About Typed Algebraic Lambda-calculi

Benoît Valiron
INRIA Saclay/LIX
Palaiseau, France
valiron@lix.polytechnique.fr

Abstract

Arrighi and Dowek (2008) introduce an untyped lambda-calculus together with a structure of module over a ring of scalars. They show that diverging terms may break confluence if a naive rewriting system is used. In this paper, we explore the semantics of a typed version of their language. First, we describe a simply-typed language with no fixpoints. We show that in this restricted setting, no diverging term exists, and the language admits a simple denotation using the monad coming from the adjunction between the category of sets and the category of modules. Then we add the possibility of fixpoints, and we examine the differences with the previous language. In particular, we show how several notions of zeros and infinities naturally arise in the presence of fixpoints and we sketch a denotational description for catching these notions.

Keywords*: typed lambda-calculus, module over ring and semi-ring, fixpoints, semantics, computational model.*

1. Introduction

Notions of lambda-calculus with vectorial structures have at least two distinct origins. The first one is the calculus of Vaux [10], building up upon the work of Ehrhard and Regnier [4]. The goal here is to capture a notion of differentiation within lambda-calculus. Another origin of algebraic lambda-calculi is the work of Arrighi and Dowek [2], defining a lambda-calculus in the style of van Tonder [9]. Here, the goal is to describe a lambda-calculus with superposition of states in a quantum computational style.

Both [2] and [10] acknowledge the fact that for an untyped lambda-calculus, a naive rewriting system renders the language inconsistent, as any term can be made equal to the zero of the vectorial space of terms. However, coming from different backgrounds, they provide different solutions to the problem. In [2], the rewriting system is restrained in order to avoid unwanted equalities of terms. In [10], the rewriting system is untouched, but the scalars over which the vectorial structures in built are made into a semiring. This also brings a consistent system.

In this paper, we describe a semantics for an algebraic lambda-calculus. Starting with an untyped lambda-calculus and a naive rewrite system, we show where inconsistencies occur. Then we construct a simply-typed version of the untyped language, together with an equational description. In this restricted setting, the rewrite system is sound, and we describe a denotational semantics using a computational model a la Moggi [8]. We then re-read the problems that occurred in the untyped world, and find a simple solution for making the system sound again in the presence of diverging terms. We demonstrate that several notions of convergence arise when considering algebraic lambda-calculus, and sketch denotational models accounting for divergence.

1.1. An untyped calculus

Suppose the existence of a ring $(\mathcal{A}, +, 0, \times, 1)$. Elements of $\mathcal{A}$ are called *scalars*. We define a language by the following term grammar:

$$\begin{aligned} s, t &::= x \mid \lambda x.s \mid st \mid s + t \mid \alpha \cdot s \mid \mathbf{0} \mid [\ s\ ] \mid \{\ s\ \}, \\ u, v &::= x \mid \lambda x.u \mid uv \mid [\ s\ ], \end{aligned}$$

where α ranges over $\mathcal{A}$, and where x ranges over a set of variables. Terms of the form s, t are called *composite* and terms of the form u, v are called *pure terms*. We define variable substitution as usual and consider term up to α-equivalence.

1.2. A naive reduction system

A very naive reduction is to make the system of term into a monad over a ring $\mathcal{A}$, with the term $\mathbf{0}$ as unit of the addition. More precisely, a term s reduces to a term t, written $s \to t$, if there exists a term s' and a term t' respectively equivalent modulo congruence, associativity and commutativity of $+$ to s and t such that the relation $s' \to t'$ is derived

from the rules of Table 1. Although we do not describe formally the system here (a complete development can be found in Section 2.1), the reduction should be straightforward enough for the remainder of the discussion.

In particular, the addition is commutative and associative, the terms $t - t$ and $0 \cdot t$ equate the term $\mathbf{0}$. All terms constructs are linear except $[\ -\ ]$, which "lift" a composite term into a pure term. One can unlift it using $\{\ -\ \}$, and retrieve the composite term. Finally, the system is call-by-value: the beta-reduction $(\lambda x.s)v$ reduces to $s[x \leftarrow v]$ only if v is a pure term.

For example, the term $(\lambda f x.(fx)x)(y + z)$ reduces to

$$(\lambda x f.(fx)x)y + (\lambda x f.(fx)x)z,$$

and then to $\lambda f.(fy)y + \lambda f.(fz)z$. On the contrary, the composite term $(\lambda x f.(f\{\ x\ \})\{\ x\ \})[\ y + z\ ]$ reduces to

$$\lambda f.(f\{\ [\ y + z\ ]\ \})\{\ [\ y + z\ ]\ \},$$

and then to

$$\lambda f.(fy)y + \lambda f.(fz)y + \lambda f.(fy)z + \lambda f.(fz)z.$$

It is possible to build the same term constructs as with the regular untyped lambda-calculus [3]. For example, the product $\langle\ s, t\ \rangle$ of two terms s and t can be encoded as $\lambda f.(fs)t$, the first projection $\pi_1(s)$ of a pair s as the term $s\,(\lambda xy.x)$ and the second projection $\pi_2(s)$ as $s\,(\lambda xy.y)$. Note that, since each usual lambda-terms constructs are linear in each variable, the new term constructs $\langle\ -, -\ \rangle$, π_1, π_2 are also linear in each variable. In particular,

$$\begin{aligned}\langle\ s + s', t + t'\ \rangle &= \lambda f.(f(s + s'))(t + t') = \\ &\lambda f.((fs)t + (fs')t + (fs)t' + (fs')t') = \\ &\lambda f.(fs)t + \lambda f.(fs')t + \lambda f.(fs)t' + \lambda f.(fs')t' = \\ &\langle\ s, t\ \rangle + \langle\ s', t\ \rangle + \langle\ s, t'\ \rangle + \langle\ s', t'\ \rangle.\end{aligned}$$

1.3. Breaking consistency

Although the set of requirements look reasonable, the equational system is not sound. Indeed, given any term b, one can construct the term

$$Y_b = \{\ (\lambda x.[\ \{\ xx\ \} + b\ ])(\lambda x.[\ \{\ xx\ \} + b\ ])\ \}.$$

This term verify the reduction

$$Y_b \to Y_b + b. \tag{1}$$

This creates a problem of consistency, as enlightened in the following sequence of equalities:

$$\mathbf{0} = Y_b - Y_b = (Y_b + b) - Y_b = b + (Y_b - Y_b) = b. \tag{2}$$

This successfully shows that any term can be equated to $\mathbf{0}$, making the system inconsistent (this argument is issued from [2]).

2. A simply-typed lambda-calculus

The problem occurring in Section 1.3 is due to the possibility of constructing diverging terms. In this section we study a simply-typed, algebraic lambda-calculus. Equipped with a naive reduction system, it verifies strong normalization. This allows us in Section 3 to analyse carefully the pitfalls occurring when adding divergence.

