities compatible with the DSI of the ε symbols and j compatible with the DSI of the η symbols, and then construct the probability on $[-1,1]^k$.

Let P_i and P_j two probability measures on respectively $[-1,1]^i$ and $[-1,1]^j$. We define the probability measure $P_i \otimes P_j$ as the unique probability measure on $[-1,1]^k$ such that, for all $A \in [-1,1]^i$ and $B \in [-1,1]^j$,

$$P_i \otimes P_j(A \times B) = P_i(A) \times P_j(B)$$

Now, given n central noise symbols ε_i and m perturbation symbols η_j, we define the probabilities on $[-1,1]^{n+m}$ compatible with them as the set of probabilities compatible with the DS structures attached to noise symbols and that are coherent with the independency of the noise symbol ε_i. Thus the ε-marginal probability is the product of the i-th marginal probabilities for $1 \leq i \leq n$. This is formally stated in Definition 3.

Definition 3 (Compatible probabilities). *Let $\varepsilon_1, \dots, \varepsilon_n$ and $\eta_1, \dots, \eta_m$ be noise symbols with attached DS structures d_{ε_i} and d_{η_j}. We define the set of compatible probabilities, denoted $P_{\varepsilon,\eta}$, as:*

$$P_{\varepsilon,\eta} = \left\{ \begin{array}{ll} & P \textit{ probabilities on } \mathbb{R}^{n+m} \textit{ such that:} \\ (1) & P \backsimeq (d_{\varepsilon_1}, \dots, d_{\varepsilon_n}, d_{\eta_1}, \dots, d_{\eta_m}) \\ (2) & P_\epsilon = P_{\varepsilon_1,\dots,\varepsilon_n} = \otimes_{i=1}^n P_{\epsilon_i} \end{array} \right\} .$$

As stated before, the concretization of a probabilistic affine form is the set of all previsions that are expressed as the image via the affine transformation of a prevision compatible with the DS structures of the noise symbols. We thus need to define the notion of image prevision (see Definition 4), then we can formally define the concretization function (see Definition 5).

Definition 4 (Probability image). *Let P be a probability on $[-1,1]^{n+m}$ and $M : [-1,1]^{n+m} \to \mathbb{R}^d$ be a measurable map. We define the probability image of P by M, denoted $M(P)$, as the probability on $\mathbb{R}^d$ given by $M(P)(B) = \lambda B.P(M^{-1}(B))$.*

Definition 5 (Concretization function). *Let $\rho^\sharp$ be a probabilistic affine form over d variables $x_1, \dots, x_d$ and with n independent noise symbols ε_i and m perturbation noise symbols η_j. For each $k \in [1,d]$, let*

$$\rho^\sharp(x_k) = \alpha_0^k + \sum_{i=1}^n \alpha_i^k \varepsilon_i + \sum_{j=1}^m \beta_j^k \eta_j$$

and let $M_{\rho^\sharp} \in \mathcal{M}^{k \times m+n}$ be the matrix as defined above. We define the concretization of $\rho^\sharp$, denoted $\gamma(\rho^\sharp)$ as:

$$\gamma(\rho^\sharp) = \Big\{ P \mid \exists P' \in P_{\varepsilon,\eta},\ P = M(P') \Big\} .$$

In other words, $\gamma(\rho^\sharp)$ is the image by the affine transformation $M_{\rho^\sharp}$ of the set of compatible probabilities $P_{\varepsilon,\eta}$.

5.2 Correctness results

Theorem 2 (Correctness of the abstraction). *Let Λ be a program and ℓ a label appearing in Λ. Let ρ be an initial environment for program variables and let $\rho^\sharp = [\![\Lambda]\!]^\sharp_\ell(\rho)$, then we have:*

$$\forall \kappa : \Sigma \to [0,1], \quad \left\{ \begin{array}{l} [\![\Lambda]\!]^-_{\ell,\kappa} 1\rho \geq \inf\Big\{ \int_{y \in \mathbb{R}^n} \kappa(y) dP \mid P \in \gamma(\rho^\sharp) \Big\} \\ [\![\Lambda]\!]^+_{\ell,\kappa} 0\rho \leq \sup\Big\{ \int_{y \in \mathbb{R}^n} \kappa(y) dP \mid P \in \gamma(\rho^\sharp) \Big\} \end{array} \right. . \qquad (4)$$

As usual, we prove this theorem by proving the correctness of each syntactic construction of the language. Due to the lack of space, we do not give all the proofs but give the main lemmas that are useful to prove this result. In particular, we show how the composition of programs impact the probabilistic semantics.

Lemma 1. *We have:* $[\![s_1;\ell' ::; s_2;\ell :]\!]^{\pm}_{\ell,\kappa} h\rho = [\![s_1]\!]^{\pm}_{\ell',[\![s_2]\!]^{\pm}_{\ell,\kappa} h} h\rho$.

Proof (Proof sketch). We prove it for $\pm = +$ and $h = 0$, the same proof runs easily for $\pm = -$ and $h = 1$. By the rules for ; and l' : of Figure 2 we deduce:

$$\begin{aligned}
[\![s_1;\ell' ::; s_2;\ell :]\!]^{+}_{\ell,\kappa} h\rho &= [\![s_1;\ell' :]\!]^{+}_{\ell,\kappa} \Big(\lambda\rho'.[\![s_2;\ell :]\!]^{+}_{\ell,\kappa} h\rho'\Big)\rho \\
&= [\![s_1;\ell' :]\!]^{+}_{\ell,\kappa} \Big(\lambda\rho'.[\![s_2]\!]^{+}_{\ell,\kappa} (\lambda\rho''.[\![\ell :]\!]^{+}_{\ell,\kappa} h\rho'')\rho'\Big)\rho \\
&= [\![s_1;\ell' :]\!]^{+}_{\ell,\kappa} \Big(\lambda\rho'[\![s_2]\!]^{+}_{\ell,\kappa} (\lambda\rho''.\kappa(\rho''))\rho'\Big)\rho \\
&= [\![s_1;\ell' :]\!]^{+}_{\ell,\kappa} \Big(\lambda\rho'[\![s_2]\!]^{+}_{\ell,\kappa} \kappa\rho'\Big)\rho \\
&= [\![s_1]\!]^{+}_{\ell,\kappa} \Big(\lambda\rho''.[\![\ell' :]\!]^{+}_{\ell,\kappa} (\lambda\rho'.[\![s_2]\!]^{+}_{\ell,\kappa} \kappa\rho')\rho''\Big)\rho \\
&= [\![s_1]\!]^{+}_{\ell,\kappa} \Big(\lambda\rho''.[\![s_1]\!]^{+}_{\ell,\kappa} \kappa\rho''\Big)\rho
\end{aligned}$$

