Effective Quantum Certification via Linear Homotopy Types

David J. Myers Mitchell Riley Hisham Sati Urs Schreiber
djm10080@nyu.edu mvr9774@nyu.edu hsati@nyu.edu us13@nyu.edu

CENTER FOR
QUANTUM &
TOPOLOGICAL
SYSTEMS
New York University
Abu Dhabi, UAE

The intricacies of realistic — namely: of classically controlled and (topologically) error-protected — quantum algorithms arguably make computer-assisted verification a practical necessity; and yet a satisfactory theory of dependent quantum data types had been missing, certainly one that would be aware of topological error-protection.

To solve this problem we present *Linear Homotopy Type Theory* (LHoTT) as a programming and certification language for universal quantum computers with classical control and topologically protected quantum gates, focusing on (1.) its categorical semantics, which is a homotopy-theoretic extension of that of Proto-Quipper and a parameterized extension of Abramsky et al.'s quantum protocols, (2.) its expression of quantum measurement as a computational effect induced from dependent linear type formation and reminiscent of Lee et al.'s dynamic lifting monad but recovering the interacting systems of Coecke et al.'s *classical structures* monads as now used in the ZX-calculus.

Namely, we have recently shown [19] that classical dependent type theory in its novel but mature full-blown form of *Homotopy Type Theory* (HoTT) is naturally a certification language for realistic topological logic gates. But given that categorical semantics of HoTT is famously provided by parameterized homotopy theory, we had argued earlier [31] for a quantum enhancement LHoTT of classical HoTT, now with semantics in parameterized *stable* homotopy theory. This linear homotopy type theory LHoTT has meanwhile been formally described [25][26]; here we explain it as the previously missing certified quantum language with monadic dynamic lifting, as announced in [32].

Concretely, we observe that besides its support, inherited from HoTT, for topological logic gates, LHoTT intrinsically provides a system of monadic computational effects which realize what in algebraic topology is known as the ambidextrous form of Grothendieck's "Motivic Yoga"; and we show how this naturally serves to code quantum circuits subject to classical control implemented via computational effects. Logically this emerges as a linearly-typed quantum version of epistemic modal logic inside LHoTT, which besides providing a philosophically satisfactory formulation of quantum measurement, makes the language validate the quantum programming language axioms proposed by Staton [34]; notably the deferred measurement principle is verified by LHoTT.

Finally we indicate the syntax of a domain-specific programming language QS (an abbreviation both for "*Quantum Systems*" and for "*QS^0-modules*" aka spectra) which sugars LHoTT to a practical quantum programming language with all these features; and we showcase QS-pseudocode for simple forms of key algorithm classes, such as teleportation, error-correction and repeat-until-success gates.

Contents

The authors acknowledge support by *Tamkeen* under *NYU AD Research Institute grant* CG008.

Submitted to:
QPL 2023

1 Introduction

The classical computational trilogy. There is a famous conceptual relation, suggestively called the "computational trilogy" (for references see [29, p. 4]) between:
(TT) *intuitionistic logic/type theory*, (CT) *category/topos-theory*, (PT) *computation/programming theory*, which is so tight that we ought to understand these as three equivalent perspectives on a unique underlying subject. A culmination of the understanding of TT↔CT was achieved more recently (for references see [19]) with the proof that intuitionistic type theory in its fully-fledged form as univalent Homotopy Type Theory (HoTT) is "the internal language" of homotopy toposes.

Completing the computational trilogy. In view of this homotopy-theoretic completion of a quest that started (somewhat coincidentally) just around the time that classical electronic computers became a practical reality, it is interesting to ask:

1. What becomes of the "computational trilogy" in view of *quantum* computation (e.g. [20]), now that quantum computers are becoming a practical reality?
2. What is homotopy-theoretic about the relation between type theory and computation (TT↔PT)?[1]

In [19] we offered a first answer to both questions at once: Homotopy computation corresponds to reversible computation by *fiber transport* (of state spaces) along — hence *path lifting* of — continuous parameter paths (the homotopy-programs), an example being the notion of "anyon braid gates" envisioned in *topological* quantum computation — the latter a famous strategy for hardware-level quantum error-protection (arguably the only way that useful quantum computation can become a practical reality).

Topological/Homotopical. Notice here (cf. [19, §2]) that the term "topological" in "topological quantum computation" serves as a synonym for what the pedantic homotopy theorist would call "homotopical" — but less-pedantic homotopy theorists are also happy to conflate "homotopical" with "topological", for instance in naming "topological Hochschild homology", following the historical origin of homotopy theory in the field of algebraic topology. Therefore, if not for historical quirks we would be speaking of "homotopical quantum computation" instead of "topological quantum computation" and the broad result of [19] would appear an almost self-evident entry in the computational trilogy:

$$\boxed{\langle\text{Homotopy Type Theory}\rangle \leftrightarrow \langle\text{Homotopy Quantum Computation}\rangle \equiv \langle\text{Topological Quantum Computation}\rangle}$$

The remaining defect of HoTT — while it readily produces the fiber transport on state spaces given by topological anyon braid gates [19, Thm. 6.8] — is that it has no native means to express, hence not to *verify* (namely: type-check), that the resulting logic gates are really *quantum* gates:

Nature of quantum data types. The hallmark of quantum processes (such as logic gates) is twofold:

1. The constraint of no-cloning / no-deletion together with the phenomenon of superposition and entanglement means that *coherent* quantum processes are described by *linear* logic and *linear* type theory.
2. De-cohering state collapse in quantum measurement, but also quantum adiabatic transport, mean that *classically controlled* quantum processes must be described by classically-*dependent* linear types.

We had argued earlier [31, 30] that where HoTT is the internal language of general ∞-toposes $\mathbf{H}$, there ought to be a linear/quantum enhancement LHoTT that is the internal language of "tangent ∞-toposes" $T\mathbf{H}$ of parameterized spectrum spectrum objects in $\mathbf{H}$, or more generally of ∞-toposes of module spectra in $\mathbf{H}$, these being the evident semantics of dependent linear homotopy types. This expectation has recently been formally brought out by [25].

[1] The traditional suggestion for impact of HoTT on actual computing/programming theory says that the structure-identity-principle (cf. [19, p. 54]) allows for practical and secure code re-use. This is a neat convenience and may eventually have practical impact, but it is hardly the answer to the question "What is homotopical computation?"

Necessity of quantum program verification. While the practical benefit of formal program verification in *classical* computation is often felt to be outweighed by the extra programming effort, it can be argued [24, p. 3-4][37] that realistic *quantum* programming makes formal verification a practical necessity; because: (1.) Real-world quantum circuits are much more complex and logically less transparent – notably when topological error-protection constrains the available machine-level gates and makes movement of qubits in the circuit topology expensive. (2.) There is no ability to inspect the state of a program to debug a quantum computer's operation, due to the non-invertible nature of quantum measurement.

Intricacy of quantum program verification. At the same time, a theoretical foundation for realistic quantum program verification had remained elusive: The problem is that plain linear types (as envisioned in [23][8][2] and implemented in languages like `QML` or `QWIRE`) only serve for basic checks of plain quantum circuits, while general quantum data specification and including the necessary classical control of quantum circuits requires a theory of classically-*dependent* linear types which conservatively extends classical dependent type theory. This was achieved only recently in [25] (see p. vii there for critique of a number of previous proposals) with the construction of (what we shall call, following [31][19]): `LHoTT`.

`LHoTT` as a quantum verification language. Here we explain `LHoTT` as a quantum language, and hence as a satisfactory quantum program verification language, entirely by "domain-specific sugaring" of its syntax to reveal the presence of language constructs which would traditionally have to be obtained by "domain-specific embedding" [11] into a host language (such as `Quipper` [9] is inside `Haskell`).

To have a name for it, we shall refer to this sugared quantum language inside `LHoTT` as "`QS`", which we suggest to read as shorthand both for:

- **Quantum Systems Language**, since even beyond quantum programming it captures key aspects of general quantum theory, such as the notion of *quantization* (whence the title of [31]);
- ***QS*-Module Language**, since even beyond dependent linear types it is a language of dependent linear *homotopy types* which in their categorical semantics in algebraic topology are (parameterized) *spectra* equivalently known as $\mathbb{S}$-*modules*, where the sphere spectrum $\mathbb{S}$, in turn, is historically denoted "QS^0" (the suspension spectrum of the 0-sphere).

Quantum language structures emergent in `LHoTT`. We explain emergence of the following features:

Features of `QS` emergent inside `LHoTT`	**much as known from...**
An adjoint system of classical and of linear (quantum) types	`QWIRE` [21]
embedded in systems of classically-dependent linear types,	Proto-`Quipper` [9][28]
equipped with closed monoidal tensor category structure	Abramsky quantum protocols [1][2]
subject to (co-)monadic type forming operations for • quantum measurement as a monadic computational effect • state preparation as comonadic computational context,	monadic dynamic lifting [13], but reproducing Coecke et al.'s classical structures (co-)monad [6][5] now of ZX-calculus fame [4]
verifying the measurement principles;	Staton's QP axioms [34]

all enhanced to handle verifiable

density matrix types (mixed state),	Selinger's original outline [33]
topological quantum gates.	our companion article [19]

Most of these features come from `LHoTT`'s verification [25, §2.4] of an axiom scheme expressing what in algebraic topology/geometry is known as the *motivic yoga of functors* (cf. [31, p. 18], and p. 5 below).

2 Semantics

For sake of space and exposition, we will here not dwell on the syntactic rules of LHoTT (which are laid out in [25] and surveyed in the companion article [26] with brief indications below in §A.1) but speak entirely category-theoretically about its type system, with the tacit understanding that all following constructions have straightforward syntactic incarnations in LHoTT, which we will discuss elsewhere, but which the expert can readily deduce.

The type system. A powerful aspect of LHoTT is that not only does it provide classically-dependent linear types, but that indeed *every* type in LHoTT is *in itself* a bundle of linear types. In full homotopy-theoretic beauty this is what makes LHoTT the internal language of tangent ∞-toposes of parameterized module spectra, but for the present purpose of quantum programming it is expedient to focus on the sub-category of types of *set-indexed complex vector spaces* (essentially the Quipper-semantics of [28], which is a very special case of parameterized $H\mathbb{C}$-module spectra, cf. [31, Ex. 3.14]).

Category of LHoTT-Types	**Fragment relevant for plain quantum computation**
$\begin{bmatrix}\mathcal{E}\\ \downarrow\\ B\end{bmatrix} \,:\, \mathrm{Type}$	objects are tuples $\mathcal{E} := \big(\mathcal{E}_b\big)_{b\colon B}$ of complex vector spaces $\mathcal{E}_b\colon$ LinType indexed by any set $B\colon$ ClType, hence equivalently complex vector bundles over discrete topological spaces B
$\begin{bmatrix}\mathcal{E}\\ \downarrow\\ B\end{bmatrix} \begin{array}{c}\xrightarrow{\quad\phi\quad}\\ \\ \xrightarrow{\quad f\quad}\end{array} \begin{bmatrix}\mathcal{E}'\\ \downarrow\\ B'\end{bmatrix}.$	morphisms (f,ϕ) from $\mathcal{E}$ to $\mathcal{E}'$ are tuples of linear maps $\big(\mathcal{E}_b \xrightarrow{\phi_b} \mathcal{E}'_{f(v)}\big)_{b\colon B}$ covering maps $B \xrightarrow{f} B'$ of index sets (of base spaces), hence equivalently are morphisms of vector bundles over arbitrary base maps
$\mathrm{ClType} \hookrightarrow \mathrm{Type}$ $B \;\mapsto\; \begin{bmatrix}0_B\\ \downarrow\\ B'\end{bmatrix}$	full inclusion of the category of sets by regarding a set B as the zero-vector bundle 0_B over it (whose fiber over any $b\colon B$ is the zero-vector space $(0_B)_b = 0$).
$\mathrm{LinType} \hookrightarrow \mathrm{Type}$ $\mathcal{V} \;\mapsto\; \begin{bmatrix}\mathcal{V}\\ \downarrow\\ *\end{bmatrix}$	Full inclusion of the category of plain vector spaces by regarding them as vector bundles over the singleton set.

The purely classical and the purely quantum types are in fact modal types for computational effects $\natural$ and $\triangleright$, respectively, which arise from the Motivic Yoga (p. 5) and whose presence on the total type system gives neat formal meaning to Bohr's infamous notion of the classical/quantum divide (see p. 6).

