Quantum Programming with Inductive Datatypes*

Romain Péchoux[a], Simon Perdrix[a], Mathys Rennela[b], Vladimir Zamdzhiev[a,*]

[a] *Université de Lorraine, CNRS, Inria, LORIA, F 54000 Nancy, France*
[b] *Leiden University, Leiden, The Netherlands*

Abstract

Inductive datatypes in programming languages allow users to define useful data structures such as natural numbers, lists, trees, and others. In this paper we show how inductive datatypes may be added to the quantum programming language QPL. We construct a sound categorical model for the language and by doing so we provide the first detailed semantic treatment of user-defined inductive datatypes in quantum programming. We also show our denotational interpretation is invariant with respect to big-step reduction, thereby establishing another novel result for quantum programming. Compared to classical programming, this property is considerably more difficult to prove and we demonstrate its usefulness by showing how it immediately implies computational adequacy at all types. To further cement our results, our semantics is entirely based on a physically natural model of von Neumann algebras, which are mathematical structures used by physicists to study quantum mechanics.

Keywords: Quantum programming, Inductive types, Adequacy, W*-algebras

1. Introduction

Quantum computing is a computational paradigm which takes advantage of quantum mechanical phenomena to perform computation. A quantum computer can solve problems which are out of reach for classical computers (e.g. factorisation of large numbers [2], solving large linear systems [3]). The recent

*This article is an extended version of our FoSSaCS paper [1].
*Corresponding author.

developments of quantum technologies point out the necessity of filling the gap between theoretical quantum algorithms and the actual (prototypes of) quantum computers. As a consequence, quantum software and in particular quantum programming languages play a key role in the future development of quantum computing. The present paper makes several theoretical contributions towards the design and denotational semantics of quantum programming languages.

1.1. Our Contribution

Our development is based around the quantum programming language QPL [4] which we extend with inductive datatypes. Our paper is the first to construct a denotational semantics for user-defined inductive datatypes in quantum programming. In the spirit of the original QPL, our type system is *affine* (discarding of arbitrary variables is allowed, but copying is restricted). We also extend QPL with a copy operation for *classical data*, because this is an admissible operation in quantum mechanics which improves programming convenience. The addition of inductive datatypes requires a departure from the original denotational semantics of QPL, which are based on finite-dimensional quantum structures, and we consider instead (possibly infinite-dimensional) quantum structures based on *W*-algebras* (also known as *von Neumann algebras*), which have been used by physicists in the study of quantum foundations [5]. As such, our semantic treatment is physically natural and our model is more accessible to physicists and experts in quantum computing compared to most other denotational models.

QPL is a first-order programming language which has *procedures*, but it does not have lambda abstractions. Thus, there is no use for a !-modality and we show how to model the copy operation by describing the canonical comonoid structure of all classical types (including the inductive ones).

An important notion in quantum mechanics is the idea of *causality* which has been formulated in a variety of different ways. In this paper, we consider a simple operational interpretation of causality: if the output of a physical process is discarded, then it does not matter which process occurred [6]. In a symmetric

monoidal category $\mathbf{C}$ with tensor unit I, this can be understood as requiring that for any morphism (process) $f : A_1 \to A_2$, it must be the case that $\diamond_{A_2} \circ f = \diamond_{A_1}$, where $\diamond_{A_i} : A_i \to I$ is the discarding map (process) at the given objects. This notion ties in very nicely with our affine language, because we have to show that the interpretation of values is causal, i.e., values are always discardable.

A major contribution of this paper is that we prove the denotational semantics is invariant with respect to both small-step reduction and big-step reduction. The latter is more difficult in quantum programming and our paper is the first to demonstrate such a result. As a corollary, we obtain computational adequacy, i.e., we provide a denotational characterisation for the probability of termination for arbitrary programs.

1.2. Overview and Summary of Results

The main contributions of this article are the following results:

- An extension of QPL with inductive datatypes and a copy operation for classical data (§2);
- A type safe operational semantics based on *finite-dimensional* quantum operations and classical control structures (§3);
- A *physically natural* denotational model for quantum programming using W*-algebras, which are mathematical structures used by physicists to study quantum mechanics (§4);
- A detailed semantic treatment of user-defined inductive datatypes: we describe the causal structure of all types and we describe the comonoid structure of classical types (§5).
- Invariance of the denotational semantics with respect to small-step and big-step reduction and a computational adequacy result (implied by causality): we provide a denotational characterisation of the probability of termination for programs of arbitrary types (§5.6).

1.3. Publication History

This article is an extended version of the FoSSaCS paper [1]. Compared to the conference version, the main additions in this article are two extensive subsections: Subsection 5.3 which describes in detail the comonoid structure of classical types; and Subsection 5.7 which describes our proof of the invariance of the denotational semantics with respect to big-step reduction and also our computational adequacy result. We have also improved the exposition by adding examples and providing more detailed descriptions of some concepts.

2. Syntax of QPL

The syntax of QPL (including our extensions) is summarised in Figure 1. A well-formed type context, denoted $\vdash \Theta$, is simply a list of distinct type variables. A type A is well-formed in type context Θ, denoted $\Theta \vdash A$, if the judgement can be derived according to the following rules (see [7, 8] for a more detailed exposition):

$$\frac{\vdash \Theta}{\Theta \vdash \Theta_i} \quad \frac{\vdash \Theta}{\Theta \vdash I} \quad \frac{\vdash \Theta}{\Theta \vdash \mathbf{qbit}} \quad \frac{\Theta \vdash A \qquad \Theta \vdash B}{\Theta \vdash A \star B}\ \star \in \{+, \otimes\} \quad \frac{\Theta, X \vdash A}{\Theta \vdash \mu X.A}$$

A type A is *closed* if $\cdot \vdash A$. Note that nested type induction is allowed. Henceforth, we implicitly assume that all types we are dealing with are well-formed.

Example 1. The type of natural numbers is defined as $\mathbf{Nat} \equiv \mu X.I + X$. Lists of a closed type $\cdot \vdash A$ are defined as $\mathbf{List}(A) \equiv \mu Y.I + A \otimes Y$.

Notice that our type system is not equipped with a !-modality. Indeed, in the absence of function types, there is no reason to introduce it. Instead, we specify the subset of types where copying is an admissible operation. The *classical types* are a subset of our types defined in Figure 1. They are characterised by the property that variables of classical types may be copied, whereas variables of non-classical types may not be copied (see the rule for copying in Figure 2).

We use small Latin letters (e.g. x, y, u, q, b) to range over *term variables*. More specifically, q ranges over variables of type $\mathbf{qbit}$, u over variables of unit

Type Variables	X, Y, Z		
Term Variables	x, y, q, b, u		
Procedure Names	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	$::=$	$X \mid I \mid P + R \mid P \otimes R \mid \mu X.P$
Terms	M, N	$::=$	$\mathbf{new\ unit}\ u \mid \mathbf{discard}\ x \mid$
			$y = \mathbf{copy}\ x \mid \mathbf{new\ qbit}\ q \mid$
			$b = \mathbf{measure}\ q \mid q_1, \ldots, q_n\ \ *= S \mid$
			$M; N \mid \mathbf{skip} \mid \mathbf{while}\ b\ \mathbf{do}\ M \mid$
			$x = \mathbf{left}_{A,B} M \mid x = \mathbf{right}_{A,B} M \mid$
			$\mathbf{case}\ y\ \mathbf{of}\ \{\mathbf{left}\ x_1 \to M \mid \mathbf{right}\ x_2 \to N\} \mid$
			$x = (x_1, x_2) \mid (x_1, x_2) = x \mid$
			$y = \mathbf{fold}\ x \mid y = \mathbf{unfold}\ x \mid$
			$\mathbf{proc}\ f :: x : A \to y : B\ \{M\}\ \mid y = f(x)$
Type contexts	Θ	$::=$	$X_1, X_2, \ldots, X_n$
Variable contexts	Γ, Σ	$::=$	$x_1 : A_1, \ldots, x_n : A_n$
Procedure contexts	Π	$::=$	$f_1 : A_1 \to B_1, \ldots, f_n : A_n \to B_n$
Type Judgements	$\Theta \vdash A$		
Term Judgements	$\Pi \vdash \langle \Gamma \rangle\ M\ \langle \Sigma \rangle$		

Figure 1: Syntax of QPL.

type I, b over variables of type **bit** $:= I + I$ and x, y range over variables of arbitrary type. We use Γ and Σ to range over *variable contexts*. A variable context is a function from term variables to *closed types*, which we write as $\Gamma = x_1 : A_1, \ldots, x_n : A_n$.

We use f, g to range over *procedure names*. Every procedure name f has an *input type* A and an *output type* B, denoted $f : A \to B$, where A and B are closed types. We use Π to range over *procedure contexts*. A procedure context is a function from procedure names to pairs of procedure input-output types, denoted $\Pi = f_1 : A_1 \to B_1, \ldots, f_n : A_n \to B_n$.

Remark 2. Unlike lambda abstractions, procedures cannot be passed to other procedures as input arguments, nor can they be returned as output.

A *term judgement* has the form $\Pi \vdash \langle \Gamma \rangle\ M\ \langle \Sigma \rangle$ (see Figure 2) and indicates that term M is well-formed in procedure context Π with input variable context Γ and output variable context Σ. All types occurring within it are closed.

The intended interpretation of the quantum rules are as follows. The term **new qbit** q prepares a new qubit q in state $|0\rangle\langle 0|$. The term $q_1, \ldots, q_n \ *= S$ applies a unitary transformation S to a sequence of qubits in the standard way. The term $b =$ **measure** q performs a quantum measurement on qubit q in the standard basis $\{|0\rangle, |1\rangle\}$ and stores the measurement outcome in bit b. The measured qubit is destroyed in the process.

Remark 3. In the syntax of QPL, unitary transformations are referred to by names (e.g. *H*, *CNOT*, etc.) and not by their matrix representations. However, to describe the operational and denotational semantics, it is necessary to map these names to specific matrix representations. For simplicity, we assume that this mapping is given and we shall use the same notation for both the names and the matrix representations of the unitary operations which hopefully should not lead to confusion. We also leave the set of available unitary transformations unspecified, but for most purposes it suffices to assume that it contains the *H,* T and *CNOT* unitary maps.

The no-cloning theorem of quantum mechanics [9] shows that arbitrary

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

$$\frac{P \text{ is a classical type}}{\Pi \vdash \langle \Gamma, x : P \rangle\ y = \mathbf{copy}\ x\ \langle \Gamma, x : P, y : P \rangle}$$

$$\frac{}{\Pi \vdash \langle \Gamma \rangle\ \mathbf{skip}\ \langle \Gamma \rangle} \qquad \frac{\Pi \vdash \langle \Gamma \rangle\ M\ \langle \Gamma' \rangle \qquad \Pi \vdash \langle \Gamma' \rangle\ N\ \langle \Sigma \rangle}{\Pi \vdash \langle \Gamma \rangle\ M; N\ \langle \Sigma \rangle}$$

$$\frac{\Pi \vdash \langle \Gamma, b : \mathbf{bit} \rangle\ M\ \langle \Gamma, b : \mathbf{bit} \rangle}{\Pi \vdash \langle \Gamma, b : \mathbf{bit} \rangle\ \mathbf{while}\ b\ \mathbf{do}\ M\ \langle \Gamma, b : \mathbf{bit} \rangle}$$

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

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

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

$$\frac{}{\Pi \vdash \langle \Gamma, x : A \rangle\ y = \mathbf{left}_{A,B}\ x\ \langle \Gamma, y : A + B \rangle}$$

$$\frac{}{\Pi \vdash \langle \Gamma, x : B \rangle\ y = \mathbf{right}_{A,B}\ x\ \langle \Gamma, y : A + B \rangle}$$

$$\frac{\Pi \vdash \langle \Gamma, x_1 : A \rangle\ M_1\ \langle \Sigma \rangle \qquad \Pi \vdash \langle \Gamma, x_2 : B \rangle\ M_2\ \langle \Sigma \rangle}{\Pi \vdash \langle \Gamma, y : A + B \rangle\ \mathbf{case}\ y\ \mathbf{of}\ \{\mathbf{left}_{A,B}\ x_1 \to M_1 \mid \mathbf{right}_{A,B}\ x_2 \to M_2\ \}\ \langle \Sigma \rangle}$$

$$\frac{}{\Pi \vdash \langle \Gamma, x_1 : A, x_2 : B \rangle\ x = (x_1, x_2)\ \langle \Gamma, x : A \otimes B \rangle}$$

$$\frac{}{\Pi \vdash \langle \Gamma, x : A \otimes B \rangle\ (x_1, x_2) = x\ \langle \Gamma, x_1 : A, x_2 : B \rangle}$$

$$\frac{}{\Pi \vdash \langle \Gamma, x : A[\mu X.A/X] \rangle\ y = \mathbf{fold}_{\mu X.A}\ x\ \langle \Gamma, y : \mu X.A \rangle}$$

$$\frac{}{\Pi \vdash \langle \Gamma, x : \mu X.A \rangle\ y = \mathbf{unfold}\ x\ \langle \Gamma, y : A[\mu X.A/X] \rangle}$$

$$\frac{\Pi, f : A \to B \vdash \langle x : A \rangle\ M\ \langle y : B \rangle}{\Pi \vdash \langle \Gamma \rangle\ \mathbf{proc}\ f :: \ x : A \to y : B\ \{M\}\ \langle \Gamma \rangle}$$

$$\frac{}{\Pi, f : A \to B \vdash \langle \Gamma, x : A \rangle\ y = f(x)\ \langle \Gamma, y : B \rangle}$$

Figure 2: Formation rules for QPL terms.

qubits cannot be copied. Because of this, copying is restricted only to classical types, as indicated in Figure 2, and this allows us to avoid runtime errors. Like the original QPL [4], our type system is also *affine* and so any variable can be discarded (see the formation rule for the term **discard** x in Figure 2).

3. Operational Semantics of QPL

In this section we describe the operational semantics of QPL. The central notion is that of a *program configuration* which provides a complete description of the current state of program execution. It consists of four components that must satisfy some coherence properties: (1) the term which remains to be executed; (2) a *value assignment*, which is a function that assigns formal expressions to variables as a result of execution; (3) a *procedure store* which keeps track of what procedures have been defined so far and (4) the *quantum state* computed so far.

