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

Martín Escardó

School of Computer Science, Birmingham University, UK

MFPS 2009, Oxford, Friday 3rd April 2009

Main result

Theorem

May, must and probabilistic testing are semi-decidable, in a fairly general setting including higher-types.

Observations:

1. Must testing is perhaps surprising:
   It involves universal quantification over an infinite set.

2. The other two involve existential quantification and integration.

Ingredient 1

Can reduce to quantification and integration over the Cantor space.

This is the space of infinite sequences of binary digits.

Ingredient 2

Can algorithmically quantify and integrate over the Cantor space.

Quantification amounts exhaustive search in finite time.

Organization

1. A programming language for non-determinism and probability.
2. Logical types. For results of semi-decisions.
3. An executable program logic.
4. Operational semantics of the executable logic. Algorithms.
5. Denotational semantics of the executable logic. Correctness.

Brief discussion of effects

ML way.

1. All effects are possible at all types.
2. Come up with a monad that combines all effects.
3. The semantics is in the Kleisli category of that big monad.

Haskell way.

1. Explicitly define various monads as type constructors.
2. For each effect, or maybe for each combination of a set of effects.
3. Several monads are used in the same program.
4. The programmer decides which monads he wants for each sub-program.

We develop our results in the Haskell way.

A programming language for non-determinism and probability

Ground types:

$$\gamma := \texttt{Bool} \mid \texttt{Nat}$$

Powertype constructors:

$$F ::= \texttt{H} \mid \texttt{S} \mid \texttt{P} \mid \texttt{V}$$

1. Hoare, Smyth, Plotkin, Probabilistic.
2. May, must, may/must, on average.
3. Angelic, demonic, human.

Types:

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

Cartesian closed language.

Example

The type

$$\sigma \times \tau \to \mathtt{V}\,\tau$$

can be used to code labeled Markov processes with:

1. label space $A = \sigma$,
2. state space $S = \tau$, and
3. transition function $t : A \times S \to \mathtt{V}\,S$.

Terms

For the sublanguage over the PCF types

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

we take the PCF terms.

(Conditional, arithmetic, λ-calculus, fixed-point recursion.)

So no non-determinism or probability.

Non-deterministic choice constants

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

$$(\varovee^{\sigma})\colon F\sigma \times F\sigma \to F\sigma,$$

Idea. The term

$$\text{this} \varovee \text{that}$$

non-deterministically evaluates to this or that, angelically or demonically.

Probabilistic choice constants

For each type σ, we have an infix constant

$$(\oplus^\sigma)\colon\ \mathtt{V}\,\sigma \times \mathtt{V}\,\sigma \longrightarrow \mathtt{V}\,\sigma.$$

Idea. The term

$$\text{this} \oplus \text{that}$$

non-deterministically evaluates to this or that, with equal probability.

Monad syntax

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

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

Unit. For each type σ, we have a term

$$\eta_F^\sigma : \sigma \to F\sigma.$$

Multiplication. For each type σ, we have a constant

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

Strength. Left to the audience.

Remark

We could have worked with monads as Kleisli triples (as in Haskell).

This makes no difference, but our choice is presentationally more convenient.

Example

$\eta(\lambda x.0) \oslash \eta(\lambda x.1) \colon F(\sigma \to \mathtt{Nat})$
$\lambda x.\eta(0) \oslash \eta(1) \colon \sigma \to F\mathtt{Nat}$

Remark. If we apply the ML way to a call-by-name language, the terms

$$(\lambda x.0) \oslash (\lambda x.1)$$

and

$$\lambda x.(0 \oslash 1)$$

behave in the same way!

Example: randomly choose an infinite sequence of booleans with uniform distribution

$$\texttt{Cantor} = (\texttt{Nat} \to \texttt{Bool}).$$

$$\mathsf{cons}\colon \texttt{Bool} \to \texttt{Cantor} \to \texttt{Cantor}.$$

$$\mathsf{prefix}\colon \texttt{Bool} \to \texttt{V}\,\texttt{Cantor} \to \texttt{V}\,\texttt{Cantor}.$$

$$\mathsf{prefix}\, p = \texttt{V}(\mathsf{cons}\, p).$$

$$\mathsf{random}\colon\ \texttt{V}\,\texttt{Cantor}.$$

$$\mathsf{random} = (\mathsf{prefix}\,\mathsf{False}\,\mathsf{random}) \oplus (\mathsf{prefix}\,\mathsf{True}\,\mathsf{random}).$$

Possible-results operational semantics

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

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

Schedulers

Think of elements of the Cantor space as "schedulers".

