A Linear Dependent Type Theory with Identity Types as a Quantum Verification Language

Mitchell Riley
mitchell.v.riley@nyu.edu

Mathematics, Division of Science
New York University Abu Dhabi

The combination of linear and dependent types makes for a pleasant programming experience when writing quantum programs: linearity enforces the no-cloning principle throughout the code, and dependency allows classically parameterised families of circuits to be described and composed in a type-safe manner.

Proto-Quipper-D is a language with both of these features, but lacks one further selling point of dependent types: the ability to formally verify properties of the program by writing proofs in the same language, as can be done in (non-linear) languages such as Coq and Agda.

We suggest that a bunched dependent type theory (previously described in the thesis of the author) could fill this gap. This is an extension of ordinary non-linear Martin-Löf dependent type theory with linear type formers, capable of describing quantum circuits using its linear type formers and writing formal proofs concerning their properties using the non-linear type formers.

After summarising the rules of the bunched type theory, we sketch a translation of the non-dependent Proto-Quipper-M into the bunched theory and show that it can be used to verify a simple quantum circuit. We expect the dependent Proto-Quipper-D to support a similar translation, which would give us the power of dependent types both when constructing circuits and when verifying them.

1 Introduction

The Quipper family of circuit description languages originates as an embedded language in Haskell [3, 4]. Haskell lacks linear types, and so there are no compile-time guarantees that the code you have written describes a circuit that obeys the no-cloning principle: nothing prevents you from using a the same qubit or wire in multiple places. To rectify this, the Proto-Quipper [17, 18] line of work describes standalone type theories based on linear types that enforce these restrictions.

Proto-Quipper has since been extended to include linear dependent types in Proto-Quipper-D [1, 2], increasing the type-safety of the language. For example, [1, §3.3] defines families of circuits, one for each of number of qubits n. The type system can then *guarantee* that, for any input $n : \mathbb{N}$, the output circuit is indeed a circuit on n qubits.

Proto-Quipper-D lacks one type former that many non-linear dependent type theories include: Id-types, i.e. identity types/equality types. These internalise the notion of equality as a type, and allow us to reason about the equality of terms without leaving the theory we are working in. This has been put

The author acknowledges support by *Tamkeen* under *NYUAD Research Institute grant* CG008.

Submitted to:
QPL 2023

to spectacular use in the formal verification of program properties and the formalisation of mathematics, using proof assistants such as Coq, Agda and Lean.

Our goal is to use Id-types to formally verify quantum circuits described using a language in the Quipper family. This means using a type theory that integrates linear types and Id types. Unfortunately Id-types are not something that can simply be thrown into Proto-Quipper-D: the semantic interpretation of Id-types intrinsically requires a diagonal morphism, both in the interpretation of the type itself and of the reflexivity term. For 'parameter types' P, it is likely some version of the standard rules for Id-types are be sound, because such types can be equipped with a diagonal morphism $P \to P \otimes P$. But this does not help with our original problem: verifying properties of a circuit will certainly involve equalities of the linear terms defining the circuit.

One type theory that can combine linearity and ordinary Id-types is the *bunched dependent type theory* of [14]. This is an extension of ordinary Martin-Löf [5] type theory to include linear type formers. Importantly, the rules for the ordinary non-linear type formers are imported unchanged, so in particular, Id-types work exactly as we are used to.

After summarising the rules of the bunched type theory (Section 2), we describe a translation from a fragment of Proto-Quipper-M into it (Section 3). The specific fragment we intend to translate is given in Figure 1. We demonstrate the possibility of using this for the verification of quantum programs by translating a (very) simple circuit and verifying one of its properties (Section 4).

Remark 1.1. It is not surprising that Proto-Quipper and this bunched theory are related: their intended semantics are quite similar. Proto-Quipper-M and Proto-Quipper-D have semantics in set-indexed families of objects of a monoidal category where the category is required to have certain additional structure; more generally, in fibrations of a closed symmetric monoidal category (representing the linear state) over a locally cartesian closed category (representing the non-linear parameters). The bunched type theory has semantics in a similar setting, but is slightly more restricted: each of the fibers of the fibration must have a zero object[1], and the domain of the fibration must itself also be a locally cartesian closed category.

Remark 1.2. The most comparable work on formal verification for a quantum circuit language is QWIRE [12, 10, 13], but there are some important differences. QWIRE is domain specific language embedded in Coq, with quantum 'wire types' and circuits described as elements of an inductive data type. The dependent types (including Id-types) of the host language can then be used to verify properties of these circuits. There is a strict separation between the terms and types of the host language and the constructions in the circuit language, making it more difficult to manipulate parameters and state together within a program, and the fixed set of 'wire types' forecloses the ability to include dependent types or quantum data types in the linear language.