Value Assignments. A *value* is an expression defined by the following grammar:

$$v, w ::= * \mid n \mid \mathbf{left}_{A,B} v \mid \mathbf{right}_{A,B} v \mid (v, w) \mid \mathbf{fold}_{\mu X.A} v$$

where n ranges over the natural numbers. Think of $*$ as representing the unique value of unit type I and of n as representing a pointer to the n-th qubit of a quantum state ρ. Specific values of interest are $\mathtt{ff} := \mathbf{left}_{I,I} *$ and $\mathtt{tt} := \mathbf{right}_{I,I} *$ which correspond to **false** and **true** respectively. Values play an important role in the operational semantics. They are assigned to variables and formally represent the computational data computed so far that is stored in each variable.

A *qubit pointer context* is a set Q of natural numbers. Given two *disjoint* qubit pointer contexts Q_1 and Q_2, we write Q_1, Q_2 for the union of the two contexts, as usual. A value v of type A is well-formed in qubit pointer context Q, denoted $Q \vdash v : A$, if the judgement is derivable from the following rules:

$$\frac{}{\cdot \vdash * : I} \qquad \frac{}{\{n\} \vdash n : \mathbf{qbit}} \qquad \frac{Q \vdash v : A}{Q \vdash \mathbf{left}_{A,B} v : A + B} \qquad \frac{Q \vdash v : B}{Q \vdash \mathbf{right}_{A,B} v : A + B}$$

$$\frac{Q_1 \vdash v : A \qquad Q_2 \vdash w : B \qquad Q_1 \cap Q_2 = \varnothing}{Q_1, Q_2 \vdash (v, w) : A \otimes B} \qquad \frac{Q \vdash v : A[\mu X.A/X]}{Q \vdash \mathbf{fold}_{\mu X.A} v : \mu X.A}$$

We see that if $Q \vdash v : A$, then the set of natural numbers appearing within v is exactly Q. Each such natural number i within v should be thought of as a pointer to the i-th qubit of some quantum state. Moreover, if v is well-formed, then its type and qubit pointer context are uniquely determined.

Remark 4. We wish to point out that it is not possible to locally and individually equip each value with quantum information due to the possibility of quantum entanglement, i.e., the global quantum state which is being manipulated may not be separable. Because of this, we use pointers that identify individual qubits of the global quantum state which we manipulate.

If v is a value with $Q \vdash v : P$ where P is classical, then we say v is a *classical value.*

Lemma 5. *If $Q \vdash v : P$ is a well-formed classical value, then $Q = \varnothing$.*

Therefore, classical values cannot refer to any quantum data.

A *value assignment* is a function from term variables to values, which we write as $V = \{x_1 = v_1, \ldots, x_n = v_n\}$, where x_i are variables and v_i are values. A value assignment is *well-formed* in qubit pointer context Q and variable context Γ, denoted $Q; \Gamma \vdash V$, if V has exactly the same variables as Γ, so that $\Gamma = \{x_1 : A_1, \ldots, x_n : A_n\}$, and $Q = Q_1, \ldots, Q_n$, s.t. $Q_i \vdash v_i : A_i$. Such a splitting of Q is necessarily unique, if it exists, and some of the Q_i may be empty.

Procedure Stores. A *procedure store* is a set of procedure definitions, written as:

$$\Omega = \{f_1 :: x_1 : A_1 \to y_1 : B_1 \ \{M_1\}, \ldots, f_n :: x_n : A_n \to y_n : B_n \ \{M_n\}\}\,.$$

A procedure store is *well-formed* in procedure context Π, written $\Pi \vdash \Omega$, if the judgement is derivable via the following rules:

$$\frac{}{\cdot \vdash \cdot} \qquad \frac{\Pi \vdash \Omega \qquad \Pi, f : A \to B \vdash \langle x : A \rangle \ M \ \langle y : B \rangle}{\Pi, f : A \to B \vdash \Omega, f :: x : A \to y : B \ \{M\}}$$

Program Configurations. A *program configuration* is a quadruple $(M \mid V \mid \Omega \mid \rho)$, where M is a term, V is a value assignment, Ω is a procedure store and $\rho \in \mathbb{C}^{2^n \times 2^n}$ is a finite-dimensional density matrix with $0 \leq \mathrm{tr}(\rho) \leq 1$. The density matrix ρ represents a (mixed) quantum state and its trace may be smaller than one because we also use it to encode probability information (see Remark 6). We write $\mathit{size}(\rho) = n$ to indicate that ρ is an n-qubit state.

A *well-formed* program configuration is a configuration $(M \mid V \mid \Omega \mid \rho)$, where there exist (necessarily unique) Π, Γ, Σ, Q, such that: (1) $\Pi \vdash \langle \Gamma \rangle\ M\ \langle \Sigma \rangle$ is a well-formed term; (2) $Q; \Gamma \vdash V$ is a well-formed value assignment; (3) $\Pi \vdash \Omega$ is a well-formed procedure store; and (4) $Q = \{1, 2, \ldots, \mathit{size}(\rho)\}$. We write $\Pi; \Gamma; \Sigma; Q \vdash (M \mid V \mid \Omega \mid \rho)$ to indicate this situation. The formation rules enforce that the qubits of ρ and the qubit pointers from V are in a 1-1 correspondence.

The small step semantics is defined for configurations $(M \mid V \mid \Omega \mid \rho)$ by induction on M in Figure 3 and we now explain the notations used therein.

In the rule for discarding, we use two functions that depend on a value v. They are tr_v, which modifies the quantum state ρ by tracing out all of its qubits which are used in v, and r_v which simply reindexes the value assignment, so that the pointers within $r_v(V)$ correctly point to the corresponding qubits of $tr_v(\rho)$, which is potentially of smaller dimension than ρ. Formally, for a well-formed value v, let Q and A be the unique qubit pointer context and type, such that $Q \vdash v : A$. Then $tr_v(\rho)$ is the quantum state obtained from ρ by tracing out all qubits specified by Q. Given a value assignment $V = \{x_1 = v_1, \ldots, x_n = v_n\}$,

$$\frac{}{(\textbf{new unit } u \mid V \mid \Omega \mid \rho) \rightsquigarrow (\textbf{skip} \mid V, u = * \mid \Omega \mid \rho)}$$

$$\frac{}{(\textbf{discard } x \mid V, x = v \mid \Omega \mid \rho) \rightsquigarrow (\textbf{skip} \mid r_v(V) \mid \Omega \mid tr_v(\rho))}$$

$$\frac{}{(y = \textbf{copy } x \mid V, x = v \mid \Omega \mid \rho) \rightsquigarrow (\textbf{skip} \mid V, x = v, y = v \mid \Omega \mid \rho)}$$

$$\frac{}{(\textbf{new qbit } q \mid V \mid \Omega \mid \rho) \rightsquigarrow (\textbf{skip} \mid V, q = \mathit{size}(\rho) + 1 \mid \Omega \mid \rho \otimes |0\rangle\langle 0|)}$$

$$\frac{}{(\vec{q} \mathrel{*}= S \mid V, \vec{q} = \vec{m} \mid \Omega \mid \rho) \rightsquigarrow (\textbf{skip} \mid V, \vec{q} = \vec{m} \mid \Omega \mid S_{\vec{m}}(\rho))}$$

$$\frac{}{(b = \textbf{measure } q \mid V, q = m \mid \Omega \mid \rho) \rightsquigarrow (\textbf{skip} \mid r_m(V), b = \texttt{ff} \mid \Omega \mid {}_m\langle 0|\rho|0\rangle_m)}$$

$$\frac{}{(b = \textbf{measure } q \mid V, q = m \mid \Omega \mid \rho) \rightsquigarrow (\textbf{skip} \mid r_m(V), b = \texttt{tt} \mid \Omega \mid {}_m\langle 1|\rho|1\rangle_m)}$$

$$\frac{}{(\textbf{skip}; P \mid V \mid \Omega \mid \rho) \rightsquigarrow (P \mid V \mid \Omega \mid \rho)} \qquad \frac{(P \mid V \mid \Omega \mid \rho) \rightsquigarrow (P' \mid V' \mid \Omega' \mid \rho')}{(P; Q \mid V \mid \Omega \mid \rho) \rightsquigarrow (P'; Q \mid V' \mid \Omega' \mid \rho')}$$

$$\frac{}{(\textbf{while } b \textbf{ do } M \mid V, b = \texttt{ff} \mid \Omega \mid \rho) \rightsquigarrow (\textbf{skip} \mid V, b = \texttt{ff} \mid \Omega \mid \rho)}$$

$$\frac{}{(\textbf{while } b \textbf{ do } M \mid V, b = \texttt{tt} \mid \Omega \mid \rho) \rightsquigarrow (M; \textbf{while } b \textbf{ do } M \mid V, b = \texttt{tt} \mid \Omega \mid \rho)}$$

$$\frac{}{(y = \textbf{left } x \mid V, x = v \mid \Omega \mid \rho) \rightsquigarrow (\textbf{skip} \mid V, y = \textbf{left } v \mid \Omega \mid \rho)}$$

$$\frac{}{(y = \textbf{right } x \mid V, x = v \mid \Omega \mid \rho) \rightsquigarrow (\textbf{skip} \mid V, y = \textbf{right } v \mid \Omega \mid \rho)}$$

$$\frac{}{\begin{array}{c}(\textbf{case } y \textbf{ of } \{\textbf{left } x_1 \to M_1 \mid \textbf{right } x_2 \to M_2 \} \mid V, y = \textbf{left } v \mid \Omega \mid \rho) \rightsquigarrow \\ (M_1 \mid V, x_1 = v \mid \Omega \mid \rho)\end{array}}$$

$$\frac{}{\begin{array}{c}(\textbf{case } y \textbf{ of } \{\textbf{left } x_1 \to M_1 \mid \textbf{right } x_2 \to M_2 \} \mid V, y = \textbf{right } v \mid \Omega \mid \rho) \rightsquigarrow \\ (M_2 \mid V, x_2 = v \mid \Omega \mid \rho)\end{array}}$$

$$\frac{}{(x = (x_1, x_2) \mid V, x_1 = v_1, x_2 = v_2 \mid \Omega \mid \rho) \rightsquigarrow (\textbf{skip} \mid V, x = (v_1, v_2) \mid \Omega \mid \rho)}$$

$$\frac{}{((x_1, x_2) = x \mid V, x = (v_1, v_2) \mid \Omega \mid \rho) \rightsquigarrow (\textbf{skip} \mid V, x_1 = v_1, x_2 = v_2 \mid \Omega \mid \rho)}$$

$$\frac{}{(y = \textbf{fold } x \mid V, x = v \mid \Omega \mid \rho) \rightsquigarrow (\textbf{skip} \mid V, y = \textbf{fold } v \mid \Omega \mid \rho)}$$

$$\frac{}{(y = \textbf{unfold } x \mid V, x = \textbf{fold } v \mid \Omega \mid \rho) \rightsquigarrow (\textbf{skip} \mid V, y = v \mid \Omega \mid \rho)}$$

$$\frac{}{(\textbf{proc } f :: \ x : A \to y : B \ \{M\} \mid V \mid \Omega \mid \rho) \rightsquigarrow (\textbf{skip} \mid V \mid \Omega, f :: \ x : A \to y : B \ \{M\} \mid \rho)}$$

$$\frac{}{\begin{array}{c}(y_1 = f(x_1) \mid V, x_1 = v \mid \Omega, f :: \ x_2 : A \to y_2 : B \ \{M\} \mid \rho) \rightsquigarrow \\ (M_\alpha \mid V, x_1 = v \mid \Omega, f :: \ x_2 : A \to y_2 : B \ \{M\} \mid \rho)\end{array}}$$

Figure 3: Small-step operational semantics of QPL.

then $r_v(V) = \{x_1 = r'_v(v_1), \ldots, x_n = r'_v(v_n)\}$, where:

$$r'_v(w) = \begin{cases} *, & \text{if } w = * \\ k - |\{i \in Q \mid i < k\}|, & \text{if } w = k \in \mathbb{N} \text{ and where } Q \vdash v : A \\ \textbf{left } r'_v(w'), & \text{if } w = \textbf{left } w' \\ \textbf{right } r'_v(w'), & \text{if } w = \textbf{right } w' \\ (r'_v(w_1), r'_v(w_2)) & \text{if } w = (w_1, w_2) \\ \textbf{fold } r'_v(w'), & \text{if } w = \textbf{fold } w' \end{cases}$$

In the rule for unitary transformations, the superoperator $S_{\vec{m}}$ applies the unitary map S to the vector of qubits specified by $\vec{m}$. In the rules for measurement, the m-th qubit of ρ is measured in the computational basis, the measured qubit is destroyed in the process and the measurement outcome is stored in the bit b. More specifically, $|i\rangle_m = I_{2^{m-1}} \otimes |i\rangle \otimes I_{2^{n-m}}$ and ${}_m\langle i|$ is its adjoint, for $i \in \{0, 1\}$, and where I_n is the identity matrix in $\mathbb{C}^{n \times n}$.

Remark 6. Because of the way we decided to handle measurements, reduction $(- \rightsquigarrow -)$ is a *nondeterministic* operation, where we encode the probabilities of reduction within the trace of our density matrices in a similar way to [10]. Equivalently, we may see the reduction relation as *probabilistic* provided that we normalise all density matrices and decorate the reductions with the appropriate probability information as specified by the Born rule of quantum mechanics. The nondeterministic view leads to a more concise and clear presentation and because of this we have chosen it over the probabilistic view.

The introduction rule for procedures simply defines a procedure which is added to the procedure store. In the rule for calling procedures, the term M_α is α-equivalent to M and is obtained from it by renaming the input x_2 to x_1, renaming the output y_2 to y_1 and renaming all other variables within M to some fresh names, so as to avoid conflicts with the input, output and the rest of the variables within V.

Theorem 7 (Subject reduction)**.** *Assume that* $\Pi; \Gamma; \Sigma; Q \vdash (M \mid V \mid \Omega \mid \rho)$*. If*

$(M \mid V \mid \Omega \mid \rho) \rightsquigarrow (M' \mid V' \mid \Omega' \mid \rho')$, *then* $\Pi'; \Gamma'; \Sigma; Q' \vdash (M' \mid V' \mid \Omega' \mid \rho')$, *for some (necessarily unique) contexts* Π', Γ', Q'.

Remark 8. Notice that in the above theorem, the output context Σ is invariant.

Assumption 9. From now on we assume all configurations are well-formed.

A configuration $(M \mid V \mid \Omega \mid \rho)$ is said to be *terminal* if $M = \mathbf{skip}$. Program execution finishes at terminal configurations, which are characterised by the property that they do not reduce any further. We will use calligraphic letters $(\mathcal{C}, \mathcal{D}, \ldots)$ to range over configurations and we will use $\mathcal{T}$ to range over terminal configurations. For a configuration $\mathcal{C} = (M \mid V \mid \Omega \mid \rho)$, we write for brevity $\mathrm{tr}(\mathcal{C}) \coloneqq \mathrm{tr}(\rho)$ and we shall say $\mathcal{C}$ is *normalised* whenever $\mathrm{tr}(\mathcal{C}) = 1$. We say that a configuration $\mathcal{C}$ is *impossible* if $\mathrm{tr}(\mathcal{C}) = 0$ and we say it is *possible* otherwise.