And we also have, for all $\kappa : \Sigma \to [0,1]$: $[\![s_1;\ell' :]\!]^{+}_{\ell',\kappa} 0\rho = [\![s_1]\!]^{+}_{\ell',\kappa} (\kappa)\rho$, which ends the proof using the correct κ. □

We use Lemma 1 to prove that the abstract semantics is correct for Λ of the form $s_1;\ell' ::; x := e;\ell :$. Let thus $\rho \in \Sigma$ be the initial environment and let $\kappa : \Sigma \to [0,1]$ be a measurable map. Let $\rho^\sharp_1 = [\![s_1]\!]^\sharp_{\ell'}(\rho)$ and $\rho^\sharp = [\![\Lambda]\!]^\sharp_{\ell}(\rho)$. We have: $\rho^\sharp = [\![x := e]\!]^\sharp(\rho^\sharp_1)$, i.e. $\rho^\sharp$ is obtained by evaluating the assignment $x := e$ using probabilistic affine forms. We assume that Equation (4) is true for $\rho^\sharp_1$ and show that it remains true for $\rho^\sharp$. We thus have (for + and 0) $[\![s_1]\!]^{+}_{\ell',\kappa} 0\rho \le \sup_{P\in\gamma(\rho^\sharp_1)} \int_{y\in\mathbb{R}^n} \kappa(y)dP$. Now, using Lemma 1, we have $[\![\Lambda]\!]^{+}_{\ell,\kappa} 0h \le \sup_{P\in\gamma(\rho^\sharp_1)} \int_{y\in\mathbb{R}^n} [\![x := e; l :]\!]^{+}_{\ell,\kappa} 0ydP$ and $\int_{y\in\mathbb{R}^n} [\![x := e; l :]\!]^{+}_{\ell,\kappa} 0ydP = \int_{y\in\mathbb{R}^n} \kappa(y[x \mapsto [\![e]\!]y])dP$. Using the image-measure property, we get $\int_{y\in\mathbb{R}^n} [\![x := e; l :]\!]^{+}_{\ell,\kappa} 0ydP = \int_{y\in\mathbb{R}^n} \kappa(y)df(P)$ where $f : \mathbb{R}^n \to \mathbb{R}^n$ is the function $f(y)$ is y except for the dimension of x which is changed to $[\![e]\!]y$. According to the rules of our abstract semantics, we know that $\{f(P) \,|\, P \in \gamma(\rho^\sharp_1)\} \subseteq \gamma(\rho^\sharp)$, so we get:

$$[\![\Lambda]\!]^{+}_{\ell,\kappa} 0h \le \sup_{P\in\gamma(\rho^\sharp)} \int_{y\in\mathbb{R}^n} \kappa(y)dP$$

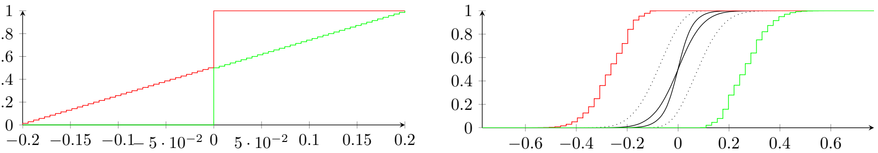
which ends the proof. The proofs for other statements are similar.

6 Experimentations

6.1 Running example

In this section, we describe on our running example the results of our analyzer which implements the abstract semantics we defined in Section 4. To assert the precision of our analysis, we compare these results with simulations of the same example with as inputs probability distributions within the set of possible inputs. Recall that our running example computes the iterations of the filter:

$$y_n = S = 0.7 * x_n - 1.3 * x_{n-1} + 1.1 * x_{n-2} + 1.4 * y_{n-1} - 0.7 * y_{n-2}$$



(a) Over-approximation of the input.

(b) P-box of the output (red-green) and simulations (black).

Fig. 4. Analysis on the running example.

where the inputs (x_n) are random variables following a uniform distribution between $-x$ and x, for any $x \in [0, 0.2]$. In other words, the inputs of the filter are the sets of all uniform distribution with support $[-x, x]$. For our analysis, we use as inputs a DSI that contains all these distributions; its is shown on Figure 4(a).

We first show the precision of our abstract domain by computing the 100^{th} iterate of the filter, without computing the union, i.e. we completely unfold the loop. The result is shown on Figure 4(b) on which we depict both the simulations and the P-box obtained by our abstract semantics. We can see that we obtain a correct over-approximation of all the distributions computed by the simulations. This over-approximation however is large because the input P-box we chose contains many more distributions than just the uniform ones on $[-x, x]$. We made some other simulations with such distributions (for example, distributions that follow closely the upper and lower functions of the P-box) and obtained the dotted curves of Figure 4(b) which are much closer to the P-box we computed. We get a distance between the lower and upper probabilities, in the abstract which is about twice as much as in our simulations, which is still quite precise.

6.2 Ferson polynomial

We now use an example from [13] to test the precision and performance of our abstract domain on arithmetic operations. The problem is to compute bounds on the solution of the differential equations

$$\dot{x}_1 = \theta_1 x_1 (1 - x_2) \quad \dot{x}_2 = \theta_2 x_2 (x_1 - 1) \tag{5}$$

under the assumption that the initial values are $x_1(0) = 1.2$ and $x_2(0) = 1.1$ but the parameters θ_1 and θ_2 are uncertain: they are given by a normal distribution with mean 3 and 1, resp., but with an unknown standard deviation in the range $[-0.01, 0.01]$. As in [13], we used VSPODE [28] to obtain a Taylor model polynomial that expresses the solution at $t_f = 20$ as an order 5 polynomial of θ_1 and θ_2. We then used the probabilistic affine forms to evaluate the Horner form of this polynomial. Figure 5 shows both the input DSI for θ_1 and the output DSI for x_1 at the final time. Our abstract domain is precise enough to correctly bound the output variables and the figure shows that we can also, with high probability, discard some values in the resulting interval. For example, we could show that $P(x_1 \leq 1.13) \leq 0.0552$, which is an even more precise enclosure than the one obtained by RiskCALC [13]. Our analysis took 104s on a 1.6Ghz laptop.

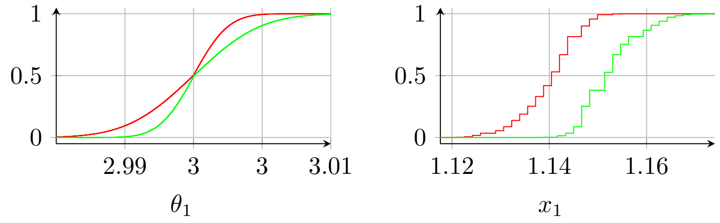


Fig. 5. DSI of the uncertain parameter and the output of problem (5).