Remark 1.3. There are some natural avenues for further exploration. In a separate article [6], we discuss how aspects of state-preparation and measurement arise naturally out of the interplay between the linear and non-linear type formers of the bunched theory. And in this note, as in Proto-Quipper, we postulate the type Qubit and the gates of interest which remain uninterpreted other than as syntactic building blocks of circuits. One of the benefits of the bunched dependent type theory is its compatibility with homotopy type theory, which is capable of directly describing some of the mathematical structures underlying quantum mechanics [7, 11]. This could lead to direct interpretations of Quipper circuits as operations on these mathematical structures, and so one could, for example, extract the unitary operator corresponding to a circuit.

The $\natural$ modality was originally described in joint work with Eric Finster and Dan Licata [15], and the further extension to linear types in work with Dan Licata [16].

[1] In the notation of [2], we require the functor q to be also left adjoint to the fibration $\sharp$, not just right adjoint.

2 Bunched Dependent Type Theory

This introduction to bunched dependent type theory will necessarily be brief and incomplete, the full details are present in [14]. Our bunched dependent type theory is an extension of Martin-Löf type theory, including ordinary Π-, Σ- and Id-types. This sidesteps the question of what 'linear identity types' ought to be: identity types with their ordinary rules are enough for our purposes.

Like Proto-Quipper-D, the bunched type theory includes *dependent* $\otimes$- and $\multimap$-types, but because the source of our translation will be non-dependent, we only need their non-dependent versions in the bunched theory. The target still contains fully dependent versions of ordinary Π-, Σ- and Id-types, however.

The way the bunched dependent theory extends MLTT is similar to the way the $\alpha\lambda$-calculus extends the ordinary λ-calculus, as part of work on bunched logics [9, 8], hence the name. In bunched implication there are two binary context-forming operations, an ordinary cartesian comma and a linear monoidal product. These notions of product can be combined in arbitrary ways, giving contexts an inhrently tree-like structure.

Palettes. In our theory, we separate the 'bunched' shape of the context from the typing information of the variables in it. A generic context $\Phi \mid \Gamma\ \mathsf{ctx}$ has two pieces, a *palette* Φ describing the tree structure of the context, and then a more typical context Γ of typed variables. Each variable is labelled with a *colour* from the palette, which places the data of that variable at the corresponding node of the tree.

On its own, a palette Φ describes a particular kind of tree, where each node represents either a cartesian product $\times$ or a tensor product $\otimes$. Palettes have no semantic interpretation on their own; they describe a shape that is later filled by the variables of the context.

Any $\times$-node can be given a label. We call these labels *colours*, and write $\Phi \vdash \mathfrak{c}\ \mathsf{colour}$ for the judgement picking out a single label. Each variable in the context will be assigned one of these labels, which corresponds to adding that variable as a child of the corresponding $\times$ node.

A typical palette may look like as follows: $\mathfrak{t} \prec (\mathfrak{a} \otimes \mathfrak{b}, \mathfrak{c} \otimes \mathfrak{d})\ \mathsf{palette}$, so that $\mathfrak{t}$ represents a $\times$-node with two children $\mathfrak{a} \otimes \mathfrak{b}$ and $\mathfrak{c} \otimes \mathfrak{d}$. The former is a $\otimes$-node with two children $\mathfrak{a}$ and $\mathfrak{b}$, and similarly for the latter. Palettes can be nested in arbitrary ways. Starting with the palette $\mathfrak{t} \prec \mathfrak{a} \otimes \mathfrak{b}\ \mathsf{palette}$, we might like for the $\mathfrak{a}$-labelled $\times$-node to itself contain a $\otimes$-node, and this would be written $\mathfrak{t} \prec (\mathfrak{a} \prec \mathfrak{p} \otimes \mathfrak{q}) \otimes \mathfrak{b}\ \mathsf{palette}$.

As well as labels and the binary palette formers, there are two special unary symbols: 1, representing the terminal object, and $\varnothing$, representing the monoidal unit. The formal rules for constructing palettes are given in the Appendix. To cut down on notation, we write $\mathfrak{r}$ as a shorthand for $\mathfrak{r} \prec 1$.