Theorem 10 (Progress)**.** *If $\mathcal{C}$ is a configuration, then either $\mathcal{C}$ is terminal or there exists a configuration $\mathcal{D}$, such that $\mathcal{C} \rightsquigarrow \mathcal{D}$. Moreover, if $\mathcal{C}$ is not terminal, then $\mathrm{tr}(\mathcal{C}) = \sum_{\mathcal{C} \rightsquigarrow \mathcal{D}} \mathrm{tr}(\mathcal{D})$ and there are at most two such configurations $\mathcal{D}$.*

In the situation of the above theorem, the probability of reduction is given by $\Pr(\mathcal{C} \rightsquigarrow \mathcal{D}) \coloneqq \mathrm{tr}(\mathcal{D})/\mathrm{tr}(\mathcal{C})$, for any possible $\mathcal{C}$ (see Remark 6) and Theorem 10 shows the total probability of all single-step reductions is 1. If $\mathcal{C}$ is impossible, then $\mathcal{C}$ occurs with probability 0 and subsequent reductions are also impossible.

Probability of Termination. Given configurations $\mathcal{C}$ and $\mathcal{D}$ let $\mathrm{Seq}_n(\mathcal{C}, \mathcal{D}) \coloneqq \{\mathcal{C}_0 \rightsquigarrow \cdots \rightsquigarrow \mathcal{C}_n \mid \mathcal{C}_0 = \mathcal{C} \text{ and } \mathcal{C}_n = \mathcal{D}\}$, and let $\mathrm{Seq}_{\leq n}(\mathcal{C}, \mathcal{D}) = \bigcup_{i=0}^{n} \mathrm{Seq}_n(\mathcal{C}, \mathcal{D})$. Finally, let $\mathrm{TerSeq}_{\leq n}(\mathcal{C}) \coloneqq \bigcup_{\mathcal{T} \text{ terminal}} \mathrm{Seq}_{\leq n}(\mathcal{C}, \mathcal{T})$. In other words, $\mathrm{TerSeq}_{\leq n}(\mathcal{C})$ is the set of all reduction sequences from $\mathcal{C}$ which terminate in at most n steps (including 0 if $\mathcal{C}$ is terminal). For every terminating reduction sequence $r = (\mathcal{C} \rightsquigarrow \cdots \rightsquigarrow \mathcal{T})$, let $\mathrm{End}(r) \coloneqq \mathcal{T}$, i.e. $\mathrm{End}(r)$ is simply the (terminal) end-point of the sequence.

For any configuration $\mathcal{C}$, the sequence $\left(\sum_{r \in \mathrm{TerSeq}_{\leq n}(\mathcal{C})} \mathrm{tr}(\mathrm{End}(r))\right)_{n \in \mathbb{N}}$ is increasing with upper bound $\mathrm{tr}(\mathcal{C})$ (follows from Theorem 10). For any possible

```
while b do {
  new qbit q;
  q *= H;
  discard b;
  b = measure q
}
```

Figure 4: A QPL program.

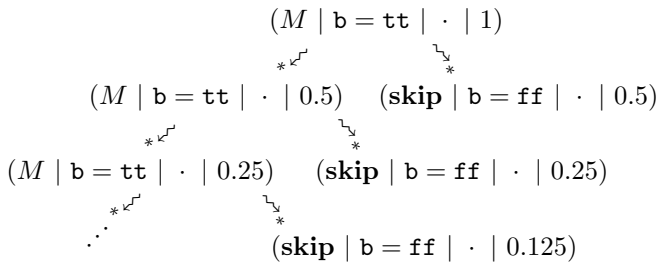

Figure 5: A reduction graph (in this case tree), where M is the program in Figure 4.

$\mathcal{C}$, we define:

$$\mathrm{Halt}(\mathcal{C}) \coloneqq \bigvee_{n=0}^{\infty} \sum_{r \in \mathrm{TerSeq}_{\leq n}(\mathcal{C})} \mathrm{tr}(\mathrm{End}(r))/\mathrm{tr}(\mathcal{C}) \tag{1}$$

which is exactly the *probability of termination* of $\mathcal{C}$. This is justified, because $\mathrm{Halt}(\mathcal{T}) = 1$, for any terminal (and possible) configuration $\mathcal{T}$ and $\mathrm{Halt}(\mathcal{C}) = \sum_{\mathcal{D} \text{ possible}}^{\mathcal{C} \rightsquigarrow \mathcal{D}} \Pr(\mathcal{C} \rightsquigarrow \mathcal{D})\mathrm{Halt}(\mathcal{D})$. We write $\rightsquigarrow_*$ for the reflexive and transitive closure of $\rightsquigarrow$.

Example 11. Let M be the term in Figure 4. The body of the **while** loop has the effect of performing a fair coin toss (realised through quantum measurement in the standard way) and storing the outcome in variable `b`. Therefore, starting from configuration $\mathcal{C} = (M \mid \mathtt{b} = \mathtt{tt} \mid \cdot \mid 1)$, as in Figure 5, the program has the effect of tossing a fair coin until `ff` shows up. The set of terminal configurations reachable from $\mathcal{C}$ is $\{(\mathbf{skip} \mid \mathtt{b} = \mathtt{ff} \mid \cdot \mid 2^{-i}) \mid i \in \mathbb{N}_{\geq 1}\}$ and the last component of each configuration is a 1×1 density matrix which is exactly the probability of reducing to the configuration. Therefore $\mathrm{Halt}(\mathcal{C}) = \sum_{i=1}^{\infty} 2^{-i} = 1$. Notice that,

```
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'))
}
```

Figure 6: Procedures for generating GHZ_n.

even though termination is guaranteed, there is no upper bound on the length of the reduction sequences of $\mathcal{C}$.

Example 12. The GHZ_n state is defined as $\gamma_n \coloneqq (|0\rangle^{\otimes n} + |1\rangle^{\otimes n})(\langle 0|^{\otimes n} + \langle 1|^{\otimes n})/2$. In Figure 6, we define a procedure `GHZ`, which given a natural number n, generates the state γ_n, which is represented as a list of qubits of length n. The procedure uses an auxiliary procedure `GHZnext`, which given a list of qubits representing the state γ_n, returns the state γ_{n+1} again represented as a list of qubits. The two procedures make use of some (hopefully obvious) syntactic sugar. In Figure 7, we also present the last few steps of a reduction sequence which produces γ_3 starting from configuration (`l = GHZ(n)` | `n = s(s(s(zero)))` | Ω | 1), where Ω contains the above mentioned procedures. In the reduction sequence we only show the term in evaluating position and we omit some intermediate steps. The type `ListQ` is a shorthand for **List**(**qbit**) from Example 1.

Note that the probability of termination of the initial configuration in Fig-

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

Figure 7: A reduction sequence producing GHZ_3.

ure 7 is 1, because it always reduces to the terminal configuration (**skip** | $\texttt{l} = 3 :: 2 :: 1 :: \texttt{nil} \mid \Omega \mid \gamma_3$) and $\mathrm{tr}(\gamma_3) = 1$.

4. W*-algebras

In this section we describe our denotational model. It is based on W*-algebras, which are algebras of observables (i.e. physical entities), with interesting domain-theoretic properties. We recall some background on W*-algebras and their categorical structure. We refer the reader to [5] for an encyclopaedic account on W*-algebras.

4.1. Domain-theoretic Preliminaries

Recall that a directed subset of a poset P is a non-empty subset $X \subseteq P$ in which every pair of elements of X has an upper bound in X. A poset P is a *directed-complete partial order* (*dcpo*) if each directed subset has a supremum. A poset P is *pointed* if it has a least element, usually denoted by $\bot$. A monotone map $f : P \to Q$ between posets is *Scott-continuous* if it preserves suprema of directed subsets. If P and Q are pointed and f preserves the least element, then

we say f is *strict*. We write **DCPO** ($\mathbf{DCPO}_{\perp!}$) for the category of (pointed) dcpo's and (strict) Scott-continuous maps between them.

4.2. Definition of W-algebras*

A *complex algebra* is a complex vector space V equipped with a bilinear multiplication $(- \cdot -) : V \times V \to V$, which we often write as juxtaposition. A *Banach algebra* A is a complex algebra A equipped with a submultiplicative norm $\| - \| : A \to \mathbb{R}_{\geq 0}$, i.e. $\forall x, y \in A : \|x \cdot y\| \leq \|x\|\|y\|$. A $*$-*algebra* A is a complex algebra A with an involution $(-)^* : A \to A$ such that $(x^*)^* = x$, $(x+y)^* = (x^* + y^*)$, $(xy)^* = y^*x^*$ and $(\lambda x)^* = \overline{\lambda} x^*$, for $x, y \in A$ and $\lambda \in \mathbb{C}$. A *C*-algebra* is a Banach $*$-algebra A which satisfies the C*-identity, i.e. $\|x^*x\| = \|x\|^2$ for all $x \in A$. A C*-algebra A is *unital* if it has an element $1 \in A$, such that for every $x \in A : x1 = 1x = x$. All C*-algebras in this paper are unital and for brevity we regard unitality as part of their definition.

Example 13. The algebra $M_n(\mathbb{C})$ of $n \times n$ complex matrices is a C*-algebra. In particular, the set of complex numbers $\mathbb{C}$ has a C*-algebra structure since $M_1(\mathbb{C}) \cong \mathbb{C}$. More generally, the $n \times n$ matrices valued in a C*-algebra A also form a C*-algebra $M_n(A)$. The C*-algebra of qubits is $\mathbf{qbit} \coloneqq M_2(\mathbb{C})$.

An element $x \in A$ of a C*-algebra A is called *positive* if $\exists y \in A : x = y^*y$. The *poset of positive elements* of A is denoted A^+ and its order is given by $x \leq y$ iff $(y - x) \in A^+$. The *unit interval* of A is the subposet $[0,1]_A \subseteq A^+$ of all positive elements x such that $0 \leq x \leq 1$.

Let $f : A \to B$ be a linear map between C*-algebras A and B. We say that f is *positive* if it preserves positive elements. We say that f is *completely positive* if it is n-positive for every $n \in \mathbb{N}$, i.e. the map $M_n(f) : M_n(A) \to M_n(B)$ defined for every matrix $[x_{i,j}]_{1 \leq i,j \leq n} \in M_n(A)$ by $M_n(f)([x_{i,j}]_{1 \leq i,j \leq n}) = [f(x_{i,j})]_{1 \leq i,j \leq n}$ is positive. The map f is called *multiplicative*, *involutive*, *unital* if it preserves multiplication, involution, and the unit, respectively. The map f is called *sub-unital* whenever the inequalities $0 \leq f(1) \leq 1$ hold. A *state* on a C*-algebra A is a completely positive unital map $s : A \to \mathbb{C}$.

Although W*-algebras are commonly defined in topological terms (as C*-algebras closed under several operator topologies) or equivalently in algebraic terms (as C*-algebras which are their own bicommutant), one can also equivalently define them in domain-theoretic terms [11], as we do next.

A completely positive map between C*-algebras is *normal* if its restriction to the unit interval is Scott-continuous [11, Proposition A.3]. A *W*-algebra* is a C*-algebra A such that the unit interval $[0,1]_A$ is a dcpo, and A has a separating set of normal states: for every $x \in A^+$, if $x \neq 0$, then there is a normal state $s : A \to \mathbb{C}$ such that $s(x) \neq 0$ [5, Theorem III.3.16].

A linear map $f : A \to B$ between W*-algebras A and B is called an *NCPSU-map* if f is normal, completely positive and subunital. The map f is called an *NMIU-map* if f is normal, multiplicative, involutive and unital. We note that every NMIU-map is necessarily an NCPSU-map and that W*-algebras are closed under formation of matrix algebras as in Example 13.

4.3. Categorical Structure

Let $\mathbf{W}^*_{\text{NCPSU}}$ be the category of W*-algebras and NCPSU-maps and let $\mathbf{W}^*_{\text{NMIU}}$ be its full-on-objects subcategory of NMIU-maps. Throughout the rest of the paper let $\mathbf{C} \coloneqq (\mathbf{W}^*_{\text{NCPSU}})^{\text{op}}$ and let $\mathbf{V} \coloneqq (\mathbf{W}^*_{\text{NMIU}})^{\text{op}}$. QPL types are interpreted as functors $[\![\Theta \vdash A]\!] : \mathbf{V}^{|\Theta|} \to \mathbf{V}$ and closed QPL types as objects $[\![A]\!] \in \text{Ob}(\mathbf{V}) = \text{Ob}(\mathbf{C})$. One should think of $\mathbf{V}$ as the category of *values*, because the interpretation of our values from §3 are indeed $\mathbf{V}$-morphisms. General QPL terms are interpreted as morphisms of $\mathbf{C}$, so one should think of $\mathbf{C}$ as the category of *computations*. We now describe the categorical structure of $\mathbf{V}$ and $\mathbf{C}$ and later we justify our choice for working in the opposite categories.

Both $\mathbf{C}$ and $\mathbf{V}$ have a symmetric monoidal structure when equipped with the spatial tensor product, denoted here by $(- \otimes -)$, and tensor unit $I \coloneqq \mathbb{C}$ [12, Section 10]. Moreover, $\mathbf{V}$ is symmetric monoidal closed and also complete and cocomplete [12]. $\mathbf{C}$ and $\mathbf{V}$ have finite coproducts, given by direct sums of W*-algebras [13, Proposition 4.7.3]. The coproduct of objects A and B is denoted by $A + B$ and the coproduct injections are denoted $\text{left}_{A,B} : A \to A + B$ and

$\mathrm{right}_{A,B} : B \to A + B$. Given morphisms $f : A \to C$ and $g : B \to C$, we write $[f, g] : A + B \to C$ for the unique cocone morphism induced by the coproduct. Moreover, coproducts distribute over tensor products [13, §4.6]. More specifically, there exists a natural isomorphism $d_{A,B,C} : A \otimes (B + C) \to (A \otimes B) + (A \otimes C)$ which satisfies the usual coherence conditions. The initial object in $\mathbf{C}$ is moreover a zero object and is denoted 0. The W*-algebra of bits is $\mathbf{bit} \coloneqq I + I = \mathbb{C} \oplus \mathbb{C}$.

