

Concrete Semantics of Programs with Non-Deterministic and Random Inputs

Assalé Adjé and Jean Goubault-Larrecq

June 15, 2013

Abstract

This document gives semantics to programs written in a C-like programming language, featuring interactions with an external environment with noisy and imprecise data.

1 Introduction

The purpose of this report is to define a concrete semantics for a toy imperative language, meant to incorporate the essential features of languages such as C, as used in numerical control programs such as those used in the ANR CPP project.

Some of the distinctive aspects of these programs are: the prominent use of floating-point operations; and the fact that these programs read inputs from sensors. Both these features imply that the values of numerical program variables are *uncertain*. Floating-point operations are vulnerable to round-off errors, which can be modeled as quantization noise. Uncertainty is probably more manifest with sensors, which return values up to some measurement error. This measurement error can be described by giving guaranteed bounds (this is *non-determinism*: any value in the interval can be the actual value), or by giving a probability distribution (this is *randomness*: some values are more likely than others), or a combination of both. To deal with the latter, more complex combinations, we rest on variants of two semantic constructions that were studied by the first author, *provisions* [Gou07] and *capacities* [GL07].

The main goal of a concrete semantics is to serve as a reference. In our case, we wish to be able to prove the validity of associated abstract semantics and static analysis algorithms, as presented in other CPP deliverables. The kind of abstract semantics we are thinking of was produced, as part of CPP, in [BGGP11]. (While it might seem strange that the publication of the abstract semantics predates the design of the concrete semantics, one might say that both were developed at roughly the same time, with an eye on each other.) So one of our constraints was to ensure that our concrete semantics should make it easy to justify the abstract semantics we intend.

Before we start, we should also mention an important point. Numerical programs manipulate floating-point values, which are values from a finite set meant to denote some approximate real values. It is customary to think of floating-point values as reals, up to some error. This is why we shall define a first semantics, called the *real semantics*, where variables hold actual reals, and no round-off is performed at all. This has well-defined mathematical contents, but is not what genuine C programs compute. So we define a second semantics, the *floating-point semantics*, which is meant to faithfully denote what C programs compute, but works on floating-point data, mathematically an extremely awkward concept: e.g., floating-point addition is not associative, has one absorbing element (NaN), has no inverse in general (the opposite of infinity `inf`, `-inf`, is not an inverse since the sum of `inf` and `-inf` is NaN, not 0). But the two semantics are related, through *quantization*, which is roughly the process of rounding a real number to the nearest floating-point value.

While the real semantics is much simpler to define than the floating-point semantics without random choice or non-determinism (e.g., the semantics of `+` is merely addition), the situation

changes completely in the presence of random or non-deterministic choice. Let us explain this briefly. The prevision semantics of the style presented in [Gou07] is based on *continuous* maps and continuous previsions. This is perfectly coherent for ordinary, non-numerical programs (or for numerical programs in the floating-point semantics, where the type of floating-point numbers is merely yet another finite data type). However, this is completely at odds with the real semantics. To give a glimpse of the difficulty, one can define the Heaviside function $\chi_{[0,+\infty)}$ as a numerical C program with real semantics, say by if $x < 0$ then 0.0 else 1.0, and this is definitely not continuous. The deep problem is that, up to some inaccuracies, continuous semantics cannot describe more than computable operations, but the real semantics must be *non-computable*: even if we restricted ourselves to computable reals, testing whether a computable real is equal to 0 is undecidable.

There are at least two ways to resolve this conundrum. The first one is to cling to the continuous semantics of random choice and non-determinism of [Gou07] or [GL07], and work not on reals (or tuples of reals, in \mathbb{R}^n , representing the list of values of all n program variables), but rather on a *computational model* of \mathbb{R}^n . The notion of computational model of a topological space originates from Lawson [Law97]. For example, the dequo of non-empty closed intervals of reals is a computational model for \mathbb{R} , and the Heaviside map would naturally be modeled as the function mapping every negative real to 0, every positive real to 1, and 0 to the interval $[0, 1]$. This is elegant, mathematically well-founded, and would allow us to reuse the continuous constructions of [Gou07] or [GL07]. But it falls short of giving an account of real number computation as operating on *reals*.

We shall explore the second way here: we shall give a real semantics in terms of *measurable*, not continuous, maps. This will give us the required degrees of freedom to define our semantics—e.g., the Heaviside map *is* measurable—while allowing us to define the semantics of random, non-deterministic and mixed choice: anticipating slightly on future sections, this involves generalized forms of integration, which will be well-defined precisely on measurable maps. We develop the required theory in sections to come, by analogy with both the classical Lebesgue theory of integration and the above cited work on continuous previsions and capacities.

In the presence of random choice only (no non-determinism), our semantics will be isomorphic to Kozen’s semantics of probabilistic programs [Koz81], and his clauses for computing expectations backwards will match our prevision-based semantics. The semantics we shall describe in the presence of other forms of choice (non-deterministic, mixed) are new.

Outline. In Section 2, we introduce the syntax of the programs analyzed. In Section 3, we define the maps which are used to pass from floating points to real numbers and vice versa. In Section 4, we define the concrete semantics of expressions and tests and prove the measurability of the semantics. In Section 5, we define our concrete semantics based as a continuation-passing semantics. We also prove in Section 5, the link between our semantics and the theory of previsions. Finally, in Section 6, we treat separately the semantics of the instructions *input*.

2 Syntax

Let \mathcal{V} be a countable set of so-called (program) variables. For each operation op on real numbers, we reserve the symbol \acute{op} for a syntactic operation meant to implement op (in the real semantics) or some approximation of op (in the floating-point semantics). The syntax of a simple imperative language working on real/floating-point values is given in Figure 1. This syntax does not include any non-deterministic or probabilistic choice construct: uncertainty will be in the initial values of the variables, and will not be created by the program while running.

<i>expr</i>	::=	<i>a</i>	$a \in \mathbb{Q}$
		<i>x</i>	$x \in \mathcal{V}$
		$\dot{-}expr$	
		$expr \dot{+} expr$	
		$expr \dot{-} expr$	
		$expr \dot{\times} expr$	
		$expr \dot{/} expr$	
<i>test</i>	::=	$expr \dot{<}= expr$	
		$expr \dot{<} expr$	
		$expr \dot{=} expr$	
		$expr \dot{!} = expr$	
		$!test$	
<i>inst</i>	::=	$\ell skip$	
		$\ell x = expr$	$x \in \mathcal{V}$
		$\ell \text{if } test \text{ then } \{inst\} \text{ else } \{inst\}$	
		$\ell \text{while } test \{inst\}$	
		$inst ; inst$	

Figure 1: Syntax of Programs

3 Conversion between Floating Point and Real Numbers

We shall consider two different semantics in Section 4. The first one implements arithmetic with floating-point numbers, while the second one relies on actual real numbers. Here, we describe the two types and how we convert between them.

However, one should first be aware of the pitfalls that are hidden in such a task [Mon08]. First and foremost, floating-point numbers are meant to give approximations to real numbers, but floating-point computations may give values that are arbitrarily far from the corresponding real number computation. Monniaux (op. cit., Section 5) gives the example of the following program:

```
double modulo(double x, double mini, double maxi) {
    double delta = maxi-mini;
    double decl = x-mini;
    double q = decl/delta;
    return x - floor(q)*delta
}

int main() {
    double m = 180.;
    double r = modulo(nextafter(m,0.), -m, m);
}
```

In a semantics working on real numbers, `modulo` would return the unique number z in the interval $[mini, maxi)$ such that $x - z$ is a multiple of the interval length $maxi - mini$. So, certainly, whatever `nextafter` actually computes, r should be in the interval $[-180, 180)$.

However, running this using IEEE 754 floating-point arithmetic may (and usually will) return -180.00000000000000284 for r . (Here we need to say that `nextafter(m,0.)` returns the floating-point that is maximal among those that are strictly smaller than m . This has no equivalent in the

world of real numbers, and accordingly our language does not include this function.) This is only logical:

- When we enter `modulo`, `x` is equal to $180 - 2^{-45}$;
- Then `maxi - mini` is computed ($= 360$), and `x - mini` is computed ($= 360 - 2^{-45}$); these values are then rounded to the nearest floating-point number, and this is 360 in *both* cases;
- so `delta`, `decl` are both equal to 360, `q` equals 1;
- so `modulo` returns (the result of rounding applied to) $(180 - 2^{-45}) - (1 \times 360) = -180 - 2^{-45} \simeq -180.0000000000000284$.

Of course, the right result, if computed using real numbers instead of floating-point numbers, should be $180 - 2^{-45} \simeq 179.9999999999999716$.

This example can be taken as an illustration of the fact that, even though one can think of each *single* operation (addition, product, etc.) as being implemented in floating-point computation as though one first computed the exact, real number result first, and then rounded it, hence obtaining a best possible approximant, this is no longer true for whole programs.

Monniaux goes further, and stresses the fact that various choices in compiler options (e.g., `x87` vs. IEEE 754 arithmetic), IEEE 754 rounding modes, abusive optimization strategies (e.g., where the compiler uses the fact that addition is associative, which is wrong in floating-point arithmetic, see op. cit., Section 4.3.2), processor-dependent optimization strategies (e.g., see op. cit., Section 3.2, about the use of the multiply-and-add assembler instruction on PowerPC micro-processors), pragmas (op. cit., Section 4.3.1), all may result in surprising changes in computed values.

This causes difficulties in defining sound semantics for floating-point programs, discussed in op. cit., Section 7.3.

But our purpose is not to verify arbitrary numerical programs, and one can make some simplifying assumptions:

1. We assume that floating-point arithmetic is performed using the IEEE 754 standard on floating-point values of a standard, fixed size, typically the 64-bit IEEE 754 (“double”) type. By this, we not only mean that the basic primitives are implemented as the standard prescribes, but that all floating-point values are stored in this format, even when stored in registers. This is meant to avoid the sundry, dreaded problems mentioned by Monniaux with the use of `x87` arithmetic (where registers hold 80-bit intermediate values).
2. We assume that the rounding mode is fixed, once and for all for all programs. In particular, calls to functions that change the rounding mode on the fly are prohibited.
3. We assume that all optimizations related to floating-point computations are turned off. This is meant to avoid abusive (unsound) optimizations (e.g., assuming associativity), and also to avoid processor-dependent optimizations (e.g., compiling $a \times x + b$ using a single multiply-and-add instruction: this skips the intermediate rounding that should have occurred when computing $a \times x$, and therefore changes the floating-point semantics).
4. We assume that the only floating-point operations allowed are arithmetic operations (i.e., $+$, $-$, \times , $/$, but not `nextafter` for example, or the `%f`, `%g` and related directives of `printf`, `scanf` and relatives; nor casts to and from the `int` type—which we shall actually omit). Library functions such as `sin`, `cos`, `exp`, `log` would be allowable in principle, and their semantics would follow the same ideas as presented below—provided we make sure that their implementations produce results that are correct in the ulp as well (i.e., that they are computed as though the exact result was computed, then rounded; the *ulp*, a.k.a., the unit in the last place, is the least significant bit of the mantissa).

These assumptions allow us to simplify our semantics considerably.