Can decorate the operational semantics with schedulers,

$$M \Downarrow^s v,$$

so that

$M \Downarrow v$ iff there is some s with $M \Downarrow^s v$.

May and must convergence

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

M *may converge* $\iff$ there are s and v with $M \Downarrow^s v$.

Our approach is based on this idea.
But we implement it in a different way.

The Sierpinski type

Term formation rules for a Sierpinski type $\mathbb{S}$:

1. $\top : \mathbb{S}$ is a term.
2. If $M : \mathbb{S}$ and $N : \sigma$ are terms then $(\text{if } M \text{ then } N) : \sigma$ is a term.
3. If $M, N : \mathbb{S}$ are terms then so is $M \vee N : \mathbb{S}$.

The only value (or canonical form) of type $\mathbb{S}$ is $\top$.

$$\frac{M \Downarrow \top \quad N \Downarrow V}{\text{if } M \text{ then } N \Downarrow V} \qquad \frac{M \Downarrow \top}{M \vee N \Downarrow \top} \qquad \frac{N \Downarrow \top}{M \vee N \Downarrow \top}.$$

Computational adequacy of Scott model

If M is a closed term of ground type and v is a value then

$$[\![M]\!] = v \text{ iff } M \Downarrow v.$$

The vertical unit-interval type $\mathtt{I}$

1. Interpretated as the cpo $([0, 1], \leq)$.
2. Computations of terms $M : \mathtt{I}$ allow to semi-decide the condition $p < M$ with p rational.
3. But not the conditions $M = p$ or $M < p$ in general.

The vertical unit-interval type $\mathtt{I}$

1. Interpretated as the cpo $([0,1],\leq)$.
2. Computations of terms $M\colon \mathtt{I}$ allow to semi-decide the condition $p < M$ with p rational.
3. But not the conditions $M = p$ or $M < p$ in general.
4. Naturally regarded as a sub-dcpo of the unit-interval domain.
5. Think of $x \in \mathtt{I}$ as the interval $[x, 1]$.

The vertical unit-interval type $\mathtt{I}$

1. Interpretated as the cpo $([0,1], \leq)$.
2. Computations of terms $M \colon \mathtt{I}$ allow to semi-decide the condition $p < M$ with p rational.
3. But not the conditions $M = p$ or $M < p$ in general.
4. Naturally regarded as a sub-dcpo of the unit-interval domain.
5. Think of $x \in \mathtt{I}$ as the interval $[x, 1]$.
6. We take the primitive operations those for Real PCF, restricted to such intervals.
7. Arithmetic functions, $p < (-) \colon \mathtt{I} \to \mathtt{S}$ and pif.
8. Same operational rules.

The vertical unit-interval type $\mathtt{I}$

1. Interpretated as the cpo $([0,1], \leq)$.
2. Computations of terms $M : \mathtt{I}$ allow to semi-decide the condition $p < M$ with p rational.
3. But not the conditions $M = p$ or $M < p$ in general.
4. Naturally regarded as a sub-dcpo of the unit-interval domain.
5. Think of $x \in \mathtt{I}$ as the interval $[x, 1]$.
6. We take the primitive operations those for Real PCF, restricted to such intervals.
7. Arithmetic functions, $p < (-) : \mathtt{I} \to \mathtt{S}$ and pif.
8. Same operational rules.

Computational adequacy

$[\![M]\!] = x$ iff for every rational number p, we have that

$$p < x \iff (p < M) \Downarrow \top.$$

Definability results

There are programs:

1. $x \oplus y = (x + y)/2$, min, max, ...
2. $\exists, \forall : (\texttt{Cantor} \to \texttt{S}) \to \texttt{S}$.
3. $\int : (\texttt{Cantor} \to \texttt{I}) \to \texttt{I}$.

Based on papers:

1. PCF extended with real numbers, 1996.
2. Integration in Real PCF (with Edalat), 2000.
3. Synthetic topology of data types and classical spaces, 2004.
4. Exhaustible sets in higher-computation, 2008.

Some code

$$
\begin{aligned}
\exists(p) &= p(\bot) \vee (\exists(\lambda s.p(\mathsf{cons\,False}\ s)) \vee \exists(\lambda s.p(\mathsf{cons\,True}\ s)))\,, \\
\forall(p) &= p(\mathsf{if}\ \forall(\lambda s.p(\mathsf{cons\,False}\ s)) \wedge \forall(\lambda s.p(\mathsf{cons\,True}\ s))\ \mathsf{then}\ c), \\
\int f &= \max\left(f(\bot), \int \lambda s.f(\mathsf{cons\,False}\ s) \oplus \int \lambda s.f(\mathsf{cons\,True}\ s) \right).
\end{aligned}
$$

Executable program logic

We extend the programming language $\mathrm{PCF} + \mathtt{S} + \mathtt{I}$ with modal operators.