The categories $\mathbf{V}, \mathbf{C}$ and $\mathbf{Set}$ are related by symmetric monoidal adjunctions:

$$\mathbf{Set} \underset{G}{\overset{F}{\underset{\longleftarrow}{\overset{\longrightarrow}{\bot}}}} \mathbf{V} \underset{R}{\overset{J}{\underset{\longleftarrow}{\overset{\hookrightarrow}{\bot}}}} \mathbf{C} \qquad \text{[14, pp. 11]}$$

and the subcategory inclusion J preserves coproducts and tensors up to equality.

Interpreting QPL within $\mathbf{C}$ and $\mathbf{V}$ is not an ad hoc trick. In physical terms, this corresponds to adopting the *Heisenberg picture* of quantum mechanics and this is usually done when working with infinite-dimensional W*-algebras (like we do). Semantically, this is necessary, because (1) our type system has conditional branching and we need to interpret QPL terms within a category with finite coproducts; (2) we have to be able to compute parameterised initial algebras to interpret inductive datatypes. The category $\mathbf{W}^*_{\mathrm{NCPSU}}$ has finite products, but it does *not* have coproducts, so by interpreting QPL terms within $\mathbf{C} = (\mathbf{W}^*_{\mathrm{NCPSU}})^{\mathrm{op}}$ we solve problem (1). For (2), the monoidal closure of $\mathbf{V} = (\mathbf{W}^*_{\mathrm{NMIU}})^{\mathrm{op}}$ is crucial, because it implies the tensor product preserves ω-colimits.

Convex Sums. In both $\mathbf{C}$ and $\mathbf{W}^*_{\mathrm{NCPSU}}$, morphisms are closed under *convex sums*, which are defined pointwise, as usual. More specifically, given NCPSU-maps $f_1, \ldots, f_n : A \to B$ and real numbers $p_i \in [0, 1]$ with $\sum_i p_i \leq 1$, then the map $\sum_i p_i f_i : A \to B$ is also an NCPSU-map.

Order-enrichment. For W*-algebras A and B, we define a partial order on $\mathbf{C}(A, B)$ by : $f \leq g$ iff $g - f$ is a completely positive map. Equipped with this order, our category $\mathbf{C}$ is $\mathbf{DCPO}_{\perp!}$-enriched [15, Theorem 4.3]. The least

$$
\begin{array}{ll}
\mathrm{tr} : M_n(\mathbb{C}) \to \mathbb{C} & \mathrm{new}_\rho : \mathbb{C} \to M_{2^n}(\mathbb{C}) \\
\mathrm{tr} :: A \mapsto \sum_i A_{i,i} & \mathrm{new}_\rho :: a \mapsto a\rho \\
\mathrm{tr}^\dagger : \mathbb{C} \to M_n(\mathbb{C}) & \mathrm{new}_\rho^\dagger : M_{2^n}(\mathbb{C}) \to \mathbb{C} \\
\mathrm{tr}^\dagger :: a \mapsto aI_n & \mathrm{new}_\rho^\dagger :: A \mapsto \mathrm{tr}(A\rho) \\
 & \\
\mathrm{meas} : M_2(\mathbb{C}) \to \mathbb{C} \oplus \mathbb{C} & \mathrm{unitary}_S : M_{2^n}(\mathbb{C}) \to M_{2^n}(\mathbb{C}) \\
\mathrm{meas} :: \begin{pmatrix} a & b \\ c & d \end{pmatrix} \mapsto \begin{pmatrix} a & d \end{pmatrix} & \mathrm{unitary}_S :: A \mapsto SAS^\dagger \\
\mathrm{meas}^\dagger : \mathbb{C} \oplus \mathbb{C} \to M_2(\mathbb{C}) & \mathrm{unitary}_S^\dagger : M_{2^n}(\mathbb{C}) \to M_{2^n}(\mathbb{C}) \\
\mathrm{meas}^\dagger :: \begin{pmatrix} a & d \end{pmatrix} \mapsto \begin{pmatrix} a & 0 \\ 0 & d \end{pmatrix} & \mathrm{unitary}_S^\dagger :: A \mapsto S^\dagger AS
\end{array}
$$

Figure 8: A selection of maps in the Schrödinger picture ($f : A \to B$) and their Hermitian adjoints ($f^\dagger : B \to A$) used in the Heisenberg picture.

element in $\mathbf{C}(A, B)$ is also a zero morphism and is given by the map $\mathbf{0} : A \to B$, defined by $\mathbf{0}(x) = 0$. Also, the coproduct structure and the symmetric monoidal structure are both $\mathbf{DCPO}_{\perp!}$-enriched [13, Corollary 4.9.15] [15, Theorem 4.5].

4.4. Quantum Operations

For convenience, our operational semantics adopts the *Schrödinger picture* of quantum mechanics, which is the picture most experts in quantum computing are familiar with. However, as we have just explained, our denotational semantics has to adopt the Heisenberg picture. The two pictures are equivalent in finite dimensions and we will now show how to translate from one to the other. By doing so, we provide an explicit description (in both pictures) of the required quantum maps that we need to interpret QPL.

Consider the maps in Figure 8. The map *tr* is used to trace out (or discard) parts of quantum states. Density matrices ρ are in 1-1 correspondence with the maps new_ρ, which we use in our semantics to describe (mixed) quantum states. The *meas* map simply measures a qubit in the computational basis and returns a bit as measurement outcome. The $unitary_S$ map is used for application of

a unitary transformation S. These maps work as described in the Schrödinger picture of quantum mechanics, i.e., the category $\mathbf{W}^*_{\mathrm{NCPSU}}$. For every map $f : A \to B$ among those mentioned, $f^\dagger : B \to A$ indicates its Hermitian adjoint [1]. In the Heisenberg picture, composition of maps is done in the opposite way, so we simply write $f^\ddagger \coloneqq (f^\dagger)^{\mathrm{op}} \in \mathbf{C}(A, B)$ for the Hermitian adjoint of f when seen as a morphism in $(\mathbf{W}^*_{\mathrm{NCPSU}})^{\mathrm{op}} = \mathbf{C}$. Thus, the mapping $(-)^\ddagger$ translates the above operations from the Schrödinger picture (the category $\mathbf{W}^*_{\mathrm{NCPSU}}$) to the Heisenberg picture (the category $\mathbf{C}$) of quantum mechanics.

4.5. Parameterised Initial Algebras

In order to interpret inductive datatypes, we need to be able to compute parameterised initial algebras for the functors induced by our type expressions. $\mathbf{V}$ is ideal for this, because it is cocomplete and monoidal closed and so all type expressions induce functors on $\mathbf{V}$ which preserve ω-colimits.

Definition 14 (cf. [8, §6.1])**.** Given a category $\mathbf{A}$ and a functor $T : \mathbf{A}^n \to \mathbf{A}$, with $n \geq 1$, a *parameterised initial algebra* for T is a pair $(T^\sharp, \phi^T)$, such that:

- $T^\sharp : \mathbf{A}^{n-1} \to \mathbf{A}$ is a functor;
- $\phi^T : T \circ \langle \mathrm{Id}, T^\sharp \rangle \Rightarrow T^\sharp : \mathbf{A}^{n-1} \to \mathbf{A}$ is a natural isomorphism;
- For every $A \in \mathrm{Ob}(\mathbf{A}^{n-1})$, the pair $(T^\sharp A, \phi^T_A)$ is an initial $T(A, -)$-algebra.

Proposition 15. *Every ω-cocontinuous functor $T : \mathbf{V}^n \to \mathbf{V}$ has a parameterised initial algebra $(T^\sharp, \phi^T)$ with $T^\sharp : \mathbf{V}^{n-1} \to \mathbf{V}$ being ω-cocontinuous.*

Proof. $\mathbf{V}$ is cocomplete, so this follows from [17, §4.3]. □

5. Denotational Semantics of QPL

In this section we describe the denotational semantics of QPL.

[1] This adjoint exists, because A and B are *finite-dimensional* W*-algebras which therefore have the structure of a Hilbert space when equipped with the Hilbert-Schmidt inner product [16, pp. 145].

5.1. Interpretation of Types

The interpretation of a type $\Theta \vdash A$ is a functor $[\![\Theta \vdash A]\!] : \mathbf{V}^{|\Theta|} \to \mathbf{V}$, defined by induction on the derivation of $\Theta \vdash A$ in Figure 9. As usual, one has to prove this assignment is well-defined by showing the required initial algebras exist.

Proposition 16. *The assignment in Figure 9 is well-defined.*

Proof. By induction, every $[\![\Theta \vdash A]\!]$ is an ω-cocontinuous functor and thus it has a parameterised initial algebra by Proposition 15. □

Lemma 17 (Type Substitution)**.** *Given types $\Theta, X \vdash A$ and $\Theta \vdash B$, then:*

$$[\![\Theta \vdash A[B/X]]\!] = [\![\Theta, X \vdash A]\!] \circ \langle Id, [\![\Theta \vdash B]\!] \rangle.$$

Proof. Straightforward induction. □

For simplicity, the interpretation of terms is only defined on closed types and so we introduce more concise notation for them. For any closed type $\cdot \vdash A$ we write for convenience $[\![A]\!] \coloneqq [\![\cdot \vdash A]\!](*) \in \mathrm{Ob}(\mathbf{V})$, where $*$ is the unique object of the terminal category $\mathbf{1}$. Notice also that $[\![A]\!] \in \mathrm{Ob}(\mathbf{C}) = \mathrm{Ob}(\mathbf{V})$.

Definition 18. Given a closed type $\cdot \vdash \mu X.A$, we define an isomorphism (in $\mathbf{V}$):

$$\mathrm{fold}_{\mu X.A} : [\![A[\mu X.A/X]]\!] = [\![X \vdash A]\!][\![\mu X.A]\!] \cong [\![\mu X.A]\!] : \mathrm{unfold}_{\mu X.A}$$

where the equality is given by Lemma 17 and the isomorphism is given by the initial algebra structure.

Example 19. The interpretation of the types from Example 1 are $[\![\mathbf{Nat}]\!] = \bigoplus_{i=0}^{\omega} \mathbb{C}$ and $[\![\mathbf{List}(A)]\!] = \bigoplus_{i=0}^{\omega} [\![A]\!]^{\otimes i}$. Specifically, $[\![\mathbf{List}(\mathbf{qbit})]\!] = \bigoplus_{i=0}^{\omega} \mathbb{C}^{2^i \times 2^i}$.

5.2. Discarding

Our type system is affine, so we have to construct discarding maps at all types. The tensor unit I is a terminal object in $\mathbf{V}$ (but not in $\mathbf{C}$) which leads us to the next definition.

$$
\begin{aligned}
[\![\Theta \vdash A]\!] &: \mathbf{V}^{|\Theta|} \to \mathbf{V} \\
[\![\Theta \vdash \Theta_i]\!] &= \Pi_i \\
[\![\Theta \vdash I]\!] &= K_I \\
[\![\Theta \vdash \mathbf{qbit}]\!] &= K_{\mathbf{qbit}} \\
[\![\Theta \vdash A + B]\!] &= + \circ \langle [\![\Theta \vdash A]\!], [\![\Theta \vdash B]\!] \rangle \\
[\![\Theta \vdash A \otimes B]\!] &= \otimes \circ \langle [\![\Theta \vdash A]\!], [\![\Theta \vdash B]\!] \rangle \\
[\![\Theta \vdash \mu X.A]\!] &= [\![\Theta, X \vdash A]\!]^\sharp
\end{aligned}
$$

Figure 9: Interpretations of types. K_A is the constant-A-functor.

Definition 20 (Discarding map)**.** For any W*-algebra A, let $\diamond_A : A \to I$ be the unique morphism of $\mathbf{V}$ with the indicated domain and codomain.

We will see that all values admit an interpretation as $\mathbf{V}$-morphisms and are therefore discardable. In physical terms, this means values are causal (in the sense mentioned in the introduction). Of course, this is not true for the interpretation of general terms (which correspond to $\mathbf{C}$-morphisms).

5.3. Copying

Our language is equipped with a copy operation on classical data, so we have to explain how to copy classical values. We do this by constructing a copy map defined at all *classical* types using results from [17, 18]. In this subsection, we will show that by using the categorical data of $\mathbf{Set} \underset{G}{\overset{F}{\underset{\longleftarrow}{\overset{\longrightarrow}{\bot}}}} \mathbf{V}$, one can define a copy map $\triangle_{[\![P]\!]} : [\![P]\!] \to [\![P]\!] \otimes [\![P]\!]$ for every classical type $\cdot \vdash P$, such that the triple $\left([\![P]\!], \triangle_{[\![P]\!]}, \diamond_{[\![P]\!]}\right)$ forms a cocommutative comonoid in $\mathbf{V}$. In later subsections, we will see that the interpretations of our *classical* values are comonoid homomorphisms (with respect to this comonoid structure) and therefore they may be copied.

5.3.1. Overview of the construction

We describe the comonoid structure of classical types within our category $\mathbf{V}$ (and therefore also in $\mathbf{C}$). Our methods are based on those of [17, 18], but there are some small differences compared to that work. In particular, we have less structure to work with, but our type expressions are also simpler. Nevertheless, the main idea is the same: we present an additional (classical) type interpretation for classical types within a cartesian category (in our case $\mathbf{Set}$) which then allows us to carry the comonoid structure into $\mathbf{V}$ via a symmetric monoidal adjunction. In our case, the adjunction is denoted $\mathbf{Set} \underset{G}{\overset{F}{\underset{\longleftarrow}{\overset{\longrightarrow}{\bot}}}} \mathbf{V}$ and it is defined in [19], but here we only use the categorical properties mentioned in §4, so we omit the definition of the adjunction.

5.3.2. Classical Interpretation of Classical Types

The classical interpretation of a classical type $\Theta \vdash P$ is given by a functor $(\!|\Theta \vdash P|\!) : \mathbf{Set}^{|\Theta|} \to \mathbf{Set}$, defined by induction on the derivation of $\Theta \vdash P$ in the following way:

$$\begin{aligned}
(\!|\Theta \vdash P|\!) &: \mathbf{Set}^{|\Theta|} \to \mathbf{Set} \\
(\!|\Theta \vdash \Theta_i|\!) &= \Pi_i \\
(\!|\Theta \vdash I|\!) &= K_1 \\
(\!|\Theta \vdash P + R|\!) &= \mathrm{II} \circ \langle (\!|\Theta \vdash P|\!), (\!|\Theta \vdash R|\!) \rangle \\
(\!|\Theta \vdash P \otimes R|\!) &= \times \circ \langle (\!|\Theta \vdash P|\!), (\!|\Theta \vdash R|\!) \rangle \\
(\!|\Theta \vdash \mu X.P|\!) &= (\!|\Theta, X \vdash P|\!)^\sharp,
\end{aligned}$$

where $\times$ is the categorical product in $\mathbf{Set}$ and where II is the categorical coproduct in $\mathbf{Set}$. This assignment is well-defined, because every $(\!|\Theta \vdash P|\!)$ is an ω-cocontinuous functor on $\mathbf{Set}$ and thus it has a parameterised initial algebra [18, §4.3]. For any closed classical type $\cdot \vdash P$, we write $(\!|P|\!) \coloneqq (\!|\cdot \vdash P|\!)(*) \in \mathrm{Ob}(\mathbf{Set})$.