Let us go on with the actual data types of floating-point, resp. real numbers. The IEEE 754 standard specifies that, in addition to values representing real numbers, floating-point values include values denoting $+\infty$ (which we write **inf**), $-\infty$ (**-inf**), and silent errors (**NaN**, for “not a number”). One can obtain the first two through arithmetic overflow, e.g., by computing $1.0/0.0$ or $-1.0/0.0$, and **NaNs**, e.g., by computing **inf** $-$ **inf**. Under Assumption (4) above, there will be no way of distinguishing any such values through the execution of expressions. We abstract them all into a unique symbol **err** (*error*).

An added benefit of this abstraction is that it dispenses us from considering the difference between the two zeroes, $+0.0$ and -0.0 , of IEEE 754 arithmetic. These are meant to satisfy $1.0/\mathbf{inf} = +0.0$, $1.0/-\mathbf{inf} = -0.0$, but are otherwise equal, in the sense that the equality predicate applied to $+0.0$ and -0.0 must return true. Collapsing **inf**, **-inf**, and **NaN** into just one value **err** therefore also allows us to confuse the two zeroes, without harm. This is important if we stick to our option that single floating-point operations should computed the exact result then round: rounding the real number 0 to the nearest would be a nonsense with two floating-point numbers representing 0.

The error **err** is absorbing for all standard arithmetic operations. This means that in our semantic definitions we assume that the error is propagated during the execution of a program which contains these special numbers. Now, we extend by the error symbol **err** the classical sets of floating points and real numbers, which we denote respectively by \mathbb{F} and \mathbb{R} . This yields two new sets: $\mathbb{F}_e = \mathbb{F} \cup \{\mathbf{err}\}$ and $\mathbb{R}_e = \mathbb{R} \cup \{\mathbf{err}\}$.

Convention 1

Let r be in \mathbb{R}_e . Let \diamond be in $\{+, -, \times, /\}$. Then:

$$\mathbf{err} \diamond r = r \diamond \mathbf{err} = r/0 = -\mathbf{err} = \mathbf{err}$$

We consider floating point as special real numbers. Formally, there is a canonical injection **inj** that lets us to convert a floating-point value (in \mathbb{F}_e) into a real number (in \mathbb{R}_e):

$$\begin{aligned} \mathbf{inj} : \mathbb{F}_e &\rightarrow \mathbb{R}_e \\ f &\mapsto \mathbf{inj}(f) = \begin{cases} \mathbf{err} & \text{if } f = \mathbf{err} \\ f & \text{otherwise} \end{cases} \end{aligned}$$

Conversely, there is a projection map **proj** _{\mathbb{F}_e} : $\mathbb{R}_e \rightarrow \mathbb{F}_e$ that converts a real number to its rounded, floating-point representation, as follows. We let \mathbf{F}_{\min} be the smallest floating point number and \mathbf{F}_{\max} be the largest. The **proj** _{\mathbb{F}_e} map is required to satisfy the following properties:

- if $r \notin [\mathbf{F}_{\min}, \mathbf{F}_{\max}]$, then **proj** _{\mathbb{F}_e} (r) = **err**;
- if $r = \mathbf{inj}(f)$ then **proj** _{\mathbb{F}_e} (r) = f .

We shall also later require **proj** _{\mathbb{F}_e} to be measurable (see Proposition 1).

This can be achieved for example by the *round-to-nearest* function, defined by:

$$\begin{aligned} \mathbf{proj}_{\mathbb{F}_e} : \mathbb{R}_e &\rightarrow \mathbb{F}_e \\ r &\mapsto \mathbf{proj}_{\mathbb{F}_e}(r) = \begin{cases} \mathbf{err} & \text{if } r \notin [\mathbf{F}_{\min}, \mathbf{F}_{\max}] \\ \operatorname{argmin}\{|f - r|, f \in \mathbb{F}\} & \text{otherwise} \end{cases} \end{aligned}$$

When $\operatorname{argmin}\{|f - r|, f \in \mathbb{F}\}$ contains two elements, the IEEE 754 standard specifies even rounding, i.e., we take the value $f \in \mathbb{F}$ whose ulp (last bit of the mantissa) is 0.

4 Concrete Semantics of Expressions and Tests

We now construct two concrete semantics, the first one denoted by $\llbracket \cdot \rrbracket_r$ on real numbers, the second one denoted by $\llbracket \cdot \rrbracket_f$ on floating-point values. The construction of these semantics is based on the two maps **inj** and **proj** _{\mathbb{F}_e} defined above.

4.1 Concrete Semantics of Expressions

Every expression will be interpreted in an *environment* ρ , which serves to specify the values of variables. Simply, ρ is a map from the set \mathcal{V} of variables to \mathbb{R}_e (in the real number semantics) or to \mathbb{F}_e (in the floating-point semantics). We denote by Σ_f the set of floating-point environments, and by Σ_r the set of real number environments.

We start with the semantics in the real model. Let ρ_r be in Σ_r . The concrete semantics $\llbracket expr \rrbracket_r$ of expressions is constructed in the obvious way:

$$\begin{aligned} \llbracket a \rrbracket_r(\rho_r) &= a \\ \llbracket x \rrbracket_r(\rho_r) &= \rho_r(x) \\ \llbracket -e \rrbracket_r(\rho_r) &= -\llbracket e \rrbracket_r(\rho_r) \\ \llbracket e_1 + e_2 \rrbracket_r(\rho_r) &= \llbracket e_1 \rrbracket_r(\rho_r) + \llbracket e_2 \rrbracket_r(\rho_r) \\ \llbracket e_1 - e_2 \rrbracket_r(\rho_r) &= \llbracket e_1 \rrbracket_r(\rho_r) - \llbracket e_2 \rrbracket_r(\rho_r) \\ \llbracket e_1 \times e_2 \rrbracket_r(\rho_r) &= \llbracket e_1 \rrbracket_r(\rho_r) \times \llbracket e_2 \rrbracket_r(\rho_r) \\ \llbracket e_1 / e_2 \rrbracket_r(\rho_r) &= \llbracket e_1 \rrbracket_r(\rho_r) / \llbracket e_2 \rrbracket_r(\rho_r) \end{aligned}$$

The operations are well-defined by Convention 1.

Now let us define the floating-point semantics. Let ρ_f be in Σ_f . The floating-point semantics $\llbracket expr \rrbracket_f$ of expressions is defined by rounding at the evaluation of each subexpression:

$$\begin{aligned} \llbracket a \rrbracket_f(\rho_f) &= \mathbf{proj}_{\mathbb{F}_e}(a) \\ \llbracket x \rrbracket_f(\rho_f) &= \rho_f(x) \\ \llbracket -e \rrbracket_f(\rho_f) &= \mathbf{proj}_{\mathbb{F}_e}(-\mathbf{inj}(\llbracket e \rrbracket_f(\rho_f))) \\ \llbracket e_1 + e_2 \rrbracket_f(\rho_f) &= \mathbf{proj}_{\mathbb{F}_e}(\mathbf{inj}(\llbracket e_1 \rrbracket_f(\rho_f)) + \mathbf{inj}(\llbracket e_2 \rrbracket_f(\rho_f))) \\ \llbracket e_1 - e_2 \rrbracket_f(\rho_f) &= \mathbf{proj}_{\mathbb{F}_e}(\mathbf{inj}(\llbracket e_1 \rrbracket_f(\rho_f)) - \mathbf{inj}(\llbracket e_2 \rrbracket_f(\rho_f))) \\ \llbracket e_1 \times e_2 \rrbracket_f(\rho_f) &= \mathbf{proj}_{\mathbb{F}_e}(\mathbf{inj}(\llbracket e_1 \rrbracket_f(\rho_f)) \times \mathbf{inj}(\llbracket e_2 \rrbracket_f(\rho_f))) \\ \llbracket e_1 / e_2 \rrbracket_f(\rho_f) &= \mathbf{proj}_{\mathbb{F}_e}(\mathbf{inj}(\llbracket e_1 \rrbracket_f(\rho_f)) / \mathbf{inj}(\llbracket e_2 \rrbracket_f(\rho_f))) \end{aligned}$$

4.2 Concrete Semantics of Tests

The semantics of tests is a bit subtler. Although one cannot distinguish \mathbf{inf} , $-\mathbf{inf}$, \mathbf{NaN} using expressions only—this justified, at least partly, our decision to abstract them as a single value \mathbf{err} —one can distinguish them using tests. Experiments with a C compiler (gcc 4.2.1 here) indeed show the following behaviors:

a	b	$a==b$	$a!=b$	$a<=b$	$a<b$	$a>=b$	$a>b$
\mathbf{inf}	\mathbf{inf}	1	0	1	0	1	0
\mathbf{inf}	$-\mathbf{inf}$	0	1	0	0	1	1
\mathbf{NaN}	\mathbf{NaN}	0	1	0	0	0	0

Note for example that an \mathbf{NaN} is not considered equal to itself, that $a!=b$ is the negation of $a==b$ but $a>b$ is not the negation of $a<=b$ (e.g., when $a = b = \mathbf{NaN}$).

There are two ways we can deal with this phenomenon. Either we abandon the confusion of \mathbf{inf} , $-\mathbf{inf}$, \mathbf{NaN} as the single value \mathbf{err} , which will allow us to replay the above behavior precisely, but will incur many complications; or we consider that the semantics of tests must be *non-deterministic*: not knowing whether \mathbf{err} means \mathbf{inf} , $-\mathbf{inf}$, \mathbf{NaN} , we are forced to consider that $\mathbf{err}==\mathbf{err}$ is *any* value in $\{0, 1\}$.

So the semantics of tests will not be a single value, but a *set* of (Boolean, in $\{0, 1\}$) values. One may say that our concrete semantics is therefore slightly of an abstract semantics. We count on the fact that \mathbf{err} abstracts (so-called silent) errors, and should occur rarely in working programs. (We are not after detecting subtle errors, but to give reasonable accuracy bounds on actual working programs.)