Quantum/Classical Data Types		**Quantum/Classical Maps**
general **bundles of linear types**	$\begin{tikzcd} \mathrm{Type} \arrow[loop left, "\natural"] \arrow[loop right, "\triangleright"] \end{tikzcd}$ $\begin{bmatrix}\mathcal{E}\\ \downarrow\\ B\end{bmatrix}$	$\mathcal{E} \longrightarrow \mathcal{E}'$ $\begin{bmatrix}\mathcal{E}\\ \downarrow\\ B\end{bmatrix} \begin{array}{c}\xrightarrow{\phi}\\ \\ =\end{array} \begin{bmatrix}f^*\mathcal{E}'\\ \downarrow\\ B\end{bmatrix} \begin{array}{c}\rightarrow\\ \\ \xrightarrow{f}\end{array} \begin{bmatrix}\mathcal{E}'\\ \downarrow\\ B'\end{bmatrix}$
purely **classical types** (bundles of zeros)	$\mathrm{ClType} \;\equiv\; \mathrm{Type}^{\natural}$ $\begin{bmatrix}B\times\{0\}\\ \downarrow\\ B\end{bmatrix}$	$B \longrightarrow B'$ $\begin{bmatrix}0_B\\ \downarrow\\ B\end{bmatrix} \begin{array}{c}\xrightarrow{\quad 0\quad}\\ \\ \xrightarrow{\quad f\quad}\end{array} \begin{bmatrix}0_{B'}\\ \downarrow\\ B'\end{bmatrix}$
purely **linear types** (bundles over point)	$\mathrm{LinType} \;\equiv\; \mathrm{Type}^{\triangleright}$ $\begin{bmatrix}\mathcal{H}\\ \downarrow\\ *\end{bmatrix}$	$\mathcal{H} \longrightarrow \mathcal{H}'$ $\begin{bmatrix}\mathcal{H}\\ \downarrow\\ *\end{bmatrix} \begin{array}{c}\xrightarrow{\quad\phi\quad}\\ \\ \xrightarrow{\quad p_*\quad}\end{array} \begin{bmatrix}\mathcal{H}'\\ \downarrow\\ *\end{bmatrix}$

Motivic Yoga. We say that a *yoga of operations* or *Motivic Yoga* (Grothendieck-Wirthmüller-style) is:

A locally cart. closed category with coproducts	Type	(1)
an ambidextrously reflected subcategory ClType (“of classical base types”), hence a functor ♮ onto a full subcategory, both left and right adjoint to the inclusion functor	$\begin{tikzcd} \mathrm{ClType} \arrow[r, hook] \arrow[r, phantom, shift left=2, "\perp"] \arrow[r, phantom, shift right=2, "\perp"'] & \mathrm{Type} \arrow[l, shift right=4, "\natural"'] \arrow[l, shift left=4, "\natural"] \arrow[loop right, "\natural"] \end{tikzcd}$	(2)
for B: ClType a symmetric closed monoidal structure $(\mathrm{Type}_B, \otimes_B, \mathbb{1}_B)$ wth coproducts on the iso-comma categories (“bundles over B”)	$\mathrm{Type}_B :\equiv \natural/B \equiv \left\{ \begin{tikzcd} \left[\mathcal{E}\right. \arrow[d] \arrow[r, "\phi"] & \left.\mathcal{E}'\right] \arrow[d] \\ \left.B\right] \arrow[r, equal] & \left[B\right] \end{tikzcd} \right\}$	(3)
for each morphism $f : B \to B'$ in ClType an adjoint triple of (“base change”) functors:	$\begin{tikzcd} \mathrm{Type}_B \arrow[r, shift left=4, "f_!"] \arrow[r, phantom, shift left=2, "\perp"] \arrow[r, phantom, shift right=2, "\perp"'] \arrow[r, shift right=4, "f_*"'] & \mathrm{Type}_{B'} \arrow[l, "f^*"description] \end{tikzcd}$	(4)

such that the following conditions hold, where FinClType ↪ ClType is the coproduct-closure of $*$:

(0) linearity: the base change (4) along finite types $B \xrightarrow{p_B} *$ is ambidextrous:

$$\text{for} \quad B : \mathrm{FinClType} \quad \text{we have} \quad (p_B)_! \simeq (p_B)_* \tag{5}$$

(i) functoriality: for composable morphisms f, g of base objects we have

$$(f^* \circ g^*) \simeq g^* \circ f^* \quad \text{and} \quad \mathrm{id}^* = \mathrm{id} \tag{6}$$

(ii) monoidalness: the pullback functors are strong monoidal in that there are natural equivalences:

$$f^*\big(\mathcal{E} \otimes_{B'} \mathcal{E}'\big) \simeq f^*\big(\mathcal{E}\big) \otimes_{B'} f^*\big(\mathcal{E}'\big) \tag{7}$$

(iii) Beck-Chevalley condition: For a pullback square in ClType the “pull-push operations” across one tip are naturally equivalent to those across the other:

$$\text{For} \quad \begin{tikzcd} & B \times_{B_0} B' \arrow[dl, "\mathrm{pr}_B"'] \arrow[dr, "\mathrm{pr}_{B'}"] & \\ B \arrow[dr, "p_B"'] & (\mathrm{pb}) & B' \arrow[dl, "p_{B'}"] \\ & B_0 & \end{tikzcd} \quad \text{we have} \quad \begin{tikzcd} & \mathrm{Type}_{B \times_{B_0} B'} \arrow[dl, "(\mathrm{pr}_B)_!"'] & \\ \mathrm{Type}_B & & \mathrm{Type}_{B'} \arrow[ul, "(\mathrm{pr}_{B'})^*"'] \arrow[dl, "(p_{B'})_!"] \\ & \mathrm{Type}_{B_0} \arrow[ul, "(p_B)^*"] & \end{tikzcd} \quad \text{and} \quad \begin{tikzcd} & \mathrm{Type}_{B \times_{B_0} B'} \arrow[dl, "(\mathrm{pr}_B)_*"'] & \\ \mathrm{Type}_B & & \mathrm{Type}_{B'} \arrow[ul, "(\mathrm{pr}_{B'})^*"'] \arrow[dl, "(p_{B'})_*"] \\ & \mathrm{Type}_{B_0} \arrow[ul, "(p_B)^*"] & \end{tikzcd} \tag{8}$$

(iv) Frobenius reciprocity / projection formula: The left pushforward of a tensor with a pullback is naturally equivalent to the tensor with the left pushforward (equivalently: f^* is strong closed):

$$f_!\big(\mathcal{E} \otimes_B f^*(\mathcal{E}')\big) \simeq f_!(\mathcal{E}) \otimes_{B'} \mathcal{E}' . \tag{9}$$

The category of indexed vector spaces, for example, satisfies the motivic yoga wrt the usual fiberwise tensor product: $\Big(\big(\mathcal{E}_b\big)_{b:\,B}\Big) \otimes_B \Big(\big(\mathcal{E}'_b\big)_{b:\,B}\Big) = \big(\mathcal{E}_b \otimes \mathcal{E}'_b\big)_{b:\,B'}$. and with left/right base change along finite types given by the direct sum of vector spaces: $B \in \mathrm{FinClType} \;\vdash\; p_!\big((\mathcal{E}_b)_{b:\,B}\big) \simeq \bigoplus_{b:\,B} \mathcal{E}_b \simeq p_*\big((\mathcal{E}_b)_{b:\,B}\big)$. This is the category that also provides semantics for Proto-`Quipper`, see [9][28][13].

Such a motivic category Type has a rich structure of globally cartesian- and linear-monoidal closure:

	Product Types	**Function Types**
classical	$\begin{bmatrix}\mathcal{V}\\ \downarrow\\ B\end{bmatrix} \times \begin{bmatrix}\mathcal{V}'\\ \downarrow\\ B'\end{bmatrix} = \begin{bmatrix}(\mathrm{pr}_B)^*\mathcal{V} \oplus (\mathrm{pr}_B)^*\mathcal{V}'\\ \downarrow\\ B\times B'\end{bmatrix}$	$\begin{bmatrix}\mathcal{V}\\ \downarrow\\ B\end{bmatrix} \to \begin{bmatrix}\mathcal{V}'\\ \downarrow\\ B'\end{bmatrix} = \begin{bmatrix}\{\,f\,\}\times\{\,\phi\,\}\times\langle\sigma : \mathbb{1}_B \to f^*\mathcal{V}'\rangle\\ \downarrow\\ \{f : B \to B'\}\times\{\phi : \mathcal{V}\to f^*\mathcal{V}'\}\end{bmatrix}$
quantum	$\begin{bmatrix}\mathcal{V}\\ \downarrow\\ B\end{bmatrix} \otimes \begin{bmatrix}\mathcal{V}'\\ \downarrow\\ B'\end{bmatrix} = \begin{bmatrix}(\mathrm{pr}_B)^*\mathcal{V} \otimes (\mathrm{pr}_B)^*\mathcal{V}'\\ \downarrow\\ B\times B'\end{bmatrix}$	$\begin{bmatrix}\mathcal{V}\\ \downarrow\\ B\end{bmatrix} \multimap \begin{bmatrix}\mathcal{V}'\\ \downarrow\\ B'\end{bmatrix} = \begin{bmatrix}\{f : B \to B'\}\times\langle\phi : \mathcal{V}\to f^*\mathcal{V}'\rangle\\ \downarrow\\ \{f : B \to B'\}\end{bmatrix}$

The Motivic Yoga induces various (co-)monadic effects on subcategories of types (background in §A.2):

First, we have these three monadic effects, capturing the quantum/classical divide inside Type:

The Quantum/Classical Divide		
modality	idempotent **monad**	**pure effect**
classical	$\natural\ :\ \mathrm{Type} \twoheadrightarrow \mathrm{ClType} \hookrightarrow \mathrm{Type}$ (strong wrt ×)	$\mathrm{return}^{\natural}_{\mathcal{E}}\ :\ \mathcal{E} \longrightarrow \natural\mathcal{E}$ $\begin{bmatrix}\mathcal{E}\\ \downarrow\\ B\end{bmatrix} \begin{matrix}\xrightarrow{0}\\ \xrightarrow{\mathrm{id}}\end{matrix} \begin{bmatrix}B\times\{0\}\\ \downarrow\\ B\end{bmatrix} \begin{matrix}\xrightarrow{0}\\ \xrightarrow{p_B}\end{matrix} \begin{bmatrix}\{0\}\\ \downarrow\\ *\end{bmatrix}$
quantum	$\triangleright\ :\ \mathrm{Type} \twoheadrightarrow \mathrm{LinType} \hookrightarrow \mathrm{Type}$ $\triangleright\ \equiv\ \mathcal{E}\ \mapsto\ \big(p_{\natural\mathcal{E}}\big)_!\mathcal{E}$ (strong wrt ⊗)	$\mathrm{return}^{\triangleright}_{\mathcal{E}}\ \mathrel{\substack{\circ\\ \circ}}\ \mathcal{E} \multimap \triangleright\mathcal{E}$ $\begin{bmatrix}\mathcal{E}\\ \downarrow\\ B\end{bmatrix} \begin{matrix}\xrightarrow{\eta^{\Diamond_B}_{\mathcal{E}}}\\ \xrightarrow{p_B}\end{matrix} \begin{bmatrix}(p_B)_!\mathcal{E}\\ \downarrow\\ *\end{bmatrix}$
quantized	$\mathrm{Q}\ :\ \mathrm{ClType} \to \mathrm{LinType} \hookrightarrow \mathrm{Type}$ $\mathrm{Q}\ \equiv\ B\ \mapsto\ \triangleright\big(B\times\mathbb{1}\big)$ (relative monad)	$\mathrm{return}^{\mathrm{Q}}_{\mathcal{E}}\ \mathrel{\substack{\circ\\ \circ}}\ B \longrightarrow \mathrm{Q}B$ $\begin{bmatrix}B\times\mathbb{1}\\ \downarrow\\ B\end{bmatrix} \begin{matrix}\xrightarrow{\eta^{\Diamond_B}_{\mathcal{E}}}\\ \xrightarrow{p_B}\end{matrix} \begin{bmatrix}(p_B)_!(B\times\mathbb{1})\\ \downarrow\\ *\end{bmatrix}$

Notice that the quantization monad Q (a relative monad [3]) expresses exactly the way in which "qbits" are the quantization of bits:

$$\begin{aligned} \mathrm{Bit} &\equiv \{0,1\} &&: \mathrm{ClType}\\ \mathrm{QBit} &\equiv \mathrm{Q}(\mathrm{Bit}) &&: \mathrm{LinType} \end{aligned}$$

Moreover, for any B : FinClType the Motivic Yoga induces a quadruple of (co-)monadic computational effects/contexts which realize a form of epistemic modal logic and which in QS serve to exhibit
1. quantum measurement as a monadic computational effect;
2. state preparation as a comonadic computational context.
We briefly indicate now how this comes about (more details in §A.3):

Via the Motivic Yoga, we find `QS` inside `LHoTT` by re-casting *dependent type formers* of `LHoTT` as *monadic computational effects* (and *comonadic computational contexts*, see §A.2) on sub-categories of types:

Classical epistemic logic from dependent types. Namely, we observe now a logical understanding of *measurement* (first classical then quantum) as a natural notion in the realm of what is traditionally known as "epistemic modal logic of possible worlds", but realized[2] here as the modal logic (cf. §A.2) induced by the standard dependent type formers in classical dependent type theory (more details in §A.3):

Given B : ClType of possible measurement outcomes ("possible worlds") the monadic effects induced by B-dependent classical data type formers constitute modalities of actual and potential B-measurements. →

In `LHoTT` the analogous construction is available for B-dependent *linear* types and yields a notion of quantum measurement which captures exactly the infamous non-deterministic branching into quantum states collapsed onto the classical measurement basis B (cf. p. 8). ↓

$$\begin{tikzcd}[column sep=large]
\text{actual data} \quad \perp & \mathrm{Type}_B \arrow[loop above, "\Diamond_B", "\text{possibly}"'] \arrow[loop below, "\Box_B"', "\text{necessarily}"] \arrow[r, shift left=3, "\coprod_B", "\text{dependent pair formation}"'] \arrow[r, shift right=3, "\prod_B"', "\text{dependent function form.}"] & \mathrm{Type} \arrow[l, "\times B" description] \arrow[loop above, "\text{\ding{73}}_B", "\text{randomly}"'] \arrow[loop below, "\bigcirc_B"', "\text{indefinitely}"] & \perp \quad \text{potential data}
\end{tikzcd}$$

necessarily $P_\bullet$ entails actually $P_\bullet$ entails possibly $P_\bullet$

$$\Box_B P_\bullet \xrightarrow{\;\epsilon^{\Box_B}_{P_\bullet}\;} P_\bullet \xrightarrow{\;\eta^{\Diamond_B}_{P_\bullet}\;} \Diamond_B P_\bullet$$

Given...

$$\underset{\text{possible world}}{b : B} \;\vdash\; \prod_{b':B} P_{b'} \xrightarrow[\text{measurement outcome}]{(p_{b'})_{b':B} \mapsto p_b} P_b \xrightarrow{p_b \mapsto (b, p_b)} \coprod_{b':B} P_{b'}$$

randomly P entails potentially P entails indefinitely P

$$\text{\ding{73}}_B P \xrightarrow{\;\epsilon^{\text{\ding{73}}_B}_{P}\;} P \xrightarrow{\;\eta^{\bigcirc_B}_{P}\;} \bigcirc_B P$$

$$\textstyle\coprod_{b:B} P \xrightarrow{(b,p) \mapsto p} P \xrightarrow{p \mapsto (b:B, p)} \prod_{b:B} P$$