Our next proposition shows that the standard and classical type interpretations are strongly related via a natural isomorphism, which is crucial for defining the comonoid structure.

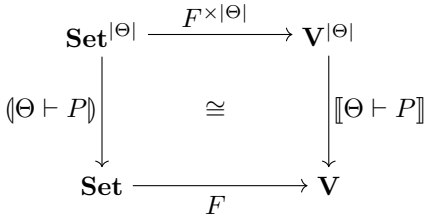

Figure 10: Relationship between type interpretations.

Proposition 21. *For any classical type $\Theta \vdash P$ there exists a natural isomorphism*

$$\iota^{\Theta \vdash P} : [\![\Theta \vdash P]\!] \circ F^{\times|\Theta|} \cong F \circ (\!|\Theta \vdash P|\!) \qquad \textit{(see Figure 10),}$$

defined by induction on the derivation of $\Theta \vdash P$. Therefore, in the special case when $\Theta = \cdot$, there exists an isomorphism $\iota^P : [\![P]\!] \cong F(\!|P|\!)$ given by $\iota^P \coloneqq \iota_*^{\cdot \vdash P}$.

Proof. The definition of $\iota^{\Theta \vdash P}$ and the proof of the theorem is essentially the same as [18, Theorem 6.1.2] and it is presented in detail in [17, A.8]. □

5.3.3. Constructing the copy map

Using ι^P, we can define a copy map on the interpretation of closed classical types.

Definition 22. For any closed classical type $\cdot \vdash P$ we define a morphism:

$$\triangle_{[\![P]\!]} \coloneqq \left([\![P]\!] \xrightarrow{\iota} F(\!|P|\!) \xrightarrow{F\langle \mathrm{id},\mathrm{id}\rangle} F\left((\!|P|\!) \times (\!|P|\!)\right) \xrightarrow{\cong} F(\!|P|\!) \otimes F(\!|P|\!) \xrightarrow{\iota^{-1} \otimes \iota^{-1}} [\![P]\!] \otimes [\![P]\!] \right)$$

called *copying*.

Proposition 23. $\left([\![P]\!], \triangle_{[\![P]\!]}, \diamond_{[\![P]\!]}\right)$ *is a cocommutative comonoid in* $\mathbf{V}$.

Proof. First, observe that $\diamond_{[\![P]\!]} = \left([\![P]\!] \xrightarrow{\iota} F(\!|P|\!) \xrightarrow{F1} F1 \xrightarrow{\cong} I \right)$, because I is terminal in $\mathbf{V}$. The rest of the proof is then identical to [18, Proposition 6.3.3]. □

Next, we identify the comonoid homomorphisms with respect to the above comonoid structure.

Definition 24. Given closed classical types P and R, we say that a morphism $f : [\![P]\!] \to [\![R]\!]$ is *classical* if $f = \left([\![P]\!] \xrightarrow{\iota} F(\!|P|\!) \xrightarrow{Ff'} F(\!|R|\!) \xrightarrow{\iota^{-1}} [\![R]\!] \right)$, for some $f' : (\!|P|\!) \to (\!|R|\!)$ in **Set**.

Proposition 25. *For every classical morphism $f : [\![P]\!] \to [\![R]\!]$, we have:*

$$\triangle_{[\![R]\!]} \circ f = (f \otimes f) \circ \triangle_{[\![P]\!]} \qquad \textit{and} \qquad \diamond_{[\![R]\!]} \circ f = \diamond_{[\![P]\!]}.$$

Therefore, f is a comonoid homomorphism with respect to Proposition 23.

Proof. Essentially the same as [18, Proposition 6.3.6]. □

5.3.4. Folding and Unfolding of Classical Types

In order to prove soundness of our semantics, we have to be able to copy all classical values. Thus, we have to show that folding and unfolding of classical inductive types are classical morphisms in the sense of Definition 24. We show this next.

Lemma 26 (Type Substitution)**.** *Let $\Theta, X \vdash P$ and $\Theta \vdash R$ be classical types. Then (see Figure 11):*

1. $(\!|\Theta \vdash P[R/X]|\!) = (\!|\Theta, X \vdash P|\!) \circ \langle Id, (\!|\Theta \vdash R|\!) \rangle$;
2. $\iota^{\Theta \vdash P[R/X]} = \iota^{\Theta, X \vdash P} \langle Id, (\!|\Theta \vdash R|\!) \rangle \circ [\![\Theta, X \vdash P]\!] \langle F^{\times |\Theta|}, \iota^{\Theta \vdash R} \rangle$

Proof. Special case of [18, Lemma 6.1.5], where $\gamma = \mathrm{id}$. Detailed proof is available in [17, A.9]. □

We may now define folding and unfolding for the classical interpretation of classical inductive types.

Definition 27. Given a closed classical type $\cdot \vdash \mu X.P$, we define an isomorphism:

$$\mathit{fold}_{\mu X.P} : (\!|P[\mu X.P/X]|\!) = (\!|X \vdash P|\!)(\!|\mu X.P|\!) \cong (\!|\mu X.P|\!) : \mathit{unfold}_{\mu X.P}$$

Like in the standard type interpretation, substitution holds up to equality, so the above folding/unfolding is given simply by the initial algebra structure of the indicated functors. We conclude the subsection with a proposition showing how the different folds/unfolds relate to one another.

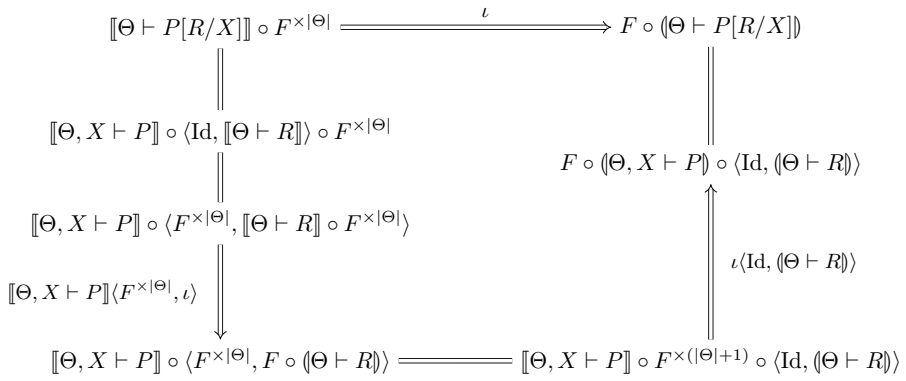

Figure 11: The commuting diagram of natural isomorphisms for Lemma 26.

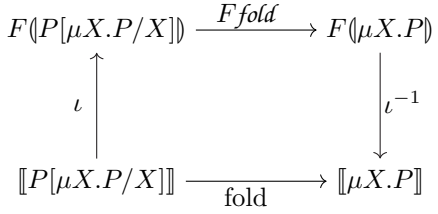

Figure 12: Relationships between the different fold/unfold maps.

Proposition 28. *Given a closed classical type* $\cdot \vdash \mu X.P$*, then*

$$\mathrm{fold}_{\mu X.P} = (\iota^{\mu X.P})^{-1} \circ F\mathit{fold}_{\mu X.P} \circ \iota^{P[\mu X.P/X]}$$

(see Figure 12).

Proof. This is simply a special case of [18, Theorem 6.1.7]. □

This shows that folding/unfolding of classical types is a classical isomorphism (Definition 24) and may thus be copied.

5.4. Interpretation of Terms

Given a variable context $\Gamma = x_1 : A_1, \ldots, x_n : A_n$, we interpret it as the object $[\![\Gamma]\!] := [\![A_1]\!] \otimes \cdots \otimes [\![A_n]\!] \in \mathrm{Ob}(\mathbf{C})$. The interpretation of a procedure context $\Pi = f_1 : A_1 \to B_1, \ldots, f_n : A_n \to B_n$ is defined to be the pointed

dcpo $[\![\Pi]\!] := \mathbf{C}(A_1, B_1) \times \cdots \times \mathbf{C}(A_n, B_n)$. A term $\Pi \vdash \langle \Gamma \rangle\ M\ \langle \Sigma \rangle$ is interpreted as a Scott-continuous function $[\![\Pi \vdash \langle \Gamma \rangle\ M\ \langle \Sigma \rangle]\!] : [\![\Pi]\!] \to \mathbf{C}([\![\Gamma]\!], [\![\Sigma]\!])$ defined by induction on the derivation of $\Pi \vdash \langle \Gamma \rangle\ M\ \langle \Sigma \rangle$ in Figure 13. For brevity, we often write $[\![M]\!] := [\![\Pi \vdash \langle \Gamma \rangle\ M\ \langle \Sigma \rangle]\!]$, when the contexts are clear or unimportant.

We now explain some of the notation used in Figure 13. The rules for manipulating qubits use the morphisms $\mathrm{new}^{\ddagger}_{|0\rangle\langle 0|}, \mathrm{meas}^{\ddagger}$ and $\mathrm{unitary}^{\ddagger}_{S}$ which are defined in §4. For the interpretation of **while** loops, given an arbitrary morphism $f : A \otimes \mathbf{bit} \to A \otimes \mathbf{bit}$ of $\mathbf{C}$, we define a Scott-continuous endofunction

$$W_f : \mathbf{C}\left(A \otimes \mathbf{bit}, A \otimes \mathbf{bit}\right) \to \mathbf{C}(A \otimes \mathbf{bit}, A \otimes \mathbf{bit})$$
$$W_f(g) = \left[\mathrm{id} \otimes \mathrm{left}_{I,I},\ g \circ f \circ (\mathrm{id} \otimes \mathrm{right}_{I,I})\right] \circ d_{A,I,I},$$

where the isomorphism $d_{A,I,I} : A \otimes (I + I) \to (A \otimes I) + (A \otimes I)$ is explained in §4. For any pointed dcpo D and Scott-continuous function $h : D \to D$, its *least fixpoint* is $\mathrm{lfp}(h) := \bigvee_{i=0}^{\infty} h^i(\bot)$, where $\bot$ is the least element of D.

Remark 29. The term semantics for defining and calling procedures does not involve any fixpoint computations. The required fixpoint computations are done when interpreting procedure stores, as we shall see next.

5.5. Interpretation of Configurations

Before we may interpret program configurations, we first have to describe how to interpret values and procedure stores.

Interpretation of Values. A qubit pointer context Q is interpreted as the object $[\![Q]\!] = \mathbf{qbit}^{\otimes |Q|}$. A value $Q \vdash v : A$ is interpreted as a morphism in $\mathbf{V}$ $[\![Q \vdash v : A]\!] : [\![Q]\!] \to [\![A]\!]$, which we abbreviate as $[\![v]\!]$ if Q and A are clear from context. It is defined by induction on the derivation of $Q \vdash v : A$ in Figure 14.

For the next theorem, recall that if $Q \vdash v : A$ is a classical value, then $Q = \cdot$.

Theorem 30. *Let $Q \vdash v : A$ be a value. Then:*

1. *$[\![v]\!]$ is discardable (i.e. causal). More specifically, $\diamond_{[\![A]\!]} \circ [\![v]\!] = \diamond_{[\![Q]\!]} = \mathrm{tr}^{\ddagger}$.*
2. *If A is classical, then $[\![v]\!]$ is copyable, i.e., $\triangle_{[\![A]\!]} \circ [\![v]\!] = ([\![v]\!] \otimes [\![v]\!]) \circ \triangle_I$.*

$$[\![\Pi \vdash \langle \Gamma \rangle \ \mathbf{new\ unit}\ u\ \langle \Gamma, u : I \rangle]\!] \coloneqq \pi \mapsto r^{-1}$$

$$[\![\Pi \vdash \langle \Gamma, x : A \rangle \ \mathbf{discard}\ x\ \langle \Gamma \rangle]\!] \coloneqq \pi \mapsto (r \circ (\mathrm{id} \otimes \diamond))$$

$$[\![\Pi \vdash \langle \Gamma, x : P \rangle\ y = \mathbf{copy}\ x\ \langle \Gamma, x : P, y : P \rangle]\!] \coloneqq \pi \mapsto (\mathrm{id} \otimes \triangle)$$

$$[\![\Pi \vdash \langle \Gamma \rangle \ \mathbf{new\ qbit}\ q\ \langle \Gamma, q : \mathbf{qbit} \rangle]\!] \coloneqq \pi \mapsto \left((\mathrm{id} \otimes \mathrm{new}^{\ddagger}_{|0\rangle\langle 0|}) \circ r^{-1}\right)$$

$$[\![\Pi \vdash \langle \Gamma, q : \mathbf{qbit} \rangle\ b = \mathbf{measure}\ q\ \langle \Gamma, b : \mathbf{bit} \rangle]\!] \coloneqq \pi \mapsto \left(\mathrm{id} \otimes \mathrm{meas}^{\ddagger}\right)$$

$$[\![\Pi \vdash \langle \Gamma, \vec{q} : \overrightarrow{\mathbf{qbit}} \rangle\ \vec{q} \mathrel{*}= S\ \langle \Gamma, \vec{q} : \overrightarrow{\mathbf{qbit}} \rangle]\!] \coloneqq \pi \mapsto \left(\mathrm{id} \otimes \mathrm{unitary}^{\ddagger}_{S}\right)$$

$$[\![\Pi \vdash \langle \Gamma \rangle\ M; N\ \langle \Sigma \rangle]\!] \coloneqq \pi \mapsto ([\![N]\!](\pi) \circ [\![M]\!](\pi))$$

$$[\![\Pi \vdash \langle \Gamma \rangle \ \mathbf{skip}\ \langle \Gamma \rangle]\!] \coloneqq \pi \mapsto \mathrm{id}$$

$$[\![\Pi \vdash \langle \Gamma, b : \mathbf{bit} \rangle \ \mathbf{while}\ b\ \mathbf{do}\ M\ \langle \Gamma, b : \mathbf{bit} \rangle]\!] \coloneqq \pi \mapsto \mathrm{lfp}(W_{[\![M]\!](\pi)})$$