We get an executable program logic, MMP.

May and must testing

The $\mathbb{S}$-valued terms are characteristic functions of open sets:

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

$$\begin{aligned} \Diamond_F^\sigma &: \mathcal{O}\,\sigma \to \mathcal{O}\,F\sigma, && \text{for } F \in \{\mathtt{H}, \mathtt{P}\}, \\ \Box_F^\sigma &: \mathcal{O}\,\sigma \to \mathcal{O}\,F\sigma, && \text{for } F \in \{\mathtt{S}, \mathtt{P}\}. \end{aligned}$$

Idea. If $u\colon \mathcal{O}\,\sigma$ and $N\colon \mathtt{P}\,\sigma$,

$$\Diamond(u)(N) = \top \iff u(x) = \top \text{ for some outcome } x \text{ of a run of } N$$

and

$$\Box(u)(N) = \top \iff u(x) = \top \text{ for all outcomes } x \text{ of runs of } N.$$

Example

1. Want to semi-decide whether $n : F\mathtt{Nat}$ must be prime.
2. Write a semi-decision term $\mathtt{prime} : \mathtt{Nat} \to \mathtt{S}$.
3. Run, in the executable logic, the ground term $\Box\,\mathtt{prime}\; n$.

Of course, on can also semi-decide whether n must be non-prime.

However:

1. It doesn't follow that primeness of all outcomes of n is decidable.
2. If n has at least one non-divergent run, then both must tests diverge.

Example

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

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

and let $\mathtt{converge} : \mathtt{Nat} \to \mathtt{S}$ be a term such that

$$\mathtt{converge}(n) = \top \iff n \neq \bot.$$

Then we intend that

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

and that

$$\Box\,\mathtt{converge}(f(0)) = \bot$$

but

$$\Box\,\mathtt{converge}(\eta(0) \oplus \eta(1)) = \top.$$

Parallel-convergence is definable from may testing

Taking $\texttt{converge}\colon \mathbb{S} \to \mathbb{S}$ as the identity, the function

$$(\vee)\colon \mathbb{S} \times \mathbb{S} \to \mathbb{S}$$

is characterized by the equation

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

However, it cannot be defined from must testing.

Notice that $(p \wedge q) = \Box\,\texttt{converge}(\eta(p) \oslash \eta(q))$.

Probabilistic testing

Define a type of expectations:

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

We add a constant to the logic:

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

For a $\{0,1\}$-valued term $u \colon\ \mathcal{E}\,\sigma$ and a term $N \colon\ \mathtt{V}\,\sigma$,

$\bigcirc(u)(N) \colon \mathtt{I}$ is the probability that u holds for outcomes of runs of N.

Example

Recursively define a term $g\colon \mathtt{Nat} \to \mathtt{V}\,\mathtt{Nat}$ by

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

Then we intend that

$$\bigcirc\, \mathtt{converge}(g(0)) = 1$$

and

$$\bigcirc\, \mathtt{converge}_n(g(0)) = 2^{-n-1}$$

where $\mathtt{converge}_n\colon \mathtt{Nat} \to \mathtt{S}$ is a term such that

$$\mathtt{converge}_n(x) = \top \iff x = n.$$

Parallel-convergence is definable from probabilistic testing

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

Example: uniform distribution on $\mathtt{I}$

Define a term $\mathsf{prefix}\colon \mathtt{I} \to \mathtt{V}\,\mathtt{I} \to \mathtt{V}\,\mathtt{I}$ by

$$\mathsf{prefix}\, x = \mathtt{V}(\lambda y. x \oplus y),$$

Define $\mathsf{random}\colon\ \mathtt{V}\,\mathtt{I}$ by

$$\mathsf{random} = (\mathsf{prefix}\, 0\, \mathsf{random}) \oplus (\mathsf{prefix}\, 1\, \mathsf{random}).$$

For example $\bigcirc(\lambda x. p < x)\, \mathsf{random} = 1 - p$ for any $p \in \mathtt{I}$.

Existential quantification in MMP

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

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

Define

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

as

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

The idea is that this stands for

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

Universal quantification in MMP

Similarly, from the must testing operator

$$\Box\colon\ \mathcal{O}\,\sigma \to \mathcal{O}\,\mathtt{S}\,\sigma,$$

we get a term

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

The Ploktin powertype has both quantifiers.

