Semi-decidability of may, must and probabilistic testing in a higher-type setting

Martín Escardó

School of Computer Science, University of Birmingham, UK

Abstract

We show that, in a fairly general setting including higher-types, may, must and probabilistic testing are semi-decidable. The case of must testing is perhaps surprising, as its mathematical definition involves universal quantification over the infinity of possible outcomes of a non-deterministic program. The other two involve existential quantification and integration. We also perform first steps towards the semi-decidability of similar tests under the simultaneous presence of non-deterministic and probabilistic choice.

Keywords: Non-deterministic and probabilistic computation, nigher-type computability theory and exhaustible sets, may and must testing, operational and denotational semantics, powerdomains.

1 Introduction

We consider a non-deterministic higher-type language, in the style of PCF, that includes angelic, demonic and probabilistic choice. The types are closed under finite products and function spaces, and certain powertype constructors, interpreted as powerdomain monads, which capture various combinations of kinds of non-determinism. Choices can only be performed at powertypes, and the different powertypes have different operational interpretations of choice.

We show that (i) may, (ii) must and (iii) probabilistic testing are semi-decidable for this language. The idea is that, given a semi-decidable property u, one can semi-decide whether a given non-deterministic program (i) has some outcome satisfying u, (ii) has all outcomes satisfying u, and (iii) has all outcomes satisfying u with probability bigger than a given number. The proofs exploit recent results on exhaustible sets in higher-type computation [9,6], and older results on exact computability and definability of integrals in PCF-like languages [5,28,26].

Even at ground types, the claim for must testing may seem suspicious: for example, it implies that for any non-deterministic program of natural number type, possibly including subterms of arbitrarily high types, it is semi-decidable whether the outcome of the program must be a prime number. The semi-decision procedure is expected to say "yes" if all possible outcomes are prime numbers, but it is not obliged to say "no" if the outcomes include composite numbers or divergent computations.

From the point of view of computability theory, we have an extensional procedure that operates on programs, but that is not definable directly on the observable input-output behaviour of programs, and hence the Rice-Shapiro theorem fails for non-deterministic programming systems. This leads us to extend the language with may, must and probabilistic testing primitives. It is interesting that these tests define basic open sets of the Scott topology of powerdomains that are not definable in the original language. The resulting language has an operational semantics, but can hardly be regarded as a programming language. We consider it as an executable program logic for semi-decidable properties, which plays the role of a sort of "Rice-Shapiro completion" of the programming language (although we don't have at present a precise formulation of such a concept).

Our semi-decision procedures are defined using operational semantics, and their correctness is proved using domain-theoretic denotational semantics. Perhaps surprisingly, may testing is harder than must testing in a sense: inclusion of the former to the language leads to definability of parallel-convergence (even on programs of deterministic type), but the latter can be defined without parallel features. Probabilistic testing also requires parallel features.

Our results on may, must and probabilistic testing are very general, but have restrictions, discussed in the body of the paper, due to open problems in domain theory involving the probabilistic powerdomain.

We also perform first steps towards the semi-decidability of similar tests under the simultaneous presence of probabilistic choice and non-deterministic choice [19,31,30]. We develop semi-decision procedures for this, but their correctness is a conjecture and their scope is open.

This work is related to Abramsky's work on logic of observable properties [1], going back to Smyth [29], but we take a different approach and also account for probabilistic computation. The relationship between the two approaches certainly deserves more scrutiny.

Organization. Section 2: A language for non-determinism and probability. Section 3: Operational semantics and semi-decision procedures. Section 4: Denotational semantics and correctness of the procedures.

Acknowledgements. A preliminary version of this work, with more conjectures than results, was presented in 2005 at a workshop in McGill University Bellairs research institute, organized by Prakash Panangaden. I thank him and the other participants for discussions, in particular Vincent Danos, Achim Jung, Klaus Keimel, Jimmie Lawson, Gordon Plotkin, Steve Vickers. I also had useful discussions with Alex Simpson and Paul Levy. And I thank Prakash again, for having invited me to take part of a similar meeting previously in 2003, which led to the paper [9] for the proceedings (and Vincent as an editor for pressing me to write it), which in turn led me to develop the ideas presented in the 2005 meeting and here as an application.

2 A language for non-determinism and probability

We study a language, which we refer to as MMP, for may, must and probabilistic testing. The language has a type system with an explicit distinction between deterministic and various kinds of non-deterministic types. A sublanguage is singled out as a programming language, and the full language is considered as an executable logic for semi-decidable properties of terms of the programming language.

Any term of deterministic type has only one outcome, including the possibility of divergence. A term of non-deterministic type has one or more runs, in general continuum many, each of which either produces a single outcome, again including the possibility of divergence, but different convergent runs may produce different outcomes.

We take the ground types to be deterministic, and the product- and function-type constructions to preserve determinism. In fact, a slightly larger deterministic language is embedded in our language, introduced in Section 2.1. In Section 3, we give an operational semantics for the full language by compositional compilation into the deterministic fragment, where the translation is the identity on this fragment.

2.1 PCF + `S` *+* `I`

We consider an extension of (call-by-name) PCF with product types and base types `S` and `I`. Thus, our ground types are

$$\gamma ::= \mathtt{Nat} \mid \mathtt{Bool} \mid \mathtt{S} \mid \mathtt{I}.$$

The types `Nat` and `Bool` for natural numbers and booleans include divergent terms but no multi-valued terms, and are intended for programming purposes.

The other two types are intended for logical purposes. The Sierpinski type `S` is for results of observations or semi-decisions, with an element $\top$ (observable true) and divergence (unobservable false). The type `I` is for observations of probabilities, and is interpreted as a "vertical unit-interval type" with elements in $[0,1] \subseteq \mathbb{R}$ under the natural order (hence zero is bottom and one is top).

For proving our claims for may and must testing, we need two terms

$$\exists, \forall \colon (\mathtt{Cantor} \to \mathtt{S}) \to \mathtt{S}.$$

Here the type

$$\mathtt{Cantor} = (\mathtt{Nat} \to \mathtt{Bool})$$

is thought of as a type of sequences of booleans, where we are mostly concerned with total sequences, which will play the role of schedulers. For definability of the existential quantifier one needs parallel-convergence, but the universal quantifier is sequentially definable in PCF+`S` [9].

For probabilistic testing, we need three terms

$$\sup, \inf, \int \colon (\mathtt{Cantor} \to \mathtt{I}) \to \mathtt{I}.$$

For integration we take the uniform distribution on the total elements of `Cantor`.

