

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
29.11.2019



Universiteit
Leiden

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.

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.

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	$::= x_1 : A_1, \dots, x_n : A_n$
Procedure cont.	Π	$::= f_1 : A_1 \rightarrow B_1, \dots, f_n : A_n \rightarrow B_n$

Some Definable Types

- The type of bits is defined as $\mathbf{bit} := I + I$.
- 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.)

Terms $M, N ::=$ **new unit** u | **new qbit** q | **discard** x | $y =$ **copy** x
 q_1, \dots, q_n * $= U$ | $M; N$ | **skip** |
 $b =$ **measure** q | **while** b **do** M |
 $x =$ **left** $_{A,B}$ M | $x =$ **right** $_{A,B}$ M |
case y **of** {**left** $x_1 \rightarrow M$ | **right** $x_2 \rightarrow N$ }
 $x = (x_1, x_2)$ | $(x_1, x_2) = x$ |
 $y =$ **fold** x | $y =$ **unfold** x |
proc f $x : A \rightarrow y : B$ { M } | $y = f(x)$

- 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 $\mathbf{bit} := I + I$.

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

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

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

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:

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

- 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

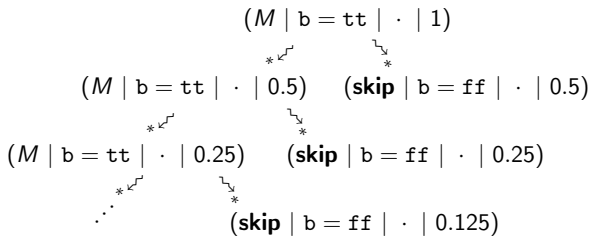
- Operational semantics is a formal specification which describes how a program is executed in a mathematically precise way.
- A *configuration* is a tuple (M, V, Ω, ρ) , where:
 - M is a well-formed term $\Pi \vdash \langle \Gamma \rangle M \langle \Sigma \rangle$.
 - V is a *value assignment*. Each input variable of M is assigned a value, e.g. $V = \{x = \text{zero}, y = \text{cons}(\text{one}, \text{nil})\}$.
 - 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 $\text{tr}(\rho')/\text{tr}(\rho)$, 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;  
  b = measure q  
}
```



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

$$\begin{array}{c}
 (1 = \text{GHZ}(n) \mid n = \text{s}(\text{s}(\text{s}(\text{zero}))) \mid \Omega \mid 1) \\
 \Downarrow \\
 (1 = \text{GHZnext}(1) \mid 1 = 2 :: 1 :: \text{nil} \mid \Omega \mid \gamma_2) \\
 \Downarrow \\
 (\text{new qbit } q; \dots \mid 1 = 2 :: 1 :: \text{nil} \mid \Omega \mid \gamma_2) \\
 \Downarrow \\
 (\text{case } 1 \text{ of } \dots \mid 1 = 2 :: 1 :: \text{nil}, q = 3 \mid \Omega \mid \gamma_2 \otimes |0\rangle \langle 0|) \\
 \Downarrow \\
 (q', q \text{ *=CNOT}; \dots \mid 1' = 1 :: \text{nil}, q = 3, q' = 2 \mid \Omega \mid \gamma_2 \otimes |0\rangle \langle 0|) \\
 \Downarrow \\
 (1 = q :: q' :: 1' \mid 1' = 1 :: \text{nil}, q = 3, q' = 2 \mid \Omega \mid \gamma_3) \\
 \Downarrow \\
 (\text{skip} \mid 1 = 3 :: 2 :: 1 :: \text{nil} \mid \Omega \mid \gamma_3)
 \end{array}$$

Denotational Semantics

- Denotational semantics is a mathematical interpretation of programs which is invariant w.r.t program execution.
- Denotationally, we have to move away from finite-dimensional quantum computing:
 - To model inductive datatypes we have to be able to solve certain *recursive domain equations*.
 - The simplest non-trivial one: $X \cong \mathbb{C} \oplus X$ cannot be solved in finite dimensions.

Denotational Semantics

- Denotational semantics is a mathematical interpretation of programs which is invariant w.r.t program execution.
- Denotationally, we have to move away from finite-dimensional quantum computing:
 - To model inductive datatypes we have to be able to solve certain *recursive domain equations*.
 - The simplest non-trivial one: $X \cong \mathbb{C} \oplus X$ cannot be solved in finite dimensions.
 - But it can be solved in infinite dimensions: take $X = \bigoplus_{\omega} \mathbb{C}$.