Quantum epistemic modal logic from dependent linear types.

$$\begin{tikzcd}
 & & & (\mathrm{LinType}_B)_{\Diamond_B} \arrow[r, hook, "\sim"] \arrow[d, phantom, "\perp"] & (\mathrm{LinType}_B)^{\Diamond_B} \arrow[dr, phantom, "\simeq"] & & \\
\text{actual data}\perp & \mathrm{LinType}_B \arrow[loop above, "\Diamond_B", "\text{possibly}"'] \arrow[loop below, "\Box_B"', "\text{necessarily}"] \arrow[r, phantom, "\simeq"] \arrow[urr, two heads, "\oplus_B", "\text{lin. dependent pair form.}"'] \arrow[drr, two heads, "\oplus_B"', "\text{lin. dependent funct. form.}"] & \mathrm{LinType}^{\bigcirc_B} & \mathrm{LinType}_{\bigcirc_B} \arrow[l, hook] \arrow[d, phantom, "\perp"] & & \mathrm{LinType} \arrow[ll, two heads, "\mathbb{1}_B \otimes" description] \arrow[loop above, "\text{\ding{73}}_B", "\text{randomly}"'] \arrow[loop below, "\bigcirc_B"', "\text{indefinitely}"] & \perp\text{potential data} \\
 & & & (\mathrm{LinType}_B)_{\Box_B} \arrow[r, hook, "\sim"'] & (\mathrm{LinType}_B)^{\Box_B} \arrow[ur, phantom, "\simeq"] & &
\end{tikzcd}$$

principle of quantum compulsion:

necessarily $\mathcal{H}_\bullet$ entails actually $\mathcal{H}_\bullet$ entails possibly $\mathcal{H}_\bullet$ is necessarily $\mathcal{H}_\bullet$

$$\Box_B \mathcal{H}_\bullet \xrightarrow{\;\epsilon^{\Box_B}_{\mathcal{H}_\bullet}\;} \mathcal{H}_\bullet \xrightarrow{\;\eta^{\Diamond_B}_{\mathcal{H}_\bullet}\;} \Diamond_B \mathcal{H}_\bullet \underset{\text{ambidexterity}}{\simeq} \Box_B \mathcal{H}_\bullet$$

Given... obtain...

$$\begin{tikzcd}[column sep=huge]
\underset{\substack{\text{possible world}\\ \text{(classical)}}}{b : B} \;\vdash & \mathcal{H} \arrow[r, two heads, "\Sigma_{b'} |\psi_{b'}\rangle \;\mapsto\; |\psi_b\rangle", "\text{quantum state collapse}"'] \arrow[rr, bend right=30, "\text{linear projector onto sub-Hilbert space } \mathcal{H}_b"'] & \mathcal{H}_b \arrow[r, hook, "|\psi_b\rangle \mapsto \oplus_{b'} \begin{cases} |\psi_b\rangle & \text{if } b'=b \\ 0 & \text{else} \end{cases}", "\text{state preparation}"'] & \mathcal{H},
\end{tikzcd} \qquad \text{where } \mathcal{H} := \bigoplus_{b':B} \mathcal{H}_{b'}$$

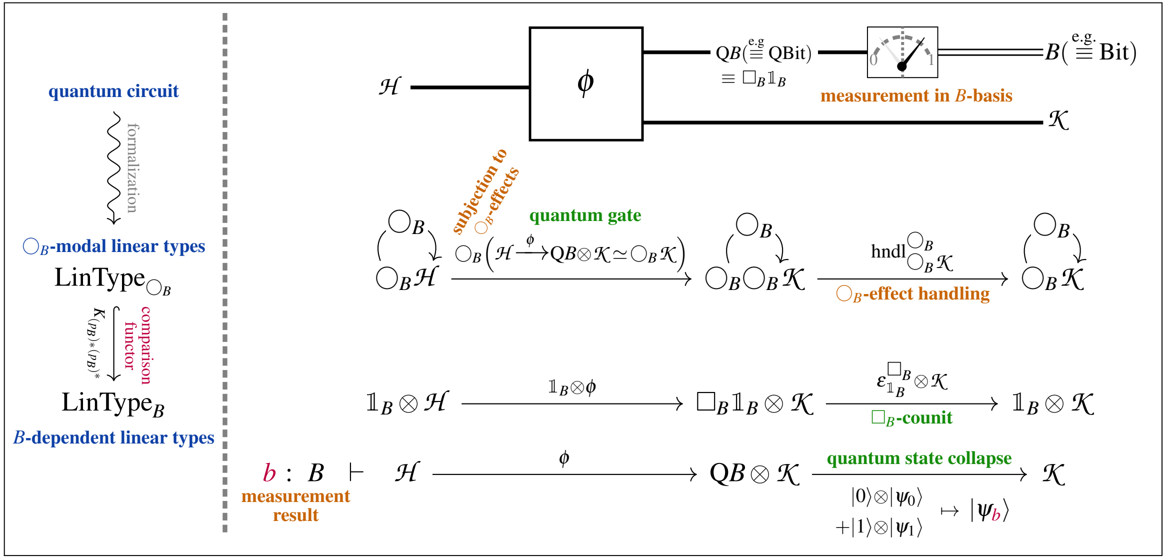
randomly $\mathcal{H}$ entails potentially $\mathcal{H}$ entails indefinitely $\mathcal{H}$

$$\text{\ding{73}}_B \mathcal{H} \xrightarrow{\;\epsilon^{\text{\ding{73}}_B}_{\mathcal{H}}\;} \mathcal{H} \xrightarrow{\;\eta^{\bigcirc_B}_{\mathcal{H}}\;} \bigcirc_B \mathcal{H}$$

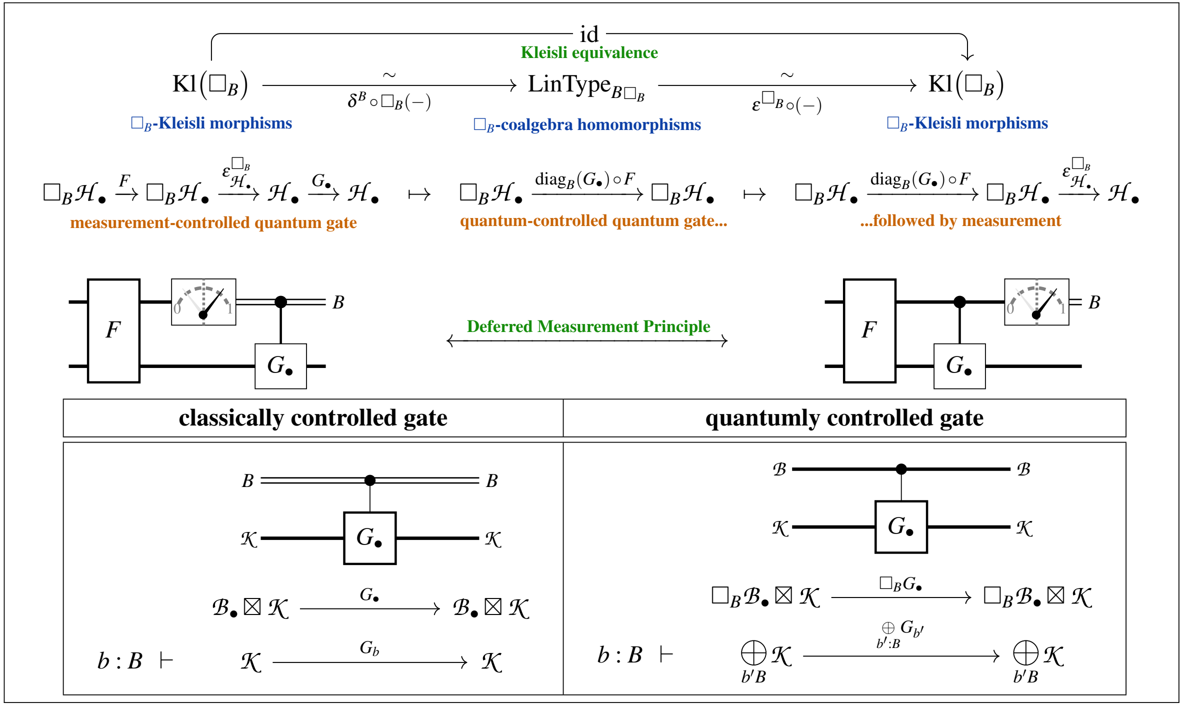
$$\bigoplus_{b:B} \mathcal{H} \xrightarrow[\text{quantum superposition}]{\oplus_b |\psi_b\rangle \mapsto \Sigma_b |\psi_b\rangle} \mathcal{H} \xrightarrow[\text{quantum parallelization}]{|\psi\rangle \mapsto \oplus_b |\psi\rangle_b} \bigoplus_{b:B} \mathcal{H}$$

[2] This fact was first observed in [ncatlab.org/nlab/revision/necessity+and+possibility/1] (2014) and highlighted in [7].

Quantum measurement as a computational effect. By the above quantum-epistemic logic, quantum B-measurement is *equivalently* (1.) the $\Box_B$-counit transformation (2.) $\bigcirc_B$-effect binding (more in §A.4):


quantum circuit
formalization
$\bigcirc_B$-modal linear types
$\mathrm{LinType}_{\bigcirc_B}$
$K_{(p_B)_*(p_B)^*}$
comparison functor
$\mathrm{LinType}_B$
B-dependent linear types
$\mathcal{H}$
ϕ
$QB(\overset{\text{e.g.}}{\equiv} \mathrm{QBit}) \equiv \Box_B \mathbb{1}_B$
measurement in B-basis
$B(\overset{\text{e.g.}}{\equiv} \mathrm{Bit})$
$\mathcal{K}$
subjection to $\bigcirc_B$-effects
quantum gate
$\bigcirc_B \mathcal{H} \xrightarrow{\bigcirc_B(\mathcal{H} \xrightarrow{\phi} QB \otimes \mathcal{K} \simeq \bigcirc_B \mathcal{K})} \bigcirc_B \bigcirc_B \mathcal{K} \xrightarrow{\mathrm{hndl}^{\bigcirc_B}_{\bigcirc_B \mathcal{K}}} \bigcirc_B \mathcal{K}$
$\bigcirc_B$-effect handling
$\mathbb{1}_B \otimes \mathcal{H} \xrightarrow{\mathbb{1}_B \otimes \phi} \Box_B \mathbb{1}_B \otimes \mathcal{K} \xrightarrow{\epsilon^{\Box_B}_{\mathbb{1}_B} \otimes \mathcal{K}} \mathbb{1}_B \otimes \mathcal{K}$
$\Box_B$-counit
$b : B \vdash \mathcal{H} \xrightarrow{\phi} QB \otimes \mathcal{K} \longrightarrow \mathcal{K}$
measurement result
quantum state collapse
$|0\rangle \otimes |\psi_0\rangle + |1\rangle \otimes |\psi_1\rangle \mapsto |\psi_b\rangle$

The deferred measurement principle (known in examples [20, §4.4] and proposed as an axiom [34, p. 6]) is now simply an instance of the Kleisli equivalence for $\Box_B$, as indicated in the following diagram:


id
Kleisli equivalence
$\mathrm{Kl}(\Box_B) \xrightarrow[\delta^B \circ \Box_B(-)]{\sim} \mathrm{LinType}_{B\Box_B} \xrightarrow[\epsilon^{\Box_B} \circ (-)]{\sim} \mathrm{Kl}(\Box_B)$
$\Box_B$-Kleisli morphisms
$\Box_B$-coalgebra homomorphisms
$\Box_B$-Kleisli morphisms
$\Box_B \mathcal{H}_\bullet \xrightarrow{F} \Box_B \mathcal{H}_\bullet \xrightarrow{\epsilon^{\Box_B}_{\mathcal{H}_\bullet}} \mathcal{H}_\bullet \xrightarrow{G_\bullet} \mathcal{H}_\bullet \mapsto \Box_B \mathcal{H}_\bullet \xrightarrow{\mathrm{diag}_B(G_\bullet) \circ F} \Box_B \mathcal{H}_\bullet \mapsto \Box_B \mathcal{H}_\bullet \xrightarrow{\mathrm{diag}_B(G_\bullet) \circ F} \Box_B \mathcal{H}_\bullet \xrightarrow{\epsilon^{\Box_B}_{\mathcal{H}_\bullet}} \mathcal{H}_\bullet$
measurement-controlled quantum gate
quantum-controlled quantum gate...
...followed by measurement
F
$G_\bullet$
B
Deferred Measurement Principle
classically controlled gate
quantumly controlled gate
B
$\mathcal{K}$
$\mathcal{B}$
$\mathcal{B}_\bullet \boxtimes \mathcal{K} \xrightarrow{G_\bullet} \mathcal{B}_\bullet \boxtimes \mathcal{K}$
$b : B \vdash \mathcal{K} \xrightarrow{G_b} \mathcal{K}$
$\Box_B \mathcal{B}_\bullet \boxtimes \mathcal{K} \xrightarrow{\Box_B G_\bullet} \Box_B \mathcal{B}_\bullet \boxtimes \mathcal{K}$
$b : B \vdash \bigoplus_{b'B} \mathcal{K} \xrightarrow{\bigoplus_{b':B} G_{b'}} \bigoplus_{b'B} \mathcal{K}$

Certifying the ZX-calculus (cf. §A.3): The canonical quantum measurement monad $\bigcirc_B$ is reminiscent of the proposal [13, §3.3], but reproduces Coecke et al.'s "classical structures" Frobenius monad [6][5]:

indefiniteness modality Coecke's classical structures randomness modality

$$\bigcirc_B \simeq B\text{-Reader} \simeq \mathbb{1}^B\text{-Writer} \simeq \mathbb{1}^B\text{-FrobWriter} \simeq \mathbb{1}^B\text{-CoWriter} \simeq B\text{-CoReader} \simeq \star_B$$

$$\mathrm{Monad} \xleftarrow{\text{quantum compulsion}} \mathrm{FrobMonad} \xrightarrow{\text{quantum compulsion}} \mathrm{CoMonad}$$

With varying bases $QB \simeq QB'$ we obtain interacting systems of such monads as used in ZX-calculus [4].