On the other hand, we do not need to specify which semantics, floating-point or real, is meant: both will work in the same way for tests. Let us introduce the new notation $\llbracket \cdot \rrbracket_\star$, where \star is either

f (floating-point) or r (real). We denote by Σ_\star the set of environment in this context. Let ρ_\star be in Σ_\star .

$$\begin{aligned}
\llbracket e_1 \dot{=} e_2 \rrbracket_\star(\rho_\star) &= \begin{cases} \{1\} & \text{if } \llbracket e_1 \rrbracket_\star(\rho_\star) \neq \mathbf{err}, \llbracket e_2 \rrbracket_\star(\rho_\star) \neq \mathbf{err}, \text{ and } \llbracket e_1 \rrbracket_\star(\rho_\star) \leq \llbracket e_2 \rrbracket_\star(\rho_\star) \\ \{0\} & \text{if } \llbracket e_1 \rrbracket_\star(\rho_\star) \neq \mathbf{err}, \llbracket e_2 \rrbracket_\star(\rho_\star) \neq \mathbf{err}, \text{ and } \llbracket e_1 \rrbracket_\star(\rho_\star) > \llbracket e_2 \rrbracket_\star(\rho_\star) \\ \{0, 1\} & \text{if } \llbracket e_1 \rrbracket_\star(\rho_\star) = \mathbf{err} \text{ or } \llbracket e_2 \rrbracket_\star(\rho_\star) = \mathbf{err} \end{cases} \\
\llbracket e_1 \dot{<} e_2 \rrbracket_\star(\rho_\star) &= \begin{cases} \{1\} & \text{if } \llbracket e_1 \rrbracket_\star(\rho_\star) \neq \mathbf{err}, \llbracket e_2 \rrbracket_\star(\rho_\star) \neq \mathbf{err}, \text{ and } \llbracket e_1 \rrbracket_\star(\rho_\star) < \llbracket e_2 \rrbracket_\star(\rho_\star) \\ \{0\} & \text{if } \llbracket e_1 \rrbracket_\star(\rho_\star) \neq \mathbf{err}, \llbracket e_2 \rrbracket_\star(\rho_\star) \neq \mathbf{err}, \text{ and } \llbracket e_1 \rrbracket_\star(\rho_\star) \geq \llbracket e_2 \rrbracket_\star(\rho_\star) \\ \{0, 1\} & \text{if } \llbracket e_1 \rrbracket_\star(\rho_\star) = \mathbf{err} \text{ or } \llbracket e_2 \rrbracket_\star(\rho_\star) = \mathbf{err} \end{cases} \\
\llbracket e_1 \dot{=} e_2 \rrbracket_\star(\rho_\star) &= \begin{cases} \{1\} & \text{if } \llbracket e_1 \rrbracket_\star(\rho_\star) \neq \mathbf{err}, \llbracket e_2 \rrbracket_\star(\rho_\star) \neq \mathbf{err}, \text{ and } \llbracket e_1 \rrbracket_\star(\rho_\star) = \llbracket e_2 \rrbracket_\star(\rho_\star) \\ \{0\} & \text{if } \llbracket e_1 \rrbracket_\star(\rho_\star) \neq \mathbf{err}, \llbracket e_2 \rrbracket_\star(\rho_\star) \neq \mathbf{err}, \text{ and } \llbracket e_1 \rrbracket_\star(\rho_\star) \neq \llbracket e_2 \rrbracket_\star(\rho_\star) \\ \{0, 1\} & \text{if } \llbracket e_1 \rrbracket_\star(\rho_\star) = \mathbf{err} \text{ or } \llbracket e_2 \rrbracket_\star(\rho_\star) = \mathbf{err} \end{cases} \\
\llbracket e_1 \dot{=} e_2 \rrbracket_\star(\rho_\star) &= \begin{cases} \{1\} & \text{if } \llbracket e_1 \rrbracket_\star(\rho_\star) \neq \mathbf{err}, \llbracket e_2 \rrbracket_\star(\rho_\star) \neq \mathbf{err}, \text{ and } \llbracket e_1 \rrbracket_\star(\rho_\star) \neq \llbracket e_2 \rrbracket_\star(\rho_\star) \\ \{0\} & \text{if } \llbracket e_1 \rrbracket_\star(\rho_\star) \neq \mathbf{err}, \llbracket e_2 \rrbracket_\star(\rho_\star) \neq \mathbf{err}, \text{ and } \llbracket e_1 \rrbracket_\star(\rho_\star) = \llbracket e_2 \rrbracket_\star(\rho_\star) \\ \{0, 1\} & \text{if } \llbracket e_1 \rrbracket_\star(\rho_\star) = \mathbf{err} \text{ or } \llbracket e_2 \rrbracket_\star(\rho_\star) = \mathbf{err} \end{cases} \\
\llbracket !t \rrbracket_\star(\rho_\star) &= \{1 - v \mid v \in \llbracket t \rrbracket_\star(\rho_\star)\}
\end{aligned}$$

The symbols \leq , $<$, \geq , $>$ in the right-hand sides above are the usual relations on \mathbb{R} . So for example, the semantics of $e_1 \dot{=} e_2$ is well-defined because we only ever compare two elements of, i.e., two elements of \mathbb{R}_e other than \mathbf{err} .

4.3 Measurability of Concrete Semantics of Expressions and Tests

In the definition of the semantics, we will work with Lebesgue integrals, or notions that generalize the Lebesgue integral. It is well-known that one cannot posit that every function is integrable without causing inconsistencies, and we shall therefore have to check that every function that we integrate is *measurable*.

Measurability concerns are (mostly) irrelevant in the floating-point semantics, if we remember that \mathbb{F} , hence \mathbb{F}_e , is finite, and that every function between finite spaces is measurable. But they are definitely important in the real number semantics.

Measurability is defined relatively to specific σ -algebras. The Borel σ -algebra on \mathbb{R} —or, more generally, on any topological space—is the smallest σ -algebra that contains all open subsets.

We extend the topology of \mathbb{R} to one on \mathbb{R}_e by extending the standard metric on \mathbb{R} to the following:

$$d(x, y) = \begin{cases} +\infty & \text{if } x = \mathbf{err} \text{ or } y = \mathbf{err} \text{ and } x \neq y \\ 0 & \text{if } x = y = \mathbf{err} \\ |x - y| & \text{if } x, y \in \mathbb{R} \end{cases}$$

The resulting topology has, as opens, all open subsets of \mathbb{R} , the singleton $\{\mathbf{err}\}$, and their unions. This makes \mathbf{err} an (the unique) isolated point of \mathbb{R}_e . Note that this topology is not the topology of the classical one-point (Alexandroff) compactification of \mathbb{R} , in which a basis of open neighborhoods of \mathbf{err} would be given by the sets $(-\infty, a) \cup (b, +\infty) \cup \{\mathbf{err}\}$, and \mathbf{err} would not be isolated. The latter would also be a possible choice, but would induce additional, irrelevant complications.

The subspace \mathbb{F}_e has the subspace topology: this is just the discrete topology, since \mathbb{F}_e is finite.

We equip Σ_r with the smallest topology that makes each map $\rho \mapsto \rho(x)$ continuous, for each $x \in \mathcal{V}$. This makes Σ_r isomorphic to $\mathbb{R}_e^\mathcal{V}$ with the product topology.

Similarly, we equip Σ_f with the subspace topology from Σ_r . This is also the product topology on $\mathbb{F}_e^\mathcal{V}$, up to isomorphism. Note that this is not the discrete topology as soon as \mathcal{V} is infinite: indeed, $\mathbb{F}_e^\mathcal{V}$ is compact and infinite in this case, but all compact discrete topological spaces are finite. This argument is however uselessly subtle: programs only use finitely many variables anyway, and for \mathcal{V} finite, Σ_f has the discrete topology.

We write $\mathcal{B}(\Sigma_r)$ and $\mathcal{B}(\Sigma_f)$ for the σ -algebras of Borel subsets of Σ_r and Σ_f respectively. By standard results in topological measure theory (and crucially using the fact that \mathcal{V} is countable), these are also the product σ -algebras on the (measure-theoretic) product of \mathcal{V} copies of \mathbb{R}_e , resp. \mathbb{F}_e . (This is because $\mathbb{R}_e, \mathbb{F}_e$ are Polish spaces, and the Borel σ -algebra on a countable topological

product of Polish spaces coincides with the σ -algebra of the measure-theoretic product of the spaces, each with their Borel σ -algebra.) This is a reassuring statement: it states that we can harmlessly say “product” without having to say whether this is a topological or measure-theoretic product. There is no such trap here.

A measurable map $f : X \rightarrow Y$ is one such that $f^{-1}(E)$ is a Borel subset for every Borel subset E ; it is equivalent to require that, for every open subset U , $f^{-1}(U)$ is Borel. In particular, every continuous map is measurable. When Y is second-countable, i.e., has certain so-called *basic* opens such that every open subset is the union of countably many basic opens, then f is measurable iff $f^{-1}(U)$ is Borel for every basic open U . We shall use this in proofs; in particular when $Y = \mathbb{R}_e$, where we can take the intervals with rational endpoints, and $\{\mathbf{err}\}$, as basic opens.

One might think that expressions have continuous real semantics, but this is wrong: x/y as a function of $x, y \in \mathbb{R}_e$ is not continuous at any point of the form $(x, 0)$. But they are measurable. This would be repaired if we had taken the topology of the 1-point compactification of \mathbb{R} on \mathbb{R}_e , but we only need measurability. On the other hand, we really need the topological and measure-theoretic products to coincide, and while this would also be true with the 1-point compactification, the argument would be slightly more complex.

Proposition 1 (Expressions are Measurable)

- For every expression e , $\rho \mapsto \llbracket e \rrbracket_r(\rho)$ is a measurable function from Σ_r to \mathbb{R}_e .
- if \mathcal{V} is finite or $\mathbf{proj}_{\mathbb{F}_e}$ is measurable, then for every expression e , $\rho \mapsto \llbracket e \rrbracket_f(\rho)$ is a measurable function from Σ_f to \mathbb{F}_e .

Proof We proceed by induction on expressions. Let $a \in \mathbb{Q}$. The function $\rho \mapsto \llbracket a \rrbracket_r(\rho)$ is a constant function and thus it is continuous, hence measurable. Let $x \in \mathcal{V}$. The function $\rho \mapsto \llbracket x \rrbracket_r(\rho)$ is the coordinate projection on the x coordinate of ρ and thus it is continuous, hence measurable. The case of expressions of the form $\dot{-}e$, $e_1 \dot{+} e_2$, $e_1 \dot{-} e_2$, $e_1 \dot{\times} e_2$ follows by induction hypothesis, using the fact that the corresponding operations on \mathbb{R}_e are continuous. To show this, it suffices to show that the inverse image of every basic open subset (i.e., open intervals of \mathbb{R} , and $\{\mathbf{err}\}$) is open in Σ_r . For example, the inverse image of an open subset of \mathbb{R} by $+$ is an open subset of $\mathbb{R} \times \mathbb{R}$, hence of $\mathbb{R}_e \times \mathbb{R}_e$, and the inverse image of the basic open subset $\{\mathbf{err}\}$ is $(\mathbb{R}_e \times \{\mathbf{err}\}) \cup (\{\mathbf{err}\} \times \mathbb{R}_e)$, hence open. The case of e_1/e_2 is slightly different as $/$ is not continuous on $\mathbb{R}_e \times \mathbb{R}_e$. But it is measurable, as we now show, by showing that the inverse image of any basic open subset is Borel. The inverse image of any open interval of \mathbb{R} is open, since division is continuous at every point (x, y) with $y \neq 0$. And the inverse image of $\{\mathbf{err}\}$ by $/$ is the union of $\mathbb{R}_e \times \{\mathbf{err}\}$, of $\{\ell\} \times \mathbb{R}_e$, and of $\mathbb{R}_e \times \{0\}$. The first two are open hence Borel, while the last one is the countable intersection $\bigcap_{n \geq 1} (\mathbb{R}_e \times (-\frac{1}{n}, \frac{1}{n}))$, hence is Borel.

The second assertion is trivial if \mathcal{V} is finite, in which case all involved σ -algebras are discrete. In the general case, it suffices to observe that \mathbf{inj} and $\mathbf{proj}_{\mathbb{F}_e}$ are measurable: \mathbf{inj} is even continuous, since any function from a discrete space is, and the fact that $\mathbf{proj}_{\mathbb{F}_e}$ is measurable is our assumption. Using the fact that the composition of measurable functions is measurable, and using a similar induction as above, we conclude. \square

All natural rounding functions $\mathbf{proj}_{\mathbb{F}_e}$ are measurable, so the assumptions we are making in Proposition 1 will be satisfied. E.g.,

Lemma 1

The round-to-nearest map, with even rounding, is measurable from \mathbb{R}_e to \mathbb{F}_e .

