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 \Gamma, \Sigma ::= x_1 : A_1, \dots, x_n : A_n
Procedure cont. \Pi ::= f_1 : A_1 \rightarrow B_1, \dots, f_n : A_n \rightarrow B_n
```

Some Definable Types

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

Syntax (contd.)

```
Terms M,N ::= new unit u \mid new qbit q \mid discard x \mid y = copy x \mid q_1,\ldots,q_n*=U \mid M;N \mid skip \mid b = measure q \mid while b do M \mid x = left<sub>A,B</sub>M \mid x = right<sub>A,B</sub>M \mid case y of {left x_1 \rightarrow M \mid right x_2 \rightarrow N} x = (x_1,x_2) \mid (x_1,x_2) = x \mid y = fold x \mid y = unfold x \mid proc f : x \mid A \rightarrow y \mid B \mid M \mid y = f(x)
```

- 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 **bit** := I + I. $\frac{}{\prod \vdash \langle \Gamma \rangle \text{ new qbit } q \ \langle \Gamma, q : \text{qbit} \rangle} \text{ (qbit)}$ S is a unitary of arity n $\frac{S \text{ is a unitary of arity } n}{\prod \vdash \langle \Gamma, q_1 : \mathsf{qbit}, \dots, q_n : \mathsf{qbit} \rangle \ q_1, \dots, q_n *= S \ \langle \Gamma, q_1 : \mathsf{qbit}, \dots, q_n : \mathsf{qbit} \rangle} \text{ (unitary)}$

Syntax: copying

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

Syntax: discarding (affine vs linear)

• If we wish to have a linear type system:

$$\frac{1}{\prod \vdash \langle \Gamma, x : I \rangle \text{ discard } x \langle \Gamma \rangle} \text{ (discard)}$$

• If we wish to have an affine type system:

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

 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. $V = \{x = zero, y = cons(one, nil)\}.$
 - 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 $tr(\rho')/tr(\rho)$, 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
}
```

A simple program for GHZ_n

```
proc GHZnext(1 : ListQ) -> 1 : ListQ {
  new qbit q;
  case 1 of
      nil \rightarrow q *= H;
              l = q :: nil
    | q' :: 1' -> q', q *= CNOT;
                    1 = q :: q' :: 1'
proc GHZ(n : Nat) -> 1 : ListQ {
  case n of
      zero \rightarrow 1 = nil
    | s(n') \rangle = GHZnext(GHZ(n'))
```

An example execution

```
(1 = GHZ(n) \mid n = s(s(s(zero))) \mid \Omega \mid 1)
               (1 = GHZnext(1) | 1 = 2 :: 1 :: nil | \Omega | \gamma_2)
                (new qbit q: \cdots \mid 1 = 2 :: 1 :: nil \mid \Omega \mid \gamma_2)
     (case 1 of \cdots | 1 = 2 :: 1 :: nil, q = 3 | \Omega | \gamma_2 \otimes |0\rangle \langle 0|)
(q',q *=CNOT; \cdots \mid 1' = 1 :: nil, q = 3, q' = 2 \mid \Omega \mid \gamma_2 \otimes |0\rangle \langle 0|)
  (1 = q :: q' :: 1' | 1' = 1 :: nil, q = 3, q' = 2 | \Omega | \gamma_3)
                     (skip \mid 1 = 3 :: 2 :: 1 :: 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 $(-\cdot -): A \times A \rightarrow A$ (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 $(\bar{x^*})^* = x$, $(x+y)^* = (x^*+y^*)$, $(xy)^* = y^*x^*$ and $(\lambda x)^* = \bar{\lambda}x^*$.
 - Subject to some additional conditions (omitted here).
- Example: The set of complex numbers 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 $(0 \le f(1) \le 1)$.
- 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 $\mathbf{W}_{\mathsf{CPSU}}^*$ be the category of W*-algebras and CPSU-maps.
- Let W^{*}_{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 V := (W^{*}_{MII})^{op}.
 - Computations are interpreted as morphisms in $\mathbf{C} \coloneqq (\mathbf{W}_{\mathsf{CPSU}}^*)^{\mathrm{op}}$.
- Both C and V are symmetric monoidal and have small coproducts.
- C is also pointed and DCPO_{⊥!}-enriched.
- There exist symmetric monoidal adjunctions

$$\mathbf{Set} \xrightarrow{F} \mathbf{V} \xleftarrow{J} \mathbf{C} .$$

Interpretation of Types

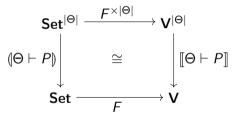
- The category V is also symmetric monoidal closed and cocomplete, which is ideal for the interpretation of inductive datatypes.
- Inductive datatypes are interpreted by constructing (parameterised) initial algebras within V.
- Every open type $\Theta \vdash A$ is interpreted as an ω -cocontinuous functor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

Copying of Classical Information

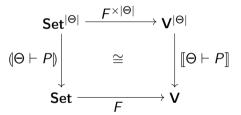
- We do not use linear logic based approaches that rely on a !-modality.
- Instead, we use techniques based on recent work [LMZ19]:
 - Abstract categorical models for linear/non-linear recursive types (! and → allowed).
 - Implicit copying and discarding for non-linear recursive types (difficult to model 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 $-\circ$.

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

• 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

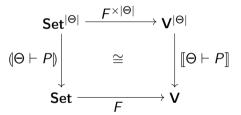


• For every classical type $\Theta \vdash P$ we present a classical interpretation $(\Theta \vdash P) : \mathbf{Set}^{|\Theta|} \to \mathbf{Set}$ which we show satisfies



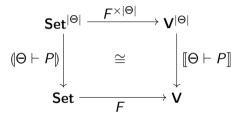
• For closed types we get an isomorphism $F(P) \cong [P]$.

• 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



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

• 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



- 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.
- Configurations are interpreted as states $[\![(M,V,\Omega,\rho)]\!]:I\to [\![\Sigma]\!].$
- This is fairly straightforward.

Soundness

Theorem (Soundness)

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

$$\llbracket \mathcal{C}
rbracket = \sum_{\mathcal{C} \leadsto \mathcal{D}} \llbracket \mathcal{D}
rbracket$$

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 C:

$$\llbracket \mathcal{C}
rbracket = \sum_{\substack{\mathcal{C} \Downarrow \mathcal{T} \ \textit{terminal}}} \llbracket \mathcal{T}
rbracket$$

- 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 $\mathcal C$:

$$(\diamond \circ \llbracket \mathcal{C} \rrbracket) (1) = \operatorname{Halt}(\mathcal{C})$$

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!