Context Formation. The colour at the top of the tree (leftmost in the palette) is used to annotate variables that are currently accessible, so palettes in the context of a term always have a label at the top level, so are of the form $\mathfrak{t} \prec \Phi \mid \Gamma \vdash a : A$. The rules for constructing contexts are:

$$\frac{\mathfrak{t} \prec \Phi\ \mathsf{palette}}{\mathfrak{t} \prec \Phi \mid \cdot\ \mathsf{ctx}}\ \textsc{ctx-empty} \qquad \frac{\mathfrak{t} \prec \Phi \vdash \mathfrak{c}\ \mathsf{colour} \quad (\mathfrak{t} \prec \Phi)^{\mathfrak{c}} \mid \Gamma^{\mathfrak{c}} \vdash A\ \mathsf{type}}{\mathfrak{t} \prec \Phi \mid \Gamma, x^{\mathfrak{c}} : A\ \mathsf{ctx}}\ \textsc{ctx-ext} \qquad \frac{\mathfrak{r} \mid \underline{\Gamma} \vdash A\ \mathsf{type}}{\mathfrak{t} \prec \Phi \mid \Gamma, \underline{x}^{\mathfrak{r}} : A\ \mathsf{ctx}}\ \textsc{ctx-ext-marked}$$

There are two ways to extend a context with a variable: either labelled with a colour from the palette, or 'marked'. A 'colourful' context extension $\mathfrak{t} \prec \Phi \mid \Gamma, x :^{\mathfrak{c}} A\ \mathsf{ctx}$ denotes an assumption whose linear data *is* accessible, much like the 1 annotations on variables in Proto-Quipper-D [2]. The $\Phi \vdash \mathfrak{c}\ \mathsf{colour}$

label describes how this linear data is combined with the rest of the context: the data is placed in the $\times$-node labelled $\mathfrak{c}$.

The type A assigned to x must be well-formed in the palette $(\mathfrak{t} \prec \Phi)^{\mathfrak{c}}$, which is defined to be the subpalette rooted at the label $\mathfrak{c}$. The context is adjusted appropriately: any variable whose colour label does not lie under $\mathfrak{c}$ becomes marked. As a special case of colourful context extension, we have ordinary context extension using the colour that is at the top of the palette, as in

$$\mathfrak{t} \prec \Phi \mid \Gamma, x :^{\mathfrak{t}} A.$$

A 'marked' context extension $\mathfrak{t} \prec \Phi \mid \Gamma, \underline{x} :^{\mathfrak{c}} A\ \mathsf{ctx}$ denotes an assumption whose linear data is *not* accessible; either because it was never accessible to begin with, or because some other part of the derivation will use that linear data. It corresponds semantically to extending Γ with the underlying indexing set of A, i.e. $\natural A$. The $\mathfrak{c}$ label here does not come from the palette: instead it is bound in this context extension. When a variable becomes marked, its connection to the ambient palette is completely severed.

The type A in such a context extension must be 'dull', i.e. all uses of variables from Γ in A must be marked uses, and nor can it use any colours form Φ: the palette is cleared to $\mathfrak{c}$, using the top colour label that the extension is annotated with.

Variable Usage. There are three ways a variable can be used.

VAR
$$\frac{}{\mathfrak{t} \prec \Phi \mid \Gamma, x^{\mathfrak{t}} : A, \Gamma' \vdash x : A}$$

VAR-ROUNDTRIP
$$\frac{\mathfrak{t} \prec \Phi \vdash \mathfrak{c}\ \mathsf{colour}}{\mathfrak{t} \prec \Phi \mid \Gamma, x^{\mathfrak{c}} : A, \Gamma' \vdash \underline{x} : \underline{A}}$$

VAR-MARKED
$$\frac{}{\mathfrak{t} \prec \Phi \mid \Gamma, \underline{x}^{\mathfrak{c}} : A, \Gamma' \vdash \underline{x} : A}$$

First, we have ordinary variable usage via the VAR rule: a colourful variable $x :^{\mathfrak{t}} A$ may be used if the colour $\mathfrak{t}$ of the variable is precisely the label at the top of the palette.

Next, any colourful variable $x :^{\mathfrak{c}} A$ can be used *marked* via VAR-ROUNDTRIP regardless of what colour that variable is labelled with in the context, yielding a term $\underline{x} : \underline{A}$. All the variables uses in the type $\underline{A}$ have also become marked. We think of $\underline{x}$ as the term that represents point of the indexing set corresponding to x, with the linear information forgotten.

Finally, any variable $\underline{x} :^{\mathfrak{c}} A$ that was added to the context via a marked context extension can also be used marked, with the same syntax as the previous rule, so $\underline{x} : A$. Here, A is already a marked type.

We have used various admissible rules in these context and variable rules to make the typing line up, and we display these rules in the Appendix.

Ordinary Type Formers. Ordinary type formers have their rules essentially unchanged, with every bound variable given the top colour of the palette as their label. For example,

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

Π-INTRO
$$\frac{\mathfrak{t} \prec \Phi \mid \Gamma, x^{\mathfrak{t}} : A \vdash b : B}{\mathfrak{t} \prec \Phi \mid \Gamma \vdash \lambda x.b : (x : A) \to B}$$

Π-ELIM
$$\frac{\mathfrak{t} \prec \Phi \mid \Gamma \vdash f : (x : A) \to B\ \mathsf{type} \qquad \mathfrak{t} \prec \Phi \mid \Gamma \vdash a : A}{\mathfrak{t} \prec \Phi \mid \Gamma \vdash f(a) : B[a/x]}$$

and similarly for Σ- and Id-types.