Exact real-number computation in PCF-like languages and domain models has been discussed extensively in the last fifteen years, by a number of authors, and we don't stop to rehearse this [4,8,22,5,28,26]. We just emphasize that, because we are not using the interval domain but rather the vertical unit interval, computations of type $\mathtt{I}$ produce (rational) approximations from below. In particular, for a term of the form $\int f$, we are able to semi-decide the condition $r < \int f$, but not the conditions $\int f = r$ or $\int f > r$.

For the sake of clarity, we use the following notation for writing terms, where the letter s ranges over the Cantor type:

$$\sup_s f[s] = \sup(\lambda s.f[s]), \quad \inf_s f[s] = \inf(\lambda s.f[s]),$$
$$\int f[s]\,\mathrm{d}s = \int(\lambda s.f[s]), \quad \exists s.p[s] = \exists(\lambda s.p[s]), \quad \forall s.p[s] = \forall(\lambda s.p[s]).$$

2.2 Extension with non-determinism and probability

We consider an extension of the above language so that the sublanguage remains deterministic when it is embedded into the extension. Non-deterministic and probabilistic terms have to be explicitly typed as such. Types that admit non-deterministic and probabilistic terms are obtained via "powertype" constructors.

Interestingly, and perhaps counter-intuitively, we shall have terms defined on non-deterministic or probabilistic types with values on deterministic types, which hence will produce deterministic outputs from non-deterministic or probabilistic inputs. These arise from the introduction of may, must and probabilistic testing constructs, and are considered part of the executable logic and not of the programming fragment.

Types.

The ground types, ranged over by γ, are those PCF + $\mathtt{S}$ + $\mathtt{I}$. General types are ranged over by σ and τ and are given by

$$\sigma, \tau ::= \gamma \mid \sigma \times \tau \mid \sigma \to \tau \mid F\sigma,$$

where F ranges over type constructors defined by

$$F ::= \mathtt{H} \mid \mathtt{S} \mid \mathtt{P} \mid \mathtt{V} \mid \mathtt{V}_{\mathtt{H}} \mid \mathtt{V}_{\mathtt{S}} \mid \mathtt{V}_{\mathtt{P}}\,.$$

The three constructors $\mathtt{H}, \mathtt{S}, \mathtt{P}$ (Hoare, Smyth and Plotkin powertypes) are for non-deterministic computation, the constructor $\mathtt{V}$ (probabilistic powertype) is for probabilistic computation, and the constructors $\mathtt{V}_{\mathtt{H}}, \mathtt{V}_{\mathtt{S}}, \mathtt{V}_{\mathtt{P}}$ (Hoare, Smyth and Plotkin probabilistic powertypes) are for computations combining probability and non-determinism. Although one can certainly consider, for any type σ, the types $\mathtt{H}\,\mathtt{V}\,\sigma$, $\mathtt{S}\,\mathtt{V}\,\sigma$ and $\mathtt{P}\,\mathtt{V}\,\sigma$, they are not quite the same as $\mathtt{V}_{\mathtt{H}}\,\sigma$, $\mathtt{V}_{\mathtt{S}}\,\sigma$ and $\mathtt{V}_{\mathtt{P}}\,\sigma$, as will become apparent in due course (see also [30]).

Terms.

We extend the inductive definition of PCF + $\mathtt{S}$ + $\mathtt{I}$ terms with the following rules.

Monad rules.

Let F be a unary type constructor as above.

Functor rule. If $f\colon \sigma \to \tau$ is a term, then so is

$$Ff\colon F\sigma \to F\tau.$$

This amounts to the fact that any deterministic function can be considered as a non-deterministic function. To compute $Ff\colon F\sigma \to F\tau$ at a given input $x\colon F\sigma$, we first compute an outcome of x and then feed it to f, which in turn gives one of the possible outcomes of Ffx.

Unit rule. For each type σ, we have a term

$$\eta_F^\sigma\colon \sigma \to F\sigma,$$

where in practice we often omit one or both super- and subscripts from η (and from other terms that have similar decorations). This amounts to the fact that any deterministic computation can be regarded as a possibly non-deterministic computation which just happens to be able to produce precisely one outcome.

Multiplication rule. For each type σ, we have a constant

$$\mu_F^\sigma\colon FF\sigma \to F\sigma.$$

This amounts to the fact that a non-deterministic computation, each of whose possible outcomes is another non-deterministic computation of an element of σ, can be seen simply as a non-deterministic computation of an element of σ. To compute an outcome of the term μX, we first compute an outcome $x\colon F\sigma$ of $X\colon FF\sigma$, and then compute an outcome of x in σ.

Strength rule. For all types σ, we have a (for the moment nameless) constant of type $\sigma \times F\tau \to F(\sigma \times \tau)$. This is needed to get terms $F\sigma_1 \times \cdots \times F\sigma_n \to F\tau$ from terms $\sigma_1 \times \cdots \times \sigma_n \to \tau$ and functoriality. An important example is the construction of various non-deterministic and probabilistic conditionals from the deterministic one.

Choice rules.

The Hoare, Smyth and Plotkin (probabilistic or not) powertypes have a binary choice operator $\varovee$. The idea is that the runs of a term $M \varovee N$ are those of the term M together with those of the term N. The four probabilistic powertypes have a binary choice operator $\oplus$. Again the runs of a term $M \oplus N$ are those of the term M together with those of the term N. However, the choice $M \varovee N$ is angelic or demonic, whereas the choice $M \oplus N$ is probabilistic. This is made precise below. Notice that the Hoare, Smyth and Plotkin probabilistic powertypes include both choice operators, and hence they combine non-determinism and probability.

Non-deterministic choice rule.

For $F \in \{\mathtt{H}, \mathtt{S}, \mathtt{P}, \mathtt{V}_\mathtt{H}, \mathtt{V}_\mathtt{S}, \mathtt{V}_\mathtt{P}\}$, and for each type σ, we have a constant

$$(\varovee_F^\sigma)\colon F\sigma \times F\sigma \to F\sigma,$$

written in infix notation.

Probabilistic choice rule.

For $F \in \{\mathtt{V}, \mathtt{V}_\mathtt{H}, \mathtt{V}_\mathtt{S}, \mathtt{V}_\mathtt{P}\}$, and for each type σ, we have an infix constant

$$(\oplus^\sigma_F)\colon F\sigma \times F\sigma \to F\sigma.$$

Example.