$$[\![\Pi \vdash \langle \Gamma, x : A \rangle\ y = \mathbf{left}_{A,B}\ x\ \langle \Gamma, y : A + B \rangle]\!] \coloneqq \pi \mapsto (\mathrm{id} \otimes \mathrm{left}_{A,B})$$

$$[\![\Pi \vdash \langle \Gamma, x : B \rangle\ y = \mathbf{right}_{A,B}\ x\ \langle \Gamma, y : A + B \rangle]\!] \coloneqq \pi \mapsto \left(\mathrm{id} \otimes \mathrm{right}_{A,B}\right)$$

$$[\![\Pi \vdash \langle \Gamma, y : A + B \rangle \ \mathbf{case}\ y\ \mathbf{of}\ \{\mathbf{left}\ x_1 \to M_1 \mid \mathbf{right}\ x_2 \to M_2\}\ \langle \Sigma \rangle]\!] \coloneqq$$
$$\pi \mapsto ([[\![M_1]\!](\pi), [\![M_2]\!](\pi)] \circ d)$$

$$[\![\Pi \vdash \langle \Gamma, x_1 : A, x_2 : B \rangle\ x = (x_1, x_2)\ \langle \Gamma, x : A \otimes B \rangle]\!] \coloneqq \pi \mapsto \mathrm{id}$$

$$[\![\Pi \vdash \langle \Gamma, x : A \otimes B \rangle\ (x_1, x_2) = x\ \langle \Gamma, x_1 : A, x_2 : B \rangle]\!] \coloneqq \pi \mapsto \mathrm{id}$$

$$[\![\Pi \vdash \langle \Gamma, x : A[\mu X.A/X] \rangle\ y = \mathbf{fold}\ x\ \langle \Gamma, y : \mu X.A \rangle]\!] \coloneqq \pi \mapsto (\mathrm{id} \otimes \mathrm{fold})$$

$$[\![\Pi \vdash \langle \Gamma, x : \mu X.A \rangle\ y = \mathbf{unfold}\ x\ \langle \Gamma, y : A[\mu X.A/X] \rangle]\!] \coloneqq \pi \mapsto (\mathrm{id} \otimes \mathrm{unfold})$$

$$[\![\Pi \vdash \langle \Gamma \rangle \ \mathbf{proc}\ f :: \ x : A \to y : B\ \{M\}\ \langle \Gamma \rangle]\!] \coloneqq \pi \mapsto \mathrm{id}$$

$$[\![\Pi, f : A \to B \vdash \langle \Gamma, x : A \rangle\ y = f(x)\ \langle \Gamma, y : B \rangle]\!] \coloneqq (\pi, f) \mapsto (\mathrm{id} \otimes f)\,,$$

where r is the right monoidal unit. For simplicity, we omit the monoidal associator.

Figure 13: Interpretation of QPL terms.

$$\begin{aligned}
[\![\cdot \vdash * : I]\!] &\coloneqq \mathrm{id}_I \\
[\![\{n\} \vdash n : \mathbf{qbit}]\!] &\coloneqq \mathrm{id}_{\mathbf{qbit}} \\
[\![Q \vdash \mathbf{left}_{A,B} v : A + B]\!] &\coloneqq \mathrm{left} \circ [\![v]\!] \\
[\![Q \vdash \mathbf{right}_{A,B} v : A + B]\!] &\coloneqq \mathrm{right} \circ [\![v]\!] \\
[\![Q_1, Q_2 \vdash (v, w) : A \otimes B]\!] &\coloneqq [\![v]\!] \otimes [\![w]\!] \\
[\![Q \vdash \mathbf{fold}_{\mu X.A} v : \mu X.A]\!] &\coloneqq \mathrm{fold} \circ [\![v]\!]
\end{aligned}$$

Figure 14: Interpretation of values.

Proof. The first statement follows immediately, because I is terminal in $\mathbf{V}$. For the second statement, the proof is essentially the same as [18, Proposition 6.3.7]. The proof begins by defining for every classical value $\cdot \vdash v : P$ a classical value interpretation $(\!|\cdot \vdash v : P|\!) : 1 \to (\!|P|\!)$ in **Set**. It is defined by induction on the derivation of $\cdot \vdash v : P$ in the following way:

$$\begin{aligned}
(\!|\cdot \vdash * : I|\!) &\coloneqq \mathrm{id}_1 \\
(\!|\cdot \vdash \mathbf{left}_{P,R} v : P + R|\!) &\coloneqq \mathrm{inl} \circ (\!|v|\!) \\
(\!|\cdot \vdash \mathbf{right}_{P,R} v : P + R|\!) &\coloneqq \mathrm{inr} \circ (\!|v|\!) \\
(\!|\cdot \vdash (v, w) : P \otimes R|\!) &\coloneqq \langle (\!|v|\!), (\!|w|\!) \rangle \\
(\!|\cdot \vdash \mathbf{fold}_{\mu X.P} v : \mu X.P|\!) &\coloneqq \mathit{fold} \circ (\!|v|\!),
\end{aligned}$$

where inl and inr are the coproduct injections in **Set**; $\langle f, g \rangle$ is the unique map induced by the product in **Set** and *fold* is defined in Definition 27. To finish the proof, we show that

$$[\![\cdot \vdash v : P]\!] = \left(I \xrightarrow{\iota^I} F1 \xrightarrow{F(\!|\cdot \vdash v : P|\!)} F(\!|P|\!) \xrightarrow{(\iota^P)^{-1}} [\![P]\!] \right),$$

i.e., we show that $[\![\cdot \vdash v : P]\!]$ is a classical morphism (Definition 24) and it is therefore copyable (Proposition 25). The **fold** case follows by Proposition 28 and the remaining cases follow easily from the axioms of symmetric monoidal adjunctions. □

We see that, as promised, interpretations of values may always be discarded and interpretations of classical values may also be copied. Next, we explain how to interpret value contexts. For a value context $Q; \Gamma \vdash V$, its interpretation is the morphism:

$$[\![Q; \Gamma \vdash V]\!] = \left([\![Q]\!] \xrightarrow{\cong} [\![Q_1]\!] \otimes \cdots \otimes [\![Q_n]\!] \xrightarrow{[\![v_1]\!] \otimes \cdots \otimes [\![v_n]\!]} [\![\Gamma]\!] \right),$$

where $Q_i \vdash v_i : A_i$ is the splitting of Q (see §3) and $[\![\Gamma]\!] = [\![A_1]\!] \otimes \cdots \otimes [\![A_n]\!]$. Some of the Q_i can be empty and this is the reason why the definition depends on a coherent natural isomorphism. We write $[\![V]\!]$ as a shorthand for $[\![Q; \Gamma \vdash V]\!]$. Obviously, $[\![V]\!]$ is also causal thanks to Theorem 30.

Interpretation of Procedure Stores. The interpretation of a well-formed procedure store $\Pi \vdash \Omega$ is an element of $[\![\Pi]\!]$, i.e. a $|\Pi|$-tuple of morphisms from $\mathbf{C}$. It is defined by induction on $\Pi \vdash \Omega$:

$$[\![\cdot \vdash \cdot]\!] = ()$$

$$[\![\Pi, f : A \to B \vdash \Omega, f :: x : A \to y : B\ \{M\}]\!] = ([\![\Omega]\!], \mathrm{lfp}([\![M]\!]([\![\Omega]\!], -))).$$

Interpretation of Configurations. Density matrices $\rho \in M_{2^n}(\mathbb{C})$ are in 1-1 correspondence with $\mathbf{W}^*_{\mathrm{NCPSU}}$-morphisms $\mathrm{new}_\rho : \mathbb{C} \to M_{2^n}(\mathbb{C})$ which are in turn in 1-1 correspondence with $\mathbf{C}$-morphisms $\mathrm{new}^\ddagger_\rho : I \to \mathbf{qbit}^{\otimes n}$. Using this observation, we can now define the interpretation of a configuration $\mathcal{C} = (M \mid V \mid \Omega \mid \rho)$ with $\Pi; \Gamma; \Sigma; Q \vdash (M \mid V \mid \Omega \mid \rho)$ to be the morphism

$$[\![\Pi; \Gamma; \Sigma; Q \vdash (M \mid V \mid \Omega \mid \rho)]\!] \coloneqq \left(I \xrightarrow{\mathrm{new}^\ddagger_\rho} \mathbf{qbit}^{\otimes \mathrm{size}(\rho)} \xrightarrow{[\![Q; \Gamma \vdash V]\!]} [\![\Gamma]\!] \xrightarrow{[\![\Pi \vdash \langle \Gamma \rangle\ M\ \langle \Sigma \rangle]\!]([\![\Pi \vdash \Omega]\!])} [\![\Sigma]\!] \right).$$

For brevity, we simply write $[\![(M \mid V \mid \Omega \mid \rho)]\!]$ or even just $[\![\mathcal{C}]\!]$ to refer to the above morphism.

5.6. Soundness, Adequacy and Big-step Invariance

Since our operational semantics allows for branching, *soundness* is showing that the interpretation of configurations is equal to the sum of small-step reducts.

Theorem 31 (Soundness)**.** *For any non-terminal configuration $\mathcal{C}$:*

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

Proof. By induction on the shape of the term component of $\mathcal{C}$. □

Remark 32. The above sum and all sums that follow are well-defined convex sums of NCPSU-maps where the probability weights p_i have been encoded in the density matrices.

A natural question to ask is whether $[\![\mathcal{C}]\!]$ is also equal to the (potentially infinite) sum of all terminal configurations that $\mathcal{C}$ reduces to. In other words, is the interpretation of configurations also invariant with respect to big-step reduction. This is indeed the case and proving this requires considerable effort.

Theorem 33 (Big-step Invariance)**.** *For any configuration $\mathcal{C}$, we have:*

$$[\![\mathcal{C}]\!] = \bigvee_{n=0}^{\infty} \sum_{r \in \mathrm{TerSeq}_{\leq n}(\mathcal{C})} [\![\mathrm{End}(r)]\!]$$

Proof. The proof of this theorem is very technical and it is deferred to the next subsection. □

Let us consider an example which shows how this theorem may be used.

Example 34. Consider the program M and configuration $\mathcal{C}$ from Example 11. We may compute the interpretation of $\mathcal{C}$ via a fixpoint computation using the definitions we have already provided, but we may also use Theorem 33 to recover

it from the set of terminal configurations $\mathcal{C}$ reduces to. Hence, we obtain:

$$\begin{aligned}
[\![\mathcal{C}]\!] &= \bigvee_{n=0}^{\infty} \sum_{r \in \mathrm{TerSeq}_{\leq n}(\mathcal{C})} [\![\mathrm{End}(r)]\!] \\
&= \bigvee_{n=0}^{\infty} \sum_{i=1}^{n} [\![(\mathbf{skip} \mid \mathtt{b} = \mathtt{ff} \mid \cdot \mid 2^{-i})]\!] \\
&= \sum_{i=1}^{\infty} [\![(\mathbf{skip} \mid \mathtt{b} = \mathtt{ff} \mid \cdot \mid 2^{-i})]\!] \\
&= \sum_{i=1}^{\infty} 2^{-i} [\![\mathtt{ff}]\!] \\
&= \left(\sum_{i=1}^{\infty} 2^{-i} \right) [\![\mathtt{ff}]\!] \\
&= [\![\mathtt{ff}]\!],
\end{aligned}$$

where $[\![\mathtt{ff}]\!] : I \to \mathbf{bit}$ is the interpretation of the **false** value. That is, $[\![\mathtt{ff}]\!] : I \to I + I$, is the $\mathbf{V}$-morphism which is opposite to the $\mathbf{W}^*_{\mathrm{NMIU}}$-map $\pi_1 : \mathbb{C} \oplus \mathbb{C} \to \mathbb{C}$ defined by $\pi_1 \begin{pmatrix} a & b \end{pmatrix} = a$, i.e., the projection on the first component.

Theorem 33 is the main result of our paper. This is a powerful result, because with big-step invariance in place, computational adequacy[2] at all types is now a simple consequence of the causal properties of our interpretation. Observe that for any configuration $\mathcal{C}$, we have a subunital map $\diamond \circ [\![\mathcal{C}]\!] : \mathbb{C} \to \mathbb{C}$ and evaluating it at 1 yields a real number $(\diamond \circ [\![\mathcal{C}]\!])(1) \in [0, 1]$.

Theorem 35 (Adequacy)**.** *For any normalised $\mathcal{C}$:* $(\diamond \circ [\![\mathcal{C}]\!])(1) = \mathrm{Halt}(\mathcal{C})$.

Proof. Before we may prove this, we need to make a simple observation about terminal configurations.

Let $\mathcal{T} = (\mathbf{skip} \mid V \mid \Omega \mid \rho)$ be a terminal configuration. Then

$$(\diamond \circ [\![\mathcal{T}]\!])(1) = \mathrm{tr}(\rho) \tag{2}$$

[2]Recall that a computational adequacy result has to establish an equivalent *purely denotational* characterisation of the operational notion of non-termination.

To see this, we reason as follows:

$$\begin{aligned}
(\diamond \circ [\![\mathcal{T}]\!])\,(1) &= \left(\diamond \circ \mathrm{id} \circ [\![V]\!] \circ \mathrm{new}_\rho^\ddagger\right)(1) && \text{(Definition)}\\
&= \left(\mathrm{tr}^\ddagger \circ \mathrm{new}_\rho^\ddagger\right)(1) && \text{(Causality of values (see §5.5))}\\
&= \mathrm{new}_\rho^\dagger(\mathrm{tr}^\dagger(1)) && \text{(Definition)}\\
&= \mathrm{tr}(\rho) && \text{(Definition)}
\end{aligned}$$

Finally, to prove adequacy, we reason as follows:

$$\begin{aligned}
(\diamond \circ [\![\mathcal{C}]\!])\,(1) &= \left(\diamond \circ \bigvee_{n=0}^{\infty} \sum_{r \in \mathrm{TerSeq}_{\leq n}(\mathcal{C})} [\![\mathrm{End}(r)]\!]\right)(1) && \text{(Theorem 33)}\\
&= \bigvee_{n=0}^{\infty} \sum_{r \in \mathrm{TerSeq}_{\leq n}(\mathcal{C})} (\diamond \circ [\![\mathrm{End}(r)]\!])\,(1)\\
&= \bigvee_{n=0}^{\infty} \sum_{r \in \mathrm{TerSeq}_{\leq n}(\mathcal{C})} \mathrm{tr}(\mathrm{End}(r)) && \text{(Equation (2))}\\
&= \mathrm{Halt}(\mathcal{C}) && \text{(Definition)}
\end{aligned}$$

□

If $\mathcal{C}$ is not normalised, then adequacy can be recovered simply by normalising: $(\diamond \circ [\![\mathcal{C}]\!])\,(1) = \mathrm{tr}(\mathcal{C})\mathrm{Halt}(\mathcal{C})$, for any possible configuration $\mathcal{C}$. The adequacy formulation of [20] and [21] is now a special case of our more general formulation.

