Inductive Datatypes for Quantum Programming

Romain Péchoux\textsuperscript{1}, Simon Perdrix\textsuperscript{1}, Mathys Rennela\textsuperscript{2} and Vladimir Zamdzhiev\textsuperscript{1}

\textsuperscript{1}Université de Lorraine, CNRS, Inria, LORIA, F 54000 Nancy, France
\textsuperscript{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.
• 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 \( A, B \) ::= \( X \mid I \mid \text{qbit} \mid A + B \mid A \otimes B \mid \mu X.A \)
Classical Types \( P, R \) ::= \( X \mid I \mid P + R \mid P \otimes R \mid \mu X.P \)
Variable contexts \( \Gamma, \Sigma \) ::= \( x_1 : A_1, \ldots, x_n : A_n \)
Procedure cont. \( \Pi \) ::= \( f_1 : A_1 \rightarrow B_1, \ldots, f_n : A_n \rightarrow B_n \)
Syntax (contd.)

Terms $M, N ::= \text{new unit } u \mid \text{new qbit } q \mid \text{discard } x \mid y = \text{copy } x$

$q_1, \ldots, q_n * = U \mid M; N \mid \text{skip}$

$b = \text{measure } q \mid \text{while } b \text{ do } M$

$x = \text{left}_{A,B} M \mid x = \text{right}_{A,B} M$

\text{case } y \text{ of } \{ \text{left } x_1 \to M \mid \text{right } x_2 \to N \}$

$x = (x_1, x_2) \mid (x_1, x_2) = x$

$y = \text{fold } x \mid y = \text{unfold } x$

\text{proc } f \ x : A \to y : B \ \{ M \} \ \text{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 $\Pi$, 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) $\Gamma$. 
Some syntactic sugar

- The type of bits is defined as $\text{bit} := I + I$.
- 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 $\text{Nat} := \mu X. I + X$.
- 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 $\text{QList} = \mu X. I + \text{qbit} \otimes X$.
Example Program - toss a coin until tail shows up

```plaintext
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 \((M, V, \Omega, \rho)\), 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. \(V = \{x = \text{zero}, y = \text{cons}(\text{one, nil})\}\).
  • \(\Omega\) is a procedure store. It keeps track of the defined procedures by mapping procedure variables to their procedure bodies (which are terms).
  • \(\rho\) 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 \(\rho'\) 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 $\mathcal{C} := (\mathcal{W}_{\text{NCPSU}}^*)^{\text{op}}$.
  • 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 $\mathcal{F} \dashv \mathcal{G} : \mathcal{C} \to \textbf{Set}$, which is crucial for the description of the copy operation.
Interpretation of Types

- Every open type $X \vdash A$ is interpreted as an endofunctor $\llbracket X \vdash A \rrbracket : C \to C$.
- Every closed type $A$ is interpreted as an object $\llbracket A \rrbracket \in \text{Ob}(C)$.
- 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 $\parallel A \parallel \in \text{Ob}(C/I)$ 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 $X \vdash P$ we present a classical interpretation $(\llbracket X \vdash P \rrbracket) : \textbf{Set} \rightarrow \textbf{Set}$ which we show satisfies $F \circ (\llbracket X \vdash P \rrbracket) \cong \llbracket X \vdash P \rrbracket \circ F$.
- For closed types we get an isomorphism $F(\llbracket P \rrbracket) \cong \llbracket P \rrbracket$.
- This isomorphism now easily allows us to define a cocommutative comonoid structure in a canonical way by using the cartesian structure of $\textbf{Set}$ and the axioms of symmetric monoidal adjunctions.
Relationship between the Type Interpretations

\[ \text{Set}^{\Theta} \xrightarrow{F \times |\Theta|} C^{\Theta} \]

\[ \{\Theta \vdash P\} \xrightarrow{\cong} \llbracket \Theta \vdash P \rrbracket \]

\[ \text{Set} \xrightarrow{F} C \]

\[ C^{\Theta} \xrightarrow{L \times |\Theta|} C_c^{\Theta} \]

\[ \llbracket \Theta \vdash A \rrbracket \xrightarrow{\cong} \|\Theta \vdash A\| \]

\[ C \xleftarrow{U} C_c \]
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
  \[ [[\prod \vdash \langle \Gamma \rangle \ M \ \langle \Sigma \rangle ]] : [[\Pi]] \to C([[\Gamma]], [[\Sigma]]). \]

• 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:

\[
\forall (M, \mathcal{V}, \Omega, \rho) \quad \text{denotes} \quad \mathcal{J}(M, \mathcal{V}, \Omega, \rho) = \sum_{(M_i, \mathcal{V}_i, \Omega_i, \rho_i) \downarrow (M, \mathcal{V}, \Omega, \rho)} \mathcal{J}(M_i, \mathcal{V}_i, \Omega_i, \rho_i)
\]
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.