6.3 Tank filling

Our final example is a simple modification of the tank filling program of [35] that can be found at `https://sites.google.com/site/probabilisticanalysis/`. It consists of a tank of volume V that is filled with water: at each time instant, water is added into the tank but an error is attached to the added volume of water and the measured volume is also equiped with a probabilistic error. The filling stops when the measured volume v_m is greater than V, and we are interested in knowing how long the filling process takes. The error on the inputs is as follows: the input volume at each time instant is given by a uniform law with support $[0.07, 0.13]$ and the error on the sensor that measures the volume is also a normal law but with support $[-0.03, 0.03]$, i.e. the sensor is very noisy.

We compute the affine form attached to the measured volume $v_m(i)$ at each time instant i and can thus bound the probability of the program ending in i time instants: we compute the upper and lower bound of the probability $P(v_m(i) < V)$ at each time instant. Then, we can prove that the program stops in less than 26 steps (with $V = 2$) as $P(v_m(26) \leq V) = 0$. We can also prove that the program ends in more than 20 steps with probability less than 0.63, which seems to be a rather precise estimate. Note that we can still slightly improve the precision of our analysis and decrease the bound 0.63 by increasing the maximal number of focal elements in DSI. This impact the performances (the computations are quadratic in the number of focal elements) but greatly increases precision. With 300 focal elements per DSI, we could prove that the program ends in more than 20 steps with probability less than 0.595.

We also made some experimentations on the EGFR example from [35] which computes the *Estimated Globular Filtration Rate* and studies the impact of noise on the computed value. Our model of probabilistic affine forms can efficiently handle such a problem as it tracks down the dependency between a variable (the EGFR) and its noisy version.

7 Conclusion

We have presented a framework for the verification of embedded programs with both probabilistic and non-deterministic inputs. In particular, we have defined a concrete collecting semantics using (higher and lower) previsions to enclose a set of probability distributions and an abstract semantics using probabilistic affine forms to efficiently compute an over-approximation of this set.

Note that our analysis is an extension of the purely non-deterministic case: if we have no information about the probability distribution associated to each noise symbol, we shall use DS structures with only one focal element and a weight of 1 and then get the same results as standard affine forms analysis.

In this work, we focused on numerical programs that usually appear in embedded systems and only treated them as open-loop programs, i.e. we ignored the feedback from the program to its plant. In the future, we shall extend this work to treat hybrid systems as in [6]. This will require to be able to handle ODEs with initial values given as probabilistic affine forms. As shown by our second benchmark, we think that we can extend guaranteed ODE solvers to make them compute with imprecise probabilities.

References

1. Arce, G.: Nonlinear Signal Processing: A Statistical Approach. Wiley (2005)
2. Auer, E., Luther, W., Rebner, G., Limbourg, P.: A verified matlab toolbox for the dempster-shafer theory. In: Workshop on the Theory of Belief Functions (2010)
3. Berleant, D., Goodman-Strauss, C.: Bounding the results of arithmetic operations on random variables of unknown dependency using intervals. Reliable Computing 4(2), 147–165 (1998)
4. Berleant, D., Xie, L., Zhang, J.: Statool: A tool for distribution envelope determination (denv), an interval-based algorithm for arithmetic on random variables. Reliable Computing 9, 91–108 (2003)
5. Bouissou, O., Goubault, E., Goubault-Larrecq, J., Putot, S.: A generalization of p-boxes to affine arithmetic. Computing pp. 1–13 (2011), 10.1007/s00607-011-0182-8
6. Bouissou, O., Goubault, E., Putot, S., Tekkal, K., Vedrine, F.: Hybridfluctuat: A static analyzer of numerical programs within a continuous environment. In: CAV. pp. 620–626. LNCS, Springer-Verlag (2009)
7. Busaba, J., Suwan, S., Kosheleva, O.: A faster algorithm for computing the sum of p-boxes. Journal of Uncertain Systems 4(4) (2010)
8. Choquet, G.: Theory of capacities. Annales de l'Institut Fourier 5, 131–295 (1953)
9. Comba, J.L.D., Stolfi, J.: Affine arithmetic and its applications to computer graphics. SEBGRAPI'93 (1993)
10. Cousot, P., Monereau, M.: Probabilistic abstract interpretation. In: Proc. ESOP'12, pp. 169–193. Springer Verlag LNCS 7211 (2012)
11. Destercke, S., Dubois, D., Chojnacki, E.: Unifying practical uncertainty representations - I: Generalized p-boxes. J. of Approximate Reasoning 49(3) (2008)
12. Destercke, S., Dubois, D., Chojnacki, E.: Unifying practical uncertainty representations. II: Clouds. Intl J. of Approximate Reasoning 49(3) (2008)
13. Enszer, J.A., Lin, Y., Ferson, S., Corliss, G.F., Stadtherr, M.A.: Probability bounds analysis for nonlinear dynamic process models. AIChE Journal 57(2) (2011)
14. Feller, W.: An Introduction to Probability Theory and Its Applications. Wiley (1968)
15. Feret, J.: Static analysis of digital filters. In: European Symposium on Programming (ESOP'04). No. 2986 in LNCS, Springer-Verlag (2004)
16. Ferson, S.: RAMAS Risk Calc 4.0 Software: Risk Assessment with Uncertain Numbers. Lewis Publishers (2002)
17. Ferson, S., Kreinovich, V., Ginzburg, L., Myers, D., Sentz, K.: Constructing probability boxes and Dempster-Shafer structures. Tech. Rep. SAND2002-4015, Sandia National Laboratories (2003)