Corollary 36. *Let M be a closed program of unit type, i.e. $\cdot \vdash \langle\cdot\rangle\ M\ \langle\cdot\rangle$. Then:*

$$[\![(M \mid \cdot \mid \cdot \mid 1)]\!]\,(1) = \mathrm{Halt}(M \mid \cdot \mid \cdot \mid 1).$$

Proof. By Theorem 35 and because $\diamond_I = \mathrm{id}$. □

5.7. Proof of Theorem 33 (Big-step Invariance)

For brevity, we define

$$[\![\mathcal{C} \Downarrow]\!] \coloneqq \bigvee_{n=0}^{\infty} \sum_{r \in \mathrm{TerSeq}_{\leq n}(\mathcal{C})} [\![\mathrm{End}(r)]\!].$$

Note, that the sequence $\left(\sum_{r\in \mathrm{TerSeq}_{\leq n}(\mathcal{C})} [\![\mathrm{End}(r)]\!]\right)_{n\in\mathbb{N}}$ is an increasing chain in our order and thus we may take its supremum, as we did above.

To show Theorem 33 is to show:

$$[\![\mathcal{C}]\!] = [\![\mathcal{C} \Downarrow]\!].$$

As a simple consequence of Soundness (Theorem 31), we get the following.

Corollary 37. *For any configuration $\mathcal{C}$, we have $[\![\mathcal{C}]\!] \geq [\![\mathcal{C} \Downarrow]\!]$.*

Proof. We have $[\![\mathcal{C}]\!] \geq \sum_{r\in \mathrm{TerSeq}_{\leq n}(\mathcal{C})} [\![\mathrm{End}(r)]\!]$ which follows by induction on n using Theorem 31. The corollary follows by taking the supremum over n. □

In the remainder of this subsection, we will show that the converse inequality also holds, thereby finishing the proof of Theorem 33.

5.7.1. Proof Strategy

Our proof strategy is based on that of [20] where the authors establish computational adequacy at unit type.

In §5.7.2 we extend the QPL language with finitary (or bounded) primitives for recursion and we update the operational and denotational semantics in an appropriate way, so that the established language properties continue to hold (Theorem 38). In §5.7.3 we prove adequacy via the following steps: we show that any finitary configuration is strongly normalising (Lemma 39) which then allows us to prove invariance of the interpretation for finitary configurations with respect to big-step reduction (Corollary 40); we show that the finitary configurations approximate the ordinary configurations both operationally (Lemma 42) and denotationally (Lemma 44); using these results we prove invariance of the denotational semantics with respect to big-step reduction for the ordinary QPL language (Theorem 33).

5.7.2. Language Extension

We extend the syntax of QPL by adding *indexed procedure names*. We use f^n, g^n with $n \in \mathbb{N}$ to range over indexed procedure names. An indexed

procedure name f^n can be used at most n times in the operational semantics (see below), whereas an (ordinary) procedure name f may be used any number of times. We write $f^n : A \to B$ to indicate that the indexed procedure name has input type A and output type B. Procedure contexts are now also allowed to contain indexed procedure names, in addition to standard procedure names. Procedure contexts which contain a procedure name f^n cannot contain the unindexed procedure name f or any other other indexed procedure names f^m with $m \neq n$.

The term language is extended by adding new terms, some of which are indexed by natural numbers $n \geq 0$. These terms are governed by the following formation rules:

$$\frac{}{\Pi \vdash \langle \Gamma \rangle\ \mathbf{0}_{\Gamma,\Sigma}\ \langle \Sigma \rangle}$$

$$\frac{\Pi \vdash \langle \Gamma, b : \mathbf{bit} \rangle\ M\ \langle \Gamma, b : \mathbf{bit} \rangle}{\Pi \vdash \langle \Gamma, b : \mathbf{bit} \rangle\ \mathbf{while}^n\ b\ \mathbf{do}\ M\ \langle \Gamma, b : \mathbf{bit} \rangle}$$

$$\frac{\Pi, f^n : A \to B \vdash \langle x : A \rangle\ M\ \langle y : B \rangle}{\Pi \vdash \langle \Gamma \rangle\ \mathbf{proc}\ f^n ::\ x : A \to y : B\ \{M\}\ \langle \Gamma \rangle}$$

$$\frac{}{\Pi, f^n : A \to B \vdash \langle \Gamma, x : A \rangle\ y = f^n(x)\ \langle \Gamma, y : B \rangle}$$

The newly added terms are the bounded primitives for recursion that we need to prove adequacy. The term $\mathbf{0}$ should be thought of as representing an aborted computation (this is different from **skip** which represents doing nothing). We will later see it is interpreted by the zero map (the linear map which sends everything to the zero element) which explains its notation. The new loop construct $\mathbf{while}^n$ should be thought of as a while loop that would perform at most n loops and any subsequent loop immediately results in an aborted computation. Similarly, a procedure name f^n can be called at most n times and any subsequent call to this procedure also results in an aborted computation.

The formation rules for procedure stores are extended by adding a rule for indexed procedures:

$$\frac{\Pi \vdash \Omega \qquad \Pi, f^n : A \to B \vdash \langle x : A \rangle\ M\ \langle y : B \rangle}{\Pi, f^n : A \to B \vdash \Omega, f^n :: x : A \to y : B\ \{M\}}$$

The notion of a well-formed configuration is defined in the same way as before provided one uses the updated notions of well-formed terms and well-formed procedure stores. The operational semantics is extended by adding the rules:

$$\frac{}{(\mathbf{0}; P \mid V \mid \Omega \mid \rho) \rightsquigarrow (\mathbf{0} \mid V \mid \Omega \mid \rho)}$$

$$\frac{}{(\mathbf{while}^0\ b\ \mathbf{do}\ M \mid V, b = v \mid \Omega \mid \rho) \rightsquigarrow (\mathbf{0} \mid V, b = v \mid \Omega \mid \rho)}$$

$$\frac{}{\begin{array}{c}(\mathbf{while}^{n+1}\ b\ \mathbf{do}\ M \mid V, b = v \mid \Omega \mid \rho) \rightsquigarrow \\ (\mathbf{if}\ b\ \mathbf{then}\ \{M; \mathbf{while}^n\ b\ \mathbf{do}\ M\}\ \mathbf{else\ skip} \mid V, b = v \mid \Omega \mid \rho)\end{array}}$$

$$\frac{}{\begin{array}{c}(\mathbf{proc}\ f^n :: \ x : A \to y : B\ \{M\} \mid V \mid \Omega \mid \rho) \rightsquigarrow \\ (\mathbf{skip} \mid V \mid \Omega, f^n :: \ x : A \to y : B\ \{M\} \mid \rho)\end{array}}$$

$$\frac{}{\begin{array}{c}(y_1 = f^0(x_1) \mid V, x_1 = v \mid \Omega, f^0 :: \ x_2 : A \to y_2 : B\ \{M\} \mid \rho) \rightsquigarrow \\ (\mathbf{0} \mid V, x_1 = v \mid \Omega, f^0 :: \ x_2 : A \to y_2 : B\ \{M\} \mid \rho)\end{array}}$$

$$\frac{}{\begin{array}{c}(y_1 = f^{n+1}(x_1) \mid V, x_1 = v \mid \Omega, f^{n+1} :: \ x_2 : A \to y_2 : B\ \{M\} \mid \rho) \rightsquigarrow \\ (M_{\alpha_1} \mid V, x_1 = v \mid \Omega_\alpha, f^n :: \ x_2 : A \to y_2 : B\ \{M_{\alpha_2}\} \mid \rho)\end{array}}$$

where in the last rule, Ω_α is the procedure store obtained from Ω by renaming f^{n+1} to f^n within all procedure bodies contained in Ω. Similarly, M_{α_2} is the result of renaming f^{n+1} to f^n within M. The term M_{α_1} is obtained from M by also renaming f^{n+1} to f^n and then doing the same term variable renamings as in the unindexed (call) rule (see §3).

We extend the denotational semantics by adding interpretations for the

newly added terms:

$$[\![\Pi \vdash \langle \Gamma \rangle\ \mathbf{0}_{\Gamma,\Sigma}\ \langle \Sigma \rangle]\!] \coloneqq \pi \mapsto \left([\![\Gamma]\!] \xrightarrow{\mathbf{0}_{[\![\Gamma]\!],[\![\Sigma]\!]}} [\![\Sigma]\!] \right)$$

$$[\![\Pi \vdash \langle \Gamma, b : \mathbf{bit} \rangle\ \mathbf{while}^n\ b\ \mathbf{do}\ M\ \langle \Gamma, b : \mathbf{bit} \rangle]\!] \coloneqq \pi \mapsto \left([\![\Gamma]\!] \otimes \mathbf{bit} \xrightarrow{W^n_{[\![M]\!](\pi)}(\mathbf{0})} [\![\Gamma]\!] \otimes \mathbf{bit} \right)$$

$$[\![\Pi \vdash \langle \Gamma \rangle\ \mathbf{proc}\ f^n :: x : A \to y : B\ \{M\}\ \langle \Gamma \rangle]\!] \coloneqq \pi \mapsto \left([\![\Gamma]\!] \xrightarrow{\mathrm{id}} [\![\Gamma]\!] \right)$$

$$[\![\Pi, f^n : A \to B \vdash \langle \Gamma, x : A \rangle\ y = f^n(x)\ \langle \Gamma, y : B \rangle]\!] \coloneqq (\pi, f) \mapsto \left([\![\Gamma]\!] \otimes [\![A]\!] \xrightarrow{\mathrm{id} \otimes f} [\![\Gamma]\!] \otimes [\![B]\!] \right)$$

Notice that the interpretations of the indexed (proc) and (call) term rules contain no fixpoint calculations, just like their unindexed counterparts. Similarly to the unindexed case, the non-trivial calculations take place in the interpretation of the procedure store rules. The newly added rule for formation of procedure stores is interpreted in the following way:

$$[\![\Pi, f^n : A \to B \vdash \Omega, f^n :: x : A \to y : B\ \{M\}]\!] = ([\![\Omega]\!], ([\![M]\!]([\![\Omega]\!], -))^n (\mathbf{0}))\,.$$

The interpretation of configurations is now defined in the same way as before, but also using the newly added rules and updated notions. Finally, the notion of terminal configuration is also updated to include configurations of the form $(\mathbf{0} \mid V \mid \Omega \mid \rho)$. With this in place, all language properties stated in the previous sections also hold true for the extended language.

Theorem 38. *Subject Reduction (Theorem 7), Progress (Theorem 10) and Soundness (Theorem 31) also hold true for the extended language (using the updated language notions).*

5.7.3. The Proof

A term in the extended language is called *finitary* if it does not contain any unindexed procedure names or while loops. A term is called *ordinary* if it does not contain any indexed procedure names, indexed while loops or $\mathbf{0}_{\Gamma,\Sigma}$ subterms.

In other words, an ordinary term is simply a term in the base QPL language as described in §2. Similarly, a *finitary (ordinary) procedure store* Ω is a procedure store where each procedure name of Ω is indexed (unindexed) and such that its procedure body is a finitary (ordinary) term. A *finitary (ordinary) configuration* is a configuration $(M \mid V \mid \Omega \mid \rho)$ where M and Ω are finitary (ordinary). A finitary configuration is true to its name, because all of its reduction sequences terminate in a finite number of steps.

Lemma 39. *(Finitary Strong Normalisation) For any finitary configuration $\mathcal{C}$, there exists $n \in \mathbb{N}$, such that the length of every reduction sequence from $\mathcal{C}$ is at most n.*

Corollary 40. *For any finitary configuration $\mathcal{C}$, we have $[\![\mathcal{C}]\!] = [\![\mathcal{C} \Downarrow]\!]$.*

Proof. Using Lemma 39, let $n \in \mathbb{N}$ be the smallest number such that the length of every reduction sequence from $\mathcal{C}$ is at most n. It follows that

$$[\![\mathcal{C}]\!] = \sum_{r \in \mathrm{TerSeq}_{\leq n}(\mathcal{C})} [\![\mathrm{End}(r)]\!],$$

which can be easily shown by induction on n using Theorem 38. The proof concludes by recognising that

$$\begin{aligned} \sum_{r \in \mathrm{TerSeq}_{\leq n}(\mathcal{C})} [\![\mathrm{End}(r)]\!] &= \bigvee_{n=0}^{\infty} \sum_{r \in \mathrm{TerSeq}_{\leq n}(\mathcal{C})} [\![\mathrm{End}(r)]\!] \\ &= [\![\mathcal{C} \Downarrow]\!] \end{aligned}$$

□

Next, we define an approximation relation $(- \blacktriangleleft -)$ between finitary terms and ordinary terms. It is defined to be the smallest relation satisfying the following rules:

$$\frac{M' \blacktriangleleft M}{\mathbf{proc}\ f^n ::\ x : A \to y : B\ \{M'\} \blacktriangleleft \mathbf{proc}\ f ::\ x : A \to y : B\ \{M\}}$$

$$\frac{}{y = f^n(x) \blacktriangleleft y = f(x)}$$

$$\frac{M' \blacktriangleleft M}{\mathbf{while}^n\ b\ \mathbf{do}\ M' \blacktriangleleft \mathbf{while}\ b\ \mathbf{do}\ M}$$

$$\frac{M' \blacktriangleleft M \qquad N' \blacktriangleleft N}{M'; N' \blacktriangleleft M; N}$$

$$\frac{M_1' \blacktriangleleft M_1 \qquad M_2' \blacktriangleleft M_2}{\begin{array}{c}\mathbf{case}\ y\ \mathbf{of}\ \{\mathbf{left}_{A,B}\ x_1 \to M_1' \mid \mathbf{right}_{A,B}\ x_2 \to M_2'\ \} \blacktriangleleft \\ \mathbf{case}\ y\ \mathbf{of}\ \{\mathbf{left}_{A,B}\ x_1 \to M_1 \mid \mathbf{right}_{A,B}\ x_2 \to M_2\ \}\end{array}}$$

$\dfrac{}{M \blacktriangleleft M}$ all other terms except $\mathbf{0}$.

