Quantum Programming with Inductive Datatypes: Causality and Affine Type Theory

Romain Péchoux[1], Simon Perdrix[1], Mathys Rennela[2] and Vladimir Zamdzhiev[1]

[1]Université de Lorraine, CNRS, Inria, LORIA, F 54000 Nancy, France
[2] Leiden Inst. Advanced Computer Sciences, Universiteit Leiden, Leiden, The Netherlands

SoftQPro Meeting
6 June 2019

Introduction

- Inductive datatypes are an important programming concept.
- No detailed treatment of inductive datatypes for quantum programming so far.
- Most type systems for quantum programming are linear. We show that *affine* type systems are more appropriate.
- Some of the main challenges in designing a categorical model for the language stem from substructural limitations imposed by quantum mechanics.
  - Can (infinite-dimensional) quantum datatypes be discarded?
  - How do we copy (infinite-dimensional) classical datatypes?
- Our model is physically natural (von Neumann algebras) and all our constructions are consistent with the laws of quantum mechanics.

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.
- 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 computational data is *causal*.
  - This is difficult. We present a novel technique for causality analysis based on a non-standard type interpretation. General abstract construction, also works for non-quantum categories.
- Our treatment shows 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.
- 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.
  - Discarding operation on all 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	$::=$	$f_1 : A_1 \to B_1, \ldots, f_n : A_n \to B_n$

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 **bit** $:= I + I$.

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

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

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

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:

$$\frac{}{\Pi \vdash \langle \Gamma \rangle\ \textbf{new unit}\ u\ \langle \Gamma, u : I \rangle}\ (\text{unit}) \qquad \frac{}{\Pi \vdash \langle \Gamma, x : I \rangle\ \textbf{discard}\ x\ \langle \Gamma \rangle}\ (\text{discard})$$

- If we wish to have an affine type system:

$$\frac{}{\Pi \vdash \langle \Gamma \rangle\ \textbf{new unit}\ u\ \langle \Gamma, u : I \rangle}\ (\text{unit}) \qquad \frac{}{\Pi \vdash \langle \Gamma, x : A \rangle\ \textbf{discard}\ x\ \langle \Gamma \rangle}\ (\text{discard})$$

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

Example Program - toss a coin until tail shows up

```
proc cointoss {
  new qbit q;
  q*=H;
  b = measure q;
  return b
};
b = cointoss;
while b do {
  b = cointoss
}
```

- This program is written using some (obvious) syntactic sugar.
- It terminates with probability 1, but there is no upper bound on the number of loops it will do.

Operational Semantics

- Operational semantics is a formal specification which describes how a program should be 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 *control value context*. It formalizes the control structure. Each input variable of M is assigned a control 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 $(M, V, \Omega, \rho) \rightsquigarrow (M', V', \Omega', \rho')$.
- However, the reduction relation may equivalently be seen as a probabilistic reduction relation, because the probability of the reduction is encoded in ρ' and may be recovered from it.
- The only source of probabilistic behaviour is given by quantum measurements.

Denotational Semantics

- Types are interpreted as W*-algebras.
  - W*-algebras were introduced by von Neumann, to aid his study of QM.
  - Example: The type of natural numbers is interpreted as $\bigoplus_{i=0}^{\omega} \mathbb{C}$.
- Programs are interpreted as normal completely positive subunital maps.
- We identify the abstract categorical structure of these operator algebras which allows us to use categorical techniques from denotational semantics.

Categorical Model

- We interpret the entire language within the category $\mathbf{C} := (\mathbf{W}^*_{\mathrm{NCPSU}})^{\mathrm{op}}$.
  - The objects are (possibly infinite-dimensional) W*-algebras.
  - The morphisms are normal completely-positive subunital maps.
- Our categorical model (and language) can largely be understood even if one does not have knowledge about infinite-dimensional quantum mechanics.
- There exists an adjunction $F \dashv G : \mathbf{C} \to \mathbf{Set}$, which is crucial for the description of the copy operation.

Interpretation of Types

- Every open type $X \vdash A$ is interpreted as an endofunctor $[\![X \vdash A]\!] : \mathbf{C} \to \mathbf{C}$.
- Every closed type A is interpreted as an object $[\![A]\!] \in \mathrm{Ob}(\mathbf{C})$.
- Inductive datatypes are interpreted by constructing initial algebras within $\mathbf{C}$.
    - The existence of these initial algebras is technically involved.