The Indexing Set Modality. The first new type constructor is the unary type operator $\natural A$, which extracts the underlying indexing set of any type A. The rules are as follows:

$$\frac{\underline{\Gamma} \vdash A \ \mathsf{type}}{\Gamma \vdash \natural A \ \mathsf{type}}\ \natural\text{-FORM} \qquad \frac{\underline{\Gamma} \vdash a : A}{\Gamma \vdash a^{\natural} : \natural A}\ \natural\text{-INTRO} \qquad \frac{\Gamma \vdash b : \natural A}{\Gamma \vdash b_{\natural} : A}\ \natural\text{-ELIM}$$

Let us refer to a term/type in context $\underline{\Gamma}$, i.e. a term/type all of whose free variables are marked, as *dull*. The formation rule says that for any dull type A there is a type $\natural A$, and for any dull term of a dull type $a : A$ there is a term $a^{\natural} : \natural A$. Note that the type A must be assumed to be dull for the type $\natural A$ in the conclusion to be well-formed. The elimination rule says that for any (not necessarily dull) term $b : \natural A$, there is a term $b_{\natural} : A$.

The rules for this type former are formally similar to Quipper's $!A$, and is used in its translation below. The 'dullness' of the context replaces the restriction of the context to only parameter types.

Tensor. The $\otimes$-type is a restricted version of the ordinary $\times$-type so that to form a $\otimes$-pair, the two elements of the pair must divide the resources of the palette in a linear way. We call a valid chunk of the palette a *slice* and a valid division of a palette into two slices a *split*, written $\Phi \vdash s_L \boxtimes s_R \ \mathsf{split}$. A split of a palette Φ into s_L and s_R semantically represents a morphism $\Phi \to s_L \otimes s_R$, consisting of the projections from ordinary products and the symmetry and re-association of tensor products necessary to divide Φ in that way.

As a piece of syntax, a slice is either:

- $\mathfrak{r}$, a single colour from Φ;
- $\mathfrak{t}. \prec \mathfrak{c}_1 \otimes \mathfrak{c}_2 \otimes \cdots \otimes \mathfrak{c}_n (\otimes 1)$, a fresh colour $\mathfrak{t}$ followed by a list of labels from Φ optionally including the special symbol 1. The period indicates that $\mathfrak{t}$ is a freshly bound colour.

Not all lists of labels form a valid slice: the choices must be compatible with the shape of the palette. For example, when we reach a comma node we must choose exactly one branch to be included in the slice. A slice also cannot contain both a colour together with any piece of the subpalette under that colour; choosing a colour already specifies that we are selecting the entire subpalette. The formal rules for slices and splits are given in the Appendix, but the way they work are clearest from examples:

$$\begin{aligned}
\mathfrak{t} \prec \mathfrak{a} \otimes \mathfrak{b} &\vdash \mathfrak{a} \boxtimes \mathfrak{b} \ \mathsf{split} \\
\mathfrak{t} \prec \mathfrak{a} \otimes \mathfrak{b} &\vdash \mathfrak{b} \boxtimes \mathfrak{a} \ \mathsf{split} \\
\mathfrak{t} \prec (\mathfrak{a} \prec \mathfrak{p} \otimes \mathfrak{q}) \otimes \mathfrak{b} &\vdash \mathfrak{a} \boxtimes \mathfrak{b} \ \mathsf{split} \\
\mathfrak{t} \prec (\mathfrak{a} \prec \mathfrak{p} \otimes \mathfrak{q}) \otimes \mathfrak{b} &\vdash \mathfrak{p} \boxtimes (\mathfrak{t}' \prec \mathfrak{q} \otimes \mathfrak{b}) \ \mathsf{split} \\
\mathfrak{t} \prec (\mathfrak{a} \prec \mathfrak{p} \otimes \mathfrak{q}) \otimes \mathfrak{b} &\vdash \mathfrak{q} \boxtimes (\mathfrak{t}' \prec \mathfrak{p} \otimes \mathfrak{b}) \ \mathsf{split} \\
\mathfrak{t} \prec (\mathfrak{a}_1 \otimes \mathfrak{b}_1, \mathfrak{a}_2 \otimes \mathfrak{b}_2) &\vdash \mathfrak{a}_1 \boxtimes \mathfrak{b}_1 \ \mathsf{split} \\
\mathfrak{t} \prec (\mathfrak{a}_1 \otimes \mathfrak{b}_1, \mathfrak{a}_2 \otimes \mathfrak{b}_2) &\vdash \mathfrak{a}_2 \boxtimes \mathfrak{b}_2 \ \mathsf{split}
\end{aligned}$$

