Inductive and Recursive Types for Quantum Programming

Vladimir Zamdzhiev

Université de Lorraine, CNRS, Inria, LORIA, F 54000 Nancy, France

Joint MFPS-QPL session on quantum programming languages
4 June 2020

Introduction: Inductive Types

- *Inductive types* (also known as algebraic data types) are an important programming concept.
- Data structures such as natural numbers, lists, trees, etc.
- Admit a Curry-Howard (logical) interpretation: structural induction.

Example (Natural Numbers)

```
datatype Nat =
  zero
| s     of Nat
```

Example (Lists of Natural Numbers)

```
datatype ListNat =
  nil
| cons of Nat * ListNat
```

In this talk I assume we are using eager (call-by-value) dynamics.

Introduction: Recursive Types

- *Recursive types* are strictly more expressive than inductive types and allow a greater range of type connectives to appear in their definition.
- Lazy (or infinite) data structures, such as streams, can be implemented.
- Type recursion induces program-level recursion.
- Recursive types do not admit a Curry-Howard (logical) interpretation.

Example (Streams of Natural Numbers)

```
datatype StreamNat =
cons of Nat * (unit --> StreamNat)
```

Introduction: Quantum Programming

- *Quantum programming* is concerned with the design and analysis of programming languages for quantum computers.
- Manipulate quantum (and classical) information.
- Often based on *linear* or *affine* type systems.

Example (Fair coin toss)

```
q = |0>;
q *= H;
if (q) { do_something } else { do_something_else }
```

Motivation

- Quantum programs without inductive/recursive types have very limited expressivity.
- Inductive/recursive types are ubiquitous in (classical) programming, so we have to incorporate them anyway.
- Useful and elegant programming primitives.

Technical Challenges

- *Denotational semantics:* finite-dimensional quantum structures alone do not suffice; extra structure is needed.
- *Program logics:* quantum program logics are usually proven sound w.r.t. denotational semantics based on density operators; less freedom when designing such semantics.
- *Operational semantics:* the computational dynamics allow for probabilistic branches with different-sized quantum data; more complicated program analysis.

Example

A program which produces the GHZ_n state with probability 2^{-n} stored in a *list*.

```
q = |1>; l = nil;
while(q) {
  l = GHZ_next(l);
  q *= H
}
```

Semantic properties of interest

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

$$[\![\mathcal{C}]\!] = \sum_{\mathcal{C} \rightsquigarrow \mathcal{D}} [\![\mathcal{D}]\!]$$

- Strong adequacy: can the interpretation of a configuration be recovered from the (potentially infinite) set of its terminal reducts? For any configuration $\mathcal{C}$:

$$[\![\mathcal{C}]\!] = \sum_{\substack{\mathcal{C} \rightsquigarrow_* \mathcal{T} \\ \mathcal{T} \text{ terminal}}} [\![\mathcal{T}]\!]$$

- The latter is useful for quantum program logics.

The lower sum should be understood as ranging over a multiset (details omitted).

More compact (structural) syntax

- The name of the inductive type and the name of the cases are not structurally important. Focus on the type and number of the cases.

Example (Natural Numbers)

Write $\texttt{Nat} = \mu X.I + X$ for:

```
datatype Nat =
  zero [of I]
| s      of Nat
```

Example (Lists of Natural Numbers)

Write $\texttt{ListNat} = \mu Y.I + \texttt{Nat} \times Y$ for:

```
datatype ListNat =
  nil [of I]
| cons of Nat * ListNat
```

Inductive vs Recursive Types in Quantum Programming

Let Θ be a distinct list of type variables.
Inductive types:

$$\frac{\vdash \Theta}{\Theta \vdash \Theta_i} \qquad \frac{\Theta \vdash A \qquad \Theta \vdash B}{\Theta \vdash A \star B}\ \star \in \{\oplus, \otimes\} \qquad \frac{\Theta, X \vdash A}{\Theta \vdash \mu X.A}$$