Definition 2.1. We suppose the existence of a ring $\mathcal{A}$, containing a multiplication and an addition. A simply-typed algebraic lambda-calculus is constructed as follows. Types are of the form

$$A, B \ ::= \ \iota \mid A \to B \mid A \times B \mid \top \mid MA,$$

where ι ranges over a set of type constants. Terms come in two flavors:

$$\begin{aligned} s, t \ &::= \ x \mid \lambda x.s \mid st \mid \langle\ s, t\ \rangle \mid \pi_1(s) \mid \pi_2(s) \mid\ * \ \mid \\ &\qquad s + t \mid \alpha \cdot s \mid \mathbf{0} \mid [\ s\ ] \mid \{\ s\ \}, \\ u, v \ &::= \ x \mid \lambda x.u \mid uv \mid \langle\ u, v\ \rangle \mid \pi_1(u) \mid \pi_2(u) \mid\ * \ \mid \\ &\qquad [\ s\ ], \end{aligned}$$

where $\alpha \in \mathcal{A}$. Terms of the form s, t are called *composite* and terms of the form u, v are called *pure terms*. The term $[\ s\ ]$ is the closure of a computations: such a term is not linear and can be duplicated "as it". The term construct $\{\ -\ \}$ breaks such a closure and "run" the computation.

We define the notions of typing context Δ and of typing derivation $\Delta \vdash s : A$ in the usual way. Terms are considered up to α-equivalence, and valid typing derivations are built using the rules of Table 2.

2.1. Operational semantics

The type system is valid with respect to the reduction system described in Table 1, modulo the addition of rules for the added term constructs concerning the product. In the following, we use the terminology of [2].

Definition 2.2. Given any relation R on terms, we say that it is a *congruent relation* if for all pairs $(s, s'), (t, t') \in R$, the pairs $(\lambda x.s, \lambda x.s')$, $(st, s't')$, $(s + t, s' + t')$, $(\langle\ s, t\ \rangle, \langle\ s', t'\ \rangle)$, $(\pi_2 s, \pi_2 s')$, $(\pi_1 s, \pi_1 s')$, $(\alpha \cdot s, \alpha \cdot s')$, $([\ s\ ], [\ s'\ ])$ and $(\{\ s\ \}, \{\ s'\ \})$ are in R.

Definition 2.3. We define $\simeq_{AC}$ to be the smallest congruent, equivalent relation on terms satisfying $s + t \simeq_{AC} t + s$ and $r + (s + t) \simeq_{AC} (r + s) + t$. We say that a relation R is *consistent with* $\simeq_{AC}$ if

$$s \simeq_{AC} s' R t' \simeq_{AC} t \quad \Rightarrow \quad sRt.$$

Group E

$$\begin{array}{cccccccccc} & \alpha\cdot\mathbf{0} & \to & \mathbf{0} & & \mathbf{0}+s & \to & s & & \alpha\cdot(\beta\cdot s) & \to & (\alpha\beta)\cdot s \\ (*) & 0\cdot s & \to & \mathbf{0} & & 1\cdot s & \to & s & & \alpha\cdot(s+t) & \to & \alpha\cdot s+\alpha\cdot t \end{array}$$

Group F

$$\begin{array}{rcrcl} \alpha\cdot s & + & \beta\cdot s & \to & (\alpha+\beta)\cdot s \\ \alpha\cdot s & + & s & \to & (\alpha+1)\cdot s \\ s & + & s & \to & (1+1)\cdot s \end{array}$$

Group A

$$\begin{array}{ccc} (s+t)r\to sr+tr & (\alpha\cdot s)r\to\alpha\cdot(sr) & \mathbf{0}r\to\mathbf{0} \\ r(s+t)\to rs+rt & r(\alpha\cdot s)\to\alpha\cdot(rs) & r\mathbf{0}\to\mathbf{0} \\ \lambda x.(s+t)\to\lambda x.s+\lambda x.t & \lambda x.(\alpha\cdot s)\to\alpha\cdot\lambda x.s & \lambda x.\mathbf{0}\to\mathbf{0} \\ \{\ s+t\ \}\to\{\ s\ \}+\{\ t\ \} & \{\ \alpha\cdot s\ \}\to\alpha\cdot\{\ s\ \} & \{\ \mathbf{0}\ \}\to\mathbf{0} \end{array}$$

Group B

$$(\lambda x.s)v\to s[x\leftarrow v] \qquad \{\ [\ s\ ]\ \}\to s$$

Table 1. Reduction system L.

$$\begin{array}{rcl} \emptyset & \Rightarrow & \Delta, x:A\vdash x^A:A, \\ \Delta, x:A\vdash s:B & \Rightarrow & \Delta\vdash\lambda x^A.s:A\to B, \\ \Delta\vdash s:A\times B & \Rightarrow & \Delta\vdash\pi_1(s):A, \\ \Delta\vdash s:A\times B & \Rightarrow & \Delta\vdash\pi_2(s):B, \\ \Delta\vdash s:A & \Rightarrow & \Delta\vdash\alpha\cdot s:A, \\ \Delta\vdash s:MA & \Rightarrow & \Delta\vdash\{\ s\ \}:A, \\ \Delta\vdash s:A & \Rightarrow & \Delta\vdash[\ s\ ]:MA. \end{array} \qquad \left.\begin{array}{l}\Delta\vdash s:A\to B\\ \Delta\vdash t:A\end{array}\right\}\ \Rightarrow\ \Delta\vdash st:B,$$

$$\left.\begin{array}{l}\Delta\vdash s:A\\ \Delta\vdash t:B\end{array}\right\}\ \Rightarrow\ \Delta\vdash\langle\ s,t\ \rangle:A\times B,$$

$$\left.\begin{array}{l}\Delta\vdash s:A\\ \Delta\vdash t:A\end{array}\right\}\ \Rightarrow\ \Delta\vdash s+t:A,$$

Table 2. Typing rules.

Definition 2.4. We define the reduction systems E, F, A and B of terms as the smallest congruent relations consistent with $\simeq_{AC}$, satisfying the rules in Table 1 where B is augmented with the rules

$$\pi_1\langle\ u,v\ \rangle\to u, \qquad \pi_2\langle\ u,v\ \rangle\to v.$$

In all the given rules, the terms r, s, t are assumed to be neutral and the terms u, v are assumed to be pure. We write L for the relation $A\cup B\cup E\cup F$.

Convention 2.5. If R is a relation, we write $s\to_R t$ in place of $(s,t)\in R$. We simply right $\to$ in place of $\to_L$, and if $s\to t$, we say that s reduces to t. We denote with $\to_R^*$ the reflexive, transitive closure of $\to_R$.

