A type system for the vectorial aspects of the linear-algebraic lambda-calculus

Pablo Arrighi\textsuperscript{1,2} \textbf{Alejandro Díaz-Caro}\textsuperscript{1} \textbf{Benoît Valiron}\textsuperscript{3,4}

\textsuperscript{1}Université de Grenoble, LIG, France
\textsuperscript{2}École Normale Supérieure de Lyon, LIP, France
\textsuperscript{3}Université de Paris-Nord, LIPN, France
\textsuperscript{4}University of Pennsylvania, USA

7\textsuperscript{th} DCM • July 3, 2011 • Zurich, Switzerland
\[ M, N ::= x \mid \lambda x.M \mid (M)N \mid M + N \mid \alpha.M \mid 0 \]

Beta reduction:
\[(\lambda x.M)N \rightarrow M[x := N] \]

“Algebraic” reductions:
\[\alpha.M + \beta.M \rightarrow (\alpha + \beta).M,\]
\[(M)(N_1 + N_2) \rightarrow (M)N_1 + (M)N_2,\]
\[\ldots\]
\[\ldots\]
\[\ldots\]
*(oriented version of the axioms of vectorial spaces)*

Two origins:
- Differential \(\lambda\)-calculus: capturing linearity à la Linear Logic
  - Removing the differential operator: *Algebraic \(\lambda\)-calculus* (\(\lambda_{\text{alg}}\)) [Vaux’09]
- Quantum computing: superposition of programs
  - *Linearity as in algebra*: *Linear-algebraic \(\lambda\)-calculus* (\(\lambda_{\text{lin}}\)) [Arrighi,Dowek’08]
\[ M, N ::= x \mid \lambda x. M \mid (M) N \mid M + N \mid \alpha. M \mid 0 \]

Beta reduction:
\[ (\lambda x. M) N \rightarrow M[x := N] \]

“Algebraic” reductions:
\[ \alpha. M + \beta. M \rightarrow (\alpha + \beta). M, \]
\[ (M)(N_1 + N_2) \rightarrow (M) N_1 + (M) N_2, \]
\[ \ldots \]
\[ \ldots \]

*(oriented version of the axioms of vectorial spaces)*

<table>
<thead>
<tr>
<th>Origin</th>
<th>[ \lambda_{\text{alg}} ]</th>
<th>[ \lambda_{\text{lin}} ]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strategy</td>
<td>Linear Logic</td>
<td>Quantum computing</td>
</tr>
<tr>
<td>Algebraic part</td>
<td>Call-by-name</td>
<td>Call-by-value</td>
</tr>
<tr>
<td></td>
<td>Equalities</td>
<td>Rewrite system</td>
</tr>
</tbody>
</table>
An infinite dimensional vectorial space of values

\[ \mathcal{B} = \{ M_i : M_i \text{ is a variable or abstraction} \} \]

Set of values ::= Span(\mathcal{B})

(Now we should call \(\lambda_{lin}\)'s strategy: “call-by-base”)
Why would it be interesting?

▶ Several theories using the concept of linear-combination of terms quantum, probabilistic, non-deterministic models, . . .

▶ “Why would vector spaces be an interesting theory?”
  Many applications and moreover, interesting by itself!

Aim of the current work:
A type system capturing the “vectorial” structure of terms

  . . . to check for probability distributions
  . . . or “quantumness” of the term
  . . . or whatever application needing the structure of the vector in normal form
  . . . a Curry-Howard approach to defining Fuzzy/Quantum/Probabilistic logics from Fuzzy/Quantum/Probabilistic programming languages.
The **Scalar Type System** [Arrighi,Díaz-Caro’09]
A polymorphic type system *tracking scalars*:
\[ \Gamma \vdash M : T \]
\[ \Gamma \vdash \alpha. M : \alpha. T \]

- Barycentric restrictions
- Characterises the “amount” of terms

The **Additive Type System** [Díaz-Caro,Petit’10]
A polymorphic type system *with sums*:
\[ \Gamma \vdash M : T \quad \Gamma \vdash N : R \]
\[ \Gamma \vdash M + N : T + R \]

