Programming unitary operators in a linear-algebraic typed lambda-calculus

Benoît Valiron, Alejandro Díaz-Caro
Mauricio Guillermo & Alexandre Miquel

UNIVERSIDAD DE LA REPUBLICA URUGUAY

FACULTAD DE INGENIERIA

September 6th, 2018 – 6ta jornada LoCIC — Buenos Aires

Aim of the work

- Present the semantics of a linear algebraic lambda-calculus based on a realizability model that captures a notion of unitarity (ℓ_2-norm)
  - lambda-calculus = functional programming (see next slides)
  - algebraic = linear combinations of terms
    (to represent superpositions of values / superpositions of programs)
  - linear = all functions are linear by construction
- **Main novelty:** The calculus is designed from a realizability model
  (a notion coming from logic, Kleene 1945)
- A language to represent:
  - classical values, classical programs
  - superposition of values, superposition of programs
  - classical programs computing superposition of values
  - superposition of programs computing superposition of values
- A semantics for quantum programming languages (Quipper)

Plan

Plan

The pure λ-calculus (1/2)

- Introduced by Alonzo Church (1903–1995) in the 1930s

  ... to solve Hilbert's *Entscheidungsproblem* (Decision problem)

- Minimal functional programming language. Only:
  - 3 syntactic constructs (variable, λ-abstraction, application)
  - 1 computation rule (β-reduction)

- Actually, the first programming language ever! ...

  ... if we do not count Charles Babbage's (partial) attempt

- Same computation strength as Turing machines

  Turing (1912-1954), who had invented his abstract machines independently, became Church's PhD student in Princeton

- The λ-calculus is now the core of all functional programming languages: Lisp, Scheme, Erlang, OCaml, Haskell, F#, etc.

The pure λ-calculus (2/2)

- Terms of the pure λ-calculus (notation: s, t, u, etc.)

$$\begin{array}{rcll} s, t, u & ::= & x & \text{(variable)} \\ & \mid & \lambda x \,.\, s & \text{(λ-abstraction)} \\ & \mid & t\, u & \text{(application)} \end{array}$$

- Computation rule: $(\lambda x \,.\, s)\, u \;\succ\!\!\succ\; s[x := u]$ (β-reduction)

- Examples:
  - $(\lambda x \,.\, x)\, y \;\succ\!\!\succ\; y$
  - $(\lambda x \,.\, x)\,(\lambda x \,.\, x) \;\succ\!\!\succ\; \lambda x \,.\, x$
  - $(\lambda x \,.\, x\, x)\,(\lambda x \,.\, x) \;\succ\!\!\succ\; (\lambda x \,.\, x)\,(\lambda x \,.\, x) \;\succ\!\!\succ\; \lambda x \,.\, x$
  - $(\lambda x \,.\, x\, x)\,(\lambda x \,.\, x\, x) \;\succ\!\!\succ\; (\lambda x \,.\, x\, x)\,(\lambda x \,.\, x\, x) \;\succ\!\!\succ\; \cdots$
  - $(\lambda z \,.\, z\, z)\,(\lambda f \,.\, \lambda x \,.\, f\,(f\,(f\, x))) \;\succ\!\!\succ\;$
    $\lambda f \,.\, \lambda x \,.\, f\,(f\,(f\,(f\,(f\,(f\,(f\,(f\,(f\,(f\,(f\,(f\,(f\,(f\,(f\,(f\,(f\,(f\,(f\,(f\,(f\, x))))))))))))))))))))$

Adding types

- To avoid undesirable phenomena (self-application, non termination, etc.) it is natural to only consider well-typed λ-terms

- A possible algebra of types (notation: A, B, C, etc.) is:

$$A, B, C \quad ::= \quad \mathbb{U} \quad | \quad A \to B \quad | \quad A \times B \quad | \quad A + B$$

$\mathbb{U}$ is the unit type, from which we can form the type of Booleans $\mathbb{B} := \mathbb{U} + \mathbb{U}$

- Well-typedness of terms is enforced using a typing judgment

$$\Gamma \vdash t : A \qquad \text{("in context } \Gamma, t \text{ has type } A\text{")}$$

where

  - Γ is a typing context, of the form $\Gamma \equiv x_1 : A_1, \ldots, x_n : A_n$
  - t is a term (possibly depending on $x_1, \ldots, x_n$)
  - A is a type