Lemma 2.6 (Substitution)**.** *Let $\Delta\vdash v:A$ and $\Delta, x:A\vdash s:B$ be two valid typing derivations, where v is a pure term. Then $\Delta\vdash s[x\leftarrow v]:B$ is a valid typing derivation.*

Proof. Proof by structural induction on the typing derivation of $\Delta, x:A\vdash s:B$. □

Lemma 2.7 (Subject reduction)**.** *Let $\Delta\vdash s:A$ be a valid typing judgement such that $s\to t$. Then $\Delta\vdash t:A$ is also valid.*

Proof. Proof by structural induction on the term s and inspection of the reduction rules, using Lemma 2.6 for the first rule of group B. □

Definition 2.8. A *neutral term* is a term built from the following grammar:

$$s,t\ ::=\ \mathbf{0}\mid u\mid\alpha\cdot u\mid s+t.$$

where u is a pure term.

A *normal* term is a term s such that there does not exists a term t such that $s\to t$. A *rewrite sequence* is a sequence $(s_i)_i$ of terms such that for all i, either $s_i\to s_{i+1}$ or s_i is normal and i is the last index of the sequence.

Lemma 2.9. *A normal term is neutral.* □

Theorem 2.10 (Safety)**.** *Suppose that $\vdash s : A$ is a valid typing judgement. Then either $s \to t$, and then $\vdash t : A$, or s is a term in neutral form.*

Proof. Proof by case distinction on the structure of s, using Lemma 2.7. □

As for the simply-typed lambda-calculus, the reduction system is normalizable. The proof uses the fact that the rewrite system consists of two parts: the rules of groups E,F,A and the rules of group B.

Lemma 2.11. *Let s be any term. There exists a natural number n such that any rewrite sequence $(s_i)_i$ in $E \cup F \cup A$ consists of at most n elements.* □

Theorem 2.12 (Normalization)**.** *Let $\vdash s : A$ a valid typing judgement. There exists an index n_s such that any rewrite sequence $(s_i)_i$ starting ar $s_0 = s$ is finite and of at most n_s elements.*

Proof. The proof uses reducibility candidates. The set RED_A of reducibility candidates of type A is constructed in the usual way. The typing judgement $\Delta \vdash s : \top$ is in $\mathrm{RED}_\top$ if and only if s is strongly normalizable. For s of type $A \times B$, $s \in \mathrm{RED}_{A\times B}$ if and only if $\pi_1(s) \in \mathrm{RED}_A$ and $\pi_2(s) \in \mathrm{RED}_B$. A term $\Delta \vdash s : MA$ is in RED_{MA} if and only if $\{\ s\ \} \in \mathrm{RED}_A$, and $\Delta \vdash s : A \to B$ is in $\mathrm{RED}_{A\to B}$ if for all term $\Delta \vdash t : A$, the term st is in RED_B. Then the proof follows the one provided in [5], using Lemma 2.11 to handle the cases where addition and multiplication by scalar are involved. □

2.2. Equational theory

Together with its type system, the simply-typed lambda-calculus shares some strong similarities with Moggi's computational lambda-calculus [8]. Indeed, there are notions of *value* and *computation*, respectively described by the notion of pure and the notion of composite term.

Definition 2.13. We define an equivalence relation $\simeq_{ax}$ on terms as the smallest equivalence relation closed under α-equivalence, the usual congruence rules, the associativity and commutativity of $+$, and the equations of Table 3.

Two valid typing judgements $\Delta \vdash s, t : A$ are said to be *axiomatically equivalent*, written $\Delta \vdash s \simeq_{ax} t : A$, if $s \simeq_{ax} t$ is provable.

Definition 2.14. We define a $\mathcal{A}$-enriched computational category to be

- a cartesian closed category $(\mathcal{C}, \times, \Rightarrow, \mathbf{1})$,
- together with a strong monad (M, η, μ, t),
- such that the Kleisli category is enriched over the category of $\mathcal{A}$-modules.

We refer the reader to the literature for the definitions (e.g. [6, 7, 8]).

We claim that the simply-typed algebraic lambda-calculus is an internal language for $\mathcal{A}$-enriched computational categories.

Definition 2.15. We define the category $\mathcal{C}_l$ as follows: objects are types and morphisms $A \to B$ are axiomatic equivalent classes of typing judgements $x : A \vdash v : B$ (where v is a pure term).

Theorem 2.16. *The category $\mathcal{C}_l$ is a $\mathcal{A}$-enriched computational category. The cartesian closed structure is given by the classical subset of the language: The unit is $\top$, the product of A and B is $A \times B$. The structural maps are*

$$\begin{aligned} \pi_1 &= x : A \times B \vdash \pi_1(x) : A, \\ \pi_2 &= x : A \times B \vdash \pi_2(x) : B, \end{aligned}$$

if f and g are respectively $x : A \vdash u : A$ and $x : A \vdash v : B$, then $\langle\ f, g\ \rangle$ is the map $x : A \vdash \langle\ u, v\ \rangle : A \times B$. The unique map sending A to $\top$ is $x : A \vdash * : \top$. The natural map from $\mathcal{C}_l(A \times B, C)$ to $\mathcal{C}_l(A, B \to C)$ is sends the morphism $x : A \times B \vdash u : C$ to $y : A \vdash \lambda z.((\lambda x.u)\langle\ y, z\ \rangle) : B \to C$.

The monad M sends A to MA and $x : A \vdash u : B$ to $y : MA \vdash [\ (\lambda x.u)\{\ y\ \}\ ] : MB$, and the three required morphisms are

$$\begin{aligned} \eta_A &= x : A \vdash [\ x\ ] : MA, \\ \mu_A &= x : MMA \vdash [\ \{\ \{\ x\ \}\ \}\ ] : MA, \\ t_{A,B} &= x : MA \times B \vdash [\ \langle\ \{\pi_1(x)\}, \pi_2(x)\ \rangle\ ] : M(A \times B). \end{aligned}$$

The enrichment of $\mathcal{C}_l(A, MB)$ is given by the module structure of the term algebra. Consider the two maps $f = (x : A \vdash u : MB)$ and $g = (x : A \vdash v : MB)$. We define

$$\begin{aligned} 0 &= (x : A \vdash [\ \mathbf{0}\ ] : MB) \\ f + g &= (x : A \vdash [\ \{\ u\ \} + \{\ v\ \}\ ] : MB), \\ \alpha \cdot f &= (x : A \vdash [\ \alpha \cdot \{\ u\ \}\ ] : MB). \end{aligned}$$