Integration in MMP

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 $u\colon \sigma \to \mathtt{I}$.

Example

Let $(\sigma, f_1, \ldots, f_n, p_1, \ldots, p_n)$ be an IFS with probabilities.

Its invariant measure $\nu\colon \mathtt{V}\,\sigma$ can be defined as

$$\nu = \text{weighted-choice}(p_1, \ldots, p_n)(\mathtt{V}(f_1)(\nu), \ldots, \mathtt{V}(f_n)(\nu)),$$

Scriven (MFPS 2008) developed a PCF program for computing integrals of functions $u\colon \sigma \to \mathtt{I}$ with respect to the invariant measure.

Here we get the alternative algorithm $\int_\nu u = \bigcirc(u)(\nu)$ in the program logic MMP instead.

Operational semantics of the executable logic MMP

1. By compositional compilation into its deterministic sub-language $\mathrm{PCF} + \mathtt{S} + \mathtt{I}$.
2. The translation is the identity on $\mathrm{PCF} + \mathtt{S} + \mathtt{I}$ terms.
3. Reduce may, must and probabilistic testing in MMP to quantification and integration in $\mathrm{PCF} + \mathtt{S} + \mathtt{I}$.

Translation of types

This is defined by induction:

$$
\begin{array}{rcl}
\phi(\gamma) & = & \gamma, \\
\phi(\sigma \times \tau) & = & \phi(\sigma) \times \phi(\tau), \\
\phi(\sigma \to \tau) & = & \phi(\sigma) \to \phi(\tau), \\
\phi(F\sigma) & = & \texttt{Cantor} \to \phi(\sigma).
\end{array}
$$

Recall that $\texttt{Cantor} = (\texttt{Nat} \to \texttt{Bool})$.

(Hence the translation is the identity on $\mathrm{PCF} + \texttt{S} + \texttt{Ic}$ types.)

Translation of terms

$$\begin{aligned}\phi(x) &= x \\ \phi(\lambda x.M) &= \lambda x.\phi(M) \\ \phi(MN) &= \phi(M)\phi(N) \\ \phi(\mathrm{PCF} + \mathtt{S} + \mathtt{I}\ \text{constant}) &= \text{itself} \\ \phi(\text{any fixed-point combinator}) &= \text{itself}\end{aligned}$$

(Hence the translation is the identity on $\mathrm{PCF} + \mathtt{S} + \mathtt{I}$ terms.)

Translation of choice operators

For $\star \in \{\varovee, \oplus\}$, we define

$$\phi(\star) \quad = \quad \lambda(k_0, k_1).\lambda s.\, \mathsf{if\ head}(s)\ \mathsf{then}\ k_0(\mathsf{tail}(s))\ \mathsf{else}\ k_1(\mathsf{tail}(s)).$$

Here k_0 and k_1 range over $\phi(F\sigma) = \mathtt{Cantor} \rightarrow \phi(\sigma)$.

Translation of the modal operators: may

Typing:

$$\begin{array}{rcl} \Diamond & : & (\sigma \to \mathtt{S}) \to (F\sigma \to \mathtt{S}), \\ \phi(\Diamond) & : & (\phi(\sigma) \to \mathtt{S}) \to ((\mathtt{Cantor} \to \phi(\sigma)) \to \mathtt{S}). \end{array}$$

We define

$$\phi(\Diamond) \quad = \quad \lambda u.\lambda k.\exists s.u(k(s)).$$

Here

$$\underbrace{(\phi(\sigma) \to \mathtt{S})}_{u} \to (\underbrace{(\underbrace{\mathtt{Cantor}}_{s} \to \phi(\sigma))}_{k} \to \mathtt{S}).$$

The quantification is over the Cantor space.

Translation of the modal operators: must

Typing:

$$\begin{array}{rcl} \Box & : & (\sigma \to \mathtt{S}) \to (F\sigma \to \mathtt{S}), \\ \phi(\Box) & : & (\phi(\sigma) \to \mathtt{S}) \to ((\mathtt{Cantor} \to \phi(\sigma)) \to \mathtt{S}). \end{array}$$

We define

$$\phi(\Box) \quad = \quad \lambda u.\lambda k.\forall s.u(k(s)).$$

Here

$$\underbrace{(\phi(\sigma) \to \mathtt{S})}_{u} \to \underbrace{((\underbrace{\mathtt{Cantor}}_{s} \to \phi(\sigma))}_{k} \to \mathtt{S}).$$

The quantification is over the Cantor space.