In this language, the terms $\eta(\lambda x.0) \varovee \eta(\lambda x.1)$ and $\lambda x.\eta(0) \varovee \eta(1)$ are distinguishable from their types $F(\sigma \to \mathtt{Nat})$ and $\sigma \to F\mathtt{Nat}$ respectively. The first defines, non-deterministically, a function, whereas the second defines a single function with non-deterministic output. In languages that omit the type distinction we are making, and hence omit the coercion η, the terms $(\lambda x.0) \varovee (\lambda x.1)$ and $\lambda x.(0 \varovee 1)$ are indistinguishable under call by name, as discussed e.g. in the introduction of [27]. Here the two terms are distinguishable in terms of their behaviour too. In the first case, once we see a zero in the output for some given input, we'll always see zeros afterwards, no matter what the input is, whereas in the second we'll be able to get zeros and ones even for the same input.

May, must and probabilistic testing rules.

None of the seven powertypes types can be distinguished on the basis of the outcomes that their terms have. In order to make the powertypes be observably different, one considers may, must and probabilistic testing. For may testing, one existentially quantifies over outcomes, for must testing one universally quantifies over outcomes, and for probabilistic testing one integrates over probability distributions on outcomes (represented as valuations in the domain model of the language). The interpretation of the probabilistic choice rule amounts to choosing each branch with equal probability. For powertypes combining non-determinism and probability, the situation is more complicated and its discussion is postponed.

The Hoare powertype will admit only may testing, the Smyth powertype will admit only must testing, and the Plotkin powertype will admit both. The probabilistic powertype will admit probabilistic testing, and the powertypes types combining non-determinism and probability, will admit may-probabilistic testing (Hoare probabilistic powertype), must-probabilistic testing (Smyth probabilistic powertype), or both (Plotkin probabilistic powertype).

May and must rules.

The $\mathtt{S}$-valued terms are characteristic functions of open sets and we define a type of opens

$$\mathcal{O}\,\sigma = (\sigma \to \mathtt{S}).$$

May and must testing can be seen as ways of obtaining open sets of $F\sigma$ from open sets of σ, for certain non-deterministic type constructors, and hence we postulate corresponding constants $\Diamond$ (may) and $\Box$ (must):

$\Diamond^{\sigma}_{\mathtt{H}} : \mathcal{O}\,\sigma \to \mathcal{O}\,\mathtt{H}\,\sigma,$
$\Box^{\sigma}_{\mathtt{S}} : \mathcal{O}\,\sigma \to \mathcal{O}\,\mathtt{S}\,\sigma,$
$\Diamond^{\sigma}_{\mathtt{P}} : \mathcal{O}\,\sigma \to \mathcal{O}\,\mathtt{P}\,\sigma,$
$\Box^{\sigma}_{\mathtt{P}} : \mathcal{O}\,\sigma \to \mathcal{O}\,\mathtt{P}\,\sigma.$

The idea in the case of the Plotkin powertype is that if $u\colon\ \mathcal{O}\,\sigma$ and $N\colon\ \mathtt{P}\,\sigma$, then $\Diamond(u)(N) = \top$ if and only $u(x) = \top$ for some outcome x of a run of N, and $\Box(u)(N) = \top$ if and only $u(x) = \top$ for all outcomes x of runs of N. This is made precise later. The idea for the Hoare and Smyth powertypes is the same, but only one kind of test is made available for each of them, as discussed above.

Example.

Suppose one wants to semi-decide whether the outcome of a term $n\colon F\mathtt{Nat}$ must be prime. Then one first writes a semi-decision term $\mathtt{prime}\colon \mathtt{Nat} \to \mathtt{S}$ and then runs, in the executable logic, the ground term $\Box\,\mathtt{prime}\,n$ of type $\mathtt{S}$. If one wants to compute the probability that $n\colon\ \mathtt{V}\,\mathtt{Nat}$ produces a prime number, one proceeds in a similar manner, as discussed below.

Probabilistic rule.

This time, from an open set of σ we get an expectation on $\mathtt{V}\,\sigma$, where an expectation on σ is an $\mathtt{I}$-valued function:

$$\mathcal{E}\,\sigma = (\sigma \to \mathtt{I}).$$

By virtue of the vertical nature of the unit-interval type $\mathtt{I}$, any Sierpinski-valued term amounts to an $\mathtt{I}$-valued term with values 0 (bottom) and 1 (top) by composition with a coercion function $\mathtt{S} \to \mathtt{I}$. Hence expectations generalize open sets. We include a constant

$\bigcirc^{\sigma}_{\mathtt{V}}\colon\ \mathcal{E}\,\sigma \to \mathcal{E}\,\mathtt{V}\,\sigma.$

For a term $u\colon\ \mathcal{O}\,\sigma$ seen as a term of type $\mathcal{E}\,\sigma$, as discussed above, and a term $x\colon\ \mathtt{V}\,\sigma$, the idea is that $\bigcirc(u)(x) \in \mathtt{I}$ is the probability that u holds for outcomes of runs of a term $x\colon\ \mathtt{V}\,\sigma$.

More examples.

Recursively define a term $f\colon \mathtt{Nat} \to \mathtt{P}\,\mathtt{Nat}$ by

$$f(n) = \eta(n) \between f(n+1),$$

and let $\mathtt{converge}\colon \mathtt{Nat} \to \mathtt{S}$ be a term such that $\mathtt{converge}(n) = \top$ iff $n \neq \bot$. Then we intend that

$\Diamond\,\mathtt{converge}(f(0)) = \top$ ("$f(0)$ may converge")

and that

$\Box\,\mathtt{converge}(f(0)) = \bot$ ("it is not the case that $f(0)$ must converge"),

but

$\Box\,\texttt{converge}(\eta(0) \circledvee \eta(1)) = \top$.

Now recursively define a term $g\colon \texttt{Nat} \to \texttt{V}\,\texttt{Nat}$ by

$$g(n) = \eta(n) \oplus g(n+1),$$

Then we intend that

$\bigcirc\,\texttt{converge}(g(0)) = 1$ (“the probability that $g(0)$ converges is 1”),

and

$\bigcirc\,\texttt{converge}_n(g(0)) = 2^{-n-1}$ (“probability[$g(0)$ converges to n]=2^{-n-1}”),

where $\texttt{converge}_n\colon \texttt{Nat} \to \texttt{S}$ is a term such that $\texttt{converge}_n(x) = \top$ iff $x = n$.