3 Pseudocode

We have seen that inside formal LHoTT the intuitionistic logic of types and the modal logic of effects apply to quantum data and processes in a natural, expressive and satisfactory way.

To make this manifest, in the boxes on this page we indicate an incremental *sugaring* of LHoTT syntax which results in a programming pseudo-language for quantum protocols with resemblance to natural language.

data	**declaration**	**semantics**
classical	$\vdash B : \mathrm{ClType}$ $\vdash b : B$	$\left[\begin{array}{c}\{0\}\\ \downarrow\\ *\end{array}\right]\begin{array}{c}\xrightarrow{0}\\ \\ \xrightarrow{b}\end{array}\left[\begin{array}{l}B\times\{0\}\\ \downarrow\\ B\end{array}\right]$
quantum	$\vdash \mathcal{H} : \mathrm{LinType}$ $\vdash \lvert\psi\rangle \mathrel{⦂} \mathcal{H}$ $\equiv$ $\mathbb{1} \vdash \lvert\psi\rangle : \mathcal{H}$	$\left[\begin{array}{c}\mathbb{1}\\ \downarrow\\ *\end{array}\right]\begin{array}{c}\xrightarrow{\lvert\psi\rangle}\\ \\ \xrightarrow{*}\end{array}\left[\begin{array}{c}\mathcal{H}\\ \downarrow\\ *\end{array}\right]$
quantized	$\vdash B : \mathrm{ClType}$ $\vdash \mathcal{H} : \mathrm{LinType}$ $\vdash \lvert-\rangle \mathrel{⦂} B \to \mathcal{H}$ $\equiv$ $\mathbb{1} \vdash \lvert-\rangle : B \to \mathcal{H}$	$\left[\begin{array}{c}\mathbb{1}\\ \downarrow\\ *\end{array}\right]\begin{array}{c}\xrightarrow{\lvert-\rangle}\\ \\ \xrightarrow{*}\end{array}\left[\begin{array}{c}\langle \mathbb{1}_B \to p_B^* \mathcal{H}\rangle\\ \downarrow\\ *\end{array}\right]$

First, by the above discussion we may give nicely suggestive names to all the return-operations of the various monads, and the extract- (i.e.: coreturn-)operations of the comonads. →

sugared syntax for the (co)pure (co)monadic (co)effects		
quantization	$\lvert-\rangle \mathrel{⦂} B \to \mathrm{Q}B$ $\lvert b\rangle \equiv \mathrm{return}^{\mathrm{Q}}_B(b)$	pure linearity
quantum measurement	$\texttt{always} \mathrel{⦂} \mathcal{H} \multimap \bigcirc_B \mathcal{H}$ $\texttt{always}\,\lvert\psi\rangle \equiv \mathrm{return}^{\bigcirc_B}_{\mathcal{H}}(\lvert\psi\rangle)$	pure indefiniteness
	$\texttt{measure} \mathrel{⦂} \bigcirc_B \mathrm{Q}B \underset{B}{\multimap} \bigcirc_B \mathbb{1}$ $\texttt{measure}\,\lvert\psi\rangle_b \equiv \mathrm{extract}^{\square_B}_{\mathbb{1}_B}(\lvert\psi\rangle_b)$	pure necessity
	$\texttt{measure} \mathrel{⦂} \mathrm{Q}B \multimap \bigcirc_B \mathbb{1}$ $\texttt{measure}\,\lvert\psi\rangle \equiv \texttt{measure}\,\texttt{always}\,\lvert\psi\rangle$	returns collapsed state & puts outcome into context
quantum state preparation	$\texttt{superpose} \mathrel{⦂} \star_B \mathcal{H} \multimap \mathcal{H}$ $\texttt{superpose}\,\lvert\psi\rangle_b \equiv \mathrm{extract}^{\star_B}_{\mathcal{H}}(\lvert\psi\rangle_b)$	pure randomness
	$\texttt{prepare} \mathrel{⦂} \star_B \mathbb{1} \multimap \star_B \mathrm{Q}B$ $\texttt{prepare}\,q_b \equiv \mathrm{return}^{\lozenge_B}_{\mathbb{1}_B}(q_b)$	pure possibility
	$\texttt{prepare} : \star_B \mathbb{1} \multimap \mathrm{Q}B$ $\texttt{prepare}\,q_b \equiv \texttt{superpose}\,\texttt{prepare}\,q_b$	prepares states in context & returns superposed state

Since declaration of $\bigcirc_B$-data is exactly conditioning on runtime measurement results ("dynamic lifting" [9, p. 5]) we denote it by an `if measured`-clause: ↓

sugared syntax for $\bigcirc_B$-data:

$$\begin{array}{ccc} (b \mapsto \lvert\psi_b\rangle) & ⦂ & (B \to \mathcal{H}) \\ \updownarrow & & \wr \\ \texttt{if measured}\ b\ \texttt{then}\ \lvert\psi_b\rangle & ⦂ & \bigcirc_B \mathcal{H} \end{array}$$

Finally, we use `for...do...`-clauses to express bind operations essentially as usual, but slightly adjusted to bring out how this encodes operations *on data-generators*, e.g. quantum gates encoded on qbit basis elements $\lvert b\rangle \equiv \mathrm{return}^{\mathrm{Q}}_B(b)$. ↓

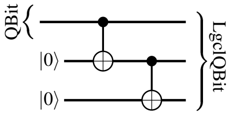
"for...do..." programming syntax for declaring effect-bound programs	
$\dfrac{\mathrm{prog} : D \to \mathscr{E}D'}{\mathrm{bind}^{\mathscr{E}}\mathrm{prog} : \mathscr{E}D \to \mathscr{E}D'}$ $\mathrm{bind}^{\mathscr{E}}\mathrm{prog} \equiv$ `for` $\boxed{\mathrm{return}^{\mathscr{E}}_D(d)}$ `do` $\mathrm{prog}(d)$	$\dfrac{\Phi : \mathscr{E}D,\quad \mathrm{prog} : D \to \mathscr{E}D'}{\phi > \mathrm{bind}^{\mathscr{E}}\mathrm{prog} : \mathscr{E}D'}$ $\phi > \mathrm{bind}^{\mathscr{E}}\mathrm{prog} \equiv$ `for` $\boxed{\mathrm{return}^{\mathscr{E}}_D(b)}$ `in` Φ `do` $\mathrm{prog}(b)$

Bit flip error correction as QS-pseudocode, is a simple but instructive example (cf. [20, §10.1.1]):

$\mathrm{LgclQBit} \;:\; \mathrm{LinType}$ $\qquad$ $\mathrm{Syndrome} \;:\; \mathrm{FinClType}$

$\mathrm{LgclQBit} \;\equiv\; \mathrm{QBit} \otimes \mathrm{QBit} \otimes \mathrm{QBit}$ $\qquad$ $\mathrm{Syndrome} \;\equiv\; \mathrm{Bit} \times \mathrm{Bit}$

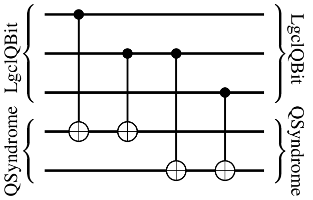
```
encode ⨾ $\mathrm{QBit} \multimap \mathrm{LgclQBit}$
encode ≡ for $|b\rangle$
           do $|b,b,b\rangle$
```



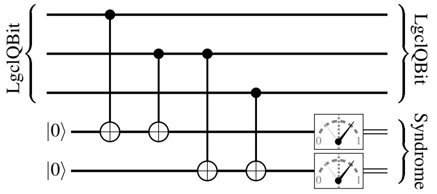
verify_circuit_encoding : encode $= (-) \otimes |0,0\rangle > \mathrm{CNOT} \otimes \mathrm{id} > \mathrm{id} \otimes \mathrm{CNOT}$

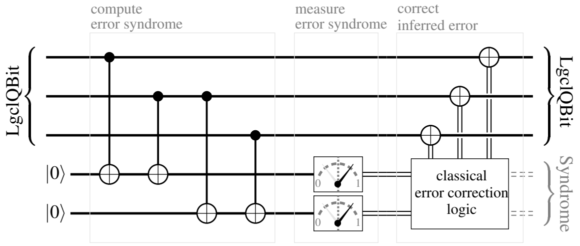
```
BitFlip ⨾ $\mathrm{Syndrome} \to (\mathrm{LgclQBit} \multimap \mathrm{LgclQBit})$
BitFlip ≡ if $(0,0)$ then $\mathrm{id} \otimes \mathrm{id} \otimes \mathrm{id}$
          if $(1,0)$ then $\mathrm{X} \otimes \mathrm{id} \otimes \mathrm{id}$
          if $(1,1)$ then $\mathrm{id} \otimes \mathrm{X} \otimes \mathrm{id}$
          if $(0,1)$ then $\mathrm{id} \otimes \mathrm{id} \otimes \mathrm{X}$
```

```
compute_syndrome ⨾ $\mathrm{QSyndrome} \otimes \mathrm{LgclQBit} \multimap \mathrm{QSyndrome} \otimes \mathrm{LgclQBit}$
compute_syndrome ≡ for $|s_1, s_2\rangle \otimes |b_1, b_2, b_3\rangle$
                     do $|s_1 + b_1 + b_2,\; s_2 + b_2 + b_3\rangle \otimes |b_1, b_2, b_2\rangle$
```



```
measure_syndrome ⨾ $\mathrm{LgclQBit} \multimap \bigcirc_{\mathrm{Syndrome}} \mathrm{LgclQBit}$
measure_syndrome ≡ for $|b_1, b_2, b_3\rangle$
                     do $|0,0\rangle \otimes |b_1, b_2, b_3\rangle$
                        > compute_syndrome
                        > measure$_{\mathrm{Syndrome}}$
```





```
correct_error ⨾ $\mathrm{LgclQBit} \multimap \bigcirc_{\mathrm{Syndrome}} \mathrm{LgclQBit}$
correct_error ≡ for $|b_1, b_2, b_3\rangle$
                  do for $|\psi\rangle$ in measure_syndrome$(|b_1, b_2, b_3\rangle)$
                       do if measured $(s_1, s_2)$ then $\mathrm{BitFlip}_{(s_1, s_2)} |\psi\rangle$
```

verify_error_correction : $(s_1, s_2 : \mathrm{Syndrome}) \to \big(\mathrm{encode} > \mathrm{BitFlip}_{s_1, s_2} > \mathrm{correct\_error} \;=\; \texttt{always}\ \mathrm{encode}\big)$

The last line asserts a term of identification type which *formally certifies* that any single bit flip on a logically encoded qbit is *always* corrected by the code (i.e.: no matter the measurement outcome). The construction of such certificates in LHoTT (not shown here, but straightforward in the present case) provides the desired formal verification of classically controlled quantum algorithms and protocols.

References

[1] S. Abramsky & B. Coecke (2004): *A categorical semantics of quantum protocols*. In: *Proceedings of IEEE Symposium on Logic in Computer Science*, pp. 415–425, doi:10.1109/LICS.2004.1319636.

[2] S. Abramsky & R. Duncan (2006): *A categorical quantum logic*. *Mathematical Structures in Computer Science* 16(3), p. 469–489, doi:10.1017/S0960129506005275.

[3] T. Altenkirch, J. Chapman & T. Uustalu (2015): *Monads need not be endofunctors*. *Logical Methods in Computer Science* Volume 11, Issue 1, doi:10.2168/LMCS-11(1:3)2015.

[4] B. Coecke & R. Duncan (2008): *Interacting Quantum Observables*. In: *Automata, Languages and Programming*, Springer, New York, pp. 298–310.

[5] B. Coecke & É O. Paquette (2008): *POVMs and Naimark's Theorem Without Sums*. *Electronic Notes in Theoretical Computer Science* 210, pp. 15–31, doi:10.1016/j.entcs.2008.04.015. QPL 2006.

[6] B. Coecke & D. Pavlović (2008): *Quantum measurements without sums*. In: *Math of Quantum Computation and Quantum Technology*, T. & F., pp. 559–596, doi:10.1201/9781584889007. arXiv:quant-ph/0608035.

[7] D Corfield (2020): *Modal homotopy type theory*. OUP. Available at `https://global.oup.com/academic/product/modal-homotopy-type-theory-9780198853404`. ISBN:9780198853404.

[8] R. Duncan (2006): *Types for quantum computing*. Ph.D. thesis, Merton College, Oxford. Available at `http://personal.strath.ac.uk/ross.duncan/papers/rduncan-thesis.pdf`.

[9] A. S. Green, P. L. Lumsdaine, N. J. Ross, P. Selinger & B. Valiron (2013): *Quipper: A Scalable Quantum Programming Language*. In: *34th ACM SIGPLAN*, PLDI '13, ACM, p. 333–342, doi:10.1145/2491956.2462177.

[10] C Heunen & J. Vicary (2019): *Categories for Quantum Theory*. Oxford University Press. ISBN:9780198739616.

[11] P. Hudak (1996): *Building Domain-Specific Embedded Languages*. *ACM Comput. Surv.* 28(4es), p. 196–es, doi:10.1145/242224.242477.

[12] A Kock (1972): *Strong functors and monoidal monads*. *Arch. Math* 23, pp. 113–120, doi:10.1007/BF01304852.

[13] D. Lee, V. Perrelle, B. Valiron & Z. Xu (2021): *Concrete Categorical Model of a Quantum Circuit Description Language with Measurement*. In: *FSTTCS 2021*, *(LIPIcs* 213, pp. 51:1–51:20, doi:10.4230/LIPIcs.FSTTCS.2021.51.

[14] S. MacLane (1971): *Categories for the Working Mathematician*. Springer, doi:10.1007/978-1-4757-4721-8.