Proof Since the Borel σ -algebra on \mathbb{F}_e is discrete, it is enough to check that the inverse image of any single element $f \in \mathbb{F}_e$ is Borel.

If $f \in (\mathbf{F}_{\min}, \mathbf{F}_{\max}) \cap \mathbb{F}$, and if the ulp of f is 0, then this inverse image is $[\frac{f+f'}{2}, \frac{f+f''}{2}]$ ($(\frac{f+f'}{2}, \frac{f+f''}{2})$ if the ulp of f is not 0), where f' is the largest element of \mathbb{F} strictly less than f and f'' is the smallest element of \mathbb{F} strictly larger than f .

If $f = \mathbf{F}_{\min}$, then the inverse image of f is $[\mathbf{F}_{\min}, \frac{f + f''}{2}]$ (if the ulp of f is 0; $[\mathbf{F}_{\min}, \frac{f + f''}{2})$ if the ulp of f is not 0), where f'' is the smallest element of \mathbb{F} strictly larger than f .

If $f = \mathbf{F}_{\max}$, then the inverse image of f is $[\frac{f + f'}{2}, \mathbf{F}_{\max}]$ (if the ulp of f is 0; $(\frac{f + f'}{2}, \mathbf{F}_{\max}]$ if the ulp of f is not 0), where f' is the largest element of \mathbb{F} strictly less than f .

Finally, the inverse image of \mathbf{err} is the union of $\{\mathbf{err}\}$, of $(-\infty, \mathbf{F}_{\min})$, and of $(\mathbf{F}_{\max}, +\infty)$.

All these sets are either open, or closed, and in any case Borel. \square

Tests are interpreted as maps from Σ_\star to $\mathbb{P}^*\{0, 1\}$, where \mathbb{P}^* denotes non-empty powerset, and are thus multifunctions. One of the standard notions of measurability for multifunctions is to say that, given topological spaces X and Y , $f : X \rightarrow \mathbb{P}^*(Y)$ is measurable if and only if $f^{-1}(\diamond U)$ is Borel for every *open* subset U of Y . ($\diamond U$ is the set of subsets that intersect U .) If we understand f as a relation between elements of X and elements of Y , this means that the elements $x \in X$ that are related to some element of a given open subset U should be Borel.

Proposition 2 (Tests are Measurable)

For every test t , $\rho \mapsto \llbracket t \rrbracket_r(\rho)$ is a measurable function from Σ_r to $\mathbb{P}^*\{0, 1\}$. If \mathcal{V} is finite or $\mathbf{proj}_{\mathbb{F}_e}$ is measurable, then $\rho \mapsto \llbracket t \rrbracket_f(\rho)$ is a measurable function from Σ_f to $\mathbb{P}^*\{0, 1\}$.

Proof It suffices to show that the inverse image of $\diamond\{0\}$ and of $\diamond\{1\}$ are Borel. We proceed by induction on t . Let \star be either f or r .

If t is of the form $e_1 \dot{<} e_2$, then $\llbracket t \rrbracket_\star(\rho_\star)$ contains 0 if and only if $\llbracket e_1 \dot{-} e_2 \rrbracket_\star(\rho_\star)$ is in $\{\mathbf{err}\} \cup (0, +\infty)$ (if $\star = r$; in $\mathbf{inj}^{-1}(\{\mathbf{err}\} \cup (0, +\infty))$ if $\star = f$). The latter is open, and $\llbracket e_1 \dot{-} e_2 \rrbracket_\star$ is measurable by Proposition 1, so $\llbracket t \rrbracket_\star^{-1}(\diamond\{0\})$ is Borel. Similarly, $\llbracket t \rrbracket_\star(\rho)$ contains 1 if and only if $\llbracket e_1 \dot{-} e_2 \rrbracket_\star(\rho)$ is in $\{\mathbf{err}\} \cup (-\infty, 0]$ (if $\star = r$; its inverse image by \mathbf{inj} if $\star = f$), which is closed, so $\llbracket t \rrbracket_\star^{-1}(\diamond\{1\})$ is Borel. We proceed similarly if t is of the form $e_1 \dot{<} e_2$, $e_1 \dot{=} e_2$, or $e_1 \dot{=} e_2$.

Finally, if t is of the form $!t'$, $\llbracket t \rrbracket_\star^{-1}(\diamond\{0\}) = \llbracket t' \rrbracket_\star^{-1}(\diamond\{1\})$, and $\llbracket t \rrbracket_\star^{-1}(\diamond\{1\}) = \llbracket t' \rrbracket_\star^{-1}(\diamond\{0\})$, which allows us to conclude immediately. \square

5 Weakest Preconditions and Continuation-Passing Style Semantics

The idea of a continuation-passing style (CPS) semantics is that the value v returned by a given program is not given explicitly. Rather, one passes a continuation parameter κ to the semantics, and the latter is defined so that it eventually calls κ on the final value v .

While this seems like a complicated and roundabout way of defining semantics, this is very useful. For example, this allows one to give semantics to exceptions, or to various forms of non-determinism and probabilistic choice [Gou07].

The continuation κ itself is a map from the domain of values to some, usually unspecified domain of *answers* \mathbf{Ans} . (In [Gou07], \mathbf{Ans} was required to be \mathbb{R}^+ .)

Also, the “final value” of a program should here be understood as the final environment ρ_\star that represents the state the program is in on termination. So a continuation κ will be a map from Σ_\star to \mathbf{Ans} .

It should also be noted that continuation-passing style semantics are nothing else than a natural generalization of Dijkstra’s *weakest preconditions*, or the computation of sets of predecessor states in transition systems. This is obtained by taking $\mathbf{Ans} = \{0, 1\}$. Then the continuations κ are merely the indicator maps of subsets E of environments (predicates P on environments), and the continuation-passing style denotation of program π in continuation κ is merely the (continuation representing) the set of environments ρ such that evaluating π starting from ρ may terminate with an environment in E (satisfying P).

Recall that an ω -cpo is a poset in which every ascending sequence $x_0 \leq x_1 \leq \dots \leq x_n \leq \dots$ has a supremum (a least upper bound).

Assumption 1

We assume that \mathbf{Ans} is an ω -cpo with a smallest element $\perp_{\mathbf{Ans}}$, and binary suprema.

We write \sup for suprema, and reserve \sup^\uparrow for suprema of ascending sequences. Assumption 1 can be stated equivalently as: \mathbf{Ans} has all countable suprema (including the supremum $\perp_{\mathbf{Ans}}$ of the empty family). If the language had been deterministic (we fall short of this because of the way \mathbf{err} is dealt with in tests), we would only need \mathbf{Ans} to be an ω -cpo, and would not have a need to binary suprema.

The typical example of such a set \mathbf{Ans} of answers is $\mathbb{R}_+ \cup \{+\infty\}$, with its usual ordering. As usual, we define the semantics of instructions by recursion on syntax:

- skip:

$$\text{wp}[\ell^1 \text{skip}, \ell_2]_\star(\kappa) = \kappa$$

- assignment:

$$\text{wp}[\ell^1 x := e, \ell_2]_\star(\kappa) = \text{fun } \rho \mapsto \kappa(\rho[x \rightarrow \llbracket e \rrbracket_\star(\rho)])$$

- sequence:

$$\text{wp}[\ell^1 P; \ell^2 Q, \ell_3]_\star(\kappa) = \text{wp}[\ell^1 P, \ell_2]_\star \left(\text{wp}[\ell^2 Q, \ell_3]_\star(\kappa) \right)$$

- tests:

$$\begin{aligned} & \text{wp}[\ell \text{if } t \text{ then } \ell^1 P_1 \text{ else } \ell^0 P_0, \ell_2]_\star(\kappa) = \\ & \text{fun } \rho \mapsto \sup_{i \in \llbracket t \rrbracket_\star(\rho)} \left(\text{wp}[\ell^i P_i, \ell_2]_\star(\kappa) \right) (\rho) \end{aligned}$$

In other words,

$$\begin{aligned} & \text{wp}[\ell \text{if } t \text{ then } \ell^1 P_1 \text{ else } \ell^0 P_0, \ell_2]_\star(\kappa) = \\ & \text{fun } \rho \mapsto \begin{cases} \left(\text{wp}[\ell^1 P_1, \ell_2]_\star(\kappa) \right) (\rho) & \text{if } \llbracket t \rrbracket_\star(\rho) = \{1\} \\ \left(\text{wp}[\ell^0 P_0, \ell_2]_\star(\kappa) \right) (\rho) & \text{if } \llbracket t \rrbracket_\star(\rho) = \{0\} \\ \sup \left(\left(\text{wp}[\ell^1 P_1, \ell_2]_\star(\kappa) \right) (\rho), \left(\text{wp}[\ell^0 P_0, \ell_2]_\star(\kappa) \right) (\rho) \right) & \text{if } \llbracket t \rrbracket_\star(\rho) = \{0, 1\} \end{cases} \end{aligned}$$

The definition of the semantics of a loop, of the form $\ell^1 \text{while } t \ell^2 P$ uses an auxiliary map. We denote by $\mathbf{F}(\Sigma_\star, \mathbf{Ans})$ the set $((\Sigma_\star \rightarrow \mathbf{Ans}) \rightarrow (\Sigma_\star \rightarrow \mathbf{Ans}))$ i.e. the set of maps from $(\Sigma_\star \rightarrow \mathbf{Ans})$ to itself. We equip $(\Sigma_\star \rightarrow \mathbf{Ans})$ with the pointwise ordering. The set $\mathbf{F}(\Sigma_\star, \mathbf{Ans})$ is also equipped with the pointwise ordering: $f \leq g$ iff for every $\kappa \in (\Sigma_\star \rightarrow \mathbf{Ans})$, for every $\rho \in \Sigma_\star$, $f(\kappa)(\rho) \leq g(\kappa)(\rho)$ in \mathbf{Ans} . For every countable family $(f_i)_{i \in I}$ of elements of $\mathbf{F}(\Sigma_\star, \mathbf{Ans})$, its supremum $\sup_{i \in I} f_i$ is then also computed pointwise:

$$\sup_{i \in I} f_i : \kappa \mapsto \left(\text{fun } \rho \mapsto \sup_{i \in I} (f_i(\kappa)(\rho)) \right) .$$

From this latter definition, we get the following lemma.