Parallel-convergence defined from may testing.

Parallel convergence

$$(\vee)\colon \texttt{S} \times \texttt{S} \to \texttt{S},$$

written in infix notation, can be defined from may testing as

$$(p \vee q) = \Diamond\,\texttt{converge}(\eta(p) \circledvee \eta(q)),$$

where $\texttt{converge}\colon \texttt{S} \to \texttt{S}$ is the identity function. It can also be defined from probabilistic testing (cf. [7] and [5,20]):

$$(p \vee q) = 0 < \bigcirc\,\texttt{converge}(\eta(p) \oplus \eta(q)).$$

But it cannot be defined from must testing, because must testing can be defined without parallel features (as we do). Notice that $(p \wedge q) = \Box\,\texttt{converge}(\eta(p) \circledvee \eta(q))$.

May-probabilistic and must-probabilistic rules.

We now account for the combination of non-determinism and probability. We need four constants

$$\begin{aligned}
\Diamond^{\sigma}_{\texttt{V}_\texttt{H}} &: \mathcal{E}\,\sigma \to \mathcal{E}\,\texttt{V}_\texttt{H}\,\sigma,\\
\Box^{\sigma}_{\texttt{V}_\texttt{S}} &: \mathcal{E}\,\sigma \to \mathcal{E}\,\texttt{V}_\texttt{S}\,\sigma,\\
\Diamond^{\sigma}_{\texttt{V}_\texttt{P}} &: \mathcal{E}\,\sigma \to \mathcal{E}\,\texttt{V}_\texttt{P}\,\sigma,\\
\Box^{\sigma}_{\texttt{V}_\texttt{P}} &: \mathcal{E}\,\sigma \to \mathcal{E}\,\texttt{V}_\texttt{P}\,\sigma.
\end{aligned}$$

The idea here is a bit subtler and is explained later.

Notational conventions.

Motivated by topological descriptions of various powerdomains [2,30] and anticipating the denotational development of Section 4, we adopt the following notation. We use the letters C (suggesting closed), Q (compact), L (lens), ν (valuation), $\mathcal{C}$, $\mathcal{Q}$ and $\mathcal{L}$ to range over terms of the powertypes, in the order they have been given above:

$$C\colon \texttt{H}\,\sigma, \quad Q\colon \texttt{S}\,\sigma, \quad L\colon \texttt{P}\,\sigma, \quad \nu\colon \texttt{V}\,\sigma, \quad \mathcal{C}\colon \texttt{V}_\texttt{H}\,\sigma, \quad \mathcal{Q}\colon \texttt{V}_\texttt{S}\,\sigma, \quad \mathcal{L}\colon \texttt{V}_\texttt{P}\,\sigma.$$

2.3 Another view of may, must and probabilistic testing

This section motivates the operational semantics developed in Section 3, and mirrors the denotational semantics given in Section 4. Consider the may testing operator

$$\Diamond\colon\ \mathcal{O}\,\sigma \to \mathcal{O}\,\mathtt{H}\,\sigma$$

and recall that $\mathcal{O}\,\sigma = (\sigma \to \mathtt{S})$. By uncurrying this operator, then twisting the product, and currying again, we get a term that will be natural to denote by

$$\exists\colon\ \mathtt{H}\,\sigma \to ((\sigma \to \mathtt{S}) \to \mathtt{S}).$$

That is,

$$\exists(C)(u) = \Diamond(u)(C).$$

For $C\colon \mathtt{H}\,\sigma$, we write $\exists_C$ rather than $\exists(C)$. Moreover, for a term $u[x]\colon \sigma \to \mathtt{S}$ possibly including a free syntactic variable x, we write

$$\exists x \in C.u[x] = \exists(C)(\lambda x.u[x]).$$

With this notation, we have

$$\Diamond(u)(C) = \exists x \in C.u(x).$$

This tautology motivates the operational semantics for $\Diamond$ given in Section 3.

Similarly, from the must testing operator $\Box\colon\ \mathcal{O}\,\sigma \to \mathcal{O}\,\mathtt{S}\,\sigma$, we get a term

$$\forall\colon\ \mathtt{S}\,\sigma \to ((\sigma \to \mathtt{S}) \to \mathtt{S}),$$

for which analogous notational conventions are adopted. For the Plotkin powertype, we get both quantifiers.

For the probabilistic powertype, recalling that $\mathcal{E}\,\sigma = (\sigma \to \mathtt{I})$, from the probabilistic testing operator

$$\bigcirc\colon\ \mathcal{E}\,\sigma \to \mathcal{E}\,\mathtt{V}\,\sigma$$

we get a term

$$\int\colon\ \mathtt{V}\,\sigma \to ((\sigma \to \mathtt{I}) \to \mathtt{I})$$

defined by

$$\int_\nu u = \bigcirc(u)(\nu).$$

where $\nu\colon \mathtt{V}\,\sigma$ and $u\colon \sigma \to \mathtt{I}$.

With the aid of these concepts and notation, we are now able to explain the intended meaning of may-probabilistic and must-probabilistic testing in powertypes combining probability and non-determinism. For the Hoare probabilistic powertype, from the may-probabilistic operator

$$\Diamond\colon\ \mathcal{E}\,\sigma \to \mathcal{E}\,\mathtt{V}_\mathtt{H}\,\sigma$$

we get a term of type

$$\mathtt{V}_\mathtt{H}\,\sigma \to ((\sigma \to \mathtt{I}) \to \mathtt{I})$$

written

$$\sup_{\nu \in \mathcal{C}} \int_\nu u = \Diamond(u)(\nu).$$

What we mean by this notation is that the application of the nameless term $\mathtt{V}_\mathtt{H}\, \sigma \to ((\sigma \to \mathtt{I}) \to \mathtt{I})$ to a term $\mathcal{C} \colon \mathtt{V}_\mathtt{H}\, \sigma$ followed by an application to a term $u \colon \sigma \to I$ is written $\sup_{\nu \in \mathcal{C}} \int_\nu u$. The bound variable is included to make the notation suggestive of the denotational semantics to be given in Section 4.

For the Smyth probabilistic powertype, we get a similar term, but we instead write $\inf_{\nu \in \mathcal{Q}} \int_\nu u$. For the Plotkin probabilistic powertype we get both.

3 Operational semantics

We first briefly discuss the operational semantics of the programming fragment of our language.