The rules for the $\otimes$-type are then

$$\otimes\text{-FORM}\quad \frac{\mathfrak{t} \mid \cdot \vdash A \;\mathsf{type} \qquad \mathfrak{t} \mid \cdot \vdash B \;\mathsf{type}}{\mathfrak{t} \prec \Phi \mid \Gamma \vdash A \otimes B \;\mathsf{type}}$$

$$\otimes\text{-INTRO}\quad \frac{\mathfrak{t} \prec \Phi \vdash s_L \boxtimes s_R \;\mathsf{split} \qquad \Phi^{s_L} \mid \Gamma^{s_L} \vdash a : A \qquad \Phi^{s_R} \mid \Gamma^{s_R} \vdash b : B}{\mathfrak{t} \prec \Phi \mid \Gamma \vdash a \;{}_{s_L}\otimes_{s_R}\; b : A \otimes B}$$

$$\otimes\text{-ELIM}\quad \frac{\begin{array}{c}\mathfrak{t} \prec \Phi \mid \Gamma, z^{\mathfrak{t}} : A \otimes B \vdash C \;\mathsf{type} \\ \mathfrak{t} \prec \Phi, \mathfrak{l} \otimes \mathfrak{r} \mid \Gamma, x^{\mathfrak{l}} : A, y^{\mathfrak{r}} : B \vdash c : C[x \;{}_{\mathfrak{l}}\otimes_{\mathfrak{r}}\; y/z] \\ \mathfrak{t} \prec \Phi \mid \Gamma \vdash s : A \otimes B\end{array}}{\mathfrak{t} \prec \Phi \mid \Gamma \vdash \mathsf{let}\; x \;{}_{\mathfrak{l}}\otimes_{\mathfrak{r}}\; y := s \;\mathsf{in}\; c : C[s/z]}$$

The $\otimes$-type $A \otimes B$ can be formed for any closed types A and B. To introduce a $\otimes$-pair $x \;{}_{s_L}\otimes_{s_R}\; y$, we must split the context into two pieces and use the pieces separately to prove x and y. This is very similar to the Quipper rule; the main difference is that the division of the context into pieces (s_L and s_R) is recorded as data in the derivation, rather than having it implicit in the context of the conclusion $\Gamma_1 + \Gamma_2$ as in Quipper.

To eliminate a $\otimes$-pair p, we may assume it is of the form $x \;{}_{\mathfrak{r}}\otimes_{\mathfrak{b}}\; y$, where x and y are assigned fresh colours $\mathfrak{r}$ and $\mathfrak{b}$ that are tensored together as $\mathfrak{r} \otimes \mathfrak{b}$ at the top level in the ambient palette. Although it seems almost identical to the Quipper rule it is different in an important way: the Quipper rule divides the context into Γ_1 and Γ_2 to form the target of the elimination and the body, whereas the bunched theory does not split the context at all. The new tensor pair $\mathfrak{r} \otimes \mathfrak{b}$ in the palette is combined with the existing palette using a palette comma, i.e. $\times$, rather than $\otimes$.

In general, we have a 'pattern matching' construct that allows us to decompose an iterated $\otimes$ all at once. This is necessary, for example, to define an associativity map via

$$\lambda t.\mathsf{let}\; (x \otimes y) \otimes z := t \;\mathsf{in}\; x \otimes (y \otimes z),$$

Unit. The unit is implemented by allowing a special 'unitor' split, where the entire palette is placed on one side, as in $\mathfrak{t} \prec \Phi \vdash \mathfrak{t} \boxtimes \varnothing \;\mathsf{split}$. The rules are:

$$I\text{-FORM}\quad \frac{}{\mathfrak{t} \prec \Phi \mid \Gamma \vdash I \;\mathsf{type}} \qquad I\text{-INTRO}\quad \frac{}{\mathfrak{t} \prec \varnothing, \Phi \mid \Gamma \vdash \natural : I}$$

$$I\text{-UNITOR-ELIM}\quad \frac{\begin{array}{c}\mathfrak{t} \mid \cdot \vdash B \;\mathsf{type} \\ \mathfrak{t} \prec \Phi \mid \Gamma, z^{\mathfrak{t}} : I \otimes B \vdash C \;\mathsf{type} \\ \mathfrak{t} \prec \Phi \mid \Gamma, y^{\mathfrak{t}} : B \vdash c : C \\ \mathfrak{t} \prec \Phi \mid \Gamma \vdash s : I \otimes B\end{array}}{\mathfrak{t} \prec \Phi \mid \Gamma \vdash \mathsf{let}\; \natural \;{}_{\varnothing}\otimes_{\mathfrak{t}}\; y := s \;\mathsf{in}\; c : C}$$

The unit type I is well formed in any context. When one of these special markers $\varnothing$ is present at the top of the palette (as a result of using a unitor split), we may use it to form a term of the unit. And to use a term of I, we apply the unitor in the other direction: this is described as a let binding that exposes the B factor of a type $I \otimes B$.