Lemma 2

The set $\mathbf{F}(\Sigma_\star, \mathbf{Ans})$ is a ω -cpo with binary suprema, and with a smallest element $\perp_{\mathbf{F}(\Sigma_\star, \mathbf{Ans})}$ defined as:

$$\perp_{\mathbf{F}(\Sigma_\star, \mathbf{Ans})}(\kappa) = \text{fun } \rho \mapsto \perp_{\mathbf{Ans}}, \forall \kappa : \Sigma_\star \mapsto \mathbf{Ans} .$$

- loops. Given a test t and an instruction ${}^{\ell_2}P, \ell_1$, let $H_{t, \ell_2 P, \ell_1}$ be the map from $\mathbf{F}(\Sigma_*, \mathbf{Ans})$ to $\mathbf{F}(\Sigma_*, \mathbf{Ans})$ defined as follows:

$$\left(H_{t, \ell_2 P, \ell_1}(\varphi) \right) (\kappa)(\rho) = \begin{cases} \left(\text{wp}[\ell_2 P, \ell_1]_\star(\varphi(\kappa)) \right) (\rho) & \text{if } \llbracket t \rrbracket_\star(\rho) = \{1\} \\ \kappa(\rho) & \text{if } \llbracket t \rrbracket_\star(\rho) = \{0\} \\ \sup(\left(\text{wp}[\ell_2 P, \ell_1]_\star(\varphi(\kappa)) \right) (\rho), \kappa(\rho)) & \text{if } \llbracket t \rrbracket_\star(\rho) = \{0, 1\} \end{cases}$$

So, for example, $\text{wp}[\text{if } t \text{ then } {}^{\ell_1}P_1 \text{ else } {}^{\ell_0}P_0, \ell_2]_\star = H_{t, \ell_1 P_1, \ell_2}(\text{wp}[\ell_0 P_0, \ell_2]_\star)$.

The semantics of the loop ${}^{\ell_1}\text{while } t \text{ } {}^{\ell_2}P, \ell_3$ is the supremum of the sequence $\perp_{\mathbf{F}(\Sigma_*, \mathbf{Ans})}$, $H_{t, \ell_2 P, \ell_3}(\perp_{\mathbf{F}(\Sigma_*, \mathbf{Ans})})$, $H_{t, \ell_2 P, \ell_3}(H_{t, \ell_2 P, \ell_3}(\perp_{\mathbf{F}(\Sigma_*, \mathbf{Ans})}))$, \dots in $\mathbf{F}(\Sigma_*, \mathbf{Ans})$, namely:

$$\text{wp}[\ell_1 \text{ while } t \text{ } {}^{\ell_2}P, \ell_3]_\star = \sup_{n \in \mathbb{N}} H_{t, \ell_2 P, \ell_3}^n(\perp_{\mathbf{F}(\Sigma_*, \mathbf{Ans})})$$

A more standard definition would have been to let $\text{wp}[\ell_1 \text{ while } t \text{ } {}^{\ell_2}P, \ell_3]_\star$ be defined as the least fixpoint of $H_{t, \ell_2 P, \ell_3}$ in $\mathbf{F}(\Sigma_*, \mathbf{Ans})$. We show below that this would be equivalent. The reason is that the map $H_{t, \ell_2 P, \ell_3}$ is ω -Scott-continuous, i.e., is monotone and preserves suprema of ascending sequences.

We prove this through two lemmas. The first one shows that $H_{t, \ell_2 P, \ell_1}$ is ω -Scott-continuous when the maps $\kappa \mapsto \text{wp}[\ell_2 P, \ell_1]_\star(\kappa)$ are ω -Scott-continuous. This second lemma says that the maps $\kappa \mapsto \text{wp}[\ell_2 P, \ell_1]_\star(\kappa)$ are actually ω -Scott-continuous.

Lemma 3

Let ${}^{\ell_2}P$ be an instruction. Let t be a test. Assume that the map

$$\kappa \mapsto \text{wp}[\ell_2 P, \ell_1]_\star(\kappa) \text{ is } \omega\text{-Scott-continuous} \quad , \quad (1)$$

then:

- The map $H_{t, \ell_2 P, \ell_1}$ is ω -Scott-continuous.
- The map $\sup_{n \in \mathbb{N}} H_{t, \ell_2 P, \ell_1}^n$ is ω -Scott-continuous.
- $\kappa \mapsto \text{wp}[\ell_1 \text{ while } t \text{ } {}^{\ell_2}P, \ell_3]_\star(\kappa)$ is ω -Scott-continuous.

Proof Let us prove the first assertion. Let $\varphi, \varphi' \in \mathbf{F}(\Sigma_*, \mathbf{Ans})$ such that $\varphi \leq \varphi'$. For every $\kappa : \Sigma_* \mapsto \mathbf{Ans}$, for every ρ such that $\llbracket t \rrbracket_\star(\rho) = \{1\}$,

$$\text{wp}[\ell_2 P, \ell_1]_\star(\varphi(\kappa))(\rho) \leq \text{wp}[\ell_2 P, \ell_1]_\star(\varphi'(\kappa))(\rho) \quad ,$$

so:

$$H_{t, \ell_2 P, \ell_1}(\varphi)(\kappa)(\rho) \leq H_{t, \ell_2 P, \ell_1}(\varphi')(\kappa)(\rho) \quad .$$

Let $(\varphi_n)_{n \in \mathbb{N}}$ be an ascending sequence in $\mathbf{F}(\Sigma_*, \mathbf{Ans})$. We also have:

$$\left(\text{wp}[\ell_2 P, \ell_1]_\star \left(\sup_{n \in \mathbb{N}}^\uparrow \varphi_n(\kappa) \right) \right) (\rho) = \sup_{n \in \mathbb{N}}^\uparrow \left(\text{wp}[\ell_2 P, \ell_1]_\star(\varphi_n(\kappa)) \right) (\rho) \quad .$$

When $\llbracket t \rrbracket_\star(\rho) = \{1\}$, this is equivalent to:

$$H_{t, \ell_2 P, \ell_1} \left(\sup_{n \in \mathbb{N}}^\uparrow \varphi_n \right) (\kappa)(\rho) = \left(\sup_{n \in \mathbb{N}}^\uparrow H_{t, \ell_2 P, \ell_1}(\varphi_n) \right) (\kappa)(\rho) \quad .$$

When $\llbracket t \rrbracket_\star(\rho) = \{0\}$, we obtain the same equality, where now both sides are the constant $\kappa(\rho)$.
When $\llbracket t \rrbracket_\star(\rho) = \{0, 1\}$,

$$\begin{aligned} H_{t, \ell_2 P, \ell_1} \left(\sup_{n \in \mathbb{N}}^\uparrow \varphi_n \right) (\kappa)(\rho) &= \sup \left(\sup_{n \in \mathbb{N}}^\uparrow \left(\text{wp} \llbracket \ell_2 P, \ell_1 \rrbracket_\star (\varphi_n(\kappa)) \right) (\rho), \kappa(\rho) \right) \\ &= \sup_{n \in \mathbb{N}}^\uparrow \left(\sup \left(\text{wp} \llbracket \ell_2 P, \ell_1 \rrbracket_\star (\varphi_n(\kappa)) (\rho), \kappa(\rho) \right) \right) \\ &= \left(\sup_{n \in \mathbb{N}}^\uparrow H_{t, \ell_2 P, \ell_1} (\varphi_n) \right) (\kappa)(\rho) . \end{aligned}$$

The second assertion follows from the first, from the fact that compositions of ω -Scott-continuous maps are again ω -Scott-continuous (hence $H_{t, \ell_2 P, \ell_1}^n$ is ω -Scott-continuous for every $n \in \mathbb{N}$), and that suprema of ω -Scott-continuous are ω -Scott-continuous.

The last assertion follows trivially from the second one, using the fact that application (of maps to $\perp_{\mathbf{F}(\Sigma_\star, \mathbf{Ans})}$) is ω -Scott-continuous. \square

Next, we prove that for every instruction ${}^\ell P$, for every label ℓ' , the map $\kappa \mapsto \text{wp} \llbracket \ell P, \ell' \rrbracket_\star(\kappa)$ is ω -Scott-continuous.

Lemma 4 (ω -Scott-Continuity of $\text{wp} \llbracket \cdot \rrbracket_\star$)

For every instruction ${}^\ell P$, for every label ℓ' , the map $\kappa \mapsto \text{wp} \llbracket \ell P, \ell' \rrbracket_\star(\kappa)$ is ω -Scott-continuous.

Proof We proceed by induction on the instructions.

- skip. The instruction skip is the identity map from $\Sigma_\star \rightarrow \mathbf{Ans}$ to itself, so it is ω -Scott-continuous.
- Assignment. Let κ, κ' be maps from Σ_\star to \mathbf{Ans} such that $\kappa \leq \kappa'$. For every $\rho \in \Sigma_\star$, $\kappa(\rho[x \rightarrow \llbracket e \rrbracket_\star(\rho)]) \leq \kappa'(\rho[x \rightarrow \llbracket e \rrbracket_\star(\rho)])$. So $\text{wp} \llbracket \ell_1 x := e, \ell_2 \rrbracket_\star$ is a monotonic map. Now, we consider an ascending sequence $(\kappa_n)_{n \in \mathbb{N}}$ of maps from Σ_\star to \mathbf{Ans} . We have:

$$\left(\sup_{n \in \mathbb{N}}^\uparrow (\kappa_n) \right) (\rho[x \rightarrow \llbracket e \rrbracket_\star(\rho)]) = \sup_{n \in \mathbb{N}}^\uparrow (\kappa_n(\rho[x \rightarrow \llbracket e \rrbracket_\star(\rho)]))$$

and then:

$$\text{wp} \llbracket \ell_1 x := e, \ell_2 \rrbracket_\star \left(\sup_{n \in \mathbb{N}}^\uparrow \kappa_n \right) = \sup_{n \in \mathbb{N}}^\uparrow \text{wp} \llbracket \ell_1 x := e, \ell_2 \rrbracket_\star(\kappa_n) .$$

- Sequence. By induction hypothesis on ${}^{\ell_1} P$ and ${}^{\ell_2} Q$, the maps $\kappa \rightarrow \text{wp} \llbracket \ell_1 P, \ell_2 \rrbracket_\star(\kappa)$ and $\kappa \rightarrow \text{wp} \llbracket \ell_2 Q, \ell_3 \rrbracket_\star(\kappa)$ are ω -Scott-continuous. Since the composition of two ω -Scott-continuous maps is ω -Scott-continuous then the sequence $\kappa \rightarrow \text{wp} \llbracket \ell_1 P; \ell_2 Q, \ell_3 \rrbracket_\star(\kappa)$ is also ω -Scott-continuous.

- Tests. By induction hypothesis on ${}^{\ell_2} P$ and ${}^{\ell_3} Q$, the maps $\kappa \rightarrow \text{wp} \llbracket \ell_2 P, \ell_4 \rrbracket_\star(\kappa)$ and $\kappa \rightarrow \text{wp} \llbracket \ell_3 Q, \ell_4 \rrbracket_\star(\kappa)$ are ω -Scott-continuous. Since we consider a pointwise order, it suffices to show that for every $\rho \in \Sigma_\star$, $\kappa \mapsto \text{wp} \llbracket \ell_1 \text{if } t \text{ then } \ell_2 P \text{ else } \ell_3 Q, \ell_4 \rrbracket_\star(\kappa)(\rho)$ is ω -Scott-continuous (from $\Sigma_\star \mapsto \mathbf{Ans}$ to \mathbf{Ans}). When we fix $\rho \in \Sigma_\star$, we get three cases whether the test is true, false or true and false. In each case, we conclude that $\text{wp} \llbracket \ell_1 \text{if } t \text{ then } \ell_2 P \text{ else } \ell_3 Q, \ell_4 \rrbracket_\star(\kappa)$ is ω -Scott-continuous by induction hypothesis.