Translation of the modal operators: probabilistic

Typing:

$$\begin{array}{rcl} \bigcirc & : & (\sigma \to \mathtt{I}) \to (\mathtt{V}\,\sigma \to \mathtt{I}), \\ \phi(\bigcirc) & : & (\phi(\sigma) \to \mathtt{I}) \to ((\mathtt{Cantor} \to \phi(\sigma)) \to \mathtt{I}). \end{array}$$

We define

$$\phi(\bigcirc) \quad = \quad \lambda u.\lambda k. \int u(k(s))\underset{\cdot}{s}.$$

Here

$$\underbrace{(\phi(\sigma) \to \mathtt{I})}_{u} \to \underbrace{((\underbrace{\mathtt{Cantor}}_{s} \to \phi(\sigma))}_{k} \to \mathtt{I}).$$

The integration is over the Cantor space.

Translation of the monad constructions: functor

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

Translation of the monad constructions: unit

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

Translation of the monad constructions: multiplication

We consider PCF terms

$$\texttt{evens}, \texttt{odds} \colon \texttt{Cantor} \longrightarrow \texttt{Cantor}$$

that take subsequences at even and odd indices.

Define:

$$\phi(\mu_F) = \lambda k.\lambda s.k(\texttt{evens}(s))(\texttt{odds}(s)).$$

Translation of the monad constructions: strength

Left as an exercise to the audience.

Ground evaluation

For MMP terms $M : \sigma$ with $\gamma \neq \mathtt{I}$ ground, define

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

Denotational semantics of the executable logic

As predicted by the audience.

Types:

1. Hoare powertype $\mapsto$ Hoare powerdomain.
2. Smyth powertype $\mapsto$ Smyth powerdomain.
3. Plotkin powertype $\mapsto$ Plotkin powerdomain.
4. Probabilistic powertype $\mapsto$ probabilistic powerdomain.

Terms:

1. These are monads, which have the binary choice operators we need.
2. The modal operators correspond to the usual descriptions of the open sets of the powerdomains.
3. The probabilistic operator is interpreted by integration.

Computational adequacy

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

Lemma

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

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

In particular, for $M : \mathtt{I}$ closed and $r \in \mathbb{Q}$,

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

Computational adequacy: technical aspects

Because the model is already known to be computationally adequate for the deterministic sub-language $\mathrm{PCF} + \mathtt{S} + \mathtt{I}$:

Lemma

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

Correctness of the semi-decision procedures

Follows directly from computational adequacy.

BUT

Trouble

1. For the proof of computational adequacy, we rely on the abstract description of the powerdomains by free algebras.

2. For the proof of correctness, we rely on the concrete descriptions of the powerdomains:
   1. Set of closed sets (Hoare).
   2. Set of compact sets (Smyth).
   3. Lenses (Plotkin).
   4. Continuous valuations with total mass 1 (Probabilistic).

3. The abstract and concrete descriptions agree only for special kinds of domains.

Partial results

Theorem

1. *For any type* σ*, may testing on terms of type* $\mathtt{H}\,\sigma$ *is semi-decidable.*
2. *For any continuous type* σ*, must testing on terms of type* $\mathtt{S}\,\sigma$ *is semi-decidable.*
3. *For any RSFP type* σ*, may and must testing on terms of type* $\mathtt{P}\,\sigma$ *are semi-decidable.*
4. *For any continuous type* σ*, probabilistic testing on terms of type* $\mathtt{V}\,\sigma$ *is semi-decidable.*

Remark

1. If we hadn't included the probabilistic powertype in our language, we wouldn't have had any of the above difficulties.
2. May and must testing would be semi-decidable for all types.
3. What causes the restrictions is the presence of the probabilistic powertype.
4. But still the restrictions are not severe in practice.
5. For example, probabilistic computations on any PCF type of any order have semi-decidable probabilistic testing.

Syntactical description of some types we account for

Define:

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

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

Proposition

1. *The interpretation of an S type is a continuous Scott domain.*
2. *The interpretation of an R type is an RSFP domain.*
3. *The interpretation of a C type is a continuous dcpo.*

End and summary

Theorem

1. *For any type σ, may testing on terms of type $\mathtt{H}\,\sigma$ is semi-decidable.*
2. *For any continuous type σ, must testing on terms of type $\mathtt{S}\,\sigma$ is semi-decidable.*
3. *For any RSFP type σ, may and must testing on terms of type $\mathtt{P}\,\sigma$ are semi-decidable.*
4. *For any continuous type σ, probabilistic testing on terms of type $\mathtt{V}\,\sigma$ is semi-decidable.*

This applies to a large class of (syntactically described) types.