Quantum Programming with Inductive Datatypes: Causality and Affine Type Theory

Vladimir Zamdzhiev

Université de Lorraine, CNRS, Inria, LORIA, F 54000 Nancy, France

Joint work with Romain Péchoux, Simon Perdrix and Mathys Rennela

Quantum Programming Overview

There are different paradigms:

- Circuit description languages. Focus on generation of circuits. Examples:
  - QWIRE (Paykin, Rand, Zdancewic. POPL 2017).
  - EWIRE (Rennela, Staton. MFPS 2017).
  - Proto-Quipper-M (Rios, Selinger. QPL 2017).
  - ECLNL (Lindenhovius, Mislove, Zamdzhiev. LICS 2018).
- Linear-algebraic lambda calculi. Superposition of terms. Examples:
  - Lineal (Arrighi, Dowek. LMCS 2017).
  - Lambda-S (Díaz-Caro, Malherbe. LSFA 2018).
- Quantum programming languages. Run on quantum hardware. Examples:
  - QPL (Selinger. MSCS. (2004)).
  - Quantum Lambda Calculus (Pagani, Selinger, Valiron. POPL 2014).

Introduction

- Inductive datatypes are an important programming concept.
  - Data structures such as natural numbers, lists, etc.; manipulate variable-sized data.
- First detailed treatment of inductive datatypes for quantum programming.
- Most type systems for quantum programming are linear (copying and discarding are restricted).
- We show that *affine* type systems (only copying is restricted) are very appropriate.
- Some of the main challenges in designing a (categorical) model for the language stem from substructural limitations imposed by quantum mechanics:
  - How to identify the causal (i.e. discardable) quantum data?
  - How do we copy (infinite-dimensional) classical datatypes?

Overview of Talk

- Extend QPL with inductive datatypes and a copy operation for classical data;
- An affine type system with first-order procedure calls. No !-modality required.
- An elegant and type safe operational semantics based on *finite-dimensional* quantum operations and classical control structures;
- A *physically natural* denotational model for quantum programming using von Neumann algebras;
- Several novel results in quantum programming:
  - Denotational semantics for user-defined inductive datatypes. We also describe the comonoid structure of classical (inductive) types.
  - Invariance of the denotational semantics w.r.t big-step reduction. This implies adequacy at all types.

Outline : Inductive Datatypes

- Syntactically, everything is very straightforward.
- Operationally, the small-step semantics can be described using finite-dimensional superoperators together with classical control structures.
- Denotationally, we have to move away from finite-dimensional quantum computing:
    - E.g. the recursive domain equation $X \cong \mathbb{C} \oplus X$ cannot be solved in finite dimensions.

Outline : Inductive Datatypes

- Syntactically, everything is very straightforward.
- Operationally, the small-step semantics can be described using finite-dimensional superoperators together with classical control structures.
- Denotationally, we have to move away from finite-dimensional quantum computing:
    - E.g. the recursive domain equation $X \cong \mathbb{C} \oplus X$ cannot be solved in finite dimensions.
    - But it can be solved in infinite dimensions: take $X = \bigoplus_\omega \mathbb{C}$.

Outline : Inductive Datatypes

- Syntactically, everything is very straightforward.
- Operationally, the small-step semantics can be described using finite-dimensional superoperators together with classical control structures.
- Denotationally, we have to move away from finite-dimensional quantum computing:
    - E.g. the recursive domain equation $X \cong \mathbb{C} \oplus X$ cannot be solved in finite dimensions.
    - But it can be solved in infinite dimensions: take $X = \bigoplus_{\omega} \mathbb{C}$.
- Naturally, we use (infinite-dimensional) W*-algebras (aka von Neumann algebras), which were introduced by von Neumann to aid his study of quantum mechanics.