3.1 Big-step semantics of the test-free fragment

For the fragment of the language without the may, must and probabilistic testing operators, we extend the big-step operational semantics of PCF + $\mathtt{S}$ + $\mathtt{I}$ with the following rules.

Firstly, we stipulate that if $v \colon \sigma$ is a value (or weak head normal form) then so is $\eta(v) \colon F\sigma$. Then, omitting types and contexts, we add the rules:

$$\frac{M \Downarrow v}{M \ovee N \Downarrow v} \qquad \frac{N \Downarrow v}{M \ovee N \Downarrow v} \qquad \frac{M \Downarrow v}{M \oplus N \Downarrow v} \qquad \frac{N \Downarrow v}{M \oplus N \Downarrow v}$$

$$\frac{M \Downarrow \eta(v) \quad f(v) \Downarrow w}{Ff(M) \Downarrow \eta(w)} \qquad \frac{M \Downarrow v}{\eta(M) \Downarrow \eta(v)} \qquad \frac{M \Downarrow \eta(V) \quad V \Downarrow \eta(W)}{\mu(M) \Downarrow \eta(W)}$$

Thus, all choice operators have the same meaning: this semantics only says what the possible outcomes of a term are, if any. The following is easy to establish (cf. [27]):

Proposition 3.1 *For the test-free fragment of the language, there is a ternary decidable relation*

$$M \Downarrow^s v,$$

where s ranges over the Cantor space of infinite binary sequences, such that

(i) *for any M and any s there is at most one v with $M \Downarrow^s v$, and*

(ii) *$M \Downarrow v$ iff there is some s with $M \Downarrow^s v$.*

The idea is that s is a scheduler that dictates which branches the choice operators have to take during evaluation. Once the scheduler is chosen, the evaluation is completely deterministic. For a test-free term M, we can say that

M must converge iff for every s there is v with $M \Downarrow^s v$.

Despite the universal quantification over an uncountable set, we can prove that must-convergence is semi-decidable for closed, test-free terms. The reason is that the Cantor space is compact and semi-decidable predicates are continuous [9,6].

However, we don't know how to define this and more general must- and probabilistic testings in the big-step style in an elegant and algorithmic way. Hence we instead translate our language into a deterministic language, using Proposition 3.1 above as the guiding idea.

3.2 Operational semantics of the full language

We define an operational semantics for our language MMP by compositional compilation into its deterministic sublanguage PCF + $\mathtt{S}$ + $\mathtt{I}$, where the translation is the identity on this sublanguage. The compilation map

$$\phi\colon \mathrm{MMP} \to \mathrm{PCF} + \mathtt{S} + \mathtt{I}$$

acts on both terms and types (like a functor): for every source term M of type σ, the translation produces a target term $\phi(M)$ of type $\phi(\sigma)$.

Translation of types.

This is defined by induction:

$$\begin{aligned} \phi(\gamma) &= \gamma, \\ \phi(\sigma \times \tau) &= \phi(\sigma) \times \phi(\tau), \\ \phi(\sigma \to \tau) &= \phi(\sigma) \to \phi(\tau), \\ \phi(F\sigma) &= \mathtt{Cantor} \to \phi(\sigma), \qquad \text{for } F \in \{\mathtt{H}, \mathtt{S}, \mathtt{P}, \mathtt{V}\}. \end{aligned}$$

Recall from Section 2.1 that $\mathtt{Cantor}$ is the type $(\mathtt{Nat} \to \mathtt{Bool})$. As in the previous section, the idea here is that the Cantor type plays the role of a type of schedulers. To run a non-deterministic program, one non-deterministically comes up with a scheduler, and then deterministically runs the program with respect to that scheduler, where the scheduler is used in order to decide which branches of the choice constructs are taken (think e.g. of false as left and of true as right). To run a probabilistic program, one first comes up with a scheduler, where the choice of scheduler is performed with uniform distribution over the Cantor space.

For non-determinism combined with probability, one needs two schedulers: one to perform the non-deterministic choices and the other to perform the probabilistic choices. The first one is chosen in an angelic or demonic way, but the second one with uniform distribution.

$$\phi(G\sigma) = \mathtt{Cantor} \times \mathtt{Cantor} \to \phi(\sigma), \qquad \text{for } G \in \{\mathtt{V}_{\mathtt{H}}, \mathtt{V}_{\mathtt{S}}, \mathtt{V}_{\mathtt{P}}\}.$$

The translation looks as if we were working with a probabilistic powertype followed by a non-deterministic powertype, and to some extent this is the case. However, the translations of the may- and must-probabilistic testing operators introduce a subtle interaction between the two [19,31,30].

Translation of terms.

This is also defined by induction. The translation of a syntactic variable is itself, with its type modified appropriately for powertypes. In order to be precise here, one needs some more-or-less evident syntactic bureaucracy which can be safely omitted for our purposes. For PCF$+$S$+$I constants, the translation is the identity. It is also the identity on all fixed-point combinators, including those of the non-deterministic and probabilistic types, with a suitable change of types for the latter. Moreover, we stipulate that the translation is a congruence:

$$\begin{aligned}\phi(MN) &= \phi(M)\phi(N),\\ \phi(\lambda x.M) &= \lambda\phi(x).\phi(M).\end{aligned}$$

This takes care of the deterministic fragment of the language, for which the translation is then the identity.

We first consider the powertypes that require only one scheduler. For the translation of the testing operators, recall that the type of may and must testing is

$$(\sigma \to \mathtt{S}) \to (F\sigma \to \mathtt{S}),$$

and hence their translations are to have type

$$(\phi(\sigma) \to \mathtt{S}) \to ((\mathtt{Cantor} \to \phi(\sigma)) \to \mathtt{S}).$$

Similarly, the translation of probabilistic testing is to have type

$$(\phi(\sigma) \to \mathtt{I}) \to ((\mathtt{Cantor} \to \phi(\sigma)) \to \mathtt{I}).$$

With this in mind, together with the discussion of Section 2.3, it should be possible to decode the following definition, where all quantifications and integrals are over the Cantor type (as introduced in Section 2.1), and we use the variable k for the type $\phi(F\sigma) = \mathtt{Cantor} \to \phi(\sigma)$:

$$\begin{aligned}\phi(\Diamond) &= \lambda u.\lambda k.\exists s.u(k(s)),\\ \phi(\Box &= \lambda u.\lambda k.\forall s.u(k(s)),\\ \phi(\bigcirc) &= \lambda u.\lambda k.\int u(k(s))\,\mathrm{d}s.\end{aligned}$$

Notice that in Section 2.3 we defined more general versions of quantification and integration. By means of this translation, we are reducing them to the special case of quantification and integration over the Cantor space.