[15] E. G. Manes (1976): *Algebraic Theories*. Springer, doi:10.1007/978-1-4612-9860-1.

[16] D McDermott & T. Uustalu (2022): *What Makes a Strong Monad?* *EPTCS* 360, pp. 113–133, doi:10.4204/EPTCS.360.6. arXiv:2207.00851.

[17] E. Moggi (1989): *Computational lambda-calculus and monads*. In: *Proceedings. Fourth Annual Symposium on Logic in Computer Science*, pp. 14–23, doi:10.1109/LICS.1989.39155.

[18] E. Moggi (1991): *Notions of computation and monads*. *Information and Computation* 93(1), pp. 55–92, doi:https://doi.org/10.1016/0890-5401(91)90052-4. Selections from 1989 IEEE Symposium on Logic in Computer Science.

[19] D. J. Myers, H. Sati & U. Schreiber (2023): *Topological Quantum Gates in HoTT*. arXiv:2303.02382.

[20] M. A. Nielsen & I. L. Chuang (2010): *Quantum Computation and Quantum Information*. CUP, doi:10.1017/CBO9780511976667.

[21] J. Paykin, R. Rand & S. Zdancewic (2017): *QWIRE: A Core Language for Quantum Circuits*. *SIGPLAN Not.* 52(1), p. 846–858, doi:10.1145/3093333.3009894.

[22] T. Petricek, D. Orchard & A. Mycroft (2013): *Coeffects: Unified Static Analysis of Context-Dependence*. In F. V. Fomin, R. Freivalds, M. Kwiatkowska & D. Peleg, editors: *Automata, Languages, and Programming*, Springer, Berlin, Heidelberg, pp. 385–397, doi:10.1007/978-3-642-39212-2_35.

[23] V. Pratt (1992): *Linear Logic For Generalized Quantum Mechanics*. In: *Workshop on Physics and Computation*, IEEE Computer Society, pp. 166–180, doi:10.1109/PHYCMP.1992.615518.

[24] R. Rand (2018): *Formally Verified Quantum Programming*. Available at `https://repository.upenn.edu/edissertations/3175`.

[25] M. Riley (2022): *A Bunched Homotopy Type Theory for Synthetic Stable Homotopy Theory*. Ph.D. thesis, Wesleyan University, doi:10.14418/wes01.3.139.

[26] M Riley (2023): *A Linear Dependent Type Theory with Identity Types as a Quantum Verification Language*. In: *QPL 2023*. (joint submission to this volume).

[27] M. Riley, E. Finster & D. Licata: *Synthetic Spectra via a Monadic and Comonadic Modality*. arXiv:2102.04099.

[28] F. Rios & P. Selinger (2018): *A categorical model for a quantum circuit description language (extended abstract)*. In: *QPL*, *EPTCS* 266, pp. 164–178, doi:10.4204/EPTCS.266.11.

[29] H. Sati & U. Schreiber (2022): *Topological Quantum Programming in* TED-K. *PlanQC* 33. arXiv:2209.08331.

[30] U. Schreiber (2014): *Differential generalized cohomology in Cohesive homotopy type theory*. Available at `https://ncatlab.org/schreiber/files/SchreiberParis2014.pdf`.

[31] U. Schreiber (2014): *Quantization via Linear Homotopy Types*. arXiv:1402.7041.

[32] U Schreiber (2022): *Quantum Data Types via Linear Homotopy Type Theory*. Quantum Software at QTML. Available at `https://ncatlab.org/schreiber/files/QuantumDataInLHoTT-221117.pdf`.

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

[34] S. Staton (2015): *Algebraic Effects, Linearity, and Quantum Programming Languages*. *SIGPLAN Not.* 50(1), p. 395–406, doi:10.1145/2775051.2676999.

[35] R. Street (2004): *Frobenius monads and pseudomonoids*. *Journal of Mathematical Physics* 45(10), pp. 3930–3948, doi:10.1063/1.1788852.

[36] T Uustalu & V. Vene (2008): *Comonadic Notions of Computation*. *Electronic Notes in Theoretical Computer Science* 203(5), pp. 263–284, doi:10.1016/j.entcs.2008.05.029. Proceedings of the Ninth Workshop on Coalgebraic Methods in Computer Science (CMCS 2008).

[37] M. Ying & Y. Feng (2018): *Model Checking Quantum Systems — A Survey*. arXiv:1807.09466.

A Appendix

In this appendix we collect some background material for reference:

§A.1 — Linear Homotopy Type Theory and Motivic Yoga
§A.2 — Computational Effects and Logical Modalities
§A.3 — Quantum Reader/Writer Monads from Dependent Linear Types
§A.4 — The Quantum Measurement Modales

Beyond this, there is certainly room for an account of more details than we provide in this short note. We will give a more extensive discussion elsewhere.

A.1 Linear Homotopy Type Theory and Motivic Yoga

An exposition of classical Homotopy Type Theory (`HoTT`) with an eye towards (topological) quantum computation may be found in [19, §5.1]. Where the types of `HoTT` may be interpreted as *parameterized homotopy types* in homotopy theory, the idea [31] of *Linear Homotopy Type Theory* `LHoTT` is to enhance `HoTT` with language structures which exhibit each type as a parameterized *stable* (∼ "linear") homotopy type, namely a bundle of spectra in the sense of algebraic topology.

For the present purpose the key point is that `LHoTT` has categorical semantics in parameterized stable homotopy theory (bundles of module spectra), as foreseen in [31] and further discussed in [27]. The upshot is that `LHoTT` naturally provides, in particular, a fully dependently-typed formal verification language for reasoning about $H\mathbb{C}$-module spectra such as the indexed complex vector spaces in §2.

As a formal language, `LHoTT` has been constructed and laid out in [25] and exposition of the basic principles of `LHoTT` as a general quantum verification language is given in the companion article [26]. We now briefly indicate how `LHoTT` verifies the *Motivic Yoga* (p. 5) that is central to the present discussion of certifying also effectful quantum computation, i.e. including the quantum measurement process and classical control:

Since every type in `LHoTT` has both a linear and non-linear aspect "$\natural$" (p. 4), the linear homotopy type theory gives us access to the non-linear (classical) data present in a variable even when its linear data has been used elsewhere in a derivation. This is done by allowing a second, 'marked' way to use every variable x, which we write $\underline{x}$. We think of this as a syntactic version of the functoriality of $\natural$ on the map represented by x. The rules for the $\natural$ operation on types then interact with this new notion of variable usage, and in [27] it is shown that this is enough to give $\natural$ all of the semantic properties we expect from (2).

To demonstrate, the $\natural$-FORM/INTRO rules

$$\natural\text{-FORM}\ \frac{\mathsf{t} \mid \underline{\Gamma}\ \vdash\ A\ \mathsf{type}}{\mathsf{t} \prec \Phi \mid \Gamma\ \vdash\ \natural A\ \mathsf{type}} \qquad\qquad \natural\text{-INTRO}\ \frac{\mathsf{t} \mid \underline{\Gamma}\ \vdash\ a : A}{\mathsf{t} \prec \Phi \mid \Gamma\ \vdash\ a^{\natural} : \natural A}$$

are the syntactic manifestation of the self-adjointness of $\natural$ as a functor Type → Type, i.e. the ambidexterity demanded in (2): First, the marking $\underline{\Gamma}$ in the assumptions of the above rules forces all uses of variables in Γ to be used via marked usages, which semantically means that the map $a : \Gamma \to A$ in the assumptions factors through a map $\natural\Gamma \to A$. From this, the conclusions of the above rules interpret as the corresponding adjunct (13) map $\Gamma \to \natural A$.

Accordingly, the universes of classical types ClType and linear types LinType (p. 4) are constructed internally as the $\natural$-modal[3] and $\natural$-connected types respectively:

$$\begin{aligned} \mathrm{ClType} &:\equiv (A : \mathrm{Type}) \times (\natural \underline{A} = A) \\ \mathrm{LinType} &:\equiv (A : \mathrm{Type}) \times (\natural \underline{A} = *) \, . \end{aligned}$$

(On the right, "=" denotes LHoTT's Martin-Löf identity type formation famous from classical HoTT.)

Given a $B : \mathrm{ClType}$, one thinks of a bundle $\mathcal{E} : \mathrm{Type}_B$ as an object that associates, to every point $b : B$, a linear type $\mathcal{E}_b : \mathrm{LinType}$. In LHoTT this holds literally, namely syntactically, in that:

$$\mathrm{Type}_B :\equiv B \to \mathrm{LinType} \, ,$$

whence $\mathcal{E}_b$ is given simply by evaluating (the fiber-assigning function that is) $\mathcal{E}$ at b.

With this understood, the base change operations (4) are almost trivial to describe type-theoretically: Given a map of classical types $f : B \to B'$, the pullback $f^* \mathcal{E}'$ is simply the precomposition $\mathcal{E}' \circ f$. And the pushforward $f_* \mathcal{E}$ is defined over a point b' as the space of sections of the bundle $\mathcal{E}$ over the fibre:

$$f_* \mathcal{E}(b') :\equiv ((x, p) : \mathsf{fib}_f(b')) \to \mathcal{E}(x) \, .$$

Last not least, LHoTT provides a rule for the formation of tensor products of linear types

$$\otimes\text{-FORM} \; \frac{\mathfrak{t} \mid \underline{\Gamma} \;\vdash\; A \;\mathsf{type} \qquad \mathfrak{t} \mid \underline{\Gamma}, \, \underline{x}^{\mathfrak{t}} : \underline{A} \;\vdash\; B \;\mathsf{type}}{\mathfrak{t} \prec \Phi \mid \Gamma \;\vdash\; (\underline{x} : A) \otimes B \;\mathsf{type}}$$

which permits the use of *marked* variables from the context: This means that the type former is computing a *fibrewise* tensor product over the parameter space of the context and makes $\otimes$ interact smoothly with $\natural$. For example, the $\otimes_B$ operation (3) on Type_B is defined simply as

$$(\mathcal{E}_1 \otimes_B \mathcal{E}_2)(b) :\equiv \mathcal{E}_1(\underline{b}) \otimes \mathcal{E}_1(\underline{b})$$

That these definitions have the properties claimed in §2 is fairly easy to check, for the full details see [25, §2.4].

A.2 Computational effects and Logical modalities.

We give a lightning survey[4] of computational effects (and computational contexts) understood as (co)monads on the type system, and of the Eilenberg-Moore-Kleisli theory of the corresponding effect handlers (context providers) understood as (co)modules, in fact as (co)modal types.

Computational effects and monads on the type system. The idea is that a computation which *nominally* produces data of some type D while however causing some computational side-effect must *de facto* produce data of some adjusted type $\mathscr{E}(D)$ which is such that the effect-part of the adjusted data can be carried alongside followup programs (whence "side effect"):

[3] Since $\natural$ is an *idempotent* monad, its modal types, in the sense recalled in (A.2), are those types which are fixed by $\natural$.

[4] The seminal idea of monadic computational effects originates with [17][18]. For a commented list of the classical but scattered literature on the topic of see: ncatlab.org/nlab/show/monad+in+computer+science.

$$\begin{tikzcd}[column sep=huge]
D_1 \arrow[r, "\overset{\textbf{first program}}{\mathrm{prog}_{12}}", "\substack{\textbf{output data}\\ \textbf{of nominal type } D_2\\ \textbf{causing effects of type } \mathscr{E}(-)}"'] & \mathscr{E}(D_2) & D_2 \arrow[r, "\overset{\textbf{second program}}{\mathrm{prog}_{23}}", "\substack{\textbf{input data}\\ \textbf{of type } D_2\\ \textbf{causing effects of type } \mathscr{E}(-)}"'] & \mathscr{E}(D_3) \arrow[d, mapsto, bend left, "\substack{\textbf{bind previous effects}\\ \textbf{into second program}}"] \\
D_1 \arrow[r, "\mathrm{prog}_{12}"] & \mathscr{E}(D_2) & \mathscr{E}(D_2) \arrow[r, "\mathrm{bind}^{\mathscr{E}}\mathrm{prog}_{23}", "\substack{\textbf{carry any previous}\\ \mathscr{E}(-)\textbf{-effects along}}"'] & \mathscr{E}(D_3) \arrow[d, mapsto, bend left, "\textbf{compose}"] \\
D_1 \arrow[rrr, "\overset{\mathscr{E}\textbf{-composite program}}{\mathrm{bind}^{\mathscr{E}}\mathrm{prog}_{23}\circ\mathrm{prog}_{12}}", "\textbf{causing cumulative } \mathscr{E}(-)\textbf{-effects}"'] & & & \mathscr{E}(D_3)
\end{tikzcd}$$

$$\begin{tikzcd}[column sep=huge]
D \arrow[r, "\mathrm{return}^{\mathscr{E}}_D", "\textbf{return plain data with trivial } \mathscr{E}(-)\textbf{-effect}"'] & \mathscr{E}(D) \\
\mathscr{E}(D) \arrow[r, "\mathrm{bind}^{\mathscr{E}}\mathrm{return}^{\mathscr{E}}_D", "= \mathrm{id}_{\mathscr{E}(D)}"'] & \mathscr{E}(D)
\end{tikzcd}$$