Proof. Proof by inspection of the required commutative diagrams. □

Definition 2.17. Consider a $\mathcal{A}$-enriched computational category $\mathcal{C}$. We define the interpretation of a composite typing judgement $[\![\, x_1 : A_1, \ldots, x_n : A_n \vdash t : B \,]\!]^c$ as a morphism in $\mathcal{C}_M$ and the interpretation of a pure typing judgement $[\![\, x_1 : A_1, \ldots, x_n : A_n \vdash t : B \,]\!]^v$ as a morphism in $\mathcal{C}$. They are defined inductively, together with their usual meanings: The interpretation of $\pi_2(s)$ and $\pi_1(s)$ are the projections and $\langle\ s, t\ \rangle$ the product of the cartesian structure; the lambda-abstraction corresponds to the internal homomorphism of $\mathcal{C}$; the term $[\ s\ ]$ makes a computation $A \to MB$ into a value using the unit of the monad and $\{\ s\ \}$ produces

$$
\begin{array}{rcl@{\qquad}rcl}
(*)\quad 0\cdot u & \simeq_{ax} & \mathbf{0} & u+\mathbf{0} & \simeq_{ax} & u \\
1\cdot u & \simeq_{ax} & u & \alpha\cdot u+\alpha\cdot v & \simeq_{ax} & \alpha\cdot(u+v) \\
\alpha\cdot u+\beta\cdot u & \simeq_{ax} & (\alpha+\beta)\cdot u & (r+s)+t & \simeq_{ax} & r+(s+t) \\
\alpha\cdot(\beta\cdot s) & \simeq_{ax} & (\alpha\beta)\cdot s & s+t & \simeq_{ax} & t+s \\
\\
\langle\, r+\alpha\cdot s,t\,\rangle & \simeq_{ax} & \langle\, r,t\,\rangle+\alpha\cdot\langle\, s,t\,\rangle & \pi_1(s+\alpha\cdot t) & \simeq_{ax} & \pi_1(s)+\alpha\cdot\pi_1(t) \\
\langle\, r,s+\alpha\cdot t\,\rangle & \simeq_{ax} & \langle\, r,s\,\rangle+\alpha\cdot\langle\, r,t\,\rangle & \pi_2(s+\alpha\cdot t) & \simeq_{ax} & \pi_2(s)+\alpha\cdot\pi_2(t) \\
\langle\, \mathbf{0},t\,\rangle & \simeq_{ax} & \mathbf{0} & \pi_1(\mathbf{0}) & \simeq_{ax} & \mathbf{0} \\
\langle\, t,\mathbf{0}\,\rangle & \simeq_{ax} & \mathbf{0} & \pi_2(\mathbf{0}) & \simeq_{ax} & \mathbf{0} \\
(r+\alpha\cdot s)t & \simeq_{ax} & rt+\alpha\cdot(st) & \mathbf{0}t & \simeq_{ax} & \mathbf{0} \\
r(s+\alpha\cdot t) & \simeq_{ax} & rs+\alpha\cdot(rt) & t\mathbf{0} & \simeq_{ax} & \mathbf{0} \\
\lambda x.(s+\alpha\cdot t) & \simeq_{ax} & \lambda x.s+\alpha\cdot(\lambda x.t) & \lambda x.\mathbf{0} & \simeq_{ax} & \mathbf{0} \\
\{\,(s+\alpha\cdot t\,\} & \simeq_{ax} & \{\,s\,\}+\alpha\cdot\{\,t\,\} & \{\,\mathbf{0}\,\} & \simeq_{ax} & \mathbf{0} \\
\\
\pi_1\langle\, u,v\,\rangle & \simeq_{ax} & u & [\,\{\,u\,\}\,] & \simeq_{ax} & u \\
\pi_2\langle\, u,v\,\rangle & \simeq_{ax} & v & \{\,[\,s\,]\,\} & \simeq_{ax} & s \\
\langle\, \pi_1(u),\pi_2(u)\,\rangle & \simeq_{ax} & u & (\lambda x.\{\,s\,\})t & \simeq_{ax} & \{\,(\lambda x.s)t\,\} \\
(\lambda x.u)v & \simeq_{ax} & u[v/x] & ((\lambda xy.r)s)t & \simeq_{ax} & ((\lambda yx.r)t)s \\
\lambda x.(ux) & \simeq_{ax} & u & (\lambda x.r)((\lambda y.s)t) & \simeq_{ax} & (\lambda y.(\lambda x.r)s)t \\
(\lambda x.x)s & \simeq_{ax} & s & & &
\end{array}
$$

Table 3. Axiomatic equivalence relation.

a computation through the multiplication of the monad; finally, the sum and the external multiplication are handled by the enriched structure.

Theorem 2.18. *If we interpret the simply-typed algebraic lambda-calculus in $\mathcal{C}_l$ then the equations $[\![\, x:A\vdash v:B\,]\!]^v \simeq_{ax} (x:A\vdash v:B)$ and $[\![\, x:A\vdash t:B\,]\!]^c \simeq_{ax} (x:A\vdash [\,t\,]:MB)$ hold.* □

2.3. Example: quantum computation

The main motivation behind [2] was quantum computation. In this section, we show that one can simulate quantum computation with the simply-typed algebraic lambda-calculus.

Quantum computation is a paradigm where data is encoded on the state of objects governed by the law of quantum physics. The mathematical description of a quantum boolean is a (normalized) vector in a 2-dimensional Hilbert space $\mathbb{H}$. In order to give sense to this vector, one chooses an orthonormal basis $\{|0\rangle,|1\rangle\}$. A vector $\alpha|0\rangle+\beta|1\rangle$ is understood as the "quantum superposition" of the boolean 0 and the boolean 1. If one consider two quantum boolean, the state of the corresponding system is an element of $\mathbb{H}\otimes\mathbb{H}$. The chosen basis is $\{|0\rangle\otimes|0\rangle,|0\rangle\otimes|1\rangle,|1\rangle\otimes|0\rangle,|1\rangle\otimes|1\rangle\}$. The operations one can performed on quantum booleans are of two sorts: Quantum gates and measurements. In the mathematical description, the former correspond to unitary maps. The latter has a probabilistic outcome and does not have a satisfactory description in term of vectors. To be able to interpret it, one uses the notion of *density matrices* to represent quantum booleans, that is, positive matrices of norm one. The measurement operation is then the map sending a matrix to its diagonal.

For simulating quantum computation, choose the ring $\mathcal{A}$ to be the field of complex numbers. Given an arbitrary type X, one can represent a quantum boolean in the simply-typed algebraic lambda-calculus as a closed value of type $bit=MX\to(MX\to MX)$. One encode $\alpha|0\rangle+\beta|1\rangle$ as $\lambda xy.[\,\alpha\cdot\{\,x\,\}+\beta\cdot\{\,y\,\}\,]$. One writes tt for $\lambda xy.[\,\{\,x\,\}\,]$ and ff for $\lambda xy.[\,\{\,y\,\}\,]$. The Hadamard gate, sending $|0\rangle$ to $\frac{1}{\sqrt{2}}(|0\rangle+|1\rangle)$ and $|1\rangle$ to $\frac{1}{\sqrt{2}}(|0\rangle-|1\rangle)$, can be written as the term

$$H=\lambda x.\lambda ab.[\,\{x[\,\tfrac{1}{\sqrt{2}}\cdot(\{a\}+\{b\})\,][\,\tfrac{1}{\sqrt{2}}\cdot(\{a\}-\{b\})\,]\}\,]$$

of type $bit \to bit$. Applying the Hadamard gate to a quantum boolean b is computing the term Hb.