The function type $A \to B$

- $A \to B$ is the type of functions from A to B

  Construction: $\lambda x\,.\,s$ (λ-abstraction)

  Destruction: $t\,u$ (application)

- Computation:

$$(\lambda x\,.\,s)\,u \;\rightarrowtail\; s[x := u] \qquad (\beta\text{-reduction})$$

- Typing rules:

$$\frac{\Gamma, x : A \vdash s : B}{\Gamma \vdash \lambda x\,.\,s : A \to B}$$

$$\frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\,u : B}$$

The Cartesian product $A \times B$

- $A \times B$ is the type of pairs (u, v), where $u : A$ and $v : B$

Construction:	(u, v)	(ordered pair)
Destruction:	$\texttt{let}\ (x, y) = t\ \texttt{in}\ s$	("let" for pairs)

- Computation:

$$\texttt{let}\ (x, y) = (u, v)\ \texttt{in}\ s \ \rightarrowtail\!\!\!\!\!\rightarrowtail\ s[x := u, y := v]$$

- Typing rules:

$$\frac{\Gamma \vdash u : A \qquad \Gamma \vdash v : B}{\Gamma \vdash (u, v) : A \times B}$$

$$\frac{\Gamma \vdash t : A \times B \qquad \Gamma, x : A, y : B \vdash s : C}{\Gamma \vdash \texttt{let}\ (x, y) = t\ \texttt{in}\ s : C}$$

The direct sum $A + B$

- $A + B$ is the direct sum (disjoint union) of types A and B

  Construction: $\mathtt{inl}(u), \quad \mathtt{inr}(v)$

  Destruction: $\mathtt{match}\ t\ \{\mathtt{inl}(x) \mapsto s_1 \mid \mathtt{inr}(y) \mapsto s_2\}$

- Computation:

$$\mathtt{match}\ \mathtt{inl}(u)\ \{\mathtt{inl}(x) \mapsto s_1 \mid \mathtt{inr}(y) \mapsto s_2\} \quad \succ\!\!\succ \quad s_1[x := u]$$
$$\mathtt{match}\ \mathtt{inr}(v)\ \{\mathtt{inl}(x) \mapsto s_1 \mid \mathtt{inr}(y) \mapsto s_2\} \quad \succ\!\!\succ \quad s_2[y := v]$$

- Typing rules:

$$\frac{\Gamma \vdash u : A}{\Gamma \vdash \mathtt{inl}(u) : A + B} \qquad \frac{\Gamma \vdash v : B}{\Gamma \vdash \mathtt{inr}(v) : A + B}$$

$$\frac{\Gamma \vdash t : A + B \qquad \Gamma, x : A \vdash s_1 : C \qquad \Gamma, y : B \vdash s_2 : C}{\Gamma \vdash \mathtt{match}\ t\ \{\mathtt{inl}(x) \mapsto s_1 \mid \mathtt{inr}(y) \mapsto s_2\} : C}$$

The unit type $\mathbb{U}$

- $\mathbb{U}$ is the singleton type (inhabited by a dummy value)

  Construction: $*$ (dummy value)

  Destruction: $t; s$ (sequence)

- Computation: $*; s \succ\!\succ s$

- Typing rules:

$$\frac{}{\Gamma \vdash * : \mathbb{U}} \qquad \frac{\Gamma \vdash t : \mathbb{U} \qquad \Gamma \vdash s : C}{\Gamma \vdash t; s : C}$$

- Combining $\mathbb{U}$ with $+$, we define the type of Booleans:

$$\begin{array}{rcl} \mathbb{B} & := & \mathbb{U} + \mathbb{U} \\ \mathtt{tt} & := & \mathtt{inl}(*) \\ \mathtt{ff} & := & \mathtt{inr}(*) \\ \mathtt{if}\ t\ \{s_1 \mid s_2\} & := & \mathtt{match}\ t\ \{\mathtt{inl}(x) \mapsto x; s_1 \mid \mathtt{inr}(y) \mapsto y; s_2\} \end{array}$$