18. Ferson, S.: What Monte-Carlo methods cannot do. Human and Ecological Risk Assessment 2, 990–1007 (1996)
19. Fuchs, M., Neumaier, A.: Potential based clouds in robust design optimization. J. Stat. Theory Practice 3, 225–238 (2009)
20. Ghorbal, K., Goubault, E., Putot, S.: A logical product approach to zonotope intersection. In: CAV. LNCS, vol. 6174 (2010)
21. Goubault, E., Putot, S.: A zonotopic framework for functional abstractions. CoRR abs/0910.1763 (2009)
22. Goubault, E., Putot, S.: Static analysis of finite precision computations. In: VMCAI'11. LNCS, vol. 6538, pp. 232–247 (2011)
23. Goubault-Larrecq, J.: Continuous capacities on continuous state spaces. In: ICALP'07. LNCS, vol. 4596, pp. 764–776. Springer (2007)
24. Goubault-Larrecq, J.: Continuous previsions. In: CSL'07. LNCS, vol. 4646, pp. 542–557. Springer (2007)
25. Goubault-Larrecq, J.: Prevision domains and convex powercones. In: FoSSaCS'08. LNCS, vol. 4962, pp. 318–333. Springer (2008)
26. Goubault-Larrecq, J., Keimel, K.: Choquet-Kendall-Matheron theorems for non-Hausdorff spaces. MSCS 21(3), 511–561 (2011)
27. Kwiatkowska, M., Norman, G., Parker, D.: Prism 4.0: Verification of probabilistic real-time systems. In: CAV, LNCS, vol. 6806, pp. 585–591. Springer (2011)
28. Lin, Y., Stadtherr, M.A.: Validated solution of initial value problems for odes with interval parameters. In: NSF Workshop on Reliable Engineering Computing (2006)
29. Mancini, R., Carter, B.: Op Amps for Everyone. Electronics & Electrical (2009)
30. McIver, A., Morgan, C.: Demonic, angelic and unbounded probabilistic choices in sequential programs. Acta Informatica 37(4/5), 329–354 (2001)
31. Mislove, M.: Nondeterminism and probabilistic choice: Obeying the law. In: CONCUR 2000. pp. 350–364. Springer Verlag LNCS 1877 (2000)
32. Monniaux, D.: Abstract interpretation of probabilistic semantics. In: SAS'00. pp. 322–339. No. 1824 in LNCS, Springer Verlag (2000)
33. Neumaier, A.: Clouds, fuzzy sets and probability intervals. Reliable Computing (2004)
34. Rump, S.: INTLAB - INTerval LABoratory. In: Csendes, T. (ed.) Developments in Reliable Computing, pp. 77–104. Kluwer Academic Publishers (1999)
35. Sankaranarayanan, S., Chakarov, A., Gulwani, S.: Towards static analysis for probabilistic programs. In: PLDI'13 (to appear) (2013)
36. Shafer, G.: A Mathematical Theory of Evidence. Princeton University Press (1976)
37. Sun, J., Huang, Y., Li, J., Wang, J.M.: Chebyshev affine arithmetic based parametric yield prediction under limited descriptions of uncertainty. In: ASP-DAC 2008. pp. 531–536. IEEE Computer Society Press (2008)
38. Terejanu, G., Singla, P., Singh, T., Scott, P.D.: Approximate interval method for epistemic uncertainty propagation using polynomial chaos and evidence theory. In: 2010 American Control Conference, Baltimore, Maryland (2010)
39. Tix, R.: Continuous D-Cones: Convexity and Powerdomain Constructions. Ph.D. thesis, Technische Universität Darmstadt (1999)
40. Tix, R., Keimel, K., Plotkin, G.: Semantic domains for combining probability and non-determinism. ENTCS 129, 1–104 (2005)
41. Walley, P.: Statistical Reasoning with Imprecise Probabilities. Chapman Hall (1991)
42. Williamson, R.C., Downs, T.: Probabilistic arithmetic I: Numerical methods for calculating convolutions and dependency bounds. J. Approximate Reasoning (1990)

A Revisions

List of revisions of the paper:

1. Thursday, 7 march 2013. 11:45pm (CET). Version 1.
2. Friday, 8 march 2013. 09:35 pm (CET). Version 1.1

B Operational semantics

We start with a small-step operational semantics, given in Figure 6. Its states are pairs $\big(\Lambda, \rho\big)$, where Λ is a finite list of statements, to be executed sequentially. The grammar for such lists is: $\Lambda ::= \epsilon \mid s \bullet \Lambda$. The states $\big(\epsilon, \rho\big)$ are final. The $\longrightarrow$ relation defined in Figure 6 form the deterministic part of the semantics, and should be clear. We write $\rho[x \mapsto v]$ for the environment that maps x to v, and every $y \neq x$ to $\rho(y)$.

To define the rest of the semantics, and in particular the semantics of the inputs $x_1, \ldots, x_k$:= `input`, we use judgments of the form $\ell \vdash \big(\Lambda, \rho\big) \downarrow^{\pm}_{\kappa} a$, for $\pm \in \{+, -\}$, where $\ell \in \mathcal{L}$, $a \in [0, 1]$ and $\kappa \colon \Sigma \to [0, 1]$ is a bounded measurable map[3]. When κ is the indicator map χ_E of a measurable subset E of Σ, the intended meaning will be: starting from Λ in environment ρ, the probability of reaching label ℓ with some environment in E is at least (resp. at most) a if $\pm$ is $-$ (resp. $+$). In general, it is practical to use general measurable maps κ instead of indicator maps χ_E, and the meaning of $\ell \vdash \big(\Lambda, \rho\big) \downarrow^{-}_{\kappa} a$ will be: the average value of $\kappa(\rho')$ over all environments ρ' reached when at label ℓ is at least a (at most a for $\ell \vdash \big(\Lambda, \rho\big) \downarrow^{+}_{\kappa} a$).

We should mention that our intuition here fails to capture an essential point. Consider a simple loop, say `while`(b) $\{ \ell\colon\ s_1 \}$, where s_1 may do some readings of the random inputs. There is no such thing as the probability of reaching program point ℓ. Instead, there is a probability of reaching ℓ in one turn of the loop, another probability of reaching ℓ in two turns of the loop, and so on. In general, for each $n \in \mathbb{N}$, there is a probability of reaching ℓ in exactly n turns. What we shall be interested in is the sup, resp. the inf, over all n, of these probabilities—and, more generally, the sup/inf over all n of the average value of κ over all runs that reach ℓ for the nth time. The judgment $\ell \vdash \big(\Lambda, \rho\big) \downarrow^{+}_{\kappa} a$ will state that whatever n is, the average value of κ over all runs reaching ℓ for the nth time is at most a, while $\ell \vdash \big(\Lambda, \rho\big) \downarrow^{+}_{\kappa} a$ will state that whatever n is, the average value of h over all reaching ℓ for the nth time is at least a.

A note to the expert: in effect, we are implementing a semantics with mixed non-deterministic and probabilistic choice. While inputs account for probabilistic choice, the statement ℓ: (for the given label at the left of $\vdash$) chooses non-deterministically whether it should stop right here and evaluate κ, or proceed. So our semantics is already not purely probabilistic, as in Kozen[4] and Panangaden[5].

[3] We equate Σ with $\mathbb{R}^{|\mathcal{V}ar^{+}|}$ with its standard σ-algebra.