In order to model measurements, we can use the fact that the language features higher-order and encode a positive matrix as a term of type $bit \to bit$. The quantum boolean $\alpha|0\rangle + \beta|1\rangle$ is encoded as the term v equal to

$$\lambda x.\lambda ab.[\ \{x\ [\ \alpha\bar{\alpha}\cdot\{a\} + \alpha\bar{\beta}\cdot\{b\}\ ][\ \bar{\alpha}\beta\cdot\{a\} + \beta\bar{\beta}\cdot\{b\}\ ]\}\ ].$$

The application of the Hadamard gate to v is $H'v$, where H' is the term $H' = \lambda x.H(xH)$ of type $(bit \to bit) \to (bit \to bit)$. The measurement is also of type $(bit \to bit) \to (bit \to bit)$ and can be encoded as the term P equal to

$$\lambda v.\lambda x.\lambda ab.[\ \{(v\,x)[\ \{\ a\ \}\ ][\ \mathbf{0}\ ] + (v\,x)[\ \mathbf{0}\ ][\ \{\ b\ \}\ ]\}\ ].$$

One can check that Pv is indeed equal to

$$\lambda x.\lambda ab.[\ \{x\ [\ \alpha\bar{\alpha}\cdot\{a\}\ ][\ \beta\bar{\beta}\cdot\{b\}\ ]\}\ ].$$

3. Adding controlled divergence

Because of Theorem 2.12, a term Y_b behaving as in Equation (1) is not constructable in the simply-typed algebraic lambda-calculus. In this section, we add to the language a notion of fixpoint in order to understand what goes wrong in the untyped system.

3.1. A fixpoint operator

In order to stay typed and to be able to keep most of the computational interpretation of Section 2.2 but still to be able to have a term Y_b, we add to the language a unary term operator Y verifying the reduction

$$Y(s) \to s(Y(s))$$

and satisfying the typing rule

$$\Delta \vdash s : MA \to MA \quad \Longrightarrow \quad \Delta \vdash Y(s) : MA. \tag{3}$$

We can now build a term Y_b behaving as required in Equation (1).

$$Y_b \equiv \{\ Y(\lambda x.[\ b + \{\ x\ \}\ ])\ \}. \tag{4}$$

Indeed:

$$\begin{aligned}
&\{\ Y(\lambda x.[\ b + \{\ x\ \}\ ])\ \}\\
&\to \{\ (\lambda x.[\ b + \{\ x\ \}\ ])(Y(\lambda x.[\ b + \{\ x\ \}\ ]))\ \}\\
&\to \{\ [\ b + \{\ Y(\lambda x.[\ b + \{\ x\ \}\ ])\ \}\ ]\ \}\\
&\to b + \{\ Y(\lambda x.[\ b + \{\ x\ \}\ ])\ \}.
\end{aligned}$$

Provided that $\Delta \vdash b : B$, the term Y_b satisfies the typing judgement $\Delta \vdash Y_b : B$.

Remark 3.1. Of course, if we keep the operational semantics of Section 2, the system becomes as inconsistent as with the untyped calculus. Let us review the problems that can occurs.

3.2. The zero in the algebra of terms

To understand what goes wrong, consider the typing judgement

$$x : MA \vdash x - x : MA.$$

With the equational system of Section 2.2, this typing judgement is equivalent to $x : MA \vdash \mathbf{0} : MA$. We claim that this interpretation is correct as long as the term x "does not contain any potential infinity". With the additional construct Y, one can replace x with $[\ Y_a\ ]$ (where Y_a is constructed as in (4)), for some term a of type A. Consider the two terms

$$(\lambda y.*)((\lambda x.(x - x))[\ Y_a\ ]), \tag{5}$$

$$(\lambda y.\{\ y\ \})((\lambda x.(x - x))[\ Y_a\ ]). \tag{6}$$

Term (5) reduces to $(\lambda y.*)(0 \cdot [\ Y_a\ ])$ and then to $0 \cdot *$. It is reasonable to think that this is equivalent to $\mathbf{0}$, thus making $0 \cdot [\ Y_a\ ]$ also equivalent to $\mathbf{0}$. Term (6), on the contrary, reduces to $Y_a - Y_a$, the flawed term of Equation (2).

The problem does not show up when writing the equation $[\ Y_a\ ] - [\ Y_a\ ] = 0 \cdot [\ Y_a\ ]$ but when one equates it with $\mathbf{0}$. The term $0 \cdot [\ Y_a\ ]$ is a "weak zero". It makes a computation "null" as long as it does not diverge (and there is always a diverging term of any inhabited type by using the construction (4)). Therefore, despite the fact that $\mathcal{A}$ is a ring, the set of terms of the form $\alpha \cdot s$ for a fixed term s is only a commutative monoid: addition does not admit an inverse, it only have an identity element $0 \cdot s$.

3.3. Recasting the equational theory

With the possible addition of fixpoints, the equational theory given in Section 2.2 is not valid. In the discussion of the previous section, we noted that the module of terms needs to be weakened to a commutative monoid by removing the rule $0 \cdot s \simeq_{ax} \mathbf{0}$. It is the only required modification, and one can rewrite the whole theory without this rule.

We do not include explicitly in this study the notion of fixpoints. Instead, we developed a general framework for working with diverging terms. The addition of fixpoints will be considered in the Section 4.

Definition 3.2. A *weak $\mathcal{A}$-module* is a module over $\mathcal{A}$ where $\mathcal{A}$ is seen as a semiring. In particular, a weak $\mathcal{A}$-module is only a commutative monoid. Given a set X, the *free weak $\mathcal{A}$-module over X* is the structure consisting of all the finite sums $\sum_i \alpha_i \cdot x_i$, where $\alpha_i \in \mathcal{A}$ and $x_i \in X$.

Definition 3.3. A *weak $\mathcal{A}$-enriched computational category* consists of the following data.

- A cartesian closed category $(\mathcal{C}, \times, \Rightarrow, \mathbf{1})$,
- together with a strong monad (M, η, μ, t),

- such that the Kleisli category $\mathcal{C}_M$ is enriched over the category of weak $\mathcal{A}$-modules.

Remark 3.4. As we saw in Section 3.2, the two zero-functions $x : A \vdash \mathbf{0} : A$ and $x : A \vdash 0 \cdot x : A$ behave differently in general. In a weak $\mathcal{A}$-enriched computational category, the former is interpreted as the unit element of the monoid $\mathcal{C}_M(A, A)$ whereas the latter is of the form $0 \cdot id_A$, where id_A is the identity map in $\mathcal{C}_M(A, B)$.

Lemma 3.5. *Any $\mathcal{A}$-enriched computational category is also a weak $\mathcal{A}$-enriched computational category.*

Proof. Any $\mathcal{A}$-module is also a weak $\mathcal{A}$-module. □

Remark 3.6. In particular, in a $\mathcal{A}$-enriched computational category, the two zero-functions $x : A \vdash \mathbf{0} : B$ and $x : A \vdash 0 \cdot x : B$ are identified.