- Sums ∼ Assoc., comm. pairs
- distributive w.r.t. application

Can we combine them?
The Vectorial Type System

Types:

\[ T, R, S := U \mid T + R \mid \alpha.T \]
\[ U, V, W := X \mid U \rightarrow T \mid \forall X.U \]

(U, V, W reflect the basis terms)

Equivalences:

1. \[ T \equiv T \]
2. \[ \alpha.(\beta.T) \equiv (\alpha \times \beta).T \]
3. \[ \alpha.T + \alpha.R \equiv \alpha.(T + R) \]
4. \[ \alpha.T + \beta.T \equiv (\alpha + \beta).T \]
5. \[ T + R \equiv R + T \]
6. \[ T + (R + S) \equiv (T + R) + S \]

(reflect the vectorial spaces axioms)
The factorisation rule problem

\[
\Gamma \vdash M : T \quad \Gamma \vdash M : T' \\
\overline{\quad \Gamma \vdash \alpha. M + \beta. M : \alpha. T + \beta. T' \quad}
\]

- However, \( \alpha. M + \beta. M \rightarrow (\alpha + \beta). M \)
- In general \( \alpha. T + \beta. T' \neq (\alpha + \beta). T \neq (\alpha + \beta). T' \)

(and since we are working in System F, there is no principal types neither)
Several possible solutions:

- Remove factorisation rule (Done. SR and SN both work)
  - $+$ in scalars not used anymore. Scalars $\Rightarrow$ Monoid
  - It works!… but it is no so expressive ("vectorial" structure lost)
Several possible solutions:

- Remove factorisation rule (Done. SR and SN both work)
  - + in scalars not used anymore. Scalars ⇒ Monoid
  - It works!... but it is no so expressive (“vectorial” structure lost)

- Add several typing rules to allow typing \((\alpha + \beta).M\) with \(\alpha.T + \beta.T'\)
  - As soon as we add one, we have to add many to make it work
  - Too complex and inelegant (subject reduction by axiom)
Several possible solutions:

- Remove factorisation rule (Done. SR and SN both work)
  - + in scalars not used anymore. Scalars $\Rightarrow$ Monoid
  - It works!... but it is no so expressive ("vectorial" structure lost)

- Add several typing rules to allow typing $(\alpha + \beta).M$ with $\alpha.T + \beta.T'$
  - As soon as we add one, we have to add many to make it work
  - Too complex and inelegant (subject reduction by axiom)

- Church style
  - Seems to be the natural solution
  - Big complexity with polymorphism and distributivity
Several possible solutions:

- Remove factorisation rule (Done. SR and SN both work)
  - + in scalars not used anymore. Scalars $\Rightarrow$ Monoid
  - It works!... but it is no so expressive ("vectorial" structure lost)

- Add several typing rules to allow typing $(\alpha + \beta).M$ with $\alpha.T + \beta.T'$
  - As soon as we add one, we have to add many to make it work
  - Too complex and inelegant (subject reduction by axiom)

- Church style
  - Seems to be the natural solution
  - Big complexity with polymorphism and distributivity

- Weak subject reduction (this work)
  - What is the best we can get in Curry style?
Typing rules