[4] Kozen, D.: Semantics of probabilistic programs. Journal of Computer and Systems Sciences 30(2), 162–178 (1985)

[5] Panangaden, P.: Probabilistic relations. In: Baier, C., Huth, M., Kwiatkowska, M., Ryan, M. (eds.) Proceedings of PROBMIV'98. pp. 59–74 (1998)

One may also observe that the latter semantics are unsuited to our purposes, as they only observe the probability of reaching final states. As such, they are probabilistic analogues of *big-step* semantics. In abstract interpretation, we need to evaluate probabilities (largest, smallest) of reaching states that may not be final, such as those at label ℓ in our example above.

Since this is equally easy, and is needed later, we allow the distribution $\nu_{\mathtt{inp}}$ to vary in some set of probability measures over $\mathbb{R}^k$. For our purposes, it is practical to merely give a pair of a lower prevision $F^-_{\mathtt{inp}}$ and of an upper prevision $F^+_{\mathtt{inp}}$ on $\mathbb{R}^k$, and to consider those distributions $\nu_{\mathtt{inp}}$ that lie between them:

$$F^-_{\mathtt{inp}}(f) \leq \int_{(v_1,\dots,v_k)\in\mathbb{R}^k} f(v_1,\dots,v_k)d\nu_{\mathtt{inp}} \leq F^+_{\mathtt{inp}}(f) \tag{6}$$

for every bounded measurable map $f\colon \mathbb{R}^k \to [0,1]$. ($F^-_{\mathtt{inp}}$ and $F^+_{\mathtt{inp}}$ will be described through Dempster-Shafer structures.)

$$\big(x := e \bullet \Lambda, \rho\big) \longrightarrow \big(\Lambda, \rho[x \mapsto \llbracket e \rrbracket \rho]\big) \qquad \big((s_1; s_2) \bullet \Lambda, \rho\big) \longrightarrow \big(s_1 \bullet s_2 \bullet \Lambda, \rho\big)$$

$$\big((\mathtt{if}\ (b)\ \{\, s_1 \,\}\ \mathtt{else}\ \{\, s_2 \,\}) \bullet \Lambda, \rho\big) \longrightarrow \begin{cases} \big(s_1 \bullet \Lambda, \rho\big) \text{ if } \llbracket b \rrbracket \rho = 1 \\ \big(s_2 \bullet \Lambda, \rho\big) \text{ if } \llbracket b \rrbracket \rho = 0 \end{cases}$$

$$\big((\mathtt{while}(b)\ \{\, s_1 \,\}) \bullet \Lambda, \rho\big) \longrightarrow \begin{cases} \big(s_1 \bullet (\mathtt{while}(b)\ \{\, s_1 \,\}) \bullet \Lambda, \rho\big) & \text{if } \llbracket b \rrbracket \rho = 1 \\ \big(\Lambda, \rho\big) & \text{if } \llbracket b \rrbracket \rho = 0 \end{cases}$$

$$\frac{\ell \vdash \big(\Lambda', \rho'\big) \downarrow^{\pm}_{\kappa} a \quad \big(\Lambda, \rho\big) \longrightarrow \big(\Lambda', \rho'\big)}{\ell \vdash \big(\Lambda, \rho\big) \downarrow^{\pm}_{\kappa} a}\,(Det^{\pm})$$

$$\frac{\ell \vdash \big(\Lambda, \rho\big) \downarrow^{\pm}_{\kappa} a \quad \ell' \neq \ell}{\ell \vdash \big(\ell' \colon \bullet \Lambda, \rho\big) \downarrow^{\pm}_{\kappa} a}\,(\mathcal{L}^{\pm}_{\neq}) \qquad \frac{\ell \vdash \big(\Lambda, \rho\big) \downarrow^{\pm} a \quad \kappa(\rho) \bowtie^{\pm}_{\kappa} a}{\ell \vdash \big(\ell \colon \bullet \Lambda, \rho\big) \downarrow^{\pm}_{\kappa} a}\,(\mathcal{L}^{\pm}_{=})$$

$$\frac{}{\ell \vdash \big(\epsilon, \rho\big) \downarrow^{\pm}_{\kappa} a}\,(Fin^{\pm}) \quad \frac{}{\ell \vdash \big(\Lambda, \rho\big) \downarrow^{+}_{\kappa} 1}\,(\bot^{+}) \quad \frac{}{\ell \vdash \big(\Lambda, \rho\big) \downarrow^{-}_{\kappa} 0}\,(\bot^{-})$$

$$\frac{\overbrace{\ell \vdash \big(\Lambda, \rho[x_1 \mapsto v_1, \dots, x_k \mapsto v_k]\big) \downarrow^{\pm}_{\kappa} f(v_1, \dots, v_k))}^{(v_1,\dots,v_k)\in\mathbb{R}^k} \quad F^{\pm}_{\mathtt{inp}}(f) \bowtie^{\pm} a}{\ell \vdash \big(x_1, \dots, x_k := \mathtt{input} \bullet \Lambda, \rho\big) \downarrow^{\pm}_{\kappa} a}\,(Inp^{\pm})$$

Fig. 6. Operational semantics

The result is given by the derivation rules at the bottom of Figure 6, which are in a style inspired by[6]. We write $\bowtie^{\pm}$ for $\geq$ if $\pm$ is $-$, or for $\leq$ if $\pm$ is $+$. The $(Det^{\pm})$ rule is simple: if $\big(\Lambda, \rho\big) \longrightarrow \big(\Lambda', \rho'\big)$, then this is a deterministic computation step, and there is no label to observe κ on when in state $\big(\Lambda, \rho\big)$, so

[6] Goubault-Larrecq, J.: Full abstraction for non-deterministic and probabilistic extensions of PCF I: the angelic cases. Journal of Logic and Algebraic Programming (2012), submitted. Presented at the Domains X Workshop, Swansea, UK, 2011.

the average of κ must be taken on the rest of the execution, starting from $\big(\Lambda', \rho'\big)$. If this is above a (or below a; see premise), then the average of κ starting from $\big(\Lambda, \rho\big)$ (conclusion) must also be above/below a. $(\mathcal{L}^{\pm}_{\neq})$ is explained similarly: we do not observe κ at ℓ', since $\ell' \neq \ell$, and additionally the effect of ℓ': is a no-op. $(\mathcal{L}^{\pm}_{=})$ is more interesting, and is the only place where κ is really used. Let us investigate $(\mathcal{L}^{-}_{=})$, the other case being similar. The possible averages of κ at each time we reach label ℓ are exactly the current value $\kappa(\rho)$ of κ (since we *are* at label ℓ), and those obtained when we reach ℓ later. The first premise states that the latter averages are above a, while the second premise states that $\kappa(\rho) \geq a$. In any case, the possible averages of κ must be above a, and this is the conclusion of the rule.

