Inductive Datatypes for Quantum Programming

Romain Péchoux¹, Simon Perdrix¹, Mathys Rennela² and <u>Vladimir Zamdzhiev</u>¹

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

13 May 2019











Introduction

- Inductive datatypes are an important programming paradigm.
 - Data structures such as natural numbers, lists, trees, etc.
 - Manipulate variable-sized data.
- We consider the problem of adding inductive datatypes to a quantum programming language.
- Some of the main challenges in designing a categorical model for the language stem from substructural limitations imposed by quantum mechanics.
 - Can quantum datatypes be discarded? What quantum operations are discardable?
 - How do we copy classical datatypes? Can we always duplicate the classical computational data?
- This talk describes work-in-progress.

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:
 - The type system ensures qubits cannot be copied.
 - The type system ensures that a CNOT gate cannot be applied with control and target the same qubit, etc.
- QPL is not a higher-order language: it has procedures, but does not have lambda abstractions.
- We extend QPL with inductive datatypes. This allows us to model natural numbers, lists of qubits, lists of natural numbers, etc.
- We extend QPL with a copy operation on classical types.
- We extend QPL with a discarding operation defined on all types.

Syntax

 The syntax (excerpt) of our language is presented below. The formation rules are omitted.

```
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
```

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 in N \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 .

Some syntactic sugar

- The type of bits is defined as bit := I + I.
- The program (new unit u; $b = left_{I,I} u$) creates a bit b which corresponds to false.
- The program (new unit u; $b = right_{I,I} u$) creates a bit b which corresponds to true.
- if b then P else Q can also be defined using the case term.
- The type of natural numbers is defined as $Nat := \mu X.I + X.$
- The program (new unit u; $z = left_{I,Nat} u$; $zero = fold_{Nat}z$) creates a variable zero which corresponds to 0.
- The type of lists of qubits is defined as $QList = \mu X.I + \mathbf{qbit} \otimes X$

Example Program - toss a coin until tail shows up

```
proc cointoss u:I --> b:bit {
  discard u:
  new qbit q;
  q*=H;
  b = measure q
} in
new unit u;
b = cointoss(u);
while b do {
  new unit u:
  b = cointoss(u)
```

• This program is written using the formal syntax, but it can be improved in an actual implementation of the language using syntactic sugar.

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 P 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).
 - ullet ho 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 modelled as a nondeterministic reduction relation on configurations $(M, V, \Omega, \rho) \Downarrow (M', V', \Omega', \rho')$.
- The only source of nondeterminism comes from quantum measurements. The probability of the measurement outcome is encoded in ρ' and may be recovered from it.
- The reduction relation may equivalently be seen as a probabilistic reduction relation.

Denotational Semantics

- Denotational semantics is a mathematical interpretation of programs.
- 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 $\bigoplus_{i<\omega}\mathbb{C}$.
- Programs are interpreted as completely positive subunital maps.
- We identify the abstract categorical structure of these operator algebras which allows us to use techniques from theoretical computer science.

Categorical Model

- We interpret the entire language within the category $C := (W_{NCPSU}^*)^{op}$.
 - 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 C.
 - The existence of these initial algebras is technically involved.

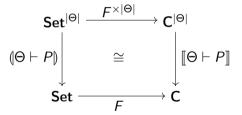
A Categorical View on Causality

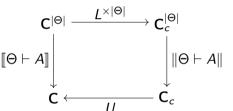
- The "no deleting" theorem of quantum mechanics shows that one cannot discard *arbitrary* quantum states.
- In mixed-state quantum mechanics, it is possible to discard certain states and operations.
- Discardable operations are called causal.
- We show the slice category C_c := C/I has sufficient structure to interpret the types within it.
 - 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.
- We present a non-standard type interpretation $||A|| \in \mathrm{Ob}(\mathbf{C}/I)$ and show the computational data is causal.

Copying of Classical Information

- To model copying of classical (nonlinear) information, we do not use linear logic based approaches that rely on a !-modality.
- Instead, for every classical type X ⊢ P we present a classical interpretation
 (|X ⊢ P|) : Set → Set which we show satisfies F ∘ (|X ⊢ P|) ≅ [|X ⊢ P|] ∘ F.
- For closed types we get an isomorphism $F(P) \cong [P]$.
- This isomorphism now easily allows us to define a cocommutative comonoid structure in a canonical way by using the cartesian structure of Set and the axioms of symmetric monoidal adjunctions.

Relationship between the Type Interpretations





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 $[\![(M,V,\Omega,\rho)]\!]:I\to [\![\Sigma]\!].$

Soundness

- We will prove the denotational semantics is sound, i.e:
 - The denotational interpretation is invariant under program execution:

$$\llbracket (\mathcal{M}, \mathcal{V}, \Omega, \rho) \rrbracket = \sum_{(\mathcal{M}, \mathcal{V}, \Omega, \rho) \Downarrow (\mathcal{M}_i, \mathcal{V}_i, \Omega_i, \rho_i)} \llbracket (\mathcal{M}_i, \mathcal{V}_i, \Omega_i, \rho_i) \rrbracket$$

Conclusion and Future Work

- We extended a quantum programming language with inductive datatypes.
- We described the causal structure of all types (including inductive ones) via a general categorical construction.
- We described the comonoid structure of all classical types using the categorical structure of models of ILL.
- Have to:
 - Finish the soundness proof.
 - Establish computational adequacy.