Such $\mathscr{E}$-effect structure on the type system is equivalently [15, p. 32][18, Prop. 1.6] a functor on the category of types

$$\begin{tikzcd}[column sep=large]
\mathscr{E} \quad : & \mathrm{Type} \arrow[r, "\text{functor underlying monad}"] & \mathrm{Type} \\
& \big(D_1 \xrightarrow{f} D_2\big) \arrow[r, mapsto] & \mathrm{bind}^{\mathscr{E}}\Big(D_1 \xrightarrow{f} D_2 \xrightarrow{\mathrm{return}^{\mathscr{E}}_D} \mathscr{E}(D_2)\Big)
\end{tikzcd} \tag{10}$$

which carries the structure of a **monad**, namely natural transformations

$$\begin{tikzcd}[column sep=large]
D \arrow[r, "\overset{\textbf{monad unit}}{\eta_D \equiv \mathrm{return}^{\mathscr{E}}_D}"] & \mathscr{E}(D) & \mathscr{E}\big(\mathscr{E}(D)\big) \arrow[r, "\overset{\textbf{monad multiplication}}{\mu_D \equiv \mathrm{bind}^{\mathscr{E}}\mathrm{id}_{\mathscr{E}(D)}}"] & \mathscr{E}(D)
\end{tikzcd} \tag{11}$$

satisfying the axioms of a unital monoid, in that they make the following natural squares commute

$$\begin{tikzcd}[column sep=large]
\mathscr{E}(D) \arrow[r, "\overset{\textbf{unitality}}{\eta_{\mathscr{E}(D)}}"] \arrow[d, "\mathscr{E}(\eta_D)"'] \arrow[dr, "\mathrm{id}_{\mathscr{E}(D)}" description] & \mathscr{E}\big(\mathscr{E}(D)\big) \arrow[d, "\mu_D"] \\
\mathscr{E}\big(\mathscr{E}(D)\big) \arrow[r, "\mu_D"] & \mathscr{E}(D)
\end{tikzcd}
\qquad
\begin{tikzcd}[column sep=large]
\mathscr{E}\Big(\mathscr{E}\big(\mathscr{E}(D)\big)\Big) \arrow[r, "\overset{\textbf{unitality}}{\mu_{\mathscr{E}(D)}}"] \arrow[d, "\mathscr{E}(\mu_D)"] & \mathscr{E}\big(\mathscr{E}(D)\big) \arrow[d, "\mu_D"] \\
\mathscr{E}\big(\mathscr{E}(D)\big) \arrow[r, "\mu_D"] & \mathscr{E}(D)
\end{tikzcd}$$

Monads induced by adjunctions. Such monads arise from (and give rise to, see (14) below) *adjunctions*, namely pairs of back-and-forth functors on the category of types

$$\begin{tikzcd}[column sep=huge]
\mathrm{Type}' \arrow[r, phantom, "\perp"] \arrow[from=1-2, to=1-1, shift right=2, "\overset{\textbf{left adjoint}}{L}"'] \arrow[r, shift right=2, "\underset{\textbf{right adjoint}}{R}"'] & \mathrm{Type} \arrow[loop right, "R\circ L =: \mathscr{E} \quad \textbf{induced monad}"]
\end{tikzcd} \tag{12}$$

equipped with a natural isomorphism (forming “adjuncts”)

$$\begin{tikzcd}
\mathrm{Hom}_{\mathrm{Type}'}\big(L(-), -\big) \arrow[r, leftrightarrow, "\widetilde{(-)}"] & \mathrm{Hom}_{\mathrm{Type}}\big(-, R(-)\big)
\end{tikzcd} \tag{13}$$

and (equivalently), with natural transformations

$$\overset{\textbf{adjunction unit}}{\eta_D \equiv \widetilde{\mathrm{id}_{L_D}} \;:\; D \longrightarrow R \circ L(D)} \qquad\qquad \overset{\textbf{adjunction co-unit}}{\varepsilon_{D'} \equiv \widetilde{\mathrm{id}_{R_{D'}}} \;:\; L \circ R(D') \longrightarrow D'}$$

satisfying the *zig-zag identities*

$$\varepsilon_{L(D)} \circ L(\eta_D) \;=\; \mathrm{id}_D \qquad\qquad R(\varepsilon_{D'}) \circ \eta_{R(D')} \;=\; \mathrm{id}_{D'}\,,$$

from which the monad structure on $\mathscr{E} := R \circ L$ is obtained as:

$$\begin{tikzcd}
D \arrow[r, "\eta_D"] \arrow[d, phantom, sloped, "\equiv"] & \mathscr{E}(D) \arrow[d, phantom, sloped, "\equiv"] \\
D \arrow[r, "\eta_D"] & R \circ L(D)
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
\mathscr{E}\big(\mathscr{E}(D)\big) \arrow[r, "\mu_D"] \arrow[d, phantom, sloped, "\equiv"] & \mathscr{E}(D) \arrow[d, phantom, sloped, "\equiv"] \\
R \circ \underbrace{L \circ R} \circ L(D) \arrow[r, "R\left(\varepsilon_{L(D)}\right)"] & R \circ L\,.
\end{tikzcd}$$

Typing of effects via Strong monads. Beware that in describing monad structure this way we are so far only looking externally at the category of types. In contrast, when encoding monadic side effects as a computational structure inside the programming language, then the above bind-operation will be typed not externally as

$$\mathrm{Hom}_{\mathrm{Type}}\big(D_1, \mathscr{E}(D_2)\big) \to \mathrm{Hom}_{\mathrm{Type}}\big(\mathscr{E}(D_1), \mathscr{E}(D_2)\big)$$

but internally as

$$\begin{aligned}
\mathrm{bind}^{\mathscr{E}}_{D_1,D_2} \;:\;\; & \big(D_1 \to \mathscr{E}(D_2)\big) \to \big(\mathscr{E}(D_1) \to \mathscr{E}(D_2)\big) \\
\simeq\;\; & \mathscr{E}(D_1) \times \big(D_1 \to \mathscr{E}(D_2)\big) \to \mathscr{E}(D_2) \\
\simeq\;\; & \mathscr{E}(D_1) \to \Big(\big(D_1 \to \mathscr{E}(D_2)\big) \to \mathscr{E}(D_2)\Big)\,.
\end{aligned}$$

where $(-) \to (-)$ denotes the formation of function types interpreted as the internal hom objects in the closed category of types.

With the above monad axioms phrased internally this way they are actually a little stronger, whence one speaks of *strong monads* (review in [16]), namely enriched monads with respect to the self-enrichment of the closed monoidal category of types. When the category of types is closed symmetric monoidal, which is the case we consider throughout, then strong monads are equivalently those which are suitably symmetric *lax* monoidal [12, Thm. 2.3]. For monads on completely classical types (in the sense of forming the category of sets) this is actually automatic (e.g. [16, Exp. 3.7]), while the monads on linear types which we consider (see appendix A.3) are actually strong symmetric monoidal.

Examples of effect monads. A couple of key examples of effect monads on type systems are considered in appendix A.3 below.

One example not central to our discussion here but illustrative of the general notion of side effects is the throwing of **exceptions**: Assuming that the category Type has coproducts and with Msg : Type some type of error messages, the exception monad is

$$\begin{aligned}
\mathrm{Exc}_{\mathrm{Msg}} \;:\; \mathrm{Type} & \longrightarrow \mathrm{Type} \\
D & \mapsto D \textstyle\coprod \mathrm{Msg}
\end{aligned}$$

whose monad unit is the coprojection of the coproduct and whose monad multiplication is given by the co-diagonal on Msg:

An $\mathrm{Exc}_{\mathrm{Msg}}$-effectful program with nominal output type D_2 is a morphism $D_1 \longrightarrow D_2 \coprod \mathrm{Msg}$ which *may* return output of type D_2 but might instead produce an (error-message) term of type Msg, in which case all subsequently $\mathrm{Exc}_{\mathrm{Msg}}$-bound programs will not execute but just hand this error message along. (Hence for $\mathrm{Msg} \equiv *$ the singleton type, this is also known as the *maybe monad*.)

In this example it is clear that one will wish for programs which can *handle* the exception, and hence in general programs which can handle a given type of side-effect.

Effect handling and modal types. Given a type of computational side effect $\mathscr{E}$ as above, a program of nominal input type D_1 which can *handle* the effect will have actual input type $\mathscr{E}(D_1)$, and handle the effect-part of $\mathscr{E}(D)$ in a way compatible with the incremental binding of effects:

$$\begin{tikzcd}
D_1 \arrow[rr, "\mathrm{prog}_{12}", "\text{in-effectful program}"'] && D_2 \arrow[d, mapsto, bend left, "\substack{\text{incorporate handling}\\ \text{of } \mathscr{E}(-)\text{-effects}}"] \\
\mathscr{E}(D_1) \arrow[rr, "\mathrm{hndl}^{\mathscr{E}}_{D_2}\mathrm{prog}_{12}", "\substack{\text{in-effectful program}\\ \text{handling effects of type } \mathscr{E}(-)}"'] && D_2
\end{tikzcd}$$

$$\begin{tikzcd}
D_1 \arrow[r, "\mathrm{return}^{\mathscr{E}}_{D_1}", "\substack{\text{produce}\\ \text{trivial effect}}"'] \arrow[rr, bend right=40, "\substack{\mathrm{prog}_{12}\\ \text{no effect}}"'] & \mathscr{E}(D_1) \arrow[r, "\mathrm{hndl}^{\mathscr{E}}_{D_2}\mathrm{prog}_{12}", "\substack{\text{handle effects}\\ \text{running program}}"'] & D_2 & \text{consistency conditions} \\
\mathscr{E}(D_0) \arrow[r, "\mathrm{bind}^{\mathscr{E}}\mathrm{prog}_{01}", "\substack{\text{carry effects}\\ \text{along}}"'] \arrow[rr, bend right=40, "\mathrm{hndl}^{\mathscr{E}}_{D_2}\big(D_0 \xrightarrow{\mathrm{prog}_{01}} \mathscr{E}(D_1) \xrightarrow{\mathrm{hndl}^{\mathscr{E}}_{D_2}\mathrm{prog}_{12}} D_2\big)"', "\substack{\text{handle effects...} \quad \text{consecutively}}"'] & \mathscr{E}(D_1) \arrow[r, "\mathrm{hndl}^{\mathscr{E}}_{D_2}\mathrm{prog}_{12}", "\substack{\text{handle}\\ \text{cumulative effects}}"'] & D_2
\end{tikzcd}$$

Such $\mathscr{E}$-effect handling structure on a type D is equivalent to $\mathscr{E}$-**modale**-structure on D (also known as an *$\mathscr{E}$-module* or *$\mathscr{E}$-algebra* structure), namely a morphism

$$\overset{\text{monad action}}{\mathscr{E}(D) \xrightarrow{\rho \,\equiv\, \mathrm{hndl}^{\mathscr{E}}_{D}\mathrm{id}_D} D}$$

satisfying the axioms of a monoid action, in that it makes the following squares commute:

$$\begin{tikzcd}
& \text{unitality} & \\
D \arrow[d, "\eta_D"'] \arrow[rr, bend left, "\mathrm{id}"] & \mathrm{utl}_{\mathscr{E}}(\rho) & D \\
\mathscr{E}(D) \arrow[rr, "\rho"'] && D
\end{tikzcd}
\qquad
\begin{tikzcd}
& \text{action property} & \\
\mathscr{E}\big(\mathscr{E}(D)\big) \arrow[rr, "\mathscr{E}(\rho)"] \arrow[d, "\mu_D"] & \mathrm{act}_{\mathscr{E}}(\rho) & \mathscr{E}(D) \arrow[d, "\rho"] \\
\mathscr{E}(D) \arrow[rr, "\rho"'] && D
\end{tikzcd}$$

Categories of effect-handling types A *homomorphism* $(D_1, \rho_1) \to (D_2, \rho_2)$ of $\mathscr{E}$-effect handlers, hence of $\mathscr{E}$-modales, is a map of the underlying data types $f : D_1 \longrightarrow D_2$ which respects the $\mathscr{E}$-action in that the following diagram commutes

$$\begin{tikzcd}
\mathscr{E}(D_1) \arrow[r, "\mathscr{E}(f)"] \arrow[d, "\rho_1"] & \mathscr{E}(D_2) \arrow[d, "\rho_2"] \\
D_1 \arrow[r, "f"] & D_2
\end{tikzcd}$$

This makes a **category of $\mathscr{E}$-modales** (traditionally known as the *Eilenberg-Moore category* of $\mathscr{E}$ and) denoted $\mathrm{Type}^{\mathscr{E}}$.

For example, for any $B : \mathrm{Type}$, the type $\mathscr{E}(B)$ carries $\mathscr{E}$-modale structure, with $\rho \equiv \mu_B$. These are called the *free* $\mathscr{E}$-modales and the full sub-category they form is traditionally denoted $\mathrm{Type}_{\mathscr{E}}$:

$$\begin{tikzcd}
 & \textbf{free construction} & \substack{\textbf{free } \mathscr{E}\textbf{-modales in } \mathrm{Type} \\ \text{(“Kleisli category”)}} & \textbf{total comparison functor} & \substack{\mathscr{E}\textbf{-modales in } \mathrm{Type} \\ \text{(“Eilenberg-Moore category”)}} \\
\mathrm{Type} \arrow[rr, two heads, "F_{\mathscr{E}}"] \arrow[rrrr, bend right=15, "F^{\mathscr{E}}"] & & \mathrm{Type}_{\mathscr{E}} \arrow[rr, hook, "K_{U^{\mathscr{E}}F^{\mathscr{E}}}"] & & \mathrm{Type}^{\mathscr{E}} \\
\big\{ B : \mathrm{Type} \big\} & & \Big\{ \mathscr{E}(B),\, \rho_B :\equiv \mu_B : \mathscr{E}\big(\mathscr{E}(B)\big) \to \mathscr{E}(B) \Big\} & & \Big\{ D : \mathrm{Type},\, \rho : \mathscr{E}(D) \to D \,\Big|\, \mathrm{untl}_{\mathscr{E}}(\rho), \mathrm{act}_{\mathscr{E}}(\rho) \Big\}
\end{tikzcd}$$