Next we consider the compilation of non-deterministic and probabilistic choice operators, for $\star \in \{\otimes, \oplus\}$:

$$\phi(\star) = \lambda(k_0, k_1).\lambda s.\,\mathrm{if}\ \mathrm{hd}(s)\ \mathrm{then}\ k_0(\mathrm{tl}(s))\ \mathrm{else}\ k_1(\mathrm{tl}(s)).$$

Here hd and tl are the head and tail maps on sequences.

We now consider the translation of the monad structure. For the functor, we define

$$\phi(Ff) = \lambda k.\lambda s.f(k(s)),$$

For the unit, we define

$$\phi(\eta_F) = \lambda x.\lambda s.x.$$

For the multiplication, we consider PCF terms

$$\mathtt{evens}, \mathtt{odds} \colon \mathtt{Cantor} \to \mathtt{Cantor}$$

that take subsequences at even and odd indices, and define:

$$\phi(\mu_F) = \lambda K.\lambda s.K(\mathtt{evens}(s))(\mathtt{odds}(s)).$$

That is, we split the scheduler into two schedulers and pass each one to a different subcomputation. In the target language, the monad laws fail for the translations (both denotationally and operationally), but they will hold modulo the appropriate notion of testing. The strength is translated in a similar, mechanical, manner.

It remains to develop the translation of powertypes that require two schedulers. The translations of may- and must-probabilistic testing then have type

$$(\phi(\sigma) \to \mathtt{I}) \to ((\mathtt{Cantor} \to \mathtt{Cantor} \to \phi(\sigma)) \to \mathtt{I}),$$

and are given as follows, where we emphasize that the infima, suprema and integrals are over the Cantor type as above:

$$\phi(\Diamond) = \lambda u.\lambda k.\sup_s\left(\int u(k(s,t))\,\mathrm{d}t\right),$$
$$\phi(\Box) = \lambda u.\lambda k.\inf_s\left(\int u(k(s,t))\,\mathrm{d}t\right),$$

The translations of the non-deterministic choice operators consume tokens from the first scheduler:

$$\phi(\varovee) = \lambda(k_0,k_1).\lambda(s,t).\,\mathrm{if}\ \mathrm{hd}(s)\ \mathrm{then}\ k_0(\mathrm{tl}(s),t)\ \mathrm{else}\ k_1(\mathrm{tl}(s),t).$$

Those of probabilistic choice operators consume tokens from the second scheduler:

$$\phi(\oplus) = \lambda(k_0,k_1).\lambda(s,t).\,\mathrm{if}\ \mathrm{hd}(t)\ \mathrm{then}\ k_0(s,\mathrm{tl}(t))\ \mathrm{else}\ k_1(s,\mathrm{tl}(t)).$$

Finally, we consider the translation of the monad structure:

$$\phi(Gf) = \lambda k.\lambda(s,t).f(k(s,t)),$$
$$\phi(\eta_G) = \lambda x.\lambda(s,t).x,$$
$$\phi(\mu_G) = \lambda K.\lambda(s,t).K(\mathtt{evens}(s),\mathtt{evens}(t))(\mathtt{odds}(s),\mathtt{odds}(t)).$$

3.3 Ground evaluation

For an MMP term $M\colon \sigma$ with $\gamma \in \{\mathtt{Nat},\mathtt{Bool},\mathtt{S}\}$, define

$$M \Downarrow v \iff \phi(M) \Downarrow v.$$

One can indirectly to account for the case $\gamma = \mathtt{I}$ by reducing to the case $\mathtt{I} = \mathtt{S}$, considering the ground term $r < M\colon \mathtt{S}$ with r rational.

4 Denotational semantics

In this section we apply domain theory to prove the correctness of the semi-decision procedures for may, must and probabilistic testing developed in the previous section. As explained in the introduction, we don't prove the correctness of the may-probabilistic and must-probabilistic decision procedures. Hence the discussion is this section excludes the types containing the powertype constructors $\mathtt{V_H}, \mathtt{V_S}, \mathtt{V_P}$, except briefly in Section 4.5.

We work in the category of dcpos and continuous maps to give the semantics, but we need to eventually consider continuous dcpos to prove the correctness of the semi-decision procedures. For the sublanguage PCF, we consider its Scott model. We interpret the type $\mathtt{S}$ as the Sierpinski domain $\{\bot, \top\}$ and $\mathtt{I}$ as the unit interval $[0,1] \subseteq \mathbb{R}$ with its natural order, getting an interpretation of the sublanguage PCF$+$ $\mathtt{S}$ $+$ $\mathtt{I}$. We then interpret the powertypes as the Hoare, Smyth, Plotkin and probabilistic powerdomains [2,17].

4.1 Computational adequacy

To establish semi-decidability of may, must and probabilistic testing, we first prove *computational adequacy* of the model:

Lemma 4.1 *For any closed term M of ground type other than $\mathtt{I}$, and all syntactical values v,*

$$[\![M]\!] = [\![v]\!] \iff M \Downarrow v.$$

In particular, this will imply, for $M\colon \mathtt{I}$ closed and $r \in \mathbb{Q}$, that

$$r < [\![M]\!] \iff r < M \Downarrow \top$$

Because the model is already known to be computationally adequate for the deterministic sublanguage PCF $+$ $\mathtt{S}$ $+$ $\mathtt{I}$, we have the following purely denotational formulation of computational adequacy for the full language, which relieves us from the burden of syntactical manipulations and arguments, as the translation ϕ is compositional.