The simply-typed λ-calculus

- Equipped with a type system such as the one presented above, the λ-calculus enjoys excellent properties:
  - Computation is ultimately deterministic: the computed value does not depend on evaluation strategy (already holds in the untyped case)
  - Types are preserved throughout computations
  - All well-typed computations terminate
- The simply-typed λ-calculus has also good semantics:
  - set-theoretic semantics, denotational semantics, categorical semantics, realizability semantics (cf later)
- Strong relationship with logic:

The Curry-Howard correspondence

$$t \text{ is a program of type } A \quad \equiv \quad t \text{ is a proof of formula } A$$

$$A \to B, \quad A \times B, \quad A + B \quad \equiv \quad A \Rightarrow B, \quad A \wedge B, \quad A \vee B$$

Plan

Aim of the calculus

- **Intuitions:** Terms of the simply-typed λ-calculus represent classical programs computing classical values
- We now want to represent
  - superposition of values
  - classical programs computing superposition of values
  - superposition of programs computing superposition of values
- For that, we extend the λ-calculus with linear combinations

$$s, t \quad ::= \quad x \quad | \quad \lambda x \,.\, t \quad | \quad s\,t \quad | \quad \cdots \quad | \quad \vec{0} \quad | \quad t + u \quad | \quad \alpha \cdot t$$

- Beware!

$$\lambda x \,.\, \left(\frac{1}{\sqrt{2}} \cdot \mathtt{tt} + \frac{1}{\sqrt{2}} \cdot \mathtt{ff}\right) \quad \neq \quad \frac{1}{\sqrt{2}} \cdot (\lambda x \,.\, \mathtt{tt}) + \frac{1}{\sqrt{2}} \cdot (\lambda x \,.\, \mathtt{ff})$$

- We also would like the type system to capture unitary operators (in an infinite dimensional space of values)

Linear combinations and non termination

- **Problem:** Linear combinations badly interact with non termination

$$\text{Let:} \quad Y_t \; := \; (\lambda x\,.\,t + xx)(\lambda x\,.\,t + xx) \qquad (t \text{ fixed term})$$
$$\succ\!\!\!\succ \; t + Y_t$$

$$\text{Hence:} \quad \vec{0} \; = \; Y_t - Y_t \; \succ\!\!\!\succ \; (t + Y_t) - Y_t \; = \; t + \vec{0} \; = \; t$$

$\Rightarrow$ Confluence is lost! (on untyped terms)

- Several solutions have been considered to fix this problem:
  - Restricting the rules of evaluation [Arrighi & Dowek '08, '17]
  - Working with positive coefficients only [Vaux '09]
  - Restricting to well-typed terms [Arrighi & Díaz-Caro '11, '12]
  - Working with weak linear combinations [Valiron '13]

Weak vector spaces (1/2)

Definition (Weak vector space) [Valiron '13]

A weak $\mathbb{C}$-vector space is a commutative monoid $(V, +, \vec{0})$ equipped with a scalar multiplication $(\cdot) : \mathbb{C} \times V \to V$ such that

$$\begin{array}{rcl} 1 \cdot u &=& u \\ \alpha \cdot (\beta \cdot u) &=& \alpha\beta \cdot u \end{array} \qquad \begin{array}{rcl} (\alpha + \beta) \cdot u &=& \alpha \cdot u + \beta \cdot u \\ \alpha \cdot (u + v) &=& \alpha \cdot u + \alpha \cdot v \end{array}$$

for all $u, v \in V$, $\alpha, \beta \in \mathbb{C}$

- **Intuition:** Weak vector space = vector space whose additive structure is not an abelian group, but a commutative monoid

  $\Rightarrow$ vectors do not have an opposite, in general

- In a weak vector space:

$$\alpha \cdot \vec{0} = \vec{0}, \qquad \text{but} \qquad 0 \cdot u \neq \vec{0} \quad \text{and} \quad (-1) \cdot u \neq -u$$

Note that $(-1) \cdot u + u = (-1) \cdot u + 1 \cdot u = (-1 + 1) \cdot u = 0 \cdot u \neq \vec{0}$

Weak vector spaces (2/2)

- Weak vector spaces already occur in mathematics!