This free construction is readily checked to be left adjoint to evident forgetful functors

$$\begin{tikzcd}
\mathrm{Type}_{\mathscr{E}} \arrow[r, hook, "K_{U^{\mathscr{E}}F^{\mathscr{E}}}"] \arrow[rr, bend left=20, "U_{\mathscr{E}}"] & \mathrm{Type}^{\mathscr{E}} \arrow[r, "U^{\mathscr{E}}", "\textbf{forgetful functor}"'] & \mathrm{Type} \\
 & \big(D, \rho : \mathscr{E}(D) \to D\big) \arrow[r, mapsto] & D
\end{tikzcd}$$

and evidently both adjunctions $F_{\mathscr{E}} \dashv U_{\mathscr{E}}$ and $F^{\mathscr{E}} \dashv U^{\mathscr{E}}$ re-induce (12) the original monad. In fact, *every* adjunction which induces $\mathscr{E}$ is “in between” these two adjunctions, in that it fits into a commuting diagram of the following form (e.g. [14, §VI.3]):

$$\begin{tikzcd}
 & & \mathrm{Type}_{\mathscr{E}} \arrow[dll, shift left, "U_{\mathscr{E}}"] \arrow[d, hook, "K_{UF}", "\substack{\textbf{initial} \\ \textbf{comparison} \\ \textbf{functor}}"'] & \substack{\textbf{free } \mathscr{E}\textbf{-modales in } \mathrm{Type} \\ \text{(“Kleisli category”)}} \\
\textbf{induced monad} \;\; \mathscr{E} \arrow[r, phantom] & \mathrm{Type} \arrow[loop left, "\mathscr{E}"] \arrow[ur, two heads, shift left, "F_{\mathscr{E}}", "B \mapsto \left(\mathscr{E}(B), \rho \equiv \mu_B\right)"'] \arrow[ur, phantom, "\perp"] \arrow[r, shift left, "F"] \arrow[r, phantom, "\perp"] \arrow[dr, shift left, "F^{\mathscr{E}}"] \arrow[dr, phantom, "\perp"] & \mathrm{Type}' \arrow[l, shift left, "U"] \arrow[d, "K^{UF}", "\substack{\textbf{terminal} \\ \textbf{comparison} \\ \textbf{functor}}"'] & \textbf{any adjunction} \text{ for } \mathscr{E} \\
 & & \mathrm{Type}^{\mathscr{E}} \arrow[ul, shift left, "U^{\mathscr{E}}", "\textbf{monadic adjunction}"'] & \substack{\mathscr{E}\textbf{-modales in } \mathrm{Type} \\ \text{(“EM-category”)}}
\end{tikzcd} \tag{14}$$

Computational contexts and co-monads on the type system. By formal duality (reversal of all arrows in the above diagrams), we have a dual notion of co-monads on the type system, which some authors refer to as “computational co-effect” but which may naturally be understood as expressing *computational contexts* [36][22] .

The idea now is that a program which *nominally* reads in data of some type D while however depending on some context must *de facto* read in data of some adjusted type $\mathscr{C}(D)$ which is such that the context-part of the adjusted data is being transferred to followup programs:

$$\begin{tikzcd}
\mathscr{C}(D_1) \arrow[r, "\overset{\textbf{first program}}{\mathrm{prog}_{12}}", "\substack{\textbf{output data}\\ \textbf{of type } D_2\\ \textbf{obtained in context of type } \mathscr{C}(-)}"'] & D_2 & \mathscr{C}(D_2) \arrow[r, "\overset{\textbf{second program}}{\mathrm{prog}_{23}}", "\substack{\textbf{input data}\\ \textbf{of nominal type } D_2\\ \textbf{having context of type } \mathscr{C}(-)}"'] & D_3 & {} \arrow[d, mapsto, bend left, "\substack{\textbf{bind previous context}\\ \textbf{into second program}}"] \\
\mathscr{C}(D_1) \arrow[r, "\mathrm{prog}_{12}"] & \mathscr{C}(D_2) & \mathscr{C}(D_2) \arrow[r, "\mathrm{bind}^{\mathscr{E}}\,\mathrm{prog}_{23}", "\substack{\textbf{carry any previous}\\ \mathscr{C}(-)\textbf{-context along}}"'] & D_3 & {} \arrow[d, mapsto, bend left, "\textbf{compose}"] \\
\mathscr{C}(D_1) \arrow[rrr, "\overset{\mathscr{C}\textbf{-composite program}}{\mathrm{bind}^{\mathscr{C}}\,\mathrm{prog}_{23} \circ \mathrm{prog}_{12}}", "\textbf{subject to cumulative } \mathscr{C}(-)\textbf{-contexts}"'] & & & D_3 & {}
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathscr{C}(D) \arrow[r, "\mathrm{extract}^{\mathscr{C}}_{D}", "\textbf{extract plain data from } \mathscr{C}(-)\textbf{-effect}"'] & D \\
\mathscr{E}(D) \arrow[r, "\mathrm{bind}^{\mathscr{C}}\,\mathrm{extract}^{\mathscr{C}}_{D}", "= \mathrm{id}_{\mathscr{E}(D)}"'] & \mathscr{E}(D)
\end{tikzcd}$$

By formal duality, all the above discussion for monadic effects and their modal types gives rise to analogous phenomena of comonadic contexts and their (co)modal types. In particular, comonad are induced on the other sides of an adjunction (12):

$$\begin{tikzcd}
\mathrm{Type}' \arrow[r, shift right=2, "\underset{\textbf{right adjoint}}{R}"'] \arrow[r, phantom, "\perp"] & \mathrm{Type} \arrow[l, shift right=2, "\overset{\textbf{right adjoint}}{L}"'] \arrow[loop right, "L \circ R =: \mathscr{C} \quad \textbf{induced co-monad}"]
\end{tikzcd} \tag{15}$$

A.3 Quantum Reader/Writer (co)Monads from Dependent Linear Types

The notions of (co)reader and (co)writer monads are well-known, in fact they are the first examples of (co)monads typically considered in computer science, as acting on cartesian monoidal categories of classical data types. We recall their definitions in streamlined form and then highlight that:

1. the (co)reader monads arise from the base change adjunctions in classical dependent type theory;
2. in their dependent-linear version over a finite classical base type B, the B-reader and B-coreader as well as the $\mathbb{1}^B$-writer and $\mathbb{1}^B$-cowriter both fuse to Frobenius monads, and as such are equivalent to each other as well as to Coecke's et al.'s "classical structures" Frobenius monad.

This is all elementary and straightforward, but the resulting picture is somewhat profound (p. 8) and seems not to have previously been appreciated much.

Some remarks and notation before we start:

- Notice that every object $B : \mathrm{ClType}$ of a cartesian monoidal category carries a unique comonoid structure whose coproduct is the cartesian diagonal map:

$$S : \mathrm{ClType} \quad \vdash \quad \big(S,\, \varepsilon : B \to *,\, \mathrm{diag} : B \to B \times V\big) \;:\; \mathrm{CoMon}\big(\mathrm{ClType}, *, \times\big) \tag{16}$$

- For $B : \mathrm{ClType}$ we may denote the objects and morphisms of the slice category ClType_B by the sequent calculus notation whose categorical semantics they are, for instance

$$b : B \;\;\vdash\;\; X_b \xrightarrow{f_b} Y_b \qquad \text{stands for} \qquad \big(X_b\big)_{b:B} \xrightarrow{(f_b)_{b:B}} \big(Y_b\big)_{b:B}\,.$$

  In particular, a context-extended B-independent type appears as:

$$b : B \;\;\vdash\; X : \mathrm{Type} \qquad\quad \text{stands for} \qquad\quad B \times X \;=\; \big(X\big)_{b:B}$$

- For review of Frobenius algebras in the present context see [10, §5]. For discussion of Frobenius monads see [35].

The classical reader monad. Classically, for B : ClType, the *B-reader monad* is the cartesian internal hom-functor $\mathrm{Map}(S,-)$ with monad structure contravariantly induced from the canonical co-monoid structure on B (16):

$$\begin{array}{rcl} B\text{-Reader} : \mathrm{ClType} & \longrightarrow & \mathrm{ClType} \\ S & \mapsto & \mathrm{Map}(B,S) \end{array} \tag{17}$$

$$\begin{array}{ccc} \mathrm{id}(S) & \xrightarrow{\;\varepsilon\;} & B\text{-Reader}(S) \\ ||| & & ||| \\ S & \longrightarrow & \mathrm{Map}(B,S) \\ s & \mapsto & \big(b \mapsto s\big) \end{array} \qquad \begin{array}{ccc} B\text{-Reader}\big(B\text{-Reader}(S)\big) & \xrightarrow{\;\mu\;} & B\text{-Reader}(S) \\ ||| & & ||| \\ \mathrm{Map}\big(B,\mathrm{Map}(B,S)\big) & \longrightarrow & \mathrm{Map}(B,S) \\ \big(b' \mapsto (b'' \mapsto s_{b',b''})\big) & \mapsto & \big(b \mapsto s_{b,b}\big) \end{array} \tag{18}$$

Proposition A.1. *The B-reader monad* (17) *is equivalently the monad* $\bigcirc_B$ *induced from the right base change adjunction*

$$\begin{tikzcd} \mathrm{ClType}_B \arrow[r, shift right=1.5ex, "\prod_B"'] \arrow[r, phantom, "\perp"] & \mathrm{ClType} \arrow[l, shift right=1.5ex, "B\times(-)"'] \arrow[loop right, "\bigcirc_B"] \end{tikzcd} \tag{19}$$

via the natural isomorphism

$$\begin{array}{rcl} \prod_B S & \xrightarrow{\;\sim\;} & \mathrm{Map}(B,S) \\ (s_b)_{b:B} & \mapsto & (b \mapsto s_b)\,. \end{array} \tag{20}$$

Proof. The $\big(B\times(-) \dashv \prod_B\big)$-adjointness relation

$$\begin{array}{rcl} & \vdash & X \xrightarrow{(f_b)_{b:B}} \prod_B Y_b \\ \hline b : B & \vdash & X \xrightarrow{\;\;f_b\;\;} Y_b \end{array}$$

gives the adjunction (co)unit η (ε) as

$$\begin{array}{rcl} b : B & \vdash & X \xrightarrow{\;\;\mathrm{id}\;\;} X \\ \hline & \vdash & X \xrightarrow{\eta\,\equiv\,(\mathrm{id})_{b:B}} \prod_B X \end{array} \qquad \begin{array}{rcl} & \vdash & \prod_{b:B} X_b \xrightarrow{\mathrm{id}\,\equiv\,(\mathrm{pr}_b)_{b:B}} \prod_{b:B} X_b \\ \hline b : B & \vdash & \prod_{b':B} X_{b'} \xrightarrow{\;\;\varepsilon_b\,\equiv\,\mathrm{pr}_b\;\;} X_b \end{array}$$

and thus under the the monad structure comes out as:

$$\begin{array}{rcl} S & \xrightarrow{\;\eta\;} & \prod_{b:B} S \\ s & \mapsto & (s)_{b:B} \end{array} \qquad \begin{array}{ccc} \prod_{b:B}\Big(\prod_{b':B} S & \xrightarrow{\;\;\varepsilon_b\;\;} & S\Big) \\ \big(b \mapsto (b' \mapsto s_{b,b'})\big) & \mapsto & \big(b \mapsto s_{b,b}\big) \end{array}$$

Under the isomorphism (20) this is accordance with (18). □

The classical coreader comonad. Dually, the classical *B-coreader* is the functor $B \times (-)$ with comonad structure induced from the comonoid structure (16) of B

$$\begin{array}{rcl} B\text{-CoReader} \;:\; \mathrm{ClType} & \longrightarrow & \mathrm{ClType} \\ S & \mapsto & B \times S \end{array} \tag{21}$$

$$\begin{array}{rcl} B\text{-CoReader}(S) & \xrightarrow{\eta} & S \\ \big(b,s\big) & \mapsto & s \end{array} \qquad \begin{array}{rcl} B\text{-CoReader}(S) & \xrightarrow{\nu} & B\text{-CoReader}\big(B\text{-CoReader}(S)\big) \\ \big(b,s\big) & \mapsto & \big(b,(b,s)\big) \end{array} \tag{22}$$

Proposition A.2. *The B-co-reader* (21) *is equivalent to the comonad induced from the left base change adjunction*

$$\begin{tikzcd} \mathrm{ClType}_B \arrow[r, shift left=2, "\coprod_B"] \arrow[r, phantom, "\perp"] & \mathrm{ClType} \arrow[l, shift left=2, "B\times(-)"] \arrow[loop right, "\star_B"] \end{tikzcd}$$

via the natural isomorphism

$$\begin{array}{rcl} \coprod_B S & \xrightarrow{\sim} & B \times S \\ (b,s) & \mapsto & (b,s)\,. \end{array}$$

Proof. The adjointness relation

$$\frac{\vdash \;\; \coprod_{b:B} X_b \xrightarrow{(f_b)_{b:B}} Y}{b:B \;\vdash \;\; X_b \xrightarrow{f_b} Y}$$

gives the adjunction (co)unit η (ε) as

$$\frac{b:B \;\vdash\;\; X \xrightarrow{\mathrm{id}} X}{\vdash \;\; \coprod_B X \xrightarrow{\varepsilon \equiv (\mathrm{id})_{b:B}} X} \qquad \frac{\vdash \;\; \coprod_{b:B} X_b \xrightarrow{\mathrm{id} \equiv (\mathrm{cpr}_b)_{b:B}} \coprod_{b:B} X_b}{b:B \;\vdash \;\; X_b \xrightarrow{\eta_b \equiv \mathrm{cpr}_b} \coprod_{b:B} X_b}$$

from which the comonad structure is

$$\begin{array}{rcl} \coprod_{b:B} S & \xrightarrow{\varepsilon} & S \\ (b,s) & \mapsto & s \end{array} \qquad \begin{array}{rcl} \coprod_{b:B} \Big(S & \xrightarrow{\eta_b} & \coprod_{b':B} S \Big) \\ (b,s) & \mapsto & \big(b,(b,s)\big) \end{array}$$

in accordance with (22). □

The classical writer monad. Now consider a monoid structure on a classical type

$$\big(A, \mathrm{e}, \cdot\big) \;:\; \mathrm{Mon}\big(\mathrm{ClType}(*,\times)\big)\,, \tag{23}$$

which, in contrast to the cartesian co-monoid structure (16) requires making a choice. (In computing practice the canonical choice for A is the *free monoid* on some alphabet, which will make the following writer monad have the side-effect of concatenating characters in this alphabet, hence of "writing strings", whence its name).