Lemma 4.2 *Computational adequacy holds if and only if $[\![M]\!] = [\![\phi(M)]\!]$ for every closed term M of ground type.*

To prove computational adequacy using this, we rely on the description of the powerdomains as free algebras for the (interpretations of) the choice operators $\otimes$ and $\oplus$. The axioms for the operations can be found in [17,23,2]. Then the semantics of the test operators $\Diamond, \Box, \bigcirc$ are uniquely determined by the conditions that $\Diamond(u)$, $\Box(u)$ and $\bigcirc(u)$ are algebra homomorphisms:

$$\begin{aligned}
\Diamond(u)(\eta(x)) = u(x), &\quad \Diamond(u)(X_0 \otimes X_1) = \Diamond(u)(X_0) \vee \Diamond(u)(X_1),\\
\Box(u)(\eta(x)) = u(x), &\quad \Box(u)(X_0 \otimes X_1) = \Box(u)(X_0) \wedge \Box(u)(X_1),\\
\bigcirc(u)(\eta(x)) = u(x), &\quad \bigcirc(u)(\nu_0 \oplus \nu_1) = \bigcirc(u)(\nu_0) \oplus \mathtt{V}(u)(\nu_1),
\end{aligned}$$

where $(\vee), (\wedge)\colon \mathtt{S} \times \mathtt{S} \to \mathtt{S}$ are the obvious operations, where $(\oplus)\colon \mathtt{I} \times I \to I$ is defined by $x \oplus y = (x+y)/2$, and where we are using the fact that $\mathtt{S}$ and $\mathtt{I}$ are algebras with these operations. This completes our proof sketch for computational adequacy.

4.2 Proof of semi-decidability of may, must and probabilistic testing

An alternative, also well-known [17,23,2], definition of the powerdomains is in terms of non-empty subsets and valuations. The elements of the Hoare power domain are the closed subsets, that of the Smyth powerdomain are the compact upper sets, that of the Plotkin powerdomain are the lenses, and that of the probabilistic powerdomain are the continuous valuations with total mass 1. The non-deterministic choice constants are interpreted as the union operation, and the probabilistic choice constant is interpreted as the convex-combination operation $(\nu_0, \nu_1) \mapsto \nu_0 \oplus \nu_1$. The logical constants are interpreted so that, as expected,

(i) $\Diamond(u)(X) = \top$ iff $u(x) = \top$ for some $x \in X$,
(ii) $\Box(u)(X) = \top$ iff $u(x) = \top$ for all $x \in X$,
(iii) $\bigcirc(u)(\nu) = \int_\nu u$.

More usually, $\Diamond$ and $\Box$ are seen as open-set constructors: The Scott topologies of the Hoare, Smyth, and Plotkin powerdomains have the following bases of open sets:

(i) Hoare: $\Diamond U$, where U ranges over open sets, and $X \in \Diamond U \iff U \cap X \neq \emptyset$.
(ii) Smyth: $\Diamond U$, where U ranges over open sets, and $X \in \Box U \iff X \subseteq U$.
(iii) Plotkin: Both $\Diamond U$ and $\Box U$, but restricted to lenses.

Hence, writing, $\mathcal{O}\, D$ for the lattice of open sets of a domain D, we have that $\Diamond$ and $\Box$ are functions:

(i) $\Diamond\colon\ \mathcal{O}\, D \to \mathcal{O}\, \mathtt{H}\, D$.
(ii) $\Box\colon\ \mathcal{O}\, D \to \mathcal{O}\, \mathtt{S}\, D$.
(iii) (a) $\Diamond\colon\ \mathcal{O}\, D \to \mathcal{O}\, \mathtt{P}\, D$.
(b) $\Box\colon\ \mathcal{O}\, D \to \mathcal{O}\, \mathtt{P}\, D$.

Now $\mathcal{O}\, D \cong (D \to \mathtt{S})$ via characteristic functions, and hence $\Diamond$ and $\Box$ can be seen as the functions discussed ealier. Moreover, it is clear that the condition $U \cap X \neq \emptyset$ amount to existential quantification over X and the condition $X \subseteq U$ amounts to universal quantification. We have to show that these functions are continuous, but this is straightforward.

So, it seems that these observations, together with computational adequacy, conclude the proof of semi-decidability of may, must and probabilistic testing. Unfortunately, this is not the case, because the concrete descriptions of the powerdomains given in this section coincide with the abstraction description given in the previous section only for certain classes of domains.

4.3 Obstacles

For the Hoare powerdomain, the abstract description given in Section 4.1 agrees with the concrete one given in Section 4.2 (see [23]). But for the Smyth, Plotkin and probabilistic powerdomains, the two descriptions agree only for continuous domains [23,17,2]. However, not all continuous domains are closed under function spaces and the Plotkin powerdomain, as is also well known, and there is no known cartesian closed category of continuous domains closed under the probabilistic powerdomain [18]. Hence we cannot expect all the types of our language to have continuous interpretations.

These and similar issues led the authors of [3] to propose an alternative form of domain theory, called topological domain theory, that overcomes these and other kinds of obstacles in semantics. We plan to investigate its use to the resolution of the problems explained here, but, for the moment, we establish partial, albeit very general, results using classical domain theory.

4.4 Partially overcoming the obstacles

We consider certain well behaved types for which the above problems don't arise. Inductively define collections of types S, R, C as follows, where γ ranges over ground types:

$$\begin{aligned} S &::= \gamma \mid S \times S \mid (C \to S) \mid \mathtt{H}\,C \mid \mathtt{S}\,C \\ R &::= S \mid R \times R \mid (R \to R) \mid \mathtt{P}\,R \\ C &::= R \mid C \times C \mid \mathtt{V}\,C \end{aligned}$$

By a *continuous Scott domain* we mean a bounded complete continuous dcpo.

Lemma 4.3

(i) *The interpretation of an S type is a continuous Scott domain.*

(ii) *The interpretation of an R type is an RSFP domain.*

(iii) *The interpretation of a C type is a continuous dcpo.*

Proof The ground types are continuous Scott domains. Continuous Scott domains are RSFP domains, and RSFP domains are continuous domains [2]. Plotkin showed that the category RSFP is cartesian closed, and closed under the Plotkin powerdomain [2]. Scott showed that the continuous Scott domains are densely injective spaces, and the densely injective spaces are an exponential ideal [10]. Schalk showed that the continuous dcpos are closed under the Hoare and Smyth powerdomains, and it is immediate that they are bounded complete [23]. Jones showed that the continuous domains are closed under the probabilistic powerdomain [17]. □

The following is an immediate consequence of this and the results of Section 4.1 and 4.2:

Theorem 4.4

(i) *For any type σ, may testing on terms of type $\mathtt{H}\,\sigma$ is semi-decidable.*

(ii) *For any continuous type σ, must testing on terms of type $\mathtt{S}\,\sigma$ is semi-decidable.*

(iii) *For any RSFP type σ, may and must testing on terms of type $\mathtt{P}\,\sigma$ are semi-decidable.*

(iv) *For any continuous type σ, probabilistic testing on terms of type $\mathtt{V}\,\sigma$ is semi-decidable.*

Notice that the smallest collection of types containing the ground types and closed under finite products, function spaces, and the Hoare, Smyth and Plotkin powertypes consists entirely of RSFP types. Hence if we hadn't included the probabilistic powertype in our language, we wouldn't have had any of the above difficulties, and may and must testing would be decidable for all types. What causes the restrictions is the presence of the probabilistic powertype. But still the restrictions are not severe in practice: for example, probabilistic computations on any PCF type of any order have semi-decidable probabilistic testing.