Outline : Causality and Linear vs Affine Type Systems

- Linear type system : only non-linear variables may be copied or discarded.
- Affine type system : only non-linear variables may be copied; all variables may be discarded.
- Syntactically, all types have an elimination rule in quantum programming.
- Operationally, all computational data may be discarded by a mix of partial trace and classical discarding.
- Denotationally, we can construct discarding maps at all types (quantum and classical) and prove the interpretation of the values is *causal*.
  - This is achieved by considering different kinds of structure-preserving superoperators.
- The "no deletion" theorem of QM is irrelevant for quantum programming. We work entirely within W*-algebras, so no violation of QM.

QPL - a Quantum Programming Language

- As a basis for our development, we describe a quantum programming language based on the language QPL of Selinger (which is also affine).
- The language is equipped with a type system which guarantees no runtime errors can occur.
- QPL is not a higher-order language: it has procedures, but does not have lambda abstractions.
- We extend QPL with :
  - Inductive datatypes.
  - Copy operation on classical types.

Syntax

- The syntax (excerpt) of our language is presented below. The formation rules are omitted. Notice there is no ! modality.

Type Var.	X, Y, Z		
Term Var.	x, q, b, u		
Procedure Var.	f, g		
Types	A, B	$::=$	$X \mid I \mid \mathbf{qbit} \mid A + B \mid A \otimes B \mid \mu X.A$
Classical Types	P, R	$::=$	$X \mid I \mid P + R \mid P \otimes R \mid \mu X.P$
Variable contexts	Γ, Σ	$::=$	$x_1 : A_1, \ldots, x_n : A_n$
Procedure cont.	Π	$::=$	$f_1 : A_1 \to B_1, \ldots, f_n : A_n \to B_n$

Some Definable Types

- The type of bits is defined as $\mathbf{bit} := I + I$.
- The type of natural numbers is defined as $\mathbf{Nat} := \mu X.I + X$.
- The type of lists of qubits is defined as $\mathbf{QList} = \mu X.I + \mathbf{qbit} \otimes X$.

Syntax (contd.)

$$
\begin{array}{llll}
\text{Terms} & M, N & ::= & \textbf{new unit } u \mid \textbf{new qbit } q \mid \textbf{discard } x \mid y = \textbf{copy } x \\
& & & q_1, \ldots, q_n \mathrel{*}= U \mid \; M; N \mid \textbf{skip} \mid \\
& & & b = \textbf{measure } q \mid \textbf{while } b \textbf{ do } M \mid \\
& & & x = \textbf{left}_{A,B} M \mid x = \textbf{right}_{A,B} M \mid \\
& & & \textbf{case } y \textbf{ of } \{\textbf{left } x_1 \to M \mid \textbf{right } x_2 \to N\} \\
& & & x = (x_1, x_2) \mid (x_1, x_2) = x \mid \\
& & & y = \textbf{fold } x \mid y = \textbf{unfold } x \mid \\
& & & \textbf{proc } f \; x : A \to y : B \; \{M\} \mid y = f(x)
\end{array}
$$

- A *term judgement* is of the form $\Pi \vdash \langle \Gamma \rangle \; P \; \langle \Sigma \rangle$, where all types are closed and all contexts are well-formed. It states that the term is well-formed in procedure context Π, given input variables $\langle \Gamma \rangle$ and output variables $\langle \Sigma \rangle$.
- A *program* is a term P, such that $\cdot \vdash \langle \cdot \rangle \; P \; \langle \Gamma \rangle$, for some (unique) Γ.

Syntax : qubits

The type of bits is (canonically) defined to be $\mathbf{bit} \coloneqq I + I$.

$$\dfrac{}{\Pi \vdash \langle \Gamma \rangle \ \textbf{new qbit} \ q \ \langle \Gamma, q : \mathbf{qbit} \rangle} \ (\text{qbit})$$

$$\dfrac{}{\Pi \vdash \langle \Gamma, q : \mathbf{qbit} \rangle \ b = \textbf{measure} \ q \ \langle \Gamma, b : \mathbf{bit} \rangle} \ (\text{measure})$$

$$\dfrac{S \text{ is a unitary of arity } n}{\Pi \vdash \langle \Gamma, q_1 : \mathbf{qbit}, \ldots, q_n : \mathbf{qbit} \rangle \ q_1, \ldots, q_n \mathrel{*=} S \ \langle \Gamma, q_1 : \mathbf{qbit}, \ldots, q_n : \mathbf{qbit} \rangle} \ (\text{unitary})$$