Then the classical *A-writer monad* is the functor $A \times (-)$ with monad structure induced from this monoid structure (23):

$$\begin{array}{rccc} A\text{-Writer} : & \mathrm{ClType} & \longrightarrow & \mathrm{ClType} \\ & S & \mapsto & A \times S \end{array} \tag{24}$$

$$\begin{tikzcd} \mathrm{id}(S) \arrow[r, "\varepsilon"] \arrow[d, phantom, sloped, "\equiv"] & A\text{-Writer}(S) \arrow[d, phantom, sloped, "\equiv"] & A\text{-Writer}\big(A\text{-Writer}(S)\big) \arrow[r, "\mu"] \arrow[d, phantom, sloped, "\equiv"] & A\text{-Writer}(S) \arrow[d, phantom, sloped, "\equiv"] \\ S \arrow[r] & A \times S & A \times A \times S \arrow[r] & A \times S \\ s \arrow[r, phantom, "\mapsto"] & (\mathrm{e}, s) & \big(a, a', s\big) \arrow[r, phantom, "\mapsto"] & \big(a \cdot a', s\big) \end{tikzcd} \tag{25}$$

The linear writer monad. These constructions of the classical (co-)monads above use only the closed monoidal structure $\big(\mathrm{ClType}, *, \times\big)$ and not that this is cartesian. Therefore analogous constructions are obtained on the non-cartesian symmetric monoidal category of linear types $\big(\mathrm{LinType}, \mathbb{1}, \otimes\big)$.

Notably, for

$$\big(A, \mathrm{e}, \cdot\big) \; : \; \mathrm{Mon}\big(\mathrm{LinType}, \mathbb{1}, \otimes\big) \tag{26}$$

a monoid structure among linear types, hence an *algebra structure*, the corresponding linear writer monad is

$$\begin{array}{rccc} A\text{-Writer} : & \mathrm{LinType} & \longrightarrow & \mathrm{LinType} \\ & \mathcal{H} & \mapsto & A \otimes \mathcal{H} \end{array} \tag{27}$$

$$\begin{tikzcd} \mathrm{id}(\mathcal{H}) \arrow[r, "\varepsilon"] \arrow[d, phantom, sloped, "\equiv"] & A\text{-Writer}(\mathcal{H}) \arrow[d, phantom, sloped, "\equiv"] & A\text{-Writer}\big(A\text{-Writer}(\mathcal{H})\big) \arrow[r, "\mu"] \arrow[d, phantom, sloped, "\equiv"] & A\text{-Writer}(\mathcal{H}) \arrow[d, phantom, sloped, "\equiv"] \\ \mathcal{H} \arrow[r] & A \otimes \mathcal{H} & A \otimes A \otimes \mathcal{H} \arrow[r] & A \otimes \mathcal{H} \\ |\psi\rangle \arrow[r, phantom, "\mapsto"] & \mathrm{e} \otimes |\psi\rangle & a \otimes a' \otimes s \arrow[r, phantom, "\mapsto"] & (a \cdot a') \otimes |\psi\rangle \end{tikzcd} \tag{28}$$

The example of interest here is the linear writer monad induced from finite direct sums of the unit algebra:

$$B : \mathrm{FinClType} \quad \vdash \quad \mathbb{1}^B \equiv (p_B)_! \mathbb{1}_B \simeq \bigoplus_{b:B} \mathbb{1} \simeq \left\{ \sum_{b:B} c_b |b\rangle \,\middle|\, c_{b:B} \in \mathbb{1} \right\} \; : \; \mathrm{Mon}\big(\mathrm{LinType}, \mathbb{1}^B, \otimes\big) \tag{29}$$

$$\begin{tikzcd} \mathbb{1} \arrow[r, "\mathrm{e}"] & \mathbb{1}^B & \mathbb{1}^B \otimes \mathbb{1}^B \arrow[r, "(-)\cdot(-)"] & \mathbb{1} \\ 1 \arrow[r, phantom, "\mapsto"] & \sum_{b:B} |b\rangle & |b\rangle \otimes |b'\rangle \arrow[r, phantom, "\mapsto"] & \delta_{b,b'} |b'\rangle \end{tikzcd} \tag{30}$$

Here, $p_B : B \to *$ is the terminal morphism of the classical type B, and $(p_B)_!$ is as in (4).

Proposition A.3. *The linear writer monad* (27) *for the direct B-sum algebra* (29) *is equivalent to the linear B-reader monad, and hence also to the linear indefiniteness modality $\bigcirc_B$:*

$$\bigcirc_B \;\simeq\; B\text{-Reader} \;\simeq\; \mathbb{1}^B\text{-Writer} \qquad : \; \mathrm{LinType} \to \mathrm{LinType}$$

Proof. Consider the following natural isomorphisms between the underlying functors:

$$\begin{array}{ccccc} \bigcirc_B \mathcal{H} & & B\text{-Reader}(\mathcal{H}) & & \mathbb{1}^B\text{-Writer}(\mathcal{H}) \\ \rotatebox{90}{$\equiv$} & & \rotatebox{90}{$\equiv$} & & \rotatebox{90}{$\equiv$} \\ \prod\limits_{b:B} \mathcal{H} & \simeq & \mathrm{Map}(B, \mathcal{H}) & \simeq & \mathbb{1}^B \otimes \mathcal{H} \\ \big(|\psi_b\rangle\big)_{b:B} & \mapsto & \big(b \mapsto \psi_b\big) & \mapsto & \sum\limits_{b:B} |b\rangle \otimes |\psi_b\rangle \end{array}$$

Using the above formulas, we check, straightforwardly, that these respect the monad structure:

$$\begin{tikzcd}
\bigcirc_B \bigcirc_B \mathcal{H} \arrow[r, "\mu"] & \bigcirc_B \mathcal{H} & & \mathbb{1}^B\text{-Writer}\big(\mathbb{1}^B\text{-Writer}(\mathcal{H})\big) \arrow[r, "\mu"] & \mathbb{1}^B\text{-Writer}(\mathcal{H})\big) \\
\rotatebox{90}{$\equiv$} & \rotatebox{90}{$\equiv$} & & \rotatebox{90}{$\equiv$} & \rotatebox{90}{$\equiv$} \\
\prod\limits_{b':B}\prod\limits_{b'':B}\mathcal{H} \arrow[r] & \prod\limits_{b:B}\mathcal{H} & \leftrightarrow & \mathbb{1}^B \otimes \mathbb{1}^B \otimes \mathcal{H} & \mathbb{1}^B \otimes \mathcal{H} \\
\big(\lvert\psi_{b',b''}\rangle\big)_{b',b'':B} \arrow[r, mapsto] & \big(\lvert\psi_{b,b}\rangle\big)_{b:B} & & \sum\limits_{b',b'':B}\lvert b'\rangle\otimes\lvert b''\rangle\otimes\lvert\psi_{b',b''}\rangle \arrow[r, mapsto] & \sum\limits_{b',b'':B}\lvert b\rangle\otimes\lvert\psi_{b,b}\rangle
\end{tikzcd}$$

$$\begin{tikzcd}
\mathrm{id}(\mathcal{H}) \arrow[r, "\varepsilon"] & \bigcirc_B \mathcal{H} & & \mathrm{id}(\mathcal{H}) \arrow[r, "\varepsilon"] & \mathbb{1}^B\text{-Writer}(\mathcal{H}) \\
\rotatebox{90}{$\equiv$} & \rotatebox{90}{$\equiv$} & & \rotatebox{90}{$\equiv$} & \rotatebox{90}{$\equiv$} \\
\mathcal{H} \arrow[r, mapsto] & \prod_B \mathcal{H} & \leftrightarrow & \mathcal{H} \arrow[r] & \mathbb{1}^B \otimes \mathcal{H} \\
\lvert\psi\rangle \arrow[r, mapsto] & \big(\lvert\psi\rangle\big)_{b:B} & & \lvert\psi\rangle \arrow[r, mapsto] & \mathrm{e}\otimes\lvert\psi\rangle = \sum_{b:B}\lvert b\rangle\otimes\lvert\psi\rangle
\end{tikzcd}$$

□

Frobenius structure on the quantum reader. The ambidexterity clause (4) implies that over finite classical types the linear version of the reader monad (19) carries the structure of a Frobenius monad, by [35, Prop. 1.5]:

$$\begin{tikzcd}
B : \mathrm{FinClType} \quad \vdash & \mathrm{LinType}_B \arrow[r, shift left=3ex, "\oplus_B"] \arrow[r, shift right=3ex, "\oplus_B"'] \arrow[r, phantom, shift left=1.5ex, "\perp"] \arrow[r, phantom, shift right=1.5ex, "\perp"] & \mathrm{LinType} \arrow[l, "(p_B)^*" description] \arrow[loop right, "\bigcirc_B", "\star_B"']
\end{tikzcd} \tag{31}$$

By dualizing the proof of Prop. A.3 we find that this Frobenius monad corresponds to the Frobenius algebra structure on (29) whose co-algebra structure is

$$\begin{tikzcd}
\mathbb{1}^B \arrow[r, "\mathrm{e}"] & \mathbb{1} & & \mathbb{1} \arrow[r, "\nabla(-)"] & \mathbb{1}^B \otimes \mathbb{1}^B \\
\sum_{b:B} c_b \lvert b\rangle \arrow[r, mapsto] & \sum_{b:B} c_b & & \lvert b\rangle \arrow[r, mapsto] & \lvert b\rangle \otimes \lvert b\rangle
\end{tikzcd} \tag{32}$$

Proposition A.4. *For* B : FinClType *the comonad* $\star_B$ (31) *is equivalent to the* $\mathbb{1}^B$ *co-writer monad induced from the co-algebra structure* (32)*:*

$$\star_B \;\simeq\; \mathbb{1}^B\text{-CoWriter}\,.$$

Proof.

$$\begin{tikzcd}
\star_B(\mathcal{H}) \arrow[r] & \mathrm{id}(\mathcal{H}) & & \mathbb{1}^B\text{-CoReader} \arrow[r] & \mathrm{id}(\mathcal{H}) \\
\rotatebox{90}{$\equiv$} & \rotatebox{90}{$\equiv$} & & \rotatebox{90}{$\equiv$} & \rotatebox{90}{$\equiv$} \\
\coprod_B \mathcal{H} \arrow[r] & \mathcal{H} & \leftrightarrow & \mathbb{1}^B \otimes \mathcal{H} \arrow[r] & \mathcal{H} \\
\big(b, \lvert\psi\rangle\big) \arrow[r, mapsto] & \lvert\psi\rangle & & & \\
\text{hence } \sum_{b:B}\big(b, \lvert\psi_b\rangle\big) \arrow[r, mapsto] & \sum_{b:B}\lvert\psi_b\rangle & & \sum_{b:B}\lvert b\rangle\otimes\lvert\psi_b\rangle \arrow[r, mapsto] & \sum_{b:B}\lvert\psi_b\rangle
\end{tikzcd}$$

$$\begin{tikzcd}
\star_B \mathcal{H} \arrow[r] \arrow[d, phantom, sloped, "\equiv"] & \star_B \star_B \mathcal{H} \arrow[d, phantom, sloped, "\equiv"] & & \mathbb{1}^B\text{-CoWriter}(\mathcal{H}) \arrow[r] \arrow[d, phantom, sloped, "\equiv"] & \mathbb{1}^B\text{-CoWriter}\big(\mathbb{1}^B\text{-CoWriter}(\mathcal{H})\big) \arrow[d, phantom, sloped, "\equiv"] \\
\coprod_B \mathcal{H} \arrow[r] & \coprod_B \coprod_B \mathcal{H} & \leftrightarrow & \mathbb{1}^B \otimes \mathcal{H} \arrow[r] & \mathbb{1}^B \otimes \mathbb{1}^B \otimes \mathcal{H} \\
\big(b, \vert\psi_b\rangle\big) \arrow[r, mapsto] & \Big(b, \big(b, \vert\psi_b\rangle\big)\Big) & & \sum_{b:B} \vert b\rangle \otimes \vert\psi_b\rangle \arrow[r, mapsto] & \sum_{b:B} \vert b\rangle \otimes \vert b\rangle \otimes \vert\psi_b\rangle
\end{tikzcd}$$

□

In conclusion: By the ambidexterity clause (4) of the Motivic Yoga verified by LHoTT, the quantum measurement monad $\bigcirc_B$ (p. 8) is Frobenius and as such equivalent to the $\mathbb{1}^B$-Writer monad, which is just Coecke et al's "classical structures"-monad [6][5].

A.4 The quantum measurement modales

The classical modales (p. 18) over the classical reader monad (17) may be somewhat exotic (cf. MO:868317) but their linear (quantum) analog is nicely behaved (as claimed on p. 7):

Proposition A.5. *The Eilenberg-Moore category of modal types for the quantum reader monad* (31)

$$B : \text{FinClType} \quad \vdash \quad \bigcirc_B : \text{LinType} \to \text{LinType}$$

is equivalently that of B-dependent linear types. Moreover on the free B-modales this equivalence restricts to the comparison functor indicated on p. 8:

$$\begin{tikzcd}
\substack{\textbf{free } \bigcirc_B\textbf{-modal} \\ \textbf{linear types}} & \substack{\bigcirc_B\textbf{-modal} \\ \textbf{linear types}} & \substack{B\textbf{-dependent} \\ \textbf{linear types}} \\
\text{LinType}_{\bigcirc_B} \arrow[r, hook] \arrow[rr, bend right, hook] & \text{LinType}^{\bigcirc_B} \arrow[r, "\sim"] & \text{LinType}_B \\
\bigcirc_B \mathcal{H} \arrow[rr, mapsto] & & \mathbb{1}_B \otimes \mathcal{H}
\end{tikzcd}$$

This is straightforward to verify but its implications may previously not have found due attention. We have spelled out detailed proofs at `ncatlab.org/nlab/show/quantum+reader+monad` and will elsewhere expand on this and other aspects touched on here.