4.5 May-probabilistic and must-probabilistic testing

The powerdomain constructions developed in [30], with the same restriction of the language as in the previous section, ought to give a computationally adequate semantics, provided they are modified to account for probability. The theory developed in *loc. cit.* works with (i) geometrically convex, closed sets of valuations, or (ii) geometrically convex, compact upper sets of valuations, or (iii) geometrically convex lenses of valuations. Here we would need to restrict to valuations with total mass 1, and it is plausible that this can be done so as to get the free-algebra properties needed to establish computational adequacy and hence semi-decidability of may-probabilistic and must-probabilistic testing.

5 Concluding remarks

We have considered the full language as an executable logic for semi-decidable properties, with a sublanguage singled out as a programming language for nondeterministic and probabilistic computation. From a different point of view, the full language can also be regarded as a deterministic programming language for computation with closed sets, compact sets, lenses, and probability distributions, and we have, in fact, given a compiler that compositionally generates deterministic code for this language.

References

[1] S. Abramsky. Domain theory in logical form. *Ann. Pure Appl. Logic*, 51(1-2):1–77, 1991.

[2] S. Abramsky and A. Jung. Domain theory. In S. Abramsky, D.M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 3, pages 1–168. Oxford science publications, 1994.

[3] I. Battenfeld, M. Schröder, and A. Simpson. A convenient category of domains. *Electron. Notes Theor. Comput. Sci.*, 172:69–99, 2007.

[4] P. Di-Gianantonio. *A Functional Approach to Computability on Real Numbers.* PhD thesis, Università Degli Studi di Pisa, Dipartamento di Informatica, 1993.

[5] A. Edalat and M.H. Escardó. Integration in Real PCF. *Inform. and Comput.*, 160:128–166, 2000.

[6] M. Escardó. Exhaustible sets in higher-type computation. *Log. Methods Comput. Sci.*, 4(3):3:3, 37, 2008.

[7] M. Escardó, M. Hofmann, and T. Streicher. On the non-sequential nature of the interval-domain model of real-number computation. *Math. Structures Comput. Sci.*, 14(6):803–814, 2004.

[8] M.H. Escardó. PCF extended with real numbers. *Theoret. Comput. Sci.*, 162(1):79–115, 1996.

[9] M.H. Escardó. Synthetic topology of data types and classical spaces. *Electron. Notes Theor. Comput. Sci.*, 87:21–156, 2004.

[10] G. Gierz, K.H. Hofmann, K. Keimel, J.D. Lawson, M. Mislove, and D.S. Scott. *Continuous Lattices and Domains.* Cambridge University Press, 2003.

[11] R. Heckmann. An upper power domain construction in terms of strongly compact sets. *Lecture Notes in Comput. Sci.*, 598:272–293, 1992.

[12] R. Heckmann. Power domains and second order predicates. *Theoret. Comput. Sci.*, 111:59–88, 1993.

[13] R. Heckmann. Spaces of valuations. In *Papers on general topology and applications (Gorham, ME, 1995)*, volume 806 of *Ann. New York Acad. Sci.*, pages 174–200. New York Acad. Sci., New York, 1996.

[14] M. Hennessy and R. Milner. On observing nondeterminism and concurrency.

[15] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *J. Assoc. Comput. Mach.*, 32(1):137–161, 1985.

[16] M. C. B. Hennessy and E. A. Ashcroft. A mathematical semantics for a nondeterministic typed λ-calculus. *Theoret. Comput. Sci.*, 11(3):227–245, 1980.

[17] C. Jones. *Probabilistic Non-determinism.* PhD thesis, Laboratory for Foundations of Computer Science, University of Edinburgh, January 1990. .

[18] A. Jung and R. Tix. The troublesome probabilistic powerdomain. *Electron. Notes Theor. Comput. Sci.*, 1997.

[19] A. K. McIver and C. Morgan. Partial correctness for probabilistic demonic programs. *Theoret. Comput. Sci.*, 266(1-2):513–541, 2001.

[20] D. Normann. Exact real number computations relative to hereditarily total functionals. *Theoret. Comput. Sci.*, 284(2):437–453, 2002.

[21] C.-H. L. Ong. Non-determinism in a functional setting (extended abstract. In *In Proceedings 8th LICS*, pages 275–286, 1993.

[22] P.J. Potts, A. Edalat, and M.H. Escardó. Semantics of exact real arithmetic. In *Proceedings of the Twelveth Annual IEEE Symposium on Logic In Computer Science*, Warsaw, Polland, Jun 1997.

[23] A. Schalk. *Algebras for Generalized Power Constructions.* PhD thesis, Technische Hochschule Darmstadt, July 1993. ftp://ftp.cl.cam.ac.uk/papers/as213/diss.dvi.gz.

[24] M. Schröder and A. Simpson. Probabilistic observations and valuations (extended abstract). *Electron. Notes Theor. Comput. Sci.*, 155:605–615, 2006.

[25] M. Schröder and A. Simpson. Representing probability measures using probabilistic processes. *J. Complexity*, 22(6):768–782, 2006.

[26] A. Scriven. A functional algorithm for exact real integration with invariant measures. *Electron. Notes Theor. Comput. Sci.*, 218:337–353, 2008.

[27] K. Sieber. Call-by-value and nondeterminism. *Lecture Notes in Comput. Sci.*, 664:376–390, 1993.

[28] A. Simpson. Lazy functional algorithms for exact real functionals. *Lec. Not. Comput. Sci.*, 1450:323–342, 1998.

[29] M.B. Smyth. Power domains and predicate transformers: a topological view. volume 154 of *Lec. Not. Comput. Sci.*, pages 662–675, 1983.

[30] R. Tix, K. Keimel, and G. Plotkin. Semantic domains for combining probability and non-determinism. *Electron Notes in Theor. Comput. Sci.*, 129, 2005.

[31] D. Varacca and G. Winskel. Distributing probability over non-determinism. *Math. Structures Comput. Sci.*, 16(1):87–113, 2006.