Hom. Finally, we have $\multimap$-types.

$$\frac{\mathfrak{r} \mid \cdot \vdash A \;\mathsf{type} \qquad \mathfrak{p} \mid \cdot \vdash B \;\mathsf{type}}{\mathfrak{t} \prec \Phi \mid \Gamma \vdash A \multimap B \;\mathsf{type}}\;\multimap\text{-FORM}$$

$$\frac{\mathfrak{p} \prec (\mathfrak{t} \prec \Phi) \otimes \mathfrak{r} \mid \Gamma, x^{\mathfrak{r}} : A \vdash b : B}{\mathfrak{t} \prec \Phi \mid \Gamma \vdash \partial^{\mathfrak{p}} x^{\mathfrak{r}}.b : A \multimap B}\;\multimap\text{-INTRO} \qquad \frac{\mathfrak{t} \prec \Phi \vdash s_L \boxtimes s_R \;\mathsf{split} \qquad (\mathfrak{t} \prec \Phi)^{s_L} \mid \Gamma^{s_L} \vdash f : A \multimap B \qquad (\mathfrak{t} \prec \Phi)^{s_R} \mid \Gamma^{s_R} \vdash a : A}{\mathfrak{t} \prec \Phi \mid \Gamma \vdash f_{s_L} \langle a \rangle_{s_R} : B}\;\multimap\text{-ELIM}$$

The type $A \multimap B$ can be formed for any closed types A and B. The $\multimap$-introduction rule is a linear version of λ-abstraction, written $(\partial^{\mathfrak{p}} x^{\mathfrak{r}}.b)$. The bound variable x is labelled with a fresh colour $\mathfrak{r}$ that is combined with the existing palette using $\otimes$. Like any term, the body of the function needs a colour to lie at the front of the context, which is the purpose of the second bound colour $\mathfrak{p}$. The $\multimap$-elimination rule follows a similar pattern to $\otimes$-introduction. The palette is divided linearly into two pieces, with one piece used to produce the hom and the other to produce its argument.

Properties of the Linear Types. We quote the following basic facts about the linear type formers for use in the verification section.

Proposition 2.1. *The following properties are all provable internal to the theory.*

- $\natural$ *is self-adjoint, in that* $\natural(\natural A \to B) \simeq \natural(A \to \natural B)$ *([14, Corollary 1.1.26])*
- $\natural A \times (B \otimes C) \simeq (B \times \natural A) \otimes C \simeq B \otimes (\natural A \times C)$ *([14, Proposition 1.3.12])*
- $\otimes$ *is associative ([14, Proposition 1.3.27]) and unital ([14, Lemma 1.4.8])*
- *A uniqueness principle for* $\otimes$ *holds: for any function* $f : A \otimes B \to C$*, and term* $p : A \otimes B$ *there is an equality* $f(p) = \mathsf{let}\, x \,{}_{\mathfrak{a}}\!\otimes_{\mathfrak{b}}\, y := p \,\mathsf{in}\, f(x \,{}_{\mathfrak{a}}\!\otimes_{\mathfrak{b}}\, y)$ *([14, Proposition 1.2.2]).*
- *Equality is a congruence on* $\otimes$*, i.e., if* $a = a'$ *and* $b = b'$ *then* $a \otimes a' = b \otimes b'$ *([14, Proposition 1.3.28]).*
- $\otimes$ *is left adjoint to* $\multimap$*, in that* $\natural(A \otimes B \to C) \simeq \natural(A \to (B \multimap C))$ *([14, Proposition 1.5.9])* □

The mix of the two kinds of arrows in the statement of adjointness is important! To make sense of the last equivalence, think of $\natural(P \to Q)$ as internalising an external judgement $P \vdash Q$: the equivalence then expresses the *external* adjointness of $\otimes$ and $\multimap$.

3 Translating from Proto-Quipper

We now sketch a translation of the fragment of Quipper[2] given in Figure 1 into this bunched type theory. The main ideas of the translation are:

- The "context comma" in Quipper corresponds to the linear tensor, so the translation of any Quipper context will have palettes of a very simple form: a single top-level colour $\mathfrak{t}$ which splits into an iterated tensor of colours $\mathfrak{c}_{\_}$, one for each Quipper variable. As the nullary case of this, the empty context must be translated to the monoidal unit, rather than just the empty context of the bunched theory, which represents the terminal object.