Results similar to Theorems 2.16 and 2.18 can be proved for this weakened system. We state them here.

Definition 3.7. Consider the typed language of Definition 2.1, with the axiomatic equivalence of Table 3 minus the very first rule, marked as $(*)$, stating $0 \cdot u \simeq_{ax} \mathbf{0}$. Let us call this language the *weak algebraic simply-typed lambda-calculus* and the corresponding category of values $\mathcal{C}_l^w$.

Theorem 3.8. *The weak algebraic simply-typed lambda-calculus is an internal language for weak $\mathcal{A}$-enriched computational categories.* □

4. An algebraic PCF

We now turn to the question of finding an operational semantics for a simply-typed algebraic lambda-calculus with fixpoints. For this purpose, we study a particular lambda-calculus in the form of a call-by-value PCF language with a fixpoint Y and an algebraic structure.

Definition 4.1. Let linPCF be the language defined as follows.

$$\begin{array}{rcl} A, B & ::= & \mathit{bit} \mid \mathit{int} \mid A \to B \mid A \times B \mid \top \mid MA, \\ r, s, t & ::= & x^A \mid \lambda x^A.s \mid st \mid \langle\, s, t \,\rangle \mid \pi_1(s) \mid \pi_2(s) \mid * \mid \\ & & Y(s) \mid \Omega \mid s + t \mid \alpha \cdot s \mid \mathbf{0} \mid [\, s \,] \mid \{\, s \,\} \mid \\ & & \mathit{tt} \mid \mathit{ff} \mid \mathit{if}\ r\ \mathit{then}\ s\ \mathit{else}\ t \mid \\ & & \bar{0} \mid \mathit{succ}(s) \mid \mathit{pred}(s) \mid \mathit{iszero}(s), \end{array}$$

where $\alpha \in \mathcal{A}$. The terms tt and ff respectively stand for the boolean true and the boolean false; the term $\mathit{if}\ r\ \mathit{then}\ s\ \mathit{else}\ t$ is the test function on r; the term $\bar{0}$ stands for the natural number 0; the term $\mathit{iszero}(s)$ tests whether s is null or not; pred and succ are respectively the predecessor and the successor function; finally, Y is the fixpoint of Section 3.1, and Ω is a diverging term (we add it for convenience: it can be simulated by $Y(\lambda x.x)$). The notion of pure term is defined as in Definition 1.1.

$$\begin{array}{rcl} u, v & ::= & x^A \mid \lambda x^A.u \mid uv \mid \langle\, u, v \,\rangle \mid * \mid [\, s \,] \mid \\ & & \mathit{tt} \mid \mathit{ff} \mid \mathit{if}\ r\ \mathit{then}\ s\ \mathit{else}\ t \mid \\ & & \bar{0} \mid \mathit{succ}(u) \mid \mathit{pred}(u) \mid \mathit{iszero}(s). \end{array}$$

We define the notions of typing context Δ and of typing derivation $\Delta \vdash s : A$ as in Definition 2.1. Terms are considered up to α-equivalence, and valid typing derivations are built using the rules of Table 2, the rule of Equation (3), and the usual rules concerning PCF. The rules for the null-ary terms are

$$\Delta \vdash \mathit{tt}, \mathit{ff} : \mathit{bit}, \qquad \Delta \vdash \bar{0} : \mathit{int}, \qquad \Delta \vdash \Omega : A,$$

the rules for the unary terms are

$$\Delta \vdash s : \mathit{int} \quad \Longrightarrow \quad \left\{ \begin{array}{l} \Delta \vdash \mathit{pred}(s), \mathit{succ}(s) : \mathit{int}, \\ \Delta \vdash \mathit{iszero}(s) : \mathit{bit}, \end{array} \right.$$

and the rule for the if-term is

$$\left. \begin{array}{l} \Delta \vdash r : \mathit{bit}, \\ \Delta \vdash s, t : A \end{array} \right\} \quad \Longrightarrow \quad \Delta \vdash \mathit{if}\ r\ \mathit{then}\ s\ \mathit{else}\ t : A.$$

4.1. Rewrite system and equational theory

The rewrite system of Section 2.1 can be reformulated for linPCF. Again, apart from the rule $(*)$ of Table 1 which is not valid, all the other one are correct.

Definition 4.2. As in Definition 2.4, we define the reduction systems E, F, A and B of terms as the smallest congruent relations consistent with $\simeq_{AC}$, satisfying the rules in Table 1 where B is augmented with the rules

$$\begin{array}{rr} \pi_1\langle\, u, v \,\rangle \to u, & \mathit{iszero}(\bar{0}) \to \mathit{tt}, \\ \pi_2\langle\, u, v \,\rangle \to v, & \mathit{iszero}(\mathit{succ}(u)) \to \mathit{ff}, \\ \mathit{if}\ \mathit{tt}\ \mathit{then}\ s\ \mathit{else}\ t \to s, & \mathit{succ}(\mathit{pred}(u)) \to u, \\ \mathit{if}\ \mathit{ff}\ \mathit{then}\ s\ \mathit{else}\ t \to t, & \Omega \to \Omega. \end{array}$$

In all the given rules, the terms r, s, t are assumed to be neutral and the terms u, v are assumed to be pure. We write L' for the relation $A \cup B \cup E \cup F$, and as before we write $\to$ in place of $\to_{L'}$.

Again, the rewrite system verifies subject reduction.

Theorem 4.3. *Suppose that $\Delta \vdash s : A$ is valid and that $s \to t$. Then $\Delta \vdash t : A$ is also valid.*

However, it does not satisfy strong normalization: the rewrite sequence

$$\Omega \to \Omega \to \Omega \to \cdots$$

is infinite.

One can define an axiomatic relation by extending the one described in Section 3.3 with the added relations in Definition 4.2.

Definition 4.4. We define an equivalent relation $\simeq_{ax}$ on typing judgements of linPCF as the smallest congruent relation, consistent with $\simeq_{AC}$, satisfying the rules of Table 3 minus the rule $(*)$, plus the added relation (made reflexive) in Definition 4.2.

Lemma 4.5. *If $s \to t$, then $s \simeq_{ax} t$.*

Proof. Proof by induction on the reduction $s \to t$. □

4.2. Case of the divergence

Again consider the term Y_b: it is equivalent to $Y_b + n \cdot b$ for all b. So in a sense, it is equivalent to $\uparrow \cdot b$, where $\uparrow$ is a scalar representing a diverging sequence of scalars.