\[
\begin{align*}
\Gamma, x : U & \vdash x : U \quad \text{ax} \\
\Gamma & \vdash 0 : 0. T \quad 0_I \\
\Gamma & \vdash \alpha.M : \alpha.T \quad \alpha_I \\
\Gamma, x : U & \vdash M : T \quad \rightarrow_I \\
\Gamma & \vdash \lambda x.M : U \rightarrow T \\
\Gamma & \vdash M : T \quad \Gamma \vdash N : R \\
\Gamma & \vdash M + N : T + R \\
\Gamma & \vdash M : U \quad X \notin \text{FV(}\Gamma) \\
\Gamma & \vdash M : \forall X.U \\
\Gamma & \vdash M : U[V/X] \\
\Gamma & \vdash (M) N : \sum_{i=1}^{n} \sum_{j=1}^{m} \alpha_i \forall \vec{X}.(U \rightarrow T_i) \\
\Gamma & \vdash \sum_{i=1}^{n} \sum_{j=1}^{m} \alpha_i \times \beta_i.V_j \\
\Gamma & \vdash \forall V_j, \exists \vec{W}_j/U[\vec{W}_j/\vec{X}]=V_j \\
\Gamma & \vdash (M)N : \sum_{i=1}^{n} \sum_{j=1}^{m} \alpha_i \times \beta_i.T_i[\vec{W}_j/\vec{X}] \\
\Gamma & \vdash (M)N : \sum_{i=1}^{n} \sum_{j=1}^{m} \alpha_i \times \beta_i.T_i[\vec{W}_j/\vec{X}] \\
\end{align*}
\]
\((\alpha + \beta).T \sqsubseteq \alpha.T + \beta.T'\) if \(\exists M \mid \Gamma \vdash M : T\) and \(\Gamma \vdash M : T'\)
(and its contextual closure)
\[(\alpha + \beta).T \subseteq \alpha.T + \beta.T'\] if \(\exists M / \Gamma \vdash M : T\) and \(\Gamma \vdash M : T'\)
(and its contextual closure)

**Theorem (A weak subject reduction)**

If \(\Gamma \vdash M : T\) and \(M \rightarrow_R N\), then

- if \(R\) is not a factorisation rule: \(\Gamma \vdash N : T\)
- if \(R\) is a factorisation rule: \(\exists S \subseteq T / \Gamma \vdash N : S\)
\[(\alpha + \beta).T \subseteq \alpha.T + \beta.T' \quad \text{if } \exists M / \Gamma \vdash M : T \text{ and } \Gamma \vdash M : T' \]
(and its contextual closure)

**Theorem (A weak subject reduction)**

If \( \Gamma \vdash M : T \) and \( M \rightarrow_R N \), then

- if \( R \) is not a factorisation rule: \( \Gamma \vdash N : T \)
- if \( R \) is a factorisation rule: \( \exists S \subseteq T / \Gamma \vdash N : S \)

**How weak?**

Let \( M \rightarrow N \),

**Subject reduction**

\[ \Gamma \vdash M : T \Rightarrow \Gamma \vdash N : T \]

**Subtyping**

\[ \Gamma \vdash M : T \Rightarrow \Gamma \vdash N : S, \text{ but } S \leq T, \text{ so } \Gamma \vdash N : T \]

**Our theorem**

\[ \Gamma \vdash M : T \Rightarrow \Gamma \vdash N : S, \text{ and } S \subseteq T \]
Confluence and Strong normalisation

In the original untyped setting: “confluence by restrictions”:

\[ Y_B = (\lambda x.(B + (x)x))\lambda x.(B + (x)x) \]

\[ Y_B \rightarrow B + Y_B \rightarrow B + B + Y_B \rightarrow \ldots \]
Confluence and Strong normalisation

In the original **untyped** setting: “confluence by restrictions”:

\[ Y_B = (\lambda x.(B + (x)x))\lambda x.(B + (x)x) \]

\[ Y_B \to B + Y_B \to B + B + Y_B \to \ldots \]

\[ Y_B + (-1).Y_B \to (1 - 1).Y_B \to^* 0 \]
\[ \downarrow \]
\[ B + Y_B + (-1).Y_B \to^* B \]

Solution in the untyped setting:

\[ \alpha.M + \beta.M \to (\alpha + \beta).M \]

only if \( M \) is closed-normal
Confluence and Strong normalisation

In the original untyped setting: “confluence by restrictions”:

\[ Y_B = (\lambda x.(B + (x)x))\lambda x.(B + (x)x) \]

\[ Y_B \rightarrow B + Y_B \rightarrow B + B + Y_B \rightarrow \ldots \]

\[ Y_B + (-1).Y_B \rightarrow (1 - 1).Y_B \rightarrow^* 0 \]

\[ B + Y_B + (-1).Y_B \]

\[ B \]

Solution in the untyped setting:
\[ \alpha.M + \beta.M \rightarrow (\alpha + \beta).M \]
only if \( M \) is closed-normal

In the typed setting: Strong normalisation solves the problem
Theorem (Strong normalisation)

Γ ⊢ M : T ⇒ M strongly normalising.