[2]We shorten Proto-Quipper-M to Quipper throughout, though we always mean the standalone type non-dependent theory described in [17].

$$\text{VAR}\ \frac{}{\Phi, x : A \vdash x : A} \qquad \otimes\text{-INTRO}\ \frac{\Phi, \Gamma_1 \vdash M : A \qquad \Phi, \Gamma_2 \vdash N : B}{\Phi, \Gamma_1, \Gamma_2 \vdash M \otimes N : A \otimes B} \qquad \otimes\text{-ELIM}\ \frac{\Phi, \Gamma_1 \vdash M : A \otimes B \qquad \Phi, \Gamma_2, x : A, y : B \vdash N : C}{\Phi, \Gamma_1, \Gamma_2 \vdash \mathsf{let}\, x \otimes y := M \,\mathsf{in}\, NC}$$

$$\multimap\text{-INTRO}\ \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \multimap B} \qquad \multimap\text{-ELIM}\ \frac{\Phi, \Gamma_1 \vdash M : A \multimap B \qquad \Phi, \Gamma_2 \vdash N : A}{\Phi, \Gamma_1, \Gamma_2 \vdash M(N) : B} \qquad !\text{-INTRO}\ \frac{\Phi \vdash M : A}{\Phi \vdash \mathbf{lift} M : !A} \qquad !\text{-ELIM}\ \frac{\Gamma \vdash M : !A}{\Gamma \vdash \mathbf{force} M : A}$$

Figure 1: A Fragment of Proto-Quipper-M

- The notion of parameter types/parameter contexts plays a dual role in Quipper: representing the indexing set in when forming types, and representing the monoidal unit (pulled back over an arbitrary indexing set) when introducing terms of **Unit** and $!A$. The interpretation of every parameter type is equivalent to a constant family at the monoidal unit over some indexing set. The factor of I is necessary because context extension is always interpreted as combination with $\otimes$, even if the only data of the type that is actually used is the indexing set itself. In the bunched type theory, where we have access to non-linear type formers, we can manipulate the indexing set directly.
- Some of functors used to define the semantics of Quipper can be defined internally to the bunched type theory. Specifically, the operation $q\sharp$ that replaces an indexed family with the constant family at the terminal object is exactly the $\natural$ modality of the bunched type theory. The operation $p\sharp$ that replaces a family A with the constant family at the monoidal unit can be defined as $I \times \natural A$. So an internal definition of the right adjoint $q\flat$ to to $p\sharp$ is easy to read off: $\natural(I \to A)$, by composing the right adjoints of the two components. This leads to the internal definition of the ! operation.

The translation is defined in Figure 2. We omit the term translations for **Unit** and **Nat** because they are not used in the example. Contexts are translated exactly as indicated above: each variable x is assigned a colour $\mathfrak{c}_x$, and these colours are all combined with $\otimes$ in the palette. The empty context becomes a single variable $\phi :^{\mathfrak{c}_\phi} I$, whose colour $\mathfrak{c}_\phi$ is added to the palette in the same way. The translation of types is straightforward: the $\otimes$ and $\multimap$ types become the corresponding types in the bunched type theory, and $!A$ is translated to .

For terms, let us first describe the translation in the easy case that the parameter part of the context Φ is empty.

The context manipulation done in the 'splitting' rules of Quipper (i.e. $\otimes$-introduction/elimination and $\multimap$-elimination) need to be written in terms of slices and splits in the bunched type theory. There is one thing to be careful of: the colour of the phase variable $\mathfrak{c}_\phi$ can only be assigned to one side of the split; we arbitrarily choose the left side. To deal with this, we define an admissible 'unit substitution' that replaces any use of ϕ with the unit term $\natural$. This is not a well-typed operation in full generality, but it is well-typed in the image of the translation.

Proposition 3.1. *Suppose Φ is either of the form $\mathfrak{t} \prec \mathfrak{c}_\phi$ or $\mathfrak{t} \prec (\mathfrak{c}_\phi \otimes \Phi')$, and $\varnothing$ does not occur in Φ'. Then the admissible substitutions*

$$\text{UNIT-SUB}\ \frac{\Phi\ \mathsf{palette}}{\Phi[\varnothing/\mathfrak{c}_\phi]\ \mathsf{palette}} \qquad \text{UNIT-SUB}\ \frac{\Phi \vdash s\ \mathsf{slice}}{\Phi[\varnothing/\mathfrak{c}_\phi] \vdash s[\varnothing/\mathfrak{c}_\phi]\ \mathsf{palette}} \qquad \text{UNIT-SUB}\ \frac{\Phi \mid \phi :^{\mathfrak{c}_\phi} I, \Gamma \vdash a : A}{\Phi[\varnothing/\mathfrak{c}_\phi] \mid \Gamma \vdash a[\natural/\phi] : A}$$

are well-typed.