Syntax : copying

$$\frac{P \text{ is a classical type}}{\Pi \vdash \langle \Gamma, x : P \rangle \; y = \textbf{copy } x \; \langle \Gamma, x : P, y : P \rangle} \text{ (copy)}$$

Syntax : discarding (affine vs linear)

- If we wish to have a linear type system:

$$\frac{}{\Pi \vdash \langle \Gamma, x : I \rangle \ \mathbf{discard}\ x \ \langle \Gamma \rangle}\ (\text{discard})$$

- If we wish to have an affine type system:

$$\frac{}{\Pi \vdash \langle \Gamma, x : A \rangle \ \mathbf{discard}\ x \ \langle \Gamma \rangle}\ (\text{discard})$$

- Since all types have an elimination rule, an affine type system is obviously more convenient.

Operational Semantics

- Operational semantics is a formal specification which describes how a program is executed in a mathematically precise way.
- A *configuration* is a tuple (M, V, Ω, ρ), where:
  - M is a well-formed term $\Pi \vdash \langle \Gamma \rangle \; M \; \langle \Sigma \rangle$.
  - V is a *value assignment*. Each input variable of M is assigned a value, e.g. $V = \{x = \mathit{zero}, y = \mathit{cons}(\mathit{one},\ \mathit{nil})\}$.
  - Ω is a *procedure store*. It keeps track of the defined procedures by mapping procedure variables to their *procedure bodies* (which are terms).
  - ρ is the (possibly not normalized) density matrix computed so far.
  - This data is subject to additional well-formedness conditions (omitted).

Operational Semantics (contd.)

- Program execution is (formally) modelled as a nondeterministic reduction relation on configurations $(M, V, \Omega, \rho) \rightsquigarrow (M', V', \Omega', \rho')$.
- The reduction relation may equivalently be seen as probabilistic, because the probability of the reduction is encoded in ρ'.
- The probability of the above reduction is then $\mathrm{tr}(\rho')/\mathrm{tr}(\rho)$, which is consistent with the Born rule of quantum mechanics.
- The only source of probabilistic behaviour is given by quantum measurements.

A simple program and its execution graph

```
while b do {
  new qbit q;
  q *= H;
  discard b;
  b = measure q
}
```

$$\begin{tikzcd}[column sep=small]
 & & & (M \mid \mathtt{b} = \mathtt{tt} \mid \cdot \mid 1) \arrow[dl, squiggly, "*"'] \arrow[dr, squiggly, "*"] & \\
 & & (M \mid \mathtt{b} = \mathtt{tt} \mid \cdot \mid 0.5) \arrow[dl, squiggly, "*"'] \arrow[dr, squiggly, "*"] & & (\textbf{skip} \mid \mathtt{b} = \mathtt{ff} \mid \cdot \mid 0.5) \\
 & (M \mid \mathtt{b} = \mathtt{tt} \mid \cdot \mid 0.25) \arrow[dl, squiggly, "*"'] \arrow[dr, squiggly, "*"] & & (\textbf{skip} \mid \mathtt{b} = \mathtt{ff} \mid \cdot \mid 0.25) & \\
\iddots & & (\textbf{skip} \mid \mathtt{b} = \mathtt{ff} \mid \cdot \mid 0.125) & &
\end{tikzcd}$$

A simple program for GHZ_n

```
proc GHZnext(l : ListQ) -> l : ListQ {
  new qbit q;
  case l of
      nil -> q *= H;
             l = q :: nil
    | q’ :: l’ -> q’,q *= CNOT;
                  l = q :: q’ :: l’
}

proc GHZ(n : Nat) -> l : ListQ {
  case n of
      zero -> l = nil
    | s(n’) -> l = GHZnext(GHZ(n’))
}
```