Copying of Classical Information

- We do not use linear logic based approaches that rely on a !-modality.
- Instead, for every classical type $X \vdash P$ we present a classical interpretation $(\!| X \vdash P |\!) : \mathbf{Set} \to \mathbf{Set}$ which we show satisfies $F \circ (\!| X \vdash P |\!) \cong [\![X \vdash P]\!] \circ F$.
- For closed types we get an isomorphism $F(\!| P |\!) \cong [\![P]\!]$.
- This isomorphism allows us to define a cocommutative comonoid structure at every classical type in a canonical way by using the cartesian structure of **Set** and the axioms of symmetric monoidal adjunctions.
- The classical computational data is a comonoid homomorphism, w.r.t. this choice.
- These techniques are inspired by recent work:
  - Bert Lindenhovius, Michael Mislove and Vladimir Zamdzhiev. Mixed Linear and Non-linear Recursive Types. To (probably) appear in ICFP'19.

A Categorical View on Causality

- Discardable operations are called *causal*.
- The causal structure of the finite-dimensional types is obvious.
- What is the causal structure of an infinite-dimensional type $[\![\mu X.A]\!]$? Is the construction of discarding maps closed under formation of initial algebras?
- We present a general categorical solution for any category $\mathbf{C}$ with a symmetric monoidal structure, finite coproducts, a zero object, and colimits of initial sequences of the relevant functors.

A Categorical View on Causality (contd.)

- Consider the slice category $\mathbf{C}_c := \mathbf{C}/I$.
  - The objects are pairs $(A, \diamond_A : A \to I)$, where $\diamond_A$ is a discarding map.
  - The morphisms are maps $f : A \to B$, s.t. $\diamond_B \circ f = \diamond_A$, i.e. causal maps.
- **Theorem:** $\mathbf{C}_c$ is symmetric monoidal and has finite coproducts.
- **Theorem:** The obvious forgetful functor $U : \mathbf{C}_c \to \mathbf{C}$ reflects small colimits.
- **Theorem:** The functor U reflects initial algebras for the class of *coherent endofunctors* on $\mathbf{C}_c$, i.e., endofunctors whose action on the $\mathbf{C}$-part of the category is independent of the choice of discarding map.
- This allows us to present a non-standard type interpretation $\|\Theta \vdash A\| : \mathbf{C}_c \to \mathbf{C}_c$, so that each closed type $\|A\| \in \mathrm{Ob}(\mathbf{C}_c)$ and $[\![A]\!] = U\|A\|$.
- We show the computational data is necessarily causal, w.r.t. this choice of discarding maps.

Relationship Between the Type Interpretations

$$
\begin{array}{ccc}
\mathbf{Set}^{|\Theta|} & \xrightarrow{F^{\times|\Theta|}} & \mathbf{C}^{|\Theta|} \\
{\scriptstyle (\!|\Theta \vdash P|\!)}\Big\downarrow & \cong & \Big\downarrow{\scriptstyle [\![\Theta \vdash P]\!]} \\
\mathbf{Set} & \xrightarrow[F]{} & \mathbf{C}
\end{array}
\qquad
\begin{array}{ccc}
\mathbf{C}^{|\Theta|} & \xrightarrow{L^{\times|\Theta|}} & \mathbf{C}_c^{|\Theta|} \\
{\scriptstyle [\![\Theta \vdash A]\!]}\Big\downarrow & & \Big\downarrow{\scriptstyle \|\Theta \vdash A\|} \\
\mathbf{C} & \xleftarrow[U]{} & \mathbf{C}_c
\end{array}
\quad ,
$$

where $L(A) = (A, \perp)$ and $L(f) = f$.

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 and Adequacy

- The denotational semantics is sound:
    - For any non-terminal configuration, the denotational interpretation is invariant under program execution:

$$[\![(M, V, \Omega, \rho)]\!] = \sum_{(M,V,\Omega,\rho)\rightsquigarrow(M_i,V_i,\Omega_i,\rho_i)} [\![(M_i, V_i, \Omega_i, \rho_i)]\!]$$

    - Computational adequacy proof will be finished soon.

Conclusion and Future Work

- We extended a quantum programming language with inductive datatypes, copying of classical variables and discarding of all variables.
- We described a natural model based on (infinite-dimensional) W*-algebras.
- We described the causal structure of all types (including the infinite-dimensional ones) via a general categorical construction.
- We described the comonoid structure of all classical types using the categorical structure of models of intuitionistic linear logic.
- We showed affine types are more appropriate compared to linear ones for QPL.
- Have to:
    - Finish the adequacy proof.