The $(\mathit{Fin}^{\pm})$ rules state what happens on termination. Since ℓ is never reached in a terminated run (of length *zero*), the possible averages of κ on this run form an empty set: all such averages are below every $a \in [0, 1]$ (rule (Fin^{+})) and above every $a \in [0, 1]$ (rule (Fin^{-})). The $(\perp^{\pm})$ rules express the trivial facts that the average of a map κ with values in $[0, 1]$ must be between 0 and 1.

The $(\mathit{Inp}^{\pm})$ rule is a bit intimidating, since it has infinitely many premises—at least as many as there are tuples $(v_1, \ldots, v_k)$ in $\mathbb{R}^k$—and is parameterized by a bounded measurable map $f \colon \mathbb{R}^k \to [0, 1]$. This is mandated by the fact that $\nu_{\texttt{inp}}$ may be an arbitrary, not discrete, measure. We should be reassured by looking at (Inp^{-}) in a simple case, say when $k = 1$, and $\nu_{\texttt{inp}}$ implements a discrete random choice between $v_1 = 1.2$ with probability $1/6$ ($= f(1.2)$), $v_1 = 1.3$ with probability $1/2$, and $v_1 = 1.4$ with probability $1/3$. (Let us also take the $\leq$ signs in (6) to be equalities.) Then (Inp^{-}) specializes to the following rule (up to irrelevant premises):

$$\frac{\begin{array}{c}\ell \vdash \big(\Lambda, \rho[x_1 \mapsto 1.2]\big) \downarrow^{-}_{\kappa} a_1 \quad \ell \vdash \big(\Lambda, \rho[x_1 \mapsto 1.3]\big) \downarrow^{-}_{\kappa} a_2 \quad \ell \vdash \big(\Lambda, \rho[x_1 \mapsto 1.4]\big) \downarrow^{-}_{\kappa} a_3 \\ 1/6\ a_1 + 1/2\ a_2 + 1/3\ a_3 \geq a\end{array}}{\ell \vdash \big(x_1, \ldots, x_k := \texttt{input} \bullet \Lambda, \rho\big) \downarrow^{-}_{\kappa} a.}$$

In particular, if you think of a_1 as the (minimal) average value of κ when x_1 is set to 1.2, and similarly for a_2 and a_3, this states that the values a below the (minimal) average value that κ takes when running $x_1, \ldots, x_k := \texttt{input} \bullet \Lambda$ are exactly those below the average $1/6\ a_1 + 1/2\ a_2 + 1/3\ a_3$ that one should expect.

C Adequacy theorem

We here prove that the operational and denotational semantics are equivalent. On the operational side, note that whenever $\ell \vdash \big(\Lambda, \rho\big) \downarrow^{-}_{\kappa} a$ is derivable and $a \geq b$, then $\ell \vdash \big(\Lambda, \rho\big) \downarrow^{-}_{\kappa} b$ is also derivable. So the set of values a such that $\ell \vdash \big(\Lambda, \rho\big) \downarrow^{-}_{\kappa} a$ is derivable is a downward-closed interval $[0, c]$ or $[0, c)$: let us write $[\Lambda]^{-}_{\ell,\kappa}\rho$ for c, the sup of these values a. Similarly, we write $[\Lambda]^{+}_{\ell,\kappa}\rho$ for the inf of the values a such that $\ell \vdash \big(\Lambda, \rho\big) \downarrow^{+}_{\kappa} a$ is derivable. Write 0 for the constant 0 map, and similarly for 1.

Theorem 3 (Adequacy). *$[\![\Lambda]\!]^{-}_{\ell,\kappa} 1 \rho = [\Lambda]^{-}_{\ell,\kappa}\rho$, and $[\![\Lambda]\!]^{+}_{\ell,\kappa} 0 \rho = [\Lambda]^{+}_{\ell,\kappa}\rho$.*

Proof. We deal with the $-$ case, as the $+$ case is similar.

($\geq$) We first show that $\llbracket \Lambda \rrbracket^-_{\ell,\kappa} 1\rho \geq [\Lambda]^-_{\ell,\kappa}\rho$. Equivalently, we show that for every $a \in [0,1]$ such that $\ell \vdash \big(\Lambda, \rho\big) \downarrow^-_\kappa a$ is derivable, then $\llbracket \Lambda \rrbracket^-_{\ell,\kappa} 1\rho \geq a$. This is by structural induction on the given derivation. We look at each rule in turn.

(Fin^-). We must show that $\llbracket \epsilon \rrbracket^-_{\ell,\kappa} 1\rho \geq a$, which is obvious since $\llbracket \epsilon \rrbracket^-_{\ell,\kappa} 1\rho = 1$.

($\bot^-$). $\llbracket \Lambda \rrbracket^-_{\ell,\kappa} 1\rho \geq 0$, by Theorem 1, first part.

(Det^-). For each rule $\big(\Lambda, \rho\big) \longrightarrow \big(\Lambda', \rho'\big)$, one checks easily that $\llbracket \Lambda \rrbracket^-_{\ell,\kappa} 1\rho = \llbracket \Lambda' \rrbracket^-_{\ell,\kappa} 1\rho$. By the induction hypothesis, the right-hand side is $\geq a$, so this is also the case of the left-hand side.

($\mathcal{L}^-_{\neq}$). We must show that $\llbracket \ell' \colon \bullet \Lambda \rrbracket^-_{\ell,\kappa} 1\rho \geq a$, where the induction hypothesis gives us $\llbracket \Lambda \rrbracket^-_{\ell,\kappa} 1\rho \geq a$. This is again clear, since $\llbracket \ell' \colon \bullet \Lambda \rrbracket^-_{\ell,\kappa} 1\rho = \llbracket \Lambda \rrbracket^-_{\ell,\kappa} 1\rho$.

($\mathcal{L}^-_{=}$). The induction hypothesis now gives us that not only $\llbracket \Lambda \rrbracket^-_{\ell,\kappa} 1\rho \geq a$, but also $\kappa(\rho) \geq a$. So $\llbracket \ell \colon \bullet \Lambda \rrbracket^-_{\ell,\kappa} 1\rho = \llbracket \ell \colon \rrbracket^-_{\ell\kappa} (\llbracket \Lambda \rrbracket^-_{\ell,\kappa} 1)\rho = \min(\kappa(\rho), \llbracket \Lambda \rrbracket^-_{\ell,\kappa} 1\rho) \geq a$.