An example execution

$$(\texttt{l = GHZ(n)} \mid \texttt{n} = \texttt{s(s(s(zero)))} \mid \Omega \mid 1)$$

$$\rightsquigarrow^{*}$$

$$(\texttt{l = GHZnext(l)} \mid \texttt{l} = 2 :: 1 :: \texttt{nil} \mid \Omega \mid \gamma_2)$$

$$\rightsquigarrow$$

$$(\texttt{new qbit q}; \cdots \mid \texttt{l} = 2 :: 1 :: \texttt{nil} \mid \Omega \mid \gamma_2)$$

$$\rightsquigarrow$$

$$(\texttt{case l of } \cdots \mid \texttt{l} = 2 :: 1 :: \texttt{nil}, \texttt{q} = 3 \mid \Omega \mid \gamma_2 \otimes |0\rangle\langle 0|)$$

$$\rightsquigarrow^{*}$$

$$(\texttt{q',q *=CNOT}; \cdots \mid \texttt{l'} = 1 :: \texttt{nil}, \texttt{q} = 3, \texttt{q'} = 2 \mid \Omega \mid \gamma_2 \otimes |0\rangle\langle 0|)$$

$$\rightsquigarrow$$

$$(\texttt{l = q :: q' :: l'} \mid \texttt{l'} = 1 :: \texttt{nil}, \texttt{q} = 3, \texttt{q'} = 2 \mid \Omega \mid \gamma_3)$$

$$\rightsquigarrow^{*}$$

$$(\textbf{skip} \mid \texttt{l} = 3 :: 2 :: 1 :: \texttt{nil} \mid \Omega \mid \gamma_3)$$

The Denotational Model

- Our denotational model is based on W*-algebras (aka von Neumann algebras).
- A W*-algebra is a complex vector space A, equipped with:
  - A bilinear multiplication $(- \cdot -) : A \times A \to A$ (written as juxtaposition).
  - A submultiplicative norm $\| - \| : A \to \mathbb{R}_{\geq 0}$, i.e. $\forall x, y \in A : \|xy\| \leq \|x\|\|y\|$.
  - An involution $(-)^* : A \to A$ such that $(x^*)^* = x$, $(x+y)^* = (x^* + y^*)$, $(xy)^* = y^* x^*$ and $(\lambda x)^* = \overline{\lambda} x^*$.
  - Subject to some additional conditions (omitted here).
- Example: The set of complex numbers $\mathbb{C}$.
- Example: The algebra $M_n(\mathbb{C})$ of $n \times n$ complex matrices.

The Denotational Model (contd.)

- We need to consider two kinds of structure-preserving linear maps.
- A linear map $f : A \to B$ is MIU, if it preserves multiplication, involution and the unit. These maps are known as *-homomorphisms.
- A linear map $f : A \to B$ is CPSU, if it is completely-positive and subunital $(0 \leq f(1) \leq 1)$.
- Every MIU map is also CPSU.
- Values are interpreted as MIU-maps, whereas computations are interpreted as CPSU-maps.

Categorical Structure of W*-algebras

- Let $\mathbf{W}^*_{\mathrm{CPSU}}$ be the category of W*-algebras and CPSU-maps.
- Let $\mathbf{W}^*_{\mathrm{MIU}}$ be the category of W*-algebras and MIU-maps.
- For the denotational semantics, we have to adopt the *Heisenberg picture* of quantum mechanics:
  - Categorically, this means our interpretations live in the opposite categories.
  - Values are interpreted as morphisms in $\mathbf{V} := (\mathbf{W}^*_{\mathrm{MIU}})^{\mathrm{op}}$.
  - Computations are interpreted as morphisms in $\mathbf{C} := (\mathbf{W}^*_{\mathrm{CPSU}})^{\mathrm{op}}$.
- Both $\mathbf{C}$ and $\mathbf{V}$ are symmetric monoidal and have small coproducts.
- $\mathbf{C}$ is also pointed and $\mathbf{DCPO}_{\perp!}$-enriched.
- There exist symmetric monoidal adjunctions

$$\begin{tikzcd} \mathbf{Set} \arrow[r, bend left, "F"] \arrow[r, phantom, "\perp"] & \mathbf{V} \arrow[l, bend left, "G"] \arrow[r, hook, bend left, "J"] \arrow[r, phantom, "\perp"] & \mathbf{C} \arrow[l, bend left, "R"] \end{tikzcd} .$$