- Loops. By induction hypothesis, $\kappa \rightarrow \text{wp} \llbracket \ell_2 P, \ell_1 \rrbracket_\star(\kappa)$ is ω -Scott-continuous. Lemma 3 immediately entails that $\kappa \mapsto \text{wp} \llbracket \ell_1 \text{while } t \ell_2 P, \ell_3 \rrbracket_\star(\kappa)$ is ω -Scott-continuous. \square

We introduce a parametric version of classical previsions. Since we work with ω -cpos, we have to consider $[0, +\infty]$, we add arithmetics conventions to deal with $+\infty$.

Convention 2 (Arithmetics in $\mathbb{R}_+ \cup \{+\infty\}$)

We add the following rules:

- $0 \times (+\infty) = (+\infty) \times 0 = 0$;
- $+\infty \times +\infty = +\infty$;

- For all $x \in [0, +\infty]$, $x + (+\infty) = (+\infty) + x = +\infty$.

Let X be a topological space. We equip X with its Borel σ -algebra. We denote by $\mathcal{M}_+(X)$ the set of positive measurable functions on X .

Definition 1 (Parametric prevision)

Let X be a non-empty set. Let F be a map from $\mathcal{M}_+(X)$ to itself. The map F is said to be a parametric prevision if:

1. F is positively homogeneous;
2. F is monotonic;

Moreover, a parametric prevision is said to be:

1. (lower) $F(f + g) \geq F(f) + F(g)$, for all functions $f, g \in \mathcal{M}_+(X)$;
2. (upper) $F(f + g) \leq F(f) + F(g)$, for all functions $f, g \in \mathcal{M}_+(X)$;
3. (linear) $F(f + g) = F(f) + F(g)$, for all functions f, g from $\mathcal{M}_+(X)$;
4. (ω -continuous) for all ascending family $(f_n)_{n \in \mathbb{N}}$, $F(\sup_{n \in \mathbb{N}}^{\uparrow} f_n) = \sup_{n \in \mathbb{N}}^{\uparrow} F(f_n)$.

We recall that the set of positive measurable functions is a convex cone stable by countable infima and suprema and pointwise limit. The set of positive measurable functions contains constant (positive), and continuous functions.

Proposition 3

1. The set of upper ω -continuous parametric prevision is ω -cpo (equipped with the pointwise ordering) with a smallest element (the null pfunctional $\underline{0}$ associates at $f \in \mathcal{M}_+(X)$ the positive measurable function $g : x \rightarrow 0$)
2. The set of upper ω -continuous parametric prevision is stable by binary suprema.
3. The set of upper ω -continuous parametric prevision is stable by composition.

Proof 1. The parametric $\underline{0}$ is clearly an upper ω -continuous parametric prevision. Since the null parametric is the smallest element of $\mathbf{F}(X, [0, +\infty])$, it is also the smallest element of the set of upper ω -continuous parametric previsions.

Let $(F_n)_{n \in \mathbb{N}}$ be an ascending sequence. Since $\mathcal{M}_+(X)$ is stable by countable suprema, then $(\sup_{n \in \mathbb{N}}^{\uparrow} F_n)(f) = \sup_{n \in \mathbb{N}}^{\uparrow} (F_n(f)) \in \mathcal{M}_+(X)$ for all $f \in \mathcal{M}_+(X)$.

Let $\alpha \geq 0$ and $f \in \mathcal{M}_+(X)$. The set $\mathcal{M}_+(X)$ is a cone thus $\alpha f \in \mathcal{M}_+(X)$. Since F_n are positively homogeneous then $(\sup_{n \in \mathbb{N}}^{\uparrow} F_n)(\alpha f) = \sup_{n \in \mathbb{N}}^{\uparrow} (F_n(\alpha f)) = \sup_{n \in \mathbb{N}}^{\uparrow} \alpha F_n(f)$ and since $\alpha \geq 0$, we conclude that $\sup_{n \in \mathbb{N}}^{\uparrow} (F_n(\alpha f)) = \alpha \sup_{n \in \mathbb{N}}^{\uparrow} F_n(f)$ and $\sup_{n \in \mathbb{N}}^{\uparrow} F_n$ is positively homogeneous.

Let $f, g \in \mathcal{M}_+(X)$ such that $f \leq g$, $(\sup_{n \in \mathbb{N}}^{\uparrow} F_n)(f) = \sup_{n \in \mathbb{N}}^{\uparrow} F_n(f)$. For all $n \in \mathbb{N}$, $F_n(f) \leq F_n(g)$ and we get $\sup_{n \in \mathbb{N}}^{\uparrow} F_n(f) \leq \sup_{n \in \mathbb{N}}^{\uparrow} F_n(g) = (\sup_{n \in \mathbb{N}}^{\uparrow} F_n)(g)$ and we conclude that $\sup_{n \in \mathbb{N}}^{\uparrow} F_n$ is monotonic.

Let $f, g \in \mathcal{M}_+(X)$. For all $n \in \mathbb{N}$, we have $F_n(f + g) \leq F_n(f) + F_n(g)$, taking the suprema we get $\sup_{n \in \mathbb{N}}^{\uparrow} F_n(f + g) \leq \sup_{n \in \mathbb{N}}^{\uparrow} (F_n(f) + F_n(g)) \leq \sup_{n \in \mathbb{N}}^{\uparrow} F_n(f) + \sup_{n \in \mathbb{N}}^{\uparrow} F_n(g)$ and $\sup_{n \in \mathbb{N}}^{\uparrow} F_n$ is an upper parametric prevision.

Now let $(f_k)_{k \in \mathbb{N}}$ be an ascending sequence of elements of $\mathcal{M}_+(X)$. The set $\mathcal{M}_+(X)$ is stable by suprema hence $\sup_{k \in \mathbb{N}}^{\uparrow} f_k \in \mathcal{M}_+(X)$. We have $\sup_{n \in \mathbb{N}}^{\uparrow} F_n(\sup_{k \in \mathbb{N}}^{\uparrow} f_k) = \sup_{n \in \mathbb{N}}^{\uparrow} \sup_{k \in \mathbb{N}}^{\uparrow} F_n(f_k)$ and the suprema commute and then $\sup_{n \in \mathbb{N}}^{\uparrow} F_n(\sup_{k \in \mathbb{N}}^{\uparrow} f_k) = \sup_{k \in \mathbb{N}}^{\uparrow} \sup_{n \in \mathbb{N}}^{\uparrow} F_n(f_k)$. We conclude that $\sup_{n \in \mathbb{N}}^{\uparrow} F_n$ is ω -continuous.

2. Let F, G be two upper ω -continuous parametric prevision. From the supremum stability property, $\sup(F, G)(f) = \sup(F(f), G(f))$ belongs to $\mathcal{M}_+(X)$ for all $f \in \mathcal{M}_+(X)$.

Let $\alpha \geq 0$ and $f \in \mathcal{M}_+(X)$ ($\alpha f \in \mathcal{M}_+(X)$), since F, G are positively homogeneous then $\sup(F, G)(\alpha f) = \sup(F(\alpha f), G(\alpha f)) = \sup(\alpha F(f), \alpha G(f))$ and since $\alpha \geq 0$, we conclude that $\sup(F, G)(\alpha f) = \alpha \sup(F(f), G(f)) = \alpha \sup(F, G)(f)$ and $\sup(F, G)$ is positively homogeneous.

The supremum of monotonic function is a monotonic function hence $\sup(F, G)$ is monotonic.

Let $f, g \in \mathcal{M}_+(X)$. We have $F(f+g) \leq F(f) + F(g)$ and $G(f+g) \leq G(f) + G(g)$, taking the supremum we get $\sup(F, G)(f+g) \leq \sup(F(f) + F(g), G(f) + G(g)) \leq \sup(F(f), G(f)) + \sup(F(g), G(g)) = \sup(F, G)(f) + \sup(F, G)(g)$ and $\sup(F, G)$ is an upper parametric prevision.

Now let $(f_k)_{k \in \mathbb{N}}$ be an ascending sequence of elements of $\mathcal{M}_+(X)$ (and thus $\sup_{k \in \mathbb{N}}^\uparrow f_k \in \mathcal{M}_+(X)$). We have $\sup(F, G)(\sup_{k \in \mathbb{N}}^\uparrow f_k) = \sup(F(\sup_{k \in \mathbb{N}}^\uparrow f_k), G(\sup_{k \in \mathbb{N}}^\uparrow f_k)) = \sup(\sup_{k \in \mathbb{N}}^\uparrow F(f_k), \sup_{k \in \mathbb{N}}^\uparrow G(f_k))$ and the suprema commute and then $\sup(F, G)(\sup_{k \in \mathbb{N}}^\uparrow f_k) = \sup_{k \in \mathbb{N}}^\uparrow \sup(F, G)(f_k)$. We conclude that $\sup(F, G)$ is ω -continuous.

3. Let F, G be two upper ω -continuous parametric prevision.

Since for all $f, g \in \mathcal{M}_+(X)$, $F(f)$ and $G(g)$ belong to $\mathcal{M}_+(X)$ then for all $h \in \mathcal{M}_+(X)$, $F(G(h))$ belongs to $\mathcal{M}_+(X)$.

Let $\alpha \geq 0$ and $f \in \mathcal{M}_+(X)$, since F, G are positively homogeneous then $F \circ G(\alpha f) = F(G(\alpha f)) = F(\alpha G(f)) = \alpha F \circ G(f)$, we conclude that $F \circ G$ is positively homogeneous.

The composition of two monotonic maps is also monotonic thus $F \circ G$ is monotonic.

Let $f, g \in \mathcal{M}_+(X)$. We have $G(f+g) \leq G(f) + G(g)$, and since F is monotonic, $F \circ G(f+g) \leq F(G(f) + G(g)) \leq F \circ G(f) + F \circ G(g)$ and $F \circ G$ is an upper parametric prevision.

Now let $(f_k)_{k \in \mathbb{N}}$ be an ascending sequence in $\mathcal{M}_+(X)$. We have $F \circ G(\sup_{k \in \mathbb{N}}^\uparrow f_k) = F(G(\sup_{k \in \mathbb{N}}^\uparrow f_k)) = F(\sup_{k \in \mathbb{N}}^\uparrow G(f_k)) = \sup_{k \in \mathbb{N}}^\uparrow F \circ G(f_k)$ and we conclude that $F \circ G$ is ω -continuous.

The main difference between prevision and parametric prevision is the co-domain. Since the domain and the co-domain are the same, we can compose two parametric previsions to construct a new one. It allows us to think about least fixed points of parametric previsions.

Definition 2 (Previsions)

Let X be a non-empty set. Let F be a map from $\mathcal{M}_+(X)$ to $[0, +\infty]$. The map F is said to be a prevision if:

1. F is positively homogeneous;
2. F is monotonic;

Moreover, a prevision is said to be:

1. (lower) $F(f+g) \geq F(f) + F(g)$, for all functions $f, g \in \mathcal{M}_+(X)$;
2. (upper) $F(f+g) \leq F(f) + F(g)$, for all functions $f, g \in \mathcal{M}_+(X)$;
3. (linear) $F(f+g) = F(f) + F(g)$, for all functions $f, g \in \mathcal{M}_+(X)$;
4. (ω -continuous) for all ascending family $(f_n)_{n \in \mathbb{N}} \in \mathcal{M}_+(X)$, $F(\sup_{n \in \mathbb{N}}^\uparrow f_n) = \sup_{n \in \mathbb{N}}^\uparrow F(f_n)$.