We also define an approximation relation $(- \lhd -)$ between finitary procedure stores and ordinary procedure stores. It is defined to be the smallest relation satisfying the following rules:

$$\frac{}{\cdot \lhd \cdot} \qquad \frac{\Omega' \lhd \Omega \qquad M' \blacktriangleleft M}{\Omega', f^n :: x : A \to y : B\ \{M'\} \lhd \Omega, f :: x : A \to y : B\ \{M\}}$$

Finally, we define an approximation relation $(- \sqsubset -)$ between finitary configurations and ordinary configurations. It is defined to be the smallest relation satisfying the following rule:

$$\frac{M' \blacktriangleleft M \qquad \Omega' \lhd \Omega}{(M' \mid V \mid \Omega' \mid \rho) \sqsubset (M \mid V \mid \Omega \mid \rho)}$$

Remark 41. By definition, if $(M' \mid V \mid \Omega' \mid \rho) \sqsubset (M \mid V \mid \Omega \mid \rho)$, then M and M' do not contain any $\mathbf{0}$ subterms, nor do Ω and Ω' in any of their procedure bodies.

We proceed with a lemma which shows the relation $(- \sqsubset -)$ approximates the ordinary operational semantics. But first, we introduce two new notions. Every terminal finitary configuration $\mathcal{T}$ is either of the form $(\mathbf{skip} \mid V \mid \Omega \mid \rho)$ or $(\mathbf{0} \mid V \mid \Omega \mid \rho)$. In the former case we say $\mathcal{T}$ is a **skip**-*configuration* and in the latter case we say $\mathcal{T}$ is a **0**-*configuration*.

Lemma 42. *Let $\mathcal{C}_0$ be an ordinary configuration and $\mathcal{C}_0'$ a finitary configuration with $\mathcal{C}_0' \sqsubset \mathcal{C}_0$. Let $r' = (\mathcal{C}_0' \rightsquigarrow \cdots \rightsquigarrow \mathcal{C}_n')$ be a terminal reduction sequence,*

*where $\mathcal{C}'_n$ is a **skip**-configuration. Then there exists a unique terminal reduction sequence $r = (\mathcal{C}_0 \rightsquigarrow \cdots \rightsquigarrow \mathcal{C}_n)$ such that $\mathcal{C}'_i \sqsubset \mathcal{C}_i$, which we will henceforth abbreviate by writing $r' \sqsubset r$.*

Proof. Let $\mathcal{C}'_0 = (M' \mid V' \mid \Omega' \mid \rho')$. The proof follows by induction on n. The base case $n = 0$ is trivial and the step case follows by case distinction on M'. □

In other words, the above lemma shows that lockstep execution occurs between any ordinary reduction sequence and any one of its approximating finitary reduction sequences, provided the latter terminates in the ordinary sense. This allows us to establish the following corollary.

Corollary 43. *Let $\mathcal{C}$ and $\mathcal{C}'$ be two configurations with $\mathcal{C}' \sqsubset \mathcal{C}$. Then*

$$[\![\mathcal{C}' \Downarrow]\!] \leq [\![\mathcal{C} \Downarrow]\!].$$

Proof. It suffices to show for any $n \in \mathbb{N}$ that

$$\sum_{r' \in \mathrm{TerSeq}_{\leq n}(\mathcal{C}')} [\![\mathrm{End}(r')]\!] \leq \sum_{r \in \mathrm{TerSeq}_{\leq n}(\mathcal{C})} [\![\mathrm{End}(r)]\!]$$

from which the corollary follows by taking suprema. Let $r' \in \mathrm{TerSeq}_{\leq n}(\mathcal{C}')$ be arbitrary. If $\mathrm{End}(r')$ is a $\mathbf{0}$-configuration, then $[\![\mathrm{End}(r')]\!] = \mathbf{0}$ and thus it contributes nothing to the sum and may be ignored. Otherwise $\mathrm{End}(r')$ is a **skip**-configuration and by Lemma 42, there exists a unique $r \in \mathrm{TerSeq}_{\leq n}(\mathcal{C})$, such that $r' \sqsubset r$. In this case $\mathrm{End}(r') \sqsubset \mathrm{End}(r)$ and since both of them are **skip**-configurations, it follows $[\![\mathrm{End}(r')]\!] = [\![\mathrm{End}(r)]\!]$. Moreover, if $r'_1, r'_2 \in \mathrm{TerSeq}_{\leq n}(\mathcal{C}')$ and also $r_1 \sqsupset r'_1 \neq r'_2 \sqsubset r_2$, then $r_1 \neq r_2$ (to see this, observe that r'_1 and r'_2 must differ due to branching from a quantum measurement and thus so do r_1 and r_2). Both sums are finite and we showed that for all non-trivial summands on the left there exist corresponding summands on the right which are equal to them, thus the required inequality holds. □

The next lemma shows that ordinary configurations are also approximated by finitary configurations in a denotational sense.

Lemma 44. *For any ordinary term M, procedure store Ω and configuration $\mathcal{C}$:*

$$[\![M]\!] = \bigvee_{M' \blacktriangleleft M} [\![M']\!] \qquad [\![\Omega]\!] = \bigvee_{\Omega' \lhd \Omega} [\![\Omega']\!] \qquad [\![\mathcal{C}]\!] = \bigvee_{\mathcal{C}' \sqsubset \mathcal{C}} [\![\mathcal{C}']\!].$$

Proof. Straightforward induction. □

Finally, we can prove Theorem 33.

Proof of Theorem 33. By Corollary 37, it suffices to show $[\![\mathcal{C}]\!] \leq [\![\mathcal{C} \Downarrow]\!]$. We have:

$$\begin{aligned} [\![\mathcal{C}]\!] &= \bigvee_{\mathcal{C}' \sqsubset \mathcal{C}} [\![\mathcal{C}']\!] && \text{(Lemma 44)} \\ &= \bigvee_{\mathcal{C}' \sqsubset \mathcal{C}} [\![\mathcal{C}' \Downarrow]\!] && \text{(Corollary 40)} \\ &\leq [\![\mathcal{C} \Downarrow]\!] && \text{(Corollary 43)} \end{aligned}$$

□

6. Conclusion and Related Work

There are many quantum programming languages described in the literature. For a survey see [22] and [23, pp. 129]. Some circuit programming languages (e.g. Proto-Quipper [24, 25, 26]), generate quantum circuits, but do not necessarily support executing quantum measurements. Here we focus on quantum languages which support measurement and which have either inductive datatypes or some computational adequacy result.

Our work is the first to present a detailed semantic treatment of user-defined inductive datatypes for quantum programming. In [20] and [21], the authors show how to interpret a quantum lambda calculus extended with a datatype for lists, but their syntax does not support any other inductive datatypes. These languages are equipped with lambda abstractions, whereas our language has only support for procedures. Lambda abstractions are modelled using constructions from quantitative semantics of linear logic in [20] and techniques from game semantics in [21]. We believe our model is simpler and certainly more

physically natural, because we work only with mathematical structures used by physicists in their study of quantum mechanics. Both [20] and [21] prove an adequacy result for programs of unit type. In [27], the authors discuss potential categorical models for inductive datatypes in quantum programming, but there is no detailed semantic treatment provided and there is no adequacy result, because the language lacks recursion.

Other quantum programming languages without inductive datatypes, but which prove computational adequacy results include [10, 28]. A model based on W*-algebras for a quantum lambda calculus without recursion or inductive datatypes was described in a recent manuscript [19]. In that model, it appears that currying is *not* a Scott-continuous operation, and if so, the addition of recursion renders the model neither sound, nor adequate. For this reason, we use procedures and not lambda abstractions in our language.

To conclude, we presented two novel results in quantum programming: (1) we provided a denotational semantics for a quantum programming language with inductive datatypes; (2) we proved that our denotational semantics is invariant with respect to big-step reduction. We also showed that the latter result is quite powerful by demonstrating how it immediately implies computational adequacy.

Our denotational model is based on the theory of W*-algebras, which are used to study quantum foundations. We hope this would make it useful for developing static analysis methods (based on abstract interpretation) that can be used for entanglement detection [29] and we plan on investigating this in future work.

Acknowledgements. We thank Andre Kornell, Bert Lindenhovius and Michael Mislove for discussions regarding this paper. We also thank the anonymous FoSSaCS referees for their feedback which led to multiple improvements to this article. MR acknowledges financial support from the Quantum Software Consortium, under the Gravitation programme of the Dutch Research Council NWO. The remaining authors were supported by the French projects ANR-17-CE25-0009 SoftQPro, ANR-17-CE24-0035 VanQuTe and PIA-GDN/Quantex.

References

[1] R. Péchoux, S. Perdrix, M. Rennela, V. Zamdzhiev, Quantum programming with inductive datatypes: Causality and affine type theory, in: J. Goubault-Larrecq, B. König (Eds.), Foundations of Software Science and Computation Structures - 23rd International Conference, FOSSACS 2020, Proceedings, Vol. 12077 of Lecture Notes in Computer Science, Springer, 2020, pp. 562–581. doi:10.1007/978-3-030-45231-5_29.

[2] P. W. Shor, Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer, SIAM Review 41 (2) (1999) 303–332. doi:10.1137/S0036144598347011.

[3] A. W. Harrow, A. Hassidim, S. Lloyd, Quantum Algorithm for Linear Systems of Equations, Phys. Rev. Lett. 103 (2009) 150502. doi:10.1103/PhysRevLett.103.150502.

[4] P. Selinger, Towards a quantum programming language, Mathematical Structures in Computer Science 14 (4) (2004) 527–586. doi:10.1017/S0960129504004256.

[5] M. Takesaki, Theory of Operator Algebras. Vol. I, II and III, Springer-Verlag, Berlin, 2002.

[6] A. Kissinger, S. Uijlen, A categorical semantics for causal structure, in: 32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017, IEEE Computer Society, 2017, pp. 1–12. doi:10.1109/LICS.2017.8005095.

[7] M. Abadi, M. P. Fiore, Syntactic Considerations on Recursive Types, in: Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science, New Brunswick, New Jersey, USA, July 27-30, 1996, IEEE Computer Society, 1996, pp. 242–252. doi:10.1109/LICS.1996.561324.

[8] M. P. Fiore, Axiomatic Domain Theory in Categories of Partial Maps, Ph.D. thesis, University of Edinburgh, UK (1994).

[9] W. K. Wootters, W. H. Zurek, A single quantum cannot be cloned, Nature 299 (5886) (1982) 802–803.

[10] I. Hasuo, N. Hoshino, Semantics of higher-order quantum computation via geometry of interaction, Ann. Pure Appl. Logic 168 (2) (2017) 404–469. doi:10.1016/j.apal.2016.10.010.

[11] M. Rennela, Operator Algebras in Quantum Computation, Master Thesis, Université Paris 7 Denis Diderot. (2013). arXiv:1510.06649.

[12] A. Kornell, Quantum collections, International Journal of Mathematics 28 (12) (2017) 1750085. arXiv:1202.2994, doi:10.1142/S0129167X17500859.

[13] K. Cho, Semantics for a Quantum Programming Language by Operator Algebras, Master Thesis, University of Tokyo. (2014).

[14] A. Westerbaan, Quantum Programs as Kleisli Maps, in: R. Duncan, C. Heunen (Eds.), Proceedings 13th International Conference on Quantum Physics and Logic, QPL 2016, Glasgow, Scotland, 6-10 June 2016., Vol. 236 of EPTCS, 2016, pp. 215–228. doi:10.4204/EPTCS.236.14.

[15] K. Cho, Semantics for a Quantum Programming Language by Operator Algebras, New Generation Comput. 34 (1-2) (2016) 25–68. doi:10.1007/s00354-016-0204-3.

[16] B. Westerbaan, Dagger and Dilation in the Category of Von Neumann algebras, Ph.D. thesis, Radboud University (2018). arXiv:1803.01911.

[17] B. Lindenhovius, M. Mislove, V. Zamdzhiev, LNL-FPC: The Linear/Nonlinear Fixpoint Calculus, submitted. arXiv:1906.09503.

[18] B. Lindenhovius, M. Mislove, V. Zamdzhiev, Mixed linear and non-linear recursive types, Proc. ACM Program. Lang. 3 (ICFP) (2019) 111:1–111:29. doi:10.1145/3341715.

[19] K. Cho, A. Westerbaan, Von Neumann Algebras form a Model for the Quantum Lambda Calculus, manuscript. (2016). arXiv:1603.02133.

[20] M. Pagani, P. Selinger, B. Valiron, Applying quantitative semantics to higher-order quantum computing, in: S. Jagannathan, P. Sewell (Eds.), The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014, ACM, 2014, pp. 647–658. doi:10.1145/2535838.2535879.

[21] P. Clairambault, M. de Visme, G. Winskel, Game semantics for quantum programming, PACMPL 3 (POPL) (2019) 32:1–32:29. doi:10.1145/3290345.

[22] S. J. Gay, Quantum programming languages: survey and bibliography, Mathematical Structures in Computer Science 16 (4) (2006) 581–600. doi:10.1017/S0960129506005378.

[23] M. Mosca, M. Roetteler, P. Selinger, Quantum Programming Languages (Dagstuhl Seminar 18381), Dagstuhl Reports 8 (9) (2019) 112–132. doi:10.4230/DagRep.8.9.112.

[24] F. Rios, P. Selinger, A Categorical Model for a Quantum Circuit Description Language, in: QPL, 2017. doi:10.4204/EPTCS.266.11.

[25] N. J. Ross, Algebraic and Logical Methods in Quantum Computation, Ph.D. thesis, Dalhousie University. (2015). arXiv:1510.02198.

[26] B. Lindenhovius, M. W. Mislove, V. Zamdzhiev, Enriching a Linear/Nonlinear Lambda Calculus: A Programming Language for String Diagrams, in: A. Dawar, E. Grädel (Eds.), Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018, ACM, 2018, pp. 659–668. doi:10.1145/3209108.3209196.

[27] M. Rennela, S. Staton, Classical Control and Quantum Circuits in Enriched Category Theory, Electr. Notes Theor. Comput. Sci. 336 (2018) 257–279. doi:10.1016/j.entcs.2018.03.027.

[28] U. D. Lago, C. Faggian, B. Valiron, A. Yoshimizu, The geometry of parallelism: classical, probabilistic, and quantum effects, in: G. Castagna, A. D. Gordon (Eds.), Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017, ACM, 2017, pp. 833–845.
URL http://dl.acm.org/citation.cfm?id=3009859

[29] S. Perdrix, Quantum Entanglement Analysis Based on Abstract Interpretation, in: M. Alpuente, G. Vidal (Eds.), Static Analysis, 15th International Symposium, SAS 2008, Valencia, Spain, July 16-18, 2008. Proceedings, Vol. 5079 of Lecture Notes in Computer Science, Springer, 2008, pp. 270–282. doi:10.1007/978-3-540-69166-2_18.