In order to grasp the difference between the simply-typed calculus without fixpoints and with fixpoints, let us look at a few examples:

$$\begin{aligned}
s_1 &\equiv Y(\lambda x.[\,\bar{0} + \alpha \cdot \{\, x \,\}\,]),\\
s_2 &\equiv Y(\lambda x.[\,\bar{0} + \alpha \cdot succ\{\, x \,\}\,]),\\
s_3 &\equiv Y(\lambda f.[\lambda x.\{\mathit{if}\ iszero(x) \mathit{then}[\,\bar{0}\,]\\
&\qquad\qquad \mathit{else}[\, x + f(pred\ x)\,]\}],\\
s_4 &\equiv Y(\lambda f.[\lambda x.\{\mathit{if}\ iszero(x) \mathit{then}[\,\bar{0}\,]\\
&\qquad\qquad \mathit{else}[\, x + f(succ\ x)\,]\}],\\
s_5 &\equiv Y(\lambda x.[\,\bar{0}\,]),\\
s_6 &\equiv Y(\lambda x.x).
\end{aligned}$$

The terms s_1 and s_2 reduces respectively to, for all n,

$$[\,(\sum_{i=0}^{n} \alpha^n) \cdot \bar{0}\,], \qquad [\,\sum_{i=0}^{n} (\alpha^n \cdot \bar{n})\,],$$

where $\bar{n}$ is the n^{th} natural number in linPCF. The terms s_3 and s_4 are functions taking as input a number x and outputting respectively $\sum_{i=0}^{x} \bar{i}$ (in a finite rewrite sequence) and $\sum_{i=0}^{\infty} \bar{i}$. The term s_5 outputs in $[\,\bar{0}\,]$ in a finite rewrite sequence, and s_6 behave like Ω: it reduces to itself.

The question is then to define what we mean by "converging". For example, if we consider that converging mean "having a finite rewrite sequence", then only s_3 and s_5 do converge. One can however be a little more general and also admits s_2, by understanding the term as converging to a set-map from int to $\mathcal{A}$. If one also add a topology on $\mathcal{A}$, one can also make s_1 converging for some values of α. With a measure on int, one can further precise the notions of convergences.

In brief, there are several notions of divergences. A term can be strongly normalizing, as s_3 or s_5, or it can normalize up to pointwise limit on values, as for example in the case s_2.

The small-step semantics describes the evolution of a given term through a set of rewriting rules. As a local description of the computation, it is not enough to capture the weak notions of convergence described above. Instead, one can directly describe the neutral form that a given term is supposed to reach, in what is called a *big-step semantics*.

In the following section, we describe a semantics for a strongly converging linPCF.

5. Strict convergence of the algebraic PCF

In this section, we study a semantics for linPCF with strict convergence. In particular, we want to identify all diverging terms of a given type, such as $\Omega : \top$ and $* + \Omega : \top$. Note however that we do not want to identify $[\,\Omega\,] : MA$ and $\Omega : MA$. Indeed, the first one is a computation encapsulated into a value, and can therefore be applied as it in a function, whereas the other one is a non-terminating computation.

5.1. Big-step semantics

The first method for describing the behavior of a term is to give the neutral term to which this term converges to. Called the big-step reduction, this relation is well-known in the literature (see e.g. [1]).

Definition 5.1. We call *state* an $\simeq_{AC}$-equivalence class of neutral terms. We write $\llcorner t \lrcorner_{AC}$ the $\simeq_{AC}$-equivalence class of a term t. We write a state as a formal linear combination of factors, and we identify $\llcorner t \lrcorner_{AC}$ with t when the context is clear. A state $\llcorner t \lrcorner_{AC}$ is a *result* if t is a neutral term.

Let s be a term of linPCF. We say that s *down-reduces to a result* $\llcorner t \lrcorner_{AC}$, denoted $s \Downarrow \llcorner t \lrcorner_{AC}$, if it is derived from the rules of Table 4, with the following modifications. In Formula $(N1)$, for all i, if t_i is of the form $pred(t'_i)$, one replaces t_i with t'_i; in Formula $(N3)$, for all i, if t_i is of the form $succ(t'_i)$, one replaces it with t'_i, and if it is $\bar{0}$, one replaces it with $\bar{0}$; in Formula $(N4)$, for all i, if t_i is of the form $\langle\, t'_i, t''_i \,\rangle$, one replaces it with t'_i; in Formula $(N5)$, for all i, if t_i is of the form $\langle\, t'_i, t''_i \,\rangle$, one replaces it with t''_i.

If there is no such result, one says that the term *strongly diverges*.

$$x \Downarrow \llcorner x \lrcorner_{AC} \quad * \Downarrow \llcorner * \lrcorner_{AC} \quad tt \Downarrow \llcorner tt \lrcorner_{AC} \quad ff \Downarrow \llcorner ff \lrcorner_{AC} \quad \bar{0} \Downarrow \llcorner \bar{0} \lrcorner_{AC} \quad \mathbf{0} \Downarrow \llcorner \mathbf{0} \lrcorner_{AC}$$

$$s \Downarrow \llcorner \sum_i \alpha_i \cdot t_i \lrcorner_{AC} \implies \begin{cases} succ(s) \Downarrow \llcorner \sum_i \alpha_i \cdot succ(t_i) \lrcorner_{AC} \ (N1) & \lambda x.s \Downarrow \llcorner \sum_i \alpha_i \cdot \lambda x.t_i \lrcorner_{AC} \\ pred(s) \Downarrow \llcorner \sum_i \alpha_i \cdot pred(t_i) \lrcorner_{AC} \ (N3) & \pi_2 s \Downarrow \llcorner \sum_i \alpha_i \cdot \pi_2 t_i \lrcorner_{AC} \ (N4) \\ iszero(s) \Downarrow \llcorner \sum_i \alpha_i \cdot iszero(t_i) \lrcorner_{AC} \ (N5) & \pi_1 s \Downarrow \llcorner \sum_i \alpha_i \cdot \pi_1 t_i \lrcorner_{AC} \ (N6) \\ \alpha \cdot s \Downarrow \llcorner \sum_i \alpha_i \alpha \cdot t_i \lrcorner_{AC} \ (N7) & [\ s\ ] \Downarrow \llcorner [\ \sum_i \alpha_i \cdot t_i\ ] \lrcorner_{AC} \end{cases}$$

$$\left.\begin{array}{l} s \Downarrow \llcorner \sum_i \alpha_i \cdot s_i \lrcorner_{AC} \\ t \Downarrow \llcorner \sum_j \beta_j \cdot t_j \lrcorner_{AC} \end{array}\right\} \implies \begin{cases} \langle\ s,t\ \rangle \Downarrow \llcorner \sum_{i,j} \alpha_i \beta_j \cdot \langle\ s_i, t_j\ \rangle \lrcorner_{AC} \\ s + t \Downarrow \llcorner \sum_i \alpha_i \cdot s_i + \sum_j \beta_j \cdot t_j \lrcorner_{AC} \end{cases}$$