(Inp^-). The induction hypothesis gives us a measurable map $f \colon \mathbb{R}^k \to [0,1]$, with the property that, for every $(v_1, \ldots, v_k) \in \mathbb{R}^k$, $\llbracket \Lambda \rrbracket^-_{\ell,\kappa} 1(\rho[x_1 \mapsto v_1, \ldots, x_k \mapsto v_k]) \geq f(v_1, \ldots, v_k)$, and $F^-_{\mathtt{inp}}(f) \geq a$. Since $F^-_{\mathtt{inp}}$ is monotonic, $F^-_{\mathtt{inp}}(\lambda v_1, \ldots, v_k \cdot \llbracket \Lambda \rrbracket^-_{\ell,\kappa} 1(\rho[x_1 \mapsto v_1, \ldots, x_k \mapsto v_k])) \geq F^-_{\mathtt{inp}}(f) \geq a$. But the left hand side is exactly $\llbracket x_1, \ldots, x_k := \mathtt{input} \rrbracket^-_{\ell,\kappa} (\llbracket \Lambda \rrbracket^-_{\ell,\kappa} 1)\rho = \llbracket x_1, \ldots, x_k := \mathtt{input} \bullet \Lambda \rrbracket^-_{\ell,\kappa} 1\rho$.

($\leq$) The converse inequality is harder. We shall show that for every $a \ll \llbracket \Lambda \rrbracket^-_{\ell,\kappa} 1\rho$ (implicitly, with $a \geq 0$), there is a derivation of $\ell \vdash \big(\Lambda, \rho\big) \downarrow^-_\kappa a$. (The $\ll$ relation is the so-called way-below relation on $[0,1]$, and is defined by $a \ll b$ iff $a < b$ or $a = 0$. Note that every $b \in [0,1]$ is the sup of the values a such that $a \ll b$. Moreover, if $a \ll b$, then for every sequence $b_0 \leq b_1 \leq \ldots \leq b_n \leq \ldots$ whose sup is at least b, then $a \leq b_n$ for n large enough, a property that we shall the Fundamental Property of $\ll$.) This is proved by double induction on the number of statements in Λ first, and when non-empty, by induction on the structure of the first statement in Λ.

Base case, $\Lambda = \epsilon$. We simply apply rule (Fin^-), since $a \in [0,1]$, which follows from the first part of Theorem 1.

In the inductive case, we consider a non-empty list, say of the form $s \bullet \Lambda$, and some $a \ll \llbracket s \bullet \Lambda \rrbracket^-_{\ell,\kappa} 1\rho = \llbracket s \rrbracket^-_{\ell,\kappa} (\llbracket \Lambda \rrbracket^-_{\ell,\kappa} 1)\rho$. We must exhibit a derivation of $\ell \vdash \big(s \bullet \Lambda, \rho\big) \downarrow^-_\kappa a$, under the following two induction hypotheses, which we name for future reference:

(H_1) for every $\rho' \in \Sigma$, for every $a' \ll \llbracket \Lambda \rrbracket^-_{\ell,\kappa} 1\rho'$, there is a derivation of $\ell \vdash \big(\Lambda, \rho'\big) \downarrow^-_\kappa a'$;

(H_2) for every proper substatement s' of s, for every list Λ', for every $\rho' \in \Sigma$, for every $a' \ll \llbracket s' \rrbracket^-_{\ell,\kappa} (\llbracket \Lambda' \rrbracket^-_{\ell,\kappa} 1)\rho'$, there is a derivation of $\ell \vdash \big(s' \bullet \Lambda', \rho'\big) \downarrow^-_\kappa a'$.

Assignment. $s = (x := e)$. By assumption, $a \ll \llbracket x := e \bullet \Lambda \rrbracket^-_{\ell,\kappa} 1\rho = \llbracket x := e \rrbracket^-_{\ell,\kappa} (\llbracket \Lambda \rrbracket^-_{\ell,\kappa} 1)\rho = \llbracket \Lambda \rrbracket^-_{\ell,\kappa} 1(\rho[x \mapsto \llbracket e \rrbracket \rho])$. Now use ($H_1$) with $\rho' = \rho[x \mapsto \llbracket e \rrbracket \rho]$ and $a' = a$. We obtain a derivation of $\ell \vdash \big(\Lambda, (\rho[x \mapsto \llbracket e \rrbracket \rho])\big) \downarrow^-_\kappa a$. Now use ($Det^-$) on the latter, and we obtain a derivation of $\ell \vdash \big(x := e \bullet \Lambda, \rho\big) \downarrow^-_\kappa a$.

The case of labels $\ell' \colon$ (with $\ell' = \ell$, or with $\ell' \neq \ell$) is similar.

Sequences. $s = (s_1; s_2)$. By assumption, $a \ll \llbracket s_1; s_2 \bullet \Lambda \rrbracket^-_{\ell,\kappa} 1\rho$, namely, $a \ll \llbracket s_1 \rrbracket^-_{\ell,\kappa}(\llbracket s_2 \rrbracket^-_{\ell,\kappa}(\llbracket \Lambda \rrbracket^-_{\ell,\kappa} 1))\rho$. Use (H_2) with $s' = s_1$, $\Lambda' = s_2 \bullet \Lambda$, $\rho' = \rho$, $a' = a$ and obtain a derivation of $\ell \vdash \big(s_1 \bullet s_2 \bullet \Lambda, \rho\big) \downarrow^-_\kappa a$. Add an instance of (Det^-) to obtain a derivation of $\ell \vdash \big((s_1; s_2) \bullet \Lambda, \rho\big) \downarrow^-_\kappa a$, and we are done.

Tests $\texttt{if}\ (b)\ \{\, s_1 \,\}\ \texttt{else}\ \{\, s_2 \,\}$ are dealt with similar, using (H_2) with $s' = s_1$ if $\llbracket b \rrbracket \rho = 1$, with $s' = s_2$ if $\llbracket b \rrbracket \rho = 0$.