Proof.

Reducibility candidates method.

Main difficulty: Show that

\[ \{ M_i \}_i \text{ strongly normalizing} \Rightarrow \sum_i \alpha_i . M_i \text{ strongly normalizing} \]

Done by using a measurement on terms decreasing on algebraic rewrites.
Theorem (Confluence)

\[ \forall M / \Gamma \vdash M : T \quad M \rightarrow^* N_1 \quad M \rightarrow^* N_2 \quad \Rightarrow \exists L \text{ such that } N_1 \rightarrow^* L \quad N_2 \rightarrow^* L \]

Proof.

1) **local confluence:**

\[ M \rightarrow N_1 \quad M \rightarrow N_2 \quad \Rightarrow \exists L \text{ such that } N_1 \rightarrow^* L \quad N_2 \rightarrow^* L \]

- Algebraic fragment: Coq proof
- Beta-reduction: Straightforward extension
- Commutation: Induction

2) **Local confluence + Strong normalisation \Rightarrow Confluence** [TeReSe’03]
Expressing matrices and vectors

Two base vectors:

\[ \text{true} = \lambda x. \lambda y. x \]
\[ \text{false} = \lambda x. \lambda y. y \]
Expressing matrices and vectors

**Two base vectors:**
- \texttt{true} = \lambda x.\lambda y.x
- \texttt{false} = \lambda x.\lambda y.y

**Their types:**
- \( \mathbb{T} = \forall XY. X \rightarrow Y \rightarrow X \)
- \( \mathbb{F} = \forall XY. X \rightarrow Y \rightarrow Y \)

\[
\vdash \alpha.\texttt{true} + \beta.\texttt{false} : \alpha.\mathbb{T} + \beta.\mathbb{F}
\]
Expressing matrices and vectors

Two base vectors:

\[ \text{true} = \lambda x. \lambda y. x \]
\[ \text{false} = \lambda x. \lambda y. y \]

Their types:

\[ T = \forall XY. X \to Y \to X \]
\[ F = \forall XY. X \to Y \to Y \]

\[ \vdash \alpha. \text{true} + \beta. \text{false} : \alpha. T + \beta. F \]

Linear map \( U \) s.t.

\[ (U)\text{true} = a.\text{true} + b.\text{false} \]
\[ (U)\text{false} = c.\text{true} + d.\text{false} \]
Expressing matrices and vectors

Two base vectors:

\[ \text{true} = \lambda x. \lambda y. x \]
\[ \text{false} = \lambda x. \lambda y. y \]

Their types:

\[ T = \forall X Y. X \to Y \to X \]
\[ F = \forall X Y. X \to Y \to Y \]

\[ \vdash \alpha. \text{true} + \beta. \text{false} : \alpha. T + \beta. F \]

Linear map \( U \) s.t.

\( (U)\text{true} = a.\text{true} + b.\text{false} \)
\( (U)\text{false} = c.\text{true} + d.\text{false} \)

\[ U := \lambda x. (((x)[a.\text{true} + b.\text{false}])[c.\text{true} + d.\text{false}]) \]

with

\[ [M] := \lambda z. M \]
\[ \{M\} := (M)_\_ \]
\[ \{[M]\} \to M \]

\[ \vdash U : \forall X.((I \to (a.T + b.F)) \to (I \to (c.T + d.F)) \to X) \to X \]
Contributions

- Scalar $\cup$ Additive ("AC, distributive pairs")
  $\Rightarrow$ linear-combination of types

- The typing gives the information of
  "how much the scalars sums" in the normal form

- Weak SR
  $\Rightarrow$ Church style captures better the vectorial structure

- Strong normalisation
  $\Rightarrow$ Confluence without restrictions

- Representation of matrices and vectors