The set $\mathcal{M}_+(X)$ is equipped with the pointwise ordering. The following proposition shows why the term *parametric* appears in Definition 1. The space of parameters is the same of domain of the functions of $\mathcal{M}_+(X)$ i.e. the set X . When we fix a parameter, we get a classical prevision.

Proposition 4 (parametric prevision and previsions)

The parametric F from $\mathcal{M}_+(X)$ to itself is a parametric (upper, lower, linear, ω -continuous) prevision iff for all $x \in X$, the map F_x from $\mathcal{M}_+(X)$ to $[0, +\infty]$ defined as $F_x(f) = F(f)(x)$ for all $f \in \mathcal{M}_+(X)$ is a (upper, lower, linear, ω -continuous) classical prevision and the maps $x \rightarrow F_x(h)$ are measurable for all $h \in \mathcal{M}_+(X)$.

The nondeterminism due to the tests and the value **err** implies that we cannot expect linearity. Indeed the binary supremum of the sum is not equal to the sum of the suprema, we have only an inequality. In the case of $Ans = \mathbb{R}_+ \cup \{+\infty\}$, we can establish that the weakest preconditions and continuation-passing style semantics defines an upper parametric prevision. To prove this result, we need a lemma which says that the semantics maps $\mathcal{M}_+(X)$ to itself.

Lemma 5

If $\kappa \in \mathcal{M}_+(\Sigma_\star)$, then, for all instructions $\ell^1 P$, $\text{wp}[\ell^1 P, \ell_2]_\star(\kappa) \in \mathcal{M}_+(\Sigma_\star)$.

Proof We prove this result by induction on the instructions.

- Since the skip is the identity map, thus $\kappa \in \mathcal{M}_+(\Sigma_\star)$ implies that $\text{wp}[\ell^1 \text{skip}]_\star(\kappa)$ also belongs to $\mathcal{M}_+(\Sigma_\star)$.

- Now, we consider the assignment. We define the map $h : \Sigma_\star \mapsto \Sigma_\star$ such that at ρ h associates $\rho(y)$ if $y \neq x$ and $\llbracket e \rrbracket_\star(\rho)$ otherwise. A coordinate of h is either coordinate projection or the concrete semantics of an expression which from Proposition 1 is measurable. We conclude that h is measurable since it is componentwise measurable. We conclude that $\text{wp}[\ell^1 x := e, \ell_2]_\star(\kappa) \in \mathcal{M}_+(\Sigma_\star)$ (composition of) for all $\kappa \in \mathcal{M}_+(\Sigma_\star)$.

- Let $\kappa \in \mathcal{M}_+(\Sigma_\star)$. The function $\text{wp}[\ell^1 P; \ell^2 Q, \ell_3]_\star(\kappa)$ is defined as $\text{wp}[\ell^1 P; \ell_2]_\star(\text{wp}[\ell^2 Q, \ell_3]_\star(\kappa))$. Suppose that $\text{wp}[\ell^1 P, \ell_2]_\star(\kappa')$ and $\text{wp}[\ell^2 Q, \ell_3]_\star(\kappa'')$ belong to $\mathcal{M}_+(\Sigma_\star)$ for all $\kappa', \kappa'' \in \mathcal{M}_+(\Sigma_\star)$. Then $\kappa' := \text{wp}[\ell^2 Q, \ell_3]_\star(\kappa)$ is positive and measurable. We conclude that $\text{wp}[\ell^1 P; \ell_2]_\star(\kappa') \in \mathcal{M}_+(\Sigma_\star)$.

- Let $\kappa \in \mathcal{M}_+(\Sigma_\star)$. We have:

$$\begin{aligned} \text{wp}[\ell^1 \text{if } t \text{ then } \ell^2 P \text{ else } \ell^3 Q, \ell_4]_\star(\kappa) &= \sup \left(\text{wp}[\ell^1 P, \ell_4]_\star(\kappa), \text{wp}[\ell^2 Q, \ell_4]_\star(\kappa) \right) \chi_{\{\rho \mid \llbracket t \rrbracket_\star(\rho) = \{0,1\}\}} \\ &+ \text{wp}[\ell^1 P, \ell_4]_\star(\kappa) \chi_{\{\rho \mid \llbracket t \rrbracket_\star(\rho) = \{1\}\}} \\ &+ \text{wp}[\ell^2 Q, \ell_4]_\star(\kappa) \chi_{\{\rho \mid \llbracket t \rrbracket_\star(\rho) = \{0\}\}} \end{aligned}$$

Suppose that $\text{wp}[\ell^1 P, \ell_4]_\star(\kappa)$ and $\text{wp}[\ell^2 Q, \ell_4]_\star(\kappa)$ are in $\mathcal{M}_+(\Sigma_\star)$. From Proposition 2 the functions $\chi_{\{\rho \mid \llbracket t \rrbracket_\star(\rho) = 1\}}$, $\chi_{\{\rho \mid \llbracket t \rrbracket_\star(\rho) = 0\}}$ and $\chi_{\{\rho \mid \llbracket t \rrbracket_\star(\rho) = \{0,1\}\}}$ are positive measurable functions. Since $\mathcal{M}_+(\Sigma_\star)$ is stable by product, sum and binary suprema, thus $\text{wp}[\ell^1 \text{if } t \text{ then } \ell^2 P \text{ else } \ell^3 Q, \ell_4]_\star(\kappa) \in \mathcal{M}_+(\Sigma_\star)$.

- Let $\kappa \in \mathcal{M}_+(\Sigma_\star)$. We have, from Lemma 3:

$$\begin{aligned} \text{wp}[\ell^1 \text{while } t \ell^2 P, \ell_3]_\star(\kappa) &= \left(\text{lfp} \left(H_{t, \ell^2 P, \ell_1} \right) \right) (\kappa) = \left(\sup_{n \in \mathbb{N}}^\uparrow H_{t, \ell^2 P, \ell_1}^n \left(\perp_{\mathbf{F}(\Sigma_\star, \overline{\mathbb{R}}_+)} \right) \right) (\kappa) \\ &= \sup_{n \in \mathbb{N}}^\uparrow \left(H_{t, \ell^2 P, \ell_1}^n \left(\perp_{\mathbf{F}(\Sigma_\star, \overline{\mathbb{R}}_+)} \right) \right) (\kappa) \end{aligned}$$

We suppose that $\text{wp}[\ell^2 P, \ell_1]_\star(\kappa') \in \mathcal{M}_+(\Sigma_\star)$ for all $\kappa' \in \mathcal{M}_+(\Sigma_\star)$. Since $\mathcal{M}_+(\Sigma_\star)$ is an ω -cpo the smallest of which is the null function. It suffices to show that for all $n \in \mathbb{N}$, $\left(H_{t, \ell^2 P, \ell_1}^n \left(\perp_{\mathbf{F}(\Sigma_\star, \overline{\mathbb{R}}_+)} \right) \right) (\kappa)$ belongs to $\mathcal{M}_+(\Sigma_\star)$. We prove this property by induction on integers. The null function is positive and measurable. Now, we suppose that there exists an integer n such that $\left(H_{t, \ell^2 P, \ell_1}^n \left(\perp_{\mathbf{F}(\Sigma_\star, \overline{\mathbb{R}}_+)} \right) \right) (\kappa)$ belongs to $\mathcal{M}_+(\Sigma_\star)$. We have:

$$\begin{aligned} \left(H_{t, \ell^2 P, \ell_1}^{n+1} \left(\perp_{\mathbf{F}(\Sigma_\star, \overline{\mathbb{R}}_+)} \right) \right) (\kappa) &= \sup \left(\text{wp}[\ell^2 P, \ell_1]_\star \left(\left(H_{t, \ell^2 P, \ell_1}^n \left(\perp_{\mathbf{F}(\Sigma_\star, \overline{\mathbb{R}}_+)} \right) \right) (\kappa) \right), \kappa \right) \chi_{\{\rho \mid \llbracket t \rrbracket_\star(\rho) = \{0,1\}\}} \\ &+ \text{wp}[\ell^2 P, \ell_1]_\star \left(\left(H_{t, \ell^2 P, \ell_1}^n \left(\perp_{\mathbf{F}(\Sigma_\star, \overline{\mathbb{R}}_+)} \right) \right) (\kappa) \right) \chi_{\{\rho \mid \llbracket t \rrbracket_\star(\rho) = \{1\}\}} \\ &+ \kappa \chi_{\{\rho \mid \llbracket t \rrbracket_\star(\rho) = \{0\}\}} \end{aligned}$$

From induction hypothesis (on instructions and integers n), Proposition 2 and stability of product, sum in $\mathcal{M}_+(\Sigma_\star)$ and binary suprema, we conclude that $\left(H_{t, \ell_2 P, \ell_1}^{n+1}(\perp_{\mathbf{F}(\Sigma_\star, \bar{\mathbb{R}}_+)})\right)(\kappa)$ belongs to $\mathcal{M}_+(\Sigma_\star)$. In conclusion, for all $n \in \mathbb{N}$, $\left(H_{t, \ell_2 P, \ell_1}^n(\perp_{\mathbf{F}(\Sigma_\star, \bar{\mathbb{R}}_+)})\right)(\kappa)$ belongs to $\mathcal{M}_+(\Sigma_\star)$ and $\text{wp}^{\llbracket \ell_1 \rrbracket}$ while $t^{\ell_2 P, \ell_3}(\kappa) \in \mathcal{M}_+(\Sigma_\star)$.

Proposition 5

When $\text{Ans} = \mathbb{R}_+ \cup \{+\infty\}$ and $X = \Sigma_\star$, for every instruction ${}^\ell P$, for every label ℓ' , $\text{wp}^{\llbracket \ell P, \ell' \rrbracket}_\star$ is an upper ω -continuous parametric prevision.

Proof The fact that for every instruction ${}^\ell P$, for every label ℓ' , $\text{wp}^{\llbracket \ell P, \ell' \rrbracket}_\star$ is ω -continuous and monotonic follows directly from Lemma 4. The measurability has just been proved in Lemma 5. It suffices to show the positive homogeneity and the "upper condition". We prove it by induction on instructions.