Observation: If V and W are (ordinary) $\mathbb{C}$-vector spaces, then the set of all unbounded operators from V to W is a weak $\mathbb{C}$-vector space

- The category of weak vector spaces has excellent properties:
  - It has all limits and all colimits (it is bicomplete)
  - It is monoidal closed ($\otimes \dashv \multimap$)
  - It has all free objects: weak linear combinations, a.k.a. distributions
    (In a distribution, summands of the form $0 \cdot u$ do not cancel)

- We should not think of algebraic programs as bounded operators, not even as totally defined operators, but as abstract unbounded operators (neither total nor continuous)

Plan

Syntax of the calculus

Pure values	v, w	$::=$	$x \mid \lambda x\,.\,\vec{s} \mid *$
			$\mid (v, w) \mid \mathtt{inl}(v) \mid \mathtt{inr}(v)$
Pure terms	s, t	$::=$	$v \mid s\,t \mid t; \vec{s}$
			$\mid \mathtt{let}\ (x_1, x_2) = t\ \mathtt{in}\ \vec{s}$
			$\mid \mathtt{match}\ t\ \{\mathtt{inl}(x_1) \mapsto \vec{s}_1 \mid \mathtt{inr}(x_2) \mapsto \vec{s}_2\}$
Value distr.	$\vec{v}, \vec{w}$	$::=$	$\vec{0} \mid v \mid \vec{v} + \vec{w} \mid \alpha \cdot \vec{v} \qquad (\alpha \in \mathbb{C})$
Term distr.	$\vec{s}, \vec{t}$	$::=$	$\vec{0} \mid t \mid \vec{s} + \vec{t} \mid \alpha \cdot \vec{t} \qquad (\alpha \in \mathbb{C})$

- Term/value distributions are endowed with the equational theory of distributions (summands of the form $0 \cdot t$ do not cancel)
- Syntactic constructs are extended by linearity:

$$
\begin{array}{lll}
(\vec{v}, \vec{w}), & \vec{s}\,\vec{t} & \text{are bilinear} \\
\mathtt{inl}(\vec{v}), & \mathtt{inl}(\vec{v}) & \text{are linear in } \vec{v} \\
\vec{t}; \vec{s}, & \mathtt{let}\ (x, y) = \vec{t}\ \mathtt{in}\ \vec{s}, & \\
\multicolumn{2}{l}{\mathtt{match}\ \vec{t}\ \{\mathtt{inl}(x) \mapsto \vec{s}_1 \mid \mathtt{inr}(y) \mapsto \vec{s}_2\}} & \text{are linear in } \vec{t}
\end{array}
$$

Evaluation

- Evaluation is defined from the ‘atomic’ rules

$$
\begin{array}{rcl}
(\lambda x\,.\,\vec{t})\,v & \succ\!\!\succ & \vec{t}[x := v] \\
*;\vec{s} & \succ\!\!\succ & \vec{s} \\
\mathtt{let}\ (x,y) = (v,w)\ \mathtt{in}\ \vec{s} & \succ\!\!\succ & \vec{s}[x := v, y := w] \\
\mathtt{match}\ \mathtt{inl}(v)\ \{\mathtt{inl}(x) \mapsto \vec{s}_1 \mid \mathtt{inr}(y) \mapsto \vec{s}_2\} & \succ\!\!\succ & \vec{s}_1[x := v] \\
\mathtt{match}\ \mathtt{inr}(v)\ \{\mathtt{inl}(x) \mapsto \vec{s}_1 \mid \mathtt{inr}(y) \mapsto \vec{s}_2\} & \succ\!\!\succ & \vec{s}_2[y := v]
\end{array}
$$

and then extended by linearity (as a relation)

- Call-by-basis strategy = call-by-value + all functions are linear

$$
\begin{array}{rcl}
(\lambda x\,.\,\vec{s})\vec{t} \ \succ\!\!\succ\ (\lambda x\,.\,\vec{s})\Big(\sum_j \beta_j \cdot v_j\Big) & = & \sum_j \beta_j \cdot (\lambda x\,.\,\vec{s})\,v_j \\
& \succ\!\!\succ & \sum_j \beta_j \cdot \vec{s}[x := v_j]
\end{array}
$$