Recursive Types:

$$\frac{\vdash \Theta}{\Theta \vdash \Theta_i} \qquad \frac{\Theta \vdash A \qquad \Theta \vdash B}{\Theta \vdash A \star B}\ \star \in \{\oplus, \otimes, \multimap\} \qquad \frac{\Theta \vdash A}{\Theta \vdash !A} \qquad \frac{\Theta, X \vdash A}{\Theta \vdash \mu X.A}$$

Interpretation of Inductive/Recursive Types in a nutshell

- To interpret inductive/recursive types, one has to solve (a system of) domain equations $X \cong D(X)$, where X is some object, e.g. Hilbert Space, W*-algebra, etc.
- **Example:** the interpretation of $\texttt{Nat} = \mu X.I \oplus X$ is the solution to the equation

$$X \cong \mathbb{C} \oplus X$$

  which is (canonically) given by $X = \bigoplus_{\omega} \mathbb{C}$.
- **Note:** non-trivial equations like the above one cannot be solved in finite-dimensions.

A simple model with inductive types

- A *category* is a collection of objects (e.g. Hilbert spaces) together with a collection of structure-preserving maps (e.g. bounded linear maps).
- Let $\mathbf{Hilb}_{\omega}^{\leq 1}$ be the category of countably-dimensional Hilbert spaces and linear maps between them of norm at most 1.
- **Theorem (Barr)**: Every endofunctor on $\mathbf{Hilb}_{\omega}^{\leq 1}$ is algebraically compact.
- **Therefore:** One can solve every domain equation in $\mathbf{Hilb}_{\omega}^{\leq 1}$ made from constants, $\otimes$ and $\oplus$.
- **Work in progress:** Show how to interpret a decent language within this model and develop methods for static analysis of quantum entanglement (joint work with Romain Péchoux and Simon Perdrix).

A model based on W*-algebras

- A W*-algebra is a complex vector space A, equipped with:
  - A bilinear multiplication $(- \cdot -) : A \times A \to A$ (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 $(x^*)^* = x$, $(x + y)^* = (x^* + y^*)$, $(xy)^* = y^* x^*$ and $(\lambda x)^* = \overline{\lambda} x^*$.
  - Subject to some additional conditions (omitted here).
- Example: The set of complex numbers $\mathbb{C}$.
- Example: The algebra $M_n(\mathbb{C})$ of $n \times n$ complex matrices.
- Example: $\mathcal{B}(H)$, the bounded linear maps on a Hilbert space H.

A model based on W*-algebras (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 ($0 \leq f(1) \leq 1$).
- Every MIU map is also CPSU.
- Values are interpreted as MIU-maps, whereas computations are interpreted as CPSU-maps.

A model based on W*-algebras (contd.)

- Let $\mathbf{W}^*_{\mathrm{CPSU}}$ be the category of W*-algebras and CPSU-maps.
- Let $\mathbf{W}^*_{\mathrm{MIU}}$ be the category of W*-algebras and MIU-maps.
- We adopt the *Heisenberg picture* of quantum mechanics:
  - Categorically, this means our interpretations live in the opposite categories.
  - Values are interpreted as morphisms in $\mathbf{V} \coloneqq (\mathbf{W}^*_{\mathrm{MIU}})^{\mathrm{op}}$.
  - Computations are interpreted as morphisms in $\mathbf{C} \coloneqq (\mathbf{W}^*_{\mathrm{CPSU}})^{\mathrm{op}}$.
- The model consists of three categories and two functors that relate them

  $\mathbf{Set} \xrightarrow{\quad F \quad} \mathbf{V} \hookrightarrow \mathbf{C}$, described in [PPRZ20].
- First-order language and we also model copying of classical information and discarding of (arbitrary) information without using a !-modality.
- First denotational semantics for full inductive types in quantum programming.