$$\left.\begin{array}{l} s \Downarrow \llcorner \sum_i \alpha_i \cdot s_i \lrcorner_{AC} \\ t \Downarrow \llcorner \sum_j \beta_j \cdot t_j \lrcorner_{AC} \\ s_i t_j \Downarrow \llcorner \sum_k \gamma_{i,j,k} \cdot r_{i,j,k} \lrcorner_{AC} \end{array}\right\} \implies \left\{\ st \Downarrow \llcorner \sum_{i,j,k} \alpha_i \beta_j \gamma_{i,j,k} \cdot r_{i,j,k} \lrcorner_{AC}\right.$$

$$\left.\begin{array}{l} s \Downarrow \llcorner \sum_i \alpha_i \cdot s_i \lrcorner_{AC} \\ \text{if for } i \in I,\ s_i = [\ \sum_j \beta_{i,j} \cdot t_{i,j}\ ] \end{array}\right\} \implies \{\ s\ \} \Downarrow \llcorner \sum_{i \notin I} \alpha_i \cdot s_i + \sum_{i \in I} \sum_j \beta_{i,j} \cdot t_{i,j} \lrcorner_{AC}$$

$$s(Y(s)) \Downarrow \llcorner \sum_i \alpha_i \cdot s_i \lrcorner_{AC} \implies Y(s) \Downarrow \llcorner \sum_i \alpha_i \cdot s_i \lrcorner_{AC}$$

Table 4. Big step semantics

Lemma 5.2. *If $s \Downarrow t$, then t is neutral and $s \to^* t$. Conversely, if $s \to^* t$ where t is neutral, then $s \Downarrow t$.* □

Definition 5.3. We define a relation on valid terms as follows: $\Delta \vdash s \sqsubseteq_{op} t : A$ if and only if for all contexts $C[\Delta \vdash - : A] : bit$, for all neutral term $r : bit$, $C[s] \Downarrow r$ implies $C[t] \Downarrow r$. We define the relation $\Delta \vdash s \simeq_{op} t : A$ by

$$\Delta \vdash s \sqsubseteq_{op} t : A \text{ and } \Delta \vdash t \sqsubseteq_{op} s : A.$$

Lemma 5.4. *The relation $\sqsubseteq_{op}$ is a partial ordering on valid typing judgements. The relation $\simeq_{op}$ is an equivalence relation.* □

5.2. Set-model for the algebraic PCF

The language linPCF is a language of the form described in Section 3.3. In order to find a model, we need to find a correct weak $\mathcal{A}$-enriched computational category matching the operational semantics.

A naive denotational model can be constructed with the cartesian category of Set and functions. We need two notions of monad to capture the operational behavior of the language: the diverging monad

$$D : \begin{array}{ccc} \mathbf{Set} & \to & \mathbf{Set} \\ X & \mapsto & X \cup \{\bot\}, \end{array}$$

and the weak-module monad

$$W : \begin{array}{ccc} \mathbf{Set} & \to & \mathbf{Set} \\ X & \mapsto & \langle\ X\ \rangle, \end{array}$$

where $\langle\ X\ \rangle$ is the free weak $\mathcal{A}$-monad generated from X. These two monads are computational monads, and one can compose them. The correct monad capturing the strong convergence is $M = D \circ W$. Indeed, a map in the Kleisli category takes a value and either diverges or return a finite linear combination of terms.

Theorem 5.5. *The category* **Set** *together with the monad M is a weak $\mathcal{A}$-enriched computational category.* □

Consider a function $f : M[\![\, A \,]\!] \to M[\![\, A \,]\!]$. For every n and for every $x \in MA$, $f^n(x)$ is an element of $M[\![\, A \,]\!]$, that is, a function $[\![\, A \,]\!] \to \mathcal{A}$.

Lemma 5.6. *Given a set X and a function $f : MX \to MX$, the sequence $(f^n(\bot))_n$ pointwise converges to an element of MX.* □

Definition 5.7. We call the *set-model* for linPCF weak $\mathcal{A}$-enriched computational category of Theorem 5.5, where we set $[\![\, int \,]\!] = \mathbb{N}$, $[\![\, bit \,]\!] = \{tt, ff\}$ and where we give to the corresponding term constructs their obvious meanings. The fixpoint is interpreted using Lemma 5.6.

Theorem 5.8 (Soundness). *If* $\Delta \vdash s \simeq_{ax} t : A$ *then* $[\![s]\!] = [\![t]\!]$. □

5.3. Limits of the model

The category of sets and functions does not provides a fully abstract with respect to the operational semantics. Indeed, consider the terms

$$(\lambda x.\mathit{if}\ x\ \mathit{then}\ [\ *\ ]\ \mathit{else}\ [\ \Omega\ ]) + (\lambda x.\mathit{if}\ x\ \mathit{then}\ [\ \Omega\ ]\ \mathit{else}\ [\ \star\ ])$$

and $2 \cdot \lambda x.[\ \Omega\ ]$, of type $bit \to \top$. Intuitively, they are operationally equivalent: they are both functions, and applying tt or ff to them will produce in both cases a diverging term.

6. Conclusion

In this paper, we sketched the required structures for a semantics for a typed algebraic lambda-calculus. We shown that the notion of divergence is not easy to capture and possesses many facets. We then sketch a model for astrongly converging, algebraic PCF language with fixpoints.

The question is now to describe a fully abstract model for this algebraic PCF language and explore the structure of weakly converging languages.

7. Acknowledgements

I would like to thank Gilles Dowek for introducing me to algebraic calculi. I would also like to thank Pablo Arrighi and the research group QCG in Grenoble for helpful discussions.

References

[1] Roberto M. Amadio and Pierre-Louis Curien. *Domains and Lambda-Calculi*, volume 45 of *Cambridge Tracts In Theoretical Computer Science*. Cambridge University Press, 1998.

[2] Pablo Arrighi and Gilles Dowek. Linear-algebraic lambda-calculus: higher-order, encodings, and confluence. In *Proceedings of the 19th international conference on Rewriting Techniques and Applications (RTA'08)*, volume 5117 of *Lecture Notes in Computer Science*, pages 17–31, 2008.

[3] Henk P. Barendregt. *The Lambda-Calculus, its Syntax and Semantics*. North Holland, 1984.

[4] Thomas Ehrhard and Laurent Regnier. The differential lambda-calculus. *Theoretical Computer Science*, 309(1–2): 1–41, 2003.

[5] Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and Types*. Cambridge University Press, 1990.

[6] Gregory M. Kelly. *Basic Concepts of Enriched Category Theory*, volume 64 of *London Mathematical Society Lecture Notes Series*. Cambridge University Press, 1982. Avalaible in Reprint in Theory and Application of Categories, No 10, 1982.

[7] Joachim Lambek and Philip Scott. *Introduction to Higher Order Categorical Logic*. Cambridge University Press, 1989.

[8] Eugenio Moggi. Notions of computation and monads. *Information and Computation*, 93:55–92, 1991.

[9] André van Tonder. A lambda calculus for quantum computation. *SIAM Journal of Computing*, 33:1109–1135, 2004.

[10] Lionel Vaux. Algebraic lambda-calculus. *Mathematical Structures in Computer Science*, 2008. To appear.