Interpretation of Types

- The category $\mathbf{V}$ is also symmetric monoidal closed and cocomplete, which is ideal for the interpretation of inductive datatypes.
- Inductive datatypes are interpreted by constructing (parameterised) initial algebras within $\mathbf{V}$.
- Every open type $\Theta \vdash A$ is interpreted as an ω-cocontinuous functor $[\![\Theta \vdash A]\!] : \mathbf{V}^{|\Theta|} \to \mathbf{V}$.
- Every closed type A is interpreted as an object $[\![A]\!] \in \mathrm{Ob}(\mathbf{V}) = \mathrm{Ob}(\mathbf{C})$.

Copying of Classical Information

- We do not use linear logic based approaches that rely on a !-modality.
- Instead, we use techniques based on recent work [LMZ19]:
  - Abstract categorical models for linear/non-linear recursive types (! and $\multimap$ allowed).
  - Implicit copying and discarding for non-linear recursive types (difficult to model denotationally).
  - New methods for solving recursive domain equations.
  - New coherence properties for parameterised initial algebras.
- Extended version of [LMZ19] submitted to LMCS (60 pages). **arXiv:1906.09503**
- The present treatment is actually a simple special case of [LMZ19], because here we do not use ! or $\multimap$.

[LMZ19] Bert Lindenhovius, Michael Mislove and Vladimir Zamdzhiev. Mixed Linear and Non-linear Recursive Types. ICFP'19.

Copying of Classical Information (contd.)

- For every classical type $\Theta \vdash P$ we present a classical interpretation $(\!|\Theta \vdash P|\!) : \mathbf{Set}^{|\Theta|} \to \mathbf{Set}$ which we show satisfies

$$
\begin{tikzcd}
\mathbf{Set}^{|\Theta|} \arrow[r, "F^{\times|\Theta|}"] \arrow[d, "(\!|\Theta \vdash P|\!)"'] \arrow[dr, phantom, "\cong"] & \mathbf{V}^{|\Theta|} \arrow[d, "[\![\Theta \vdash P]\!]"] \\
\mathbf{Set} \arrow[r, "F"'] & \mathbf{V}
\end{tikzcd}
$$

Copying of Classical Information (contd.)

- For every classical type $\Theta \vdash P$ we present a classical interpretation $(\!|\Theta \vdash P|\!) : \mathbf{Set}^{|\Theta|} \to \mathbf{Set}$ which we show satisfies

$$
\begin{tikzcd}
\mathbf{Set}^{|\Theta|} \arrow[r, "F^{\times|\Theta|}"] \arrow[d, "(\!|\Theta \vdash P|\!)"'] \arrow[dr, phantom, "\cong"] & \mathbf{V}^{|\Theta|} \arrow[d, "[\![\Theta \vdash P]\!]"] \\
\mathbf{Set} \arrow[r, "F"'] & \mathbf{V}
\end{tikzcd}
$$

- For closed types we get an isomorphism $F(\!|P|\!) \cong [\![P]\!]$.

Copying of Classical Information (contd.)

- For every classical type $\Theta \vdash P$ we present a classical interpretation $(\!|\Theta \vdash P|\!) : \mathbf{Set}^{|\Theta|} \to \mathbf{Set}$ which we show satisfies

$$\begin{tikzcd}
\mathbf{Set}^{|\Theta|} \arrow[r, "F^{\times|\Theta|}"] \arrow[d, "(\!|\Theta \vdash P|\!)"'] \arrow[dr, phantom, "\cong"] & \mathbf{V}^{|\Theta|} \arrow[d, "[\![\Theta \vdash P]\!]"] \\
\mathbf{Set} \arrow[r, "F"'] & \mathbf{V}
\end{tikzcd}$$

- For closed types we get an isomorphism $F(\!|P|\!) \cong [\![P]\!]$.
- This isomorphism allows us to define a cocommutative comonoid structure.