Theorem: Evaluation is confluent (on untyped terms)

Note: Only holds because we are using distributions (= weak linear combinations)

The realizability model

- The weak vector space $\vec{\mathrm{V}}$ of closed value distributions is equipped with the scalar product $\langle \vec{v} \mid \vec{w} \rangle$ and the ℓ_2-seminorm $\|\vec{v}\|$
- All constructions are performed in the unit sphere $\mathcal{S}_1 \subseteq \vec{\mathrm{V}}$

Definition (Types)

A type is a notation A together with a set of unit vectors $[\![A]\!] \subseteq \mathcal{S}_1$

- **Examples:**
  - The type $\mathbb{B}$ (of Booleans) is defined by $[\![\mathbb{B}]\!] := \{\mathtt{tt}, \mathtt{ff}\}$
  - To each type A, we associate the type $\sharp A$ (unitary span of A) that is defined by $[\![\sharp A]\!] := \mathrm{span}([\![A]\!]) \cap \mathcal{S}_1$
  - So that we can form the type $\sharp\mathbb{B}$ (of unitary Booleans)
- To each type A, we associate the realizability predicate

$$\vec{t} \Vdash A \quad :\equiv \quad \exists \vec{v} \in [\![A]\!], \quad \vec{t} \succ\!\succ \vec{v}$$

($\vec{t}$ evaluates to a value distribution of type A)

A simple algebra of types (1/2)

Types $\quad A, B \quad ::= \quad \mathbb{U} \quad | \quad \flat A \quad | \quad \sharp A \quad | \quad A \times B$
$\quad | \quad A + B \quad | \quad A \to B \quad | \quad A \Rightarrow B$

Abbrev.: $\quad \mathbb{B} := \mathbb{U} + \mathbb{U}, \qquad A \otimes B := \sharp(A \times B), \qquad A \oplus B := \sharp(A + B)$

- The unit type $\mathbb{U}$ is defined by $\quad [\![\mathbb{U}]\!] := \{*\}$
- The basis $\flat A$ of a type A is defined by
$$[\![\flat A]\!] := \text{smallest } X \subseteq \mathrm{V} \text{ s.t. } [\![A]\!] \subseteq \mathrm{span}(X)$$
- The unitary span $\sharp A$ of a type A is defined by
$$[\![\sharp A]\!] := \mathrm{span}([\![A]\!]) \cap \mathcal{S}_1$$
- The Cartesian product $A \times B$ of two types A and B is defined by
$$[\![A \times B]\!] := \big\{(\vec{v}, \vec{w}) \ : \ \vec{v} \in [\![A]\!],\ \vec{w} \in [\![B]\!]\big\}$$

A simple algebra of types (2/2)

$$\textbf{Types} \qquad A, B \ ::= \ \mathbb{U} \ \mid \ \flat A \ \mid \ \sharp A \ \mid \ A \times B \ \mid \ A + B \ \mid \ A \to B \ \mid \ A \Rightarrow B$$

Abbrev.: $\mathbb{B} := \mathbb{U} + \mathbb{U}$, $\quad A \otimes B := \sharp(A \times B)$, $\quad A \oplus B := \sharp(A + B)$

- The direct sum $A + B$ of two types A and B is defined by

$$[\![A + B]\!] \ := \ \{\mathtt{inl}(\vec{v}) \ : \ \vec{v} \in [\![A]\!]\} \cup \{\mathtt{inr}(\vec{w}) \ : \ \vec{w} \in [\![B]\!]\}$$

- The pure function space $A \to B$ from A to B is defined by:

$$[\![A \to B]\!] \ := \ \{\lambda x \,.\, \vec{t} \ : \ \forall \vec{v} \in [\![A]\!], \ \vec{t}\langle x := \vec{v}\rangle \Vdash B\}$$

- The unitary function space $A \Rightarrow B$ from A to B is defined by:

$$[\![A \Rightarrow B]\!] \ := \ \Big\{\Big(\textstyle\sum_{i=1}^{n} \alpha_i \cdot \lambda x \,.\, \vec{t}_i\Big) \in \mathcal{S}_1 \ : \ \forall \vec{v} \in [\![A]\!], \ \Big(\textstyle\sum_{i=1}^{n} \alpha_i \cdot \vec{t}_i\langle x := \vec{v}\rangle\Big) \Vdash B\Big\}$$