Inputs. $s = x_1, \ldots, x_k := \texttt{input}$. Define $f(v_1, \ldots, v_k) = \llbracket \Lambda \rrbracket^-_{\ell,\kappa} 1(\rho[x_1 \mapsto v_1, \ldots, x_k \mapsto v_k])$. This is a measurable map from $\mathbb{R}^k$ to $[0,1]$. For every $\epsilon > 0$, let $f_\epsilon(v_1, \ldots, v_k) = \max(f(v_1, \ldots, v_k) - \epsilon, 0)$. Note that this is way below $f(v_1, \ldots, v_k)$. By (H_1) with $\rho' = \rho[x_1 \mapsto v_1, \ldots, x_k \mapsto v_k]$, $a' = f_\epsilon(v_1, \ldots, v_k)$, there is a derivation of $\ell \vdash \big(\Lambda, ()\rho[x_1 \mapsto v_1, \ldots, x_k \mapsto v_k]\big) \downarrow^-_\kappa f_\epsilon(v_1, \ldots, v_k)$, one for each tuple $(v_1, \ldots, v_k) \in \mathbb{R}^k$. Since $F^-_{\texttt{inp}}$ is monotonic and ω-continuous, $\big(F^-_{\texttt{inp}}(f_{1/n})\big)_{n \in \mathbb{N}}$ is a monotone sequence whose sup is $F^-_{\texttt{inp}}(f)$. But $F^-_{\texttt{inp}}(f) = \llbracket x_1, \ldots, x_k := \texttt{input} \bullet \Lambda \rrbracket^-_{\ell,\kappa} 1\rho$, by definition of the right-hand side, and a is way below the latter. So $a \ll \sup_{n \in \mathbb{N}} F^-_{\texttt{inp}}(f_{1/n})$, which implies that $a \leq F^-_{\texttt{inp}}(f_{1/n})$ for n large enough, by the Fundamental Property of $\ll$. We can now apply rule (Inp^-) (with f replaced by $f_{1/n}$) and the result is a derivation of $\ell \vdash \big(x_1, \ldots, x_k := \texttt{input} \bullet \Lambda, \rho\big) \downarrow^-_\kappa a$.

While loops. $s = (\texttt{while}(b)\ \{\, s_1 \,\})$. Since $a \ll \llbracket s \rrbracket^-_{\ell,\kappa}(\llbracket \Lambda \rrbracket^-_{\ell,\kappa} 1)\rho$, it is plain to see that there is a $b \in [0,1]$ such that $a \ll b \ll \llbracket s \rrbracket^-_{\ell,\kappa}(\llbracket \Lambda \rrbracket^-_{\ell,\kappa} 1)\rho$. Since $\llbracket s \rrbracket^-_{\ell,\kappa} = \llbracket \texttt{while}(b)\ \{\, s_1 \,\} \rrbracket^-_{\ell,\kappa}$ is defined as the sup of a monotone chain, the Fundamental Property of $\ll$ applies to conclude that $a \ll b \leq H^i_{b,s_1}(\bot^-)(\llbracket \Lambda \rrbracket^-_{\ell,\kappa} 1)\rho = H^i_{b,s_1}(0)(\llbracket \Lambda \rrbracket^-_{\ell,\kappa} 1)\rho$, for some $i \in \mathbb{N}$, using the notations of Figure 2. It now suffices to show that there is a derivation of $\ell \vdash \big((\texttt{while}(b)\ \{\, s_1 \,\}) \bullet \Lambda, \rho\big) \downarrow^-_\kappa a$, and we do this by an auxiliary induction on i.

If $i = 0$, then $a \ll H^i_{b,s_1}(0)(\llbracket \Lambda \rrbracket^-_{\ell,\kappa} 1)\rho = 0$ implies $a = 0$, and we apply rule $(\bot^-)$. (This is the only purpose of this rule: to be able to derive $\ell \vdash \big((\texttt{while}(b)\ \{\, s_1 \,\}) \bullet \Lambda, \rho\big) \downarrow^-_\kappa 0$ when the while loop does not terminate; without it, we would simply have no derivation at all.) If $i \geq 1$, then we have two cases.

If $\llbracket b \rrbracket \rho = 0$, then $H^i_{b,s_1}(0)(\llbracket \Lambda \rrbracket^-_{\ell,\kappa} 1)\rho = H_{b,s_1}(H^{i-1}_{b,s_1}(0))(\llbracket \Lambda \rrbracket^-_{\ell,\kappa} 1)\rho = \llbracket \Lambda \rrbracket^-_{\ell,\kappa} 1\rho$. By (H_1) with $a' = a$ and $\rho' = \rho$, there is a derivation of $\ell \vdash \big(\Lambda, \rho\big) \downarrow^-_\kappa a$. Now we apply (Det^-) with the rule $\big((\texttt{while}(b)\ \{\, s_1 \,\}) \bullet \Lambda, \rho\big) \longrightarrow \big(\Lambda, \rho\big)$ to obtain a derivation of $\ell \vdash \big((\texttt{while}(b)\ \{\, s_1 \,\}) \bullet \Lambda, \rho\big) \downarrow^-_\kappa a$.

If $\llbracket b \rrbracket \rho = 1$, then $H^i_{b,s_1}(0)(\llbracket \Lambda \rrbracket^-_{\ell,\kappa} 1)\rho = H_{b,s_1}(H^{i-1}_{b,s_1}(0))(\llbracket \Lambda \rrbracket^-_{\ell,\kappa} 1)\rho$, which is equal to $\llbracket s_1 \rrbracket^-_{\ell,\kappa}(H^{i-1}_{b,s_1}(0)(\llbracket \Lambda \rrbracket^-_{\ell,\kappa} 1))\rho$. By the definition of the semantics of $\texttt{while}$ as a sup, $H^{i-1}_{b,s_1}(0)(\llbracket \Lambda \rrbracket^-_{\ell,\kappa} 1) \leq \llbracket \texttt{while}(b)\ \{\, s_1 \,\} \rrbracket^-_{\ell,\kappa}(\llbracket \Lambda \rrbracket^-_{\ell,\kappa} 1) = \llbracket \texttt{while}(b)\ \{\, s_1 \,\} \bullet \Lambda \rrbracket^-_{\ell,\kappa} 1$. Since $\lambda h \cdot \llbracket s_1 \rrbracket^-_{\ell\kappa} h\rho$ is monotonic (as a prevision, see Theorem 1), we obtain $a \ll \llbracket s_1 \rrbracket^-_{\ell,\kappa}(\llbracket \texttt{while}(b)\ \{\, s_1 \,\} \bullet \Lambda \rrbracket^-_{\ell,\kappa} 1)\rho$. By (H_2), we obtain a derivation of $\ell \vdash s_1, \llbracket \texttt{while}(b)\ \{\, s_1 \,\} \bullet \Lambda \rrbracket^\rho \downarrow^-_\kappa a$. Apply (Det^-) with the rule

$$\big((\texttt{while}(b)\ \{\, s_1 \,\}) \bullet \Lambda, \rho\big) \longrightarrow \big(s_1, (\texttt{while}(b)\ \{\, s_1 \,\}) \bullet \Lambda, \rho\big)$$

this yields the desired derivation of $\ell \vdash \big(\texttt{while}(b)\ \{\, s_1 \,\} \bullet \Lambda, \rho\big) \downarrow^-_\kappa a$. □