Copying of Classical Information (contd.)

- For every classical type $\Theta \vdash P$ we present a classical interpretation $(\!|\Theta \vdash P|\!) : \mathbf{Set}^{|\Theta|} \to \mathbf{Set}$ which we show satisfies

$$\begin{tikzcd}
\mathbf{Set}^{|\Theta|} \arrow[r, "F^{\times|\Theta|}"] \arrow[d, "(\!|\Theta \vdash P|\!)"'] \arrow[dr, phantom, "\cong"] & \mathbf{V}^{|\Theta|} \arrow[d, "[\![\Theta \vdash P]\!]"] \\
\mathbf{Set} \arrow[r, "F"'] & \mathbf{V}
\end{tikzcd}$$

- For closed types we get an isomorphism $F(\!|P|\!) \cong [\![P]\!]$.
- This isomorphism allows us to define a cocommutative comonoid structure.
- The classical values (including folds) are then comonoid homomorhpisms.

Discarding of (Quantum) Information

- Discardable operations are called *causal*.
- In **V**, the tensor unit I is terminal, so the discarding map is $\diamond_A : A \to I$.
- We show that all *values* are *-homomorphisms and therefore causal.
    - This includes folds, because type interpretation is done in **V**.

Interpretation of Terms and Configurations

- Most of the difficulty is in defining the interpretation of types and the substructural operations.
- Terms are interpreted as Scott-continuous functions $[\![\Pi \vdash \langle\Gamma\rangle\ M\ \langle\Sigma\rangle]\!] : [\![\Pi]\!] \to \mathbf{C}([\![\Gamma]\!], [\![\Sigma]\!])$.
- Configurations are interpreted as states $[\![(M, V, \Omega, \rho)]\!] : I \to [\![\Sigma]\!]$.
- This is fairly straightforward.

Soundness

Theorem (Soundness)

For any non-terminal configuration $\mathcal{C}$, the denotational interpretation is invariant under (small-step) program execution:

$$[\![\mathcal{C}]\!] = \sum_{\mathcal{C} \rightsquigarrow \mathcal{D}} [\![\mathcal{D}]\!]$$

Remark: The above sum is convex.

Invariance w.r.t big-step reduction

- Can the interpretation of a configuration be recovered from the (potentially infinite) set of its terminal reducts?

Theorem (Big-step invariance)

For any configuration $\mathcal{C}$:

$$[\![\mathcal{C}]\!] = \sum_{\substack{\mathcal{C} \Downarrow \mathcal{T} \\ \mathcal{T} \text{ terminal}}} [\![\mathcal{T}]\!]$$

- This is a novel result for quantum programming.
- This is a strong result, because it immediately implies computational adequacy.
- Useful for a collecting semantics for quantum (relational) program logics.

Computational Adequacy

- Can we provide a denotational formulation for the probability of termination?

Theorem (Computational Adequacy)

For any normalised configuration $\mathcal{C}$:

$$(\diamond \circ [\![\mathcal{C}]\!])\,(1) = \mathrm{Halt}(\mathcal{C})$$

Conclusion and Future Work

- We described a *natural* model based on (infinite-dimensional) W*-algebras.
- Use affine type systems instead of linear ones for quantum programming.
- Novel results for quantum programming:
  - Inductive datatypes.
  - Invariance of the interpretation w.r.t big-step reduction. This implies computational adequacy at all types.
- No !-modality:
  - Comonoid structure of all classical types using the categorical structure of models of intuitionistic linear logic.
  - Causal (discarding) structure by separating the values and computations into suitable categories.
- Do lambda abstractions in quantum programming admit a physical interpretation?
- Future work: use the model for abstract interpretation.

Thank you for your attention!