Properties of the semantic type system

- Recall that: $\vec{t} \Vdash A \quad :\equiv \quad \exists \vec{v} \in [\![A]\!],\ \ \vec{t} \succ\!\!\succ \vec{v}$

Theorem (Representation of unitary functions)

Let $\vec{t}$ be a program distribution

1. $\vec{t} \Vdash \sharp\mathbb{B} \to \sharp\mathbb{B}$ iff t computes a pure function that represents a unitary operator from $\mathbb{C}^2$ to $\mathbb{C}^2$
2. $\vec{t} \Vdash \sharp\mathbb{B} \Rightarrow \sharp\mathbb{B}$ iff t computes a unitary function distribution that represents a unitary operator from $\mathbb{C}^2$ to $\mathbb{C}^2$

- From the realizability relation, we extract a type system based on typing rules that are correct w.r.t. the semantics
- This system is an extension of the simply-typed λ-calculus (that now represents the classical part of the language)
- Moreover, the new type constructs ($\flat A$, $\sharp A$, etc.) allow to capture linearity constraints, and in particular: unitary functions

A possible type system

$$\frac{}{x : A \vdash x : A}\ (\text{Axiom}) \qquad \frac{\Gamma \vdash \vec{t} : A \quad A \leq A'}{\Gamma \vdash \vec{t} : A'}\ (\text{Sub})$$

$$\frac{\Gamma, x : A \vdash \vec{t} : B \quad \flat\Gamma \simeq \Gamma}{\Gamma \vdash \lambda x \,.\, \vec{t} : A \to B}\ (\text{PureLam}) \qquad \frac{\Gamma, x : A \vdash \vec{t} : B}{\Gamma \vdash \lambda x \,.\, \vec{t} : A \Rightarrow B}\ (\text{UnitLam})$$

$$\frac{\Gamma \vdash \vec{s} : A \Rightarrow B \quad \Delta \vdash \vec{t} : A}{\Gamma, \Delta \vdash \vec{s}\,\vec{t} : B}\ (\text{App})$$

$$\frac{}{\vdash\ * : \mathbb{U}}\ (\text{Void}) \qquad \frac{\Gamma \vdash \vec{t} : \mathbb{U} \quad \Delta \vdash \vec{s} : A}{\Gamma, \Delta \vdash \vec{t}; \vec{s} : A}\ (\text{Seq}) \qquad \frac{\Gamma \vdash \vec{t} : \sharp\mathbb{U} \quad \Delta \vdash \vec{s} : \sharp A}{\Gamma, \Delta \vdash \vec{t}; \vec{s} : \sharp A}\ (\text{SeqSharp})$$

$$\frac{\Gamma \vdash \vec{v} : A \quad \Delta \vdash \vec{w} : B}{\Gamma, \Delta \vdash (\vec{v}, \vec{w}) : A \times B}\ (\text{Pair}) \qquad \frac{\Gamma \vdash \vec{t} : A \times B \quad \Delta, x : A, y : B \vdash \vec{s} : C}{\Gamma, \Delta \vdash \texttt{let}\ (x, y) = \vec{t}\ \texttt{in}\ \vec{s} : C}\ (\text{LetPair})$$

$$\frac{\Gamma \vdash \vec{t} : A \otimes B \quad \Delta, x : \sharp A, y : \sharp B \vdash \vec{s} : \sharp C}{\Gamma, \Delta \vdash \texttt{let}\ (x, y) = \vec{t}\ \texttt{in}\ \vec{s} : \sharp C}\ (\text{LetTens})$$

$$\frac{\Gamma \vdash \vec{t} : B \quad \flat A \simeq A}{\Gamma, x : A \vdash \vec{t} : B}\ (\text{Weak}) \qquad \frac{\Gamma, x : A, y : A \vdash \vec{t} : B \quad \flat A \simeq A}{\Gamma, x : A \vdash \vec{t}[y := x] : B}\ (\text{Contr})$$

$+$ many other typing rules / subtyping rules