$$
\begin{aligned}
[\![\cdot]\!]_{\mathsf{cols}} &:\equiv \mathfrak{c}_\phi \\
[\![\cdot]\!]_{\mathsf{vars}} &:\equiv \phi :^{\mathfrak{c}_\phi} I \\
[\![\Gamma, x : A]\!]_{\mathsf{cols}} &:\equiv [\![\Gamma]\!]_{\mathsf{cols}} \otimes \mathfrak{c}_x \\
[\![\Gamma, x : A]\!]_{\mathsf{vars}} &:\equiv [\![\Gamma]\!]_{\mathsf{vars}}, x :^{\mathfrak{c}_x} [\![A]\!] \\
[\![\Gamma]\!]_{\mathsf{pal}} &:\equiv \mathfrak{t} \prec [\![\Gamma]\!]_{\mathsf{cols}}\ \mathsf{palette} \\
[\![\Gamma]\!] &:\equiv [\![\Gamma]\!]_{\mathsf{pal}} \mid [\![\Gamma]\!]_{\mathsf{vars}}\ \mathsf{ctx}
\end{aligned}
$$

$$
\begin{aligned}
[\![\mathbf{Unit}]\!] &:\equiv I \\
[\![A \otimes B]\!] &:\equiv [\![A]\!] \otimes [\![B]\!] \\
[\![A \multimap B]\!] &:\equiv [\![A]\!] \multimap [\![B]\!] \\
[\![!A]\!] &:\equiv I \times \natural (I \to [\![A]\!]) \\
[\![\mathbf{Nat}]\!] &:\equiv I \times \mathbb{N}
\end{aligned}
$$

$$
\begin{aligned}
[\![x]\!] &:\equiv \mathsf{let}\ \square \otimes x' := \phi \otimes x\ \mathsf{in}\ x' \\
[\![(M,N)]\!] &:\equiv [\![M]\!]\ {}_{[\![\Gamma_1]\!]_{\mathsf{pal}}}\otimes_{[\![\Gamma_2]\!]_{\mathsf{pal}}[\varnothing/\mathfrak{c}_\phi]}\ [\![N]\!][\square/\phi] \\
[\![\mathsf{let}\ (x,y) := M\ \mathsf{in}\ N]\!] &:\equiv \mathsf{let}\ h \otimes (x' \otimes y') := (\partial x.\partial y.[\![N]\!])\ {}_{[\![\Gamma_1]\!]_{\mathsf{pal}}}\otimes_{[\![\Gamma_2]\!]_{\mathsf{pal}}[\varnothing/\mathfrak{c}_\phi]}\ ([\![M]\!][\square/\phi])\ \mathsf{in}\ h\langle x'\rangle\langle y'\rangle \\
[\![\lambda x.M]\!] &:\equiv \partial^{\mathfrak{t}} x^{\mathfrak{c}_x}.[\![M]\!] \\
[\![MN]\!] &:\equiv [\![M]\!]_{[\![\Gamma_1]\!]_{\mathsf{pal}}}\langle [\![N]\!][\square/\phi]\rangle_{[\![\Gamma_2]\!]_{\mathsf{pal}}[\varnothing/\mathfrak{c}_\phi]} \\
[\![\mathbf{lift}\, M]\!] &:\equiv (\phi, (\lambda s.[\![M]\!][s/\phi])^\natural) \\
[\![\mathbf{force}\, M]\!] &:\equiv \mathsf{let}\ (\phi', f) := [\![M]\!]\ \mathsf{in}\ f_\natural(\phi')
\end{aligned}
$$

Figure 2: The Translation of Contexts, Types and Terms

The point of the previous is to engineer the following result:

Proposition 3.2. *For any Quipper contexts Γ_1 and Γ_2, there is a split*

$$[\![\Gamma_1, \Gamma_2]\!]_{\mathsf{pal}} \vdash [\![\Gamma_1]\!]_{\mathsf{pal}} \otimes [\![\Gamma_1]\!]_{\mathsf{pal}}[\varnothing/\mathfrak{c}_\phi]\ \mathsf{split}$$

where $[\![\Gamma_1]\!]_{\mathsf{pal}}[\varnothing/\mathfrak{c}_\phi]$ denotes the operation which

$$[\![\Gamma_1]\!]_{\mathsf{pal}}[\varnothing/\mathfrak{c}_\phi]$$

Turning to the term translations in Figure 2, we consider them one at a time:

- $[\![x]\!]$: The translated context is precisely of the form $\mathfrak{t} \prec (\mathfrak{c}_\phi \otimes \mathfrak{c}_x) \mid \phi :^{\mathfrak{c}_\phi} I, x :^{\mathfrak{