Denotational Semantics

- Denotational semantics is a mathematical interpretation of programs which is invariant w.r.t program execution.
- Denotationally, we have to move away from finite-dimensional quantum computing:
 - To model inductive datatypes we have to be able to solve certain *recursive domain equations*.
 - The simplest non-trivial one: $X \cong \mathbb{C} \oplus X$ 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.

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 $\| - \| : A \rightarrow \mathbb{R}_{\geq 0}$, i.e. $\forall x, y \in A : \|xy\| \leq \|x\| \|y\|$.
 - An involution $(-)^* : A \rightarrow A$ such that $(x^*)^* = x$, $(x + y)^* = (x^* + y^*)$, $(xy)^* = y^* x^*$ and $(\lambda x)^* = \bar{\lambda} x^*$.
 - Subject to some additional conditions (omitted here).
- Example: The complex numbers \mathbb{C} .
- Example: The algebra $M_n(\mathbb{C})$ of $n \times n$ complex matrices.
- Example: $\bigoplus_{\omega}(\mathbb{C})$, which is the interpretation of **Nat**.
- Example: $\bigoplus_{i=0}^{\omega} M_{2^i}(\mathbb{C})$, which is the interpretation of **QList**.
- Example: $\mathcal{B}(H)$, i.e. the bounded linear operators on a Hilbert space H .

The Denotational Model (contd.)

- We need to consider two kinds of structure-preserving linear maps.
- A linear map $f : A \rightarrow B$ is MIU, if it preserves multiplication, involution and the unit. These maps are known as $*$ -homomorphisms.
- A linear map $f : A \rightarrow B$ 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.
- For the denotational semantics, we have to adopt the *Heisenberg picture* of quantum mechanics.

Baby's first categorical semantics for quantum programming

- The denotational semantics relies heavily on categorical techniques (omitted here).
- Every closed type A is interpreted as a W^* -algebra $\llbracket A \rrbracket$.
- Programs are interpreted as certain order-preserving functions between partially ordered sets of CPSU-maps $\llbracket \Pi \vdash \langle \Gamma \rangle M \langle \Sigma \rangle \rrbracket : \llbracket \Pi \rrbracket \rightarrow \mathbf{C}(\llbracket \Gamma \rrbracket, \llbracket \Sigma \rrbracket)$.
- Configurations are interpreted as states $\llbracket (M, V, \Omega, \rho) \rrbracket : I \rightarrow \llbracket \Sigma \rrbracket$.
- Most of the difficulty (counter-intuitively) is in defining the substructural maps of copying and discarding.

Copying of Classical Information

- 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.
 - 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 .
- Baby version: for each classical type P , we have $\llbracket P \rrbracket \cong \bigoplus_J \mathbb{C}$ which allows us to construct a *canonical* copy map.

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

Discarding of (Quantum) Information

- Discardable operations are called *causal*.
- For a W^* -algebra A , the unique $*$ -homomorphism $\diamond_A : \mathbb{C} \rightarrow A$ is the *canonical* discarding map.
- We show that all *values* are $*$ -homomorphisms and therefore causal.
- Of course, computations are not causal, because of recursion.

Soundness

Theorem (Soundness)

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

$$\llbracket \mathcal{C} \rrbracket = \sum_{\mathcal{C} \rightsquigarrow \mathcal{D}} \llbracket \mathcal{D} \rrbracket$$

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} :

$$\llbracket \mathcal{C} \rrbracket = \sum_{\substack{\mathcal{C} \Downarrow \mathcal{T} \\ \mathcal{T} \text{ terminal}}} \llbracket \mathcal{T} \rrbracket$$

- 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} :

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

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 entanglement analysis.

Thank you for your attention!