[PPRZ20] Péchoux, Perdrix, Rennela, Zamdzhiev. Quantum Programming with Inductive Datatypes: Causality and Affine Type Theory. FoSSaCS 2020.

Other models that support some inductive types

- A model of the quantum lambda calculus with lists based on quantitative semantics of linear logic [PSV].
- A model of the same language based on game semantics [CdVW].
- Both models are also fully abstract (as Pierre told us earlier).
- A model of Proto-Quipper-{M,D} [RS, FKS] based on free cocompletions (circuit-description language).

[PSV] Pagani, Selinger, Valiron. Applying quantitative semantics to higher-order quantum computing. POPL'14.

[CdVW] Clairambault, de Visme, Winskel. Game semantics for quantum programming. POPL'19.

[RS] Rios, Selinger. A categorical model for a quantum circuit description language. QPL'17.

[FKS] Fu, Kishida, Selinger. Linear Dependent Type Theory for Quantum Programming Languages: Extended Abstract. LICS'20.

A general recipe for interpreting inductive types (in quantum programming)

- Find a category $\mathbf{C}$ where you interpret terms (programs). Find a subcategory $\mathbf{V}$ of $\mathbf{C}$ which has ω-colimits, finite coproducts, monoidal with $\otimes$ preserving ω-colimits, such that the subcategory inclusion preserves this structure.
- Interpret your types as functors $[\![\Theta \vdash A]\!] : \mathbf{V}^{|\Theta|} \to \mathbf{V}$, where you can now compute parameterised initial algebras. Interpret values in $\mathbf{V}$.
- If you have an affine type system, the tensor unit I of $\mathbf{V}$ should be terminal.
- To model copying of classical information, find a cartesian category $\mathbf{B}$ with ω-colimits, finite coproducts and a strong monoidal functor $F : \mathbf{B} \to \mathbf{V}$ that preserves these. You can now copy classical information at a wide-range of types (beyond $!A$, see [LMZ19,LMZ20]).

[LMZ19] Lindenhovius, Mislove, Zamdzhiev. Mixed linear and non-linear recursive types. ICFP'19.
[LMZ20] ____. LNL-FPC: The Linear/Non-linear Fixpoint Calculus. Minor revisions for LMCS.

But what about recursive types?

- So far, we have seen several results about (classes of) inductive types.
- What about recursive types?
- **Short answer:** no known semantics that supports recursive types and quantum measurements.
- **Longer answer:** partial results for quantum circuit description languages.
- **Main problems:** how to interpret $\multimap$, ! and compute fixpoints over them while still being *physical enough* to have measurements and combine the resulting probabilistic branches.

Circuit description languages and recursive types

- Languages focused on the generation and manipulation of quantum circuits.
- Example: Proto-Quipper-$\star$, where $\star \in \{\mathrm{S},\mathrm{D},\mathrm{M}\}$. No measurements.
- A model of Proto-Quipper-M in [LMZ18], based on an enriched Yoneda embedding, supports recursive types, but this has not been written up.
- A model of Proto-Quipper-M in [KLM20], based on Quantum CPOs, supports recursive types, but the semantics has not been written up.
- In both models, with recursive types, soundness is easy to show, but computational adequacy is currently unknown (with circuit generation being allowed).

[LMZ18] Lindenhovius, Mislove, Zamdzhiev. Enriching a Linear/Non-linear Lambda Calculus: A Programming Language for String Diagrams. LICS'18.

[KLM20] Kornell, Lindenhovius, Mislove. Quantum CPOs. QPL'20.

Conclusion

- We have good results for inductive types in quantum programming.
- Not so good results for recursive types in quantum programming.
- The above is assuming classical control. With quantum control, even less is known.
- The present treatment views inductive/recursive types as essentially classical containers with qubits at the leaves. What happens if we consider purely quantum containers which may be put into superposition and even entangled?

Thank you for your attention!