- The identity is clearly a linear ω -continuous prevision thus $\text{wp}^{\llbracket \text{skip}, \ell' \rrbracket}_\star$ is an upper parametric prevision.
- Suppose, we have a map $g : \Sigma_\star \rightarrow \Sigma_\star$ and consider a map F from $\mathcal{M}_+(\Sigma_\star)$ to itself defined by $F(f) = f \circ g$ for all $f \in \mathcal{M}_+(\Sigma_\star)$. The map F is clearly a linear ω -continuous prevision, this implies that $\text{wp}^{\llbracket \ell_1 x := e, \ell_2 \rrbracket}_\star$ is an upper parametric prevision.
- By induction hypothesis, the maps $\text{wp}^{\llbracket \ell_2 P, \ell_4 \rrbracket}_\star$ and $\text{wp}^{\llbracket \ell_3 Q, \ell_4 \rrbracket}_\star$ are upper parametric previsions by Proposition 3 (the third point) $\text{wp}^{\llbracket \ell_1 P; \ell_2 Q, \ell_3 \rrbracket}_\star$ is an upper parametric prevision.
- We use Proposition 4. For all ρ such that $\llbracket t \rrbracket_\star(\rho) = \{0\}$,

$$\kappa \mapsto \text{wp}^{\llbracket \ell_1 \text{ if } t \text{ then } \ell_2 P \text{ else } \ell_3 Q, \ell_4 \rrbracket}_\star(\kappa)(\rho) = \left(\text{wp}^{\llbracket \ell_0 P_0, \ell_2 \rrbracket}_\star(\kappa)\right)(\rho)$$

which is by induction hypothesis a classical upper prevision. For all ρ such that $\llbracket t \rrbracket_\star(\rho) = \{1\}$, the same argument leads to the result. Now suppose that $\llbracket t \rrbracket_\star(\rho) = \{0, 1\}$, by Proposition 3 (the second point), we conclude that $\kappa \mapsto \text{wp}^{\llbracket \ell_1 \text{ if } t \text{ then } \ell_2 P \text{ else } \ell_3 Q, \ell_4 \rrbracket}_\star(\kappa)(\rho)$ is, by induction hypothesis, a classical upper prevision. The map $\kappa \mapsto \text{wp}^{\llbracket \ell_1 \text{ if } t \text{ then } \ell_2 P \text{ else } \ell_3 Q, \ell_4 \rrbracket}_\star(\kappa)(\rho)$ is a classical upper prevision for all $\rho \in \Sigma_\star$ then $\text{wp}^{\llbracket \ell_1 \text{ if } t \text{ then } \ell_2 P \text{ else } \ell_3 Q, \ell_4 \rrbracket}_\star$ is an upper parametric prevision.

- By Proposition 3 (the first point), it suffices to prove that the auxiliary map $H_{t, \ell_2 P, \ell_3}(\perp_{\mathbf{F}(\Sigma_\star, \text{Ans})})$ is an upper ω -continuous parametric prevision. From Lemma 3, $H_{t, \ell_2 P, \ell_3}(\perp_{\mathbf{F}(\Sigma_\star, \text{Ans})})$ is ω -continuous and monotonic. It suffices to show that $H_{t, \ell_2 P, \ell_3}(\perp_{\mathbf{F}(\Sigma_\star, \text{Ans})})$ is positively homogeneous and upper. We prove the result by using Proposition 4. Let $\rho \in \Sigma_\star$. Suppose that $\llbracket t \rrbracket_\star(\rho) = \{1\}$. The result follows from the induction hypothesis. Now suppose that $\llbracket t \rrbracket_\star(\rho) = \{0\}$, $H_{t, \ell_2 P, \ell_3}(\perp_{\mathbf{F}(\Sigma_\star, \text{Ans})})$ is the identity and the result follows from the linearity of the identity. Finally suppose that $\llbracket t \rrbracket_\star(\rho) = \{0, 1\}$, the result follows from the stability of upper parametric prevision by binary suprema.

6 Special case of inputs

In this subsection, we are interested in interaction between the program and an external environment. This interaction can be viewed as a sensor which saves data from the external environment thanks to a command *input*. We suppose that these data are at the same time noisy and imprecise. Mathematically, it can be modelled by ω -capacities. It means that we want to represent for a fixed environment ρ the *input* as a ω -capacity. We assume that only k variables $x_{i_1}, x_{i_2}, \dots, x_{i_k}$ are affected by the input.

A ω -capacity on a topological space X is a map $\nu : \mathcal{B}(X) \mapsto \mathbb{R}_+$ such that:

$$\nu(\emptyset) = 0, \nu(U) \geq 0 \text{ and } \forall U \in \mathcal{B}(X) .$$

The ω -capacity is said to be:

- monotonic iff $\forall U, V \in \mathcal{B}(X)$:

$$U \subseteq V \implies \nu(U) \leq \nu(V) ;$$

- continuous iff for all nondecreasing sequences $(U_n)_{n \in \mathbb{N}} \subseteq \mathcal{B}(X)$:

$$\nu \left(\bigcup_{n \in \mathbb{N}}^{\uparrow} U_n \right) = \sup_{n \in \mathbb{N}}^{\uparrow} \nu(U_n) ;$$

- convex iff for all $U, V \in \mathcal{B}(X)$:

$$\nu(U \cup V) + \nu(U \cap V) \geq \nu(U) + \nu(V) ;$$

- concave iff for all $U, V \in \mathcal{B}(X)$:

$$\nu(U \cup V) + \nu(U \cap V) \leq \nu(U) + \nu(V) ;$$

We will use the following result relying convexity and sub(super)linearity of the Choquet integrals.

Proposition 6

Let ν be a topological space. Let f and g be in $\mathcal{M}_+(X)$. Let α, β two positive reals.

Let ν be a convex ω -capacity, then the Choquet integral is superlinear:

$$\int_{x \in X} \alpha f(x) + \beta g(x) d\nu \geq \alpha \int_{x \in X} f(x) d\nu + \beta \int_{x \in X} g(x) d\nu$$

Let μ be a concave ω -capacity, then the Choquet integral is sublinear:

$$\int_{x \in X} \alpha f(x) + \beta g(x) d\mu \leq \alpha \int_{x \in X} f(x) d\mu + \beta \int_{x \in X} g(x) d\mu$$

For $\rho \in \Sigma_\star$, we suppose that $\llbracket (x_{i_1}, x_{i_2}, \dots, x_{i_k} = \text{input}()) \rrbracket_c(\rho)$ is a monotonic continuous ω -capacity ν on $\mathcal{V}_I = \{x_{i_1}, x_{i_2}, \dots, x_{i_k}\}$. We denote $\mathcal{V}_{-I} = \{x \in \mathcal{V}, x \notin \mathcal{V}_I\}$ and we suppose that a certain $\rho_0 : \mathcal{V}_{-I} \mapsto \mathbb{R}_e$ (or with value in \mathbb{F}_e) is given. We want to extend the ω -capacity $\llbracket \text{input} \rrbracket_c(\rho)$ to \mathbb{R}_e (or \mathbb{F}_e) with respect to the fact that the unaffected variables are represented by a fixed environment ρ_0 . We extend $\llbracket \text{input} \rrbracket_c(\rho)$ to a ω -capacity $\overline{\llbracket \text{input} \rrbracket}_c(\rho)$ over $\Sigma_\star (\simeq \mathbb{R}_e^\mathcal{V}$ or $\simeq \mathbb{F}_e^\mathcal{V})$ as follows:

$$\overline{\llbracket \text{input} \rrbracket}_c(\rho)(C) = \llbracket \text{input} \rrbracket_c(\rho) (\{x \in \mathbb{R}_e^{\mathcal{V}_I} \mid (x, \rho_0) \in C\})$$

for all Borel sets C of Σ_\star . This latter definition means that the measure of a Borel set is completely determined by its affected part (by the instruction *input*).

Assumption 2

We assume that $\rho \mapsto \overline{\llbracket \text{input} \rrbracket}_c(\rho)(U)$ is measurable for all $U \in \mathcal{B}(\Sigma_\star)$.

We define a last semantics which is the integration of "continuation" by a ω -capacity. Let $\kappa : \Sigma_\star \mapsto \overline{\mathbb{R}}_+$ be a positive measurable function. We define the semantics of the instruction *input* as:

$$\text{wp} \llbracket \text{input} \rrbracket_\star(\kappa)(\rho) := \int_{\rho'} \kappa(\rho') d\overline{\llbracket \text{input} \rrbracket}_c(\rho)$$

Proposition 7

Under the Assumption 2, for all $\kappa \in \mathcal{M}_+(\Sigma_\star)$, the function $\rho \mapsto \text{wp}[[input]]_\star(\kappa)(\rho)$ belongs to $\mathcal{M}_+(\Sigma_\star)$.

Proof The positivity is clear from the definition of the Choquet integral. We only give a proof for the measurability. Let κ be a positive measurable function. Then κ is the nondecreasing supremum of a sequence of positive step functions $(\varphi_n)_{n \in \mathbb{N}}$ and:

$$\int_{x \in X} \kappa(x) d\overline{[input]_c}(\rho) = \int_{x \in X} \sup_{n \in \mathbb{N}}^\uparrow \varphi_n(x) d\overline{[input]_c}(\rho) .$$

From the ω -Scott continuity of Choquet integrals, we get:

$$\int_{\rho' \in \Sigma_\star} \kappa(\rho') d\overline{[input]_c}(\rho) = \sup_{n \in \mathbb{N}}^\uparrow \int_{\rho' \in \Sigma_\star} \varphi_n(\rho') d\overline{[input]_c}(\rho) .$$

The function φ_n have the form $\alpha_n \sum_{i=0}^{K_n} \chi_{A_i^n}$ with $(A_i^n)_i$ is a nonincreasing sequence of Borel sets for all $n \in \mathbb{N}$. Thus, we have:

$$\int_{\rho' \in X} \varphi_n(\rho') d\overline{[input]_c}(\rho) = \alpha_n \sum_{i=0}^{K_n} \overline{[input]_c}(\rho)(A_i^n) .$$

Hence, from the Assumption 2, the map $\rho \mapsto \overline{[input]_c}(\rho)(A_i^n)$ is measurable for all $n \in \mathbb{N}$, for all $i \in \{0, \dots, K_n\}$ and then for all $n \in \mathbb{N}$, $\rho \mapsto \alpha_n \sum_{i=0}^{K_n} \overline{[input]_c}(\rho)(A_i^n)$ is measurable. We conclude

that the map: $\rho \mapsto \int_{\rho' \in \Sigma_\star} \kappa(\rho') d\overline{[input]_c}(\rho)$ is measurable since it is the pointwise supremum of measurable functions.

Proposition 8

If the ω -capacity $\overline{[input]_c}(\rho)$ is convex (concave) and ω -continuous then $\kappa \mapsto \text{wp}[[input]]_\star(\kappa)(\rho)$ defines a upper (lower) ω -continuous prevision.

The proof of this latter proposition is left to the reader. Indeed, from Proposition 6, if the capacity is convex (concave) then the Choquet integral is superlinear (sublinear). The proof is thus reduced to show that if the capacity $\overline{[input]_c}(\rho)$ is convex or concave then the extended capacity $\overline{[input]_c}(\rho)$ fulfills the same property.

References

- [BGGP11] Olivier Bouissou, Éric Goubault, Jean Goubault-Larrecq, and Sylvie Putot. A generalization of P-boxes to affine arithmetic, and applications to static analysis of programs. In *Proceedings of the 14th GAMM-IMACS International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics (SCAN'10)*, Lyon, France, September 2011. To appear.
- [GL07] Jean Goubault-Larrecq. Continuous capacities on continuous state spaces. In *ICALP'2007*. Springer-Verlag LNCS, 2007.
- [Gou07] Jean Goubault-Larrecq. Continuous previsions. In Jacques Duparc and Thomas A. Henzinger, editors, *Proceedings of the 16th Annual EACSL Conference on Computer Science Logic (CSL'07)*, pages 542–557, Lausanne, Switzerland, September 2007. Springer-Verlag LNCS 4646.

- [Koz81] Dexter Kozen. Semantics of probabilistic programs. *Journal of Computer and System Sciences*, 22:328–350, 1981.
- [Law97] Jimmie Lawson. Spaces of maximal points. *Mathematical Structures in Computer Science*, 7:543–555, 1997.
- [Mon08] David Monniaux. The pitfalls of verifying floating-point computations. *Transactions on Programming Languages and Systems*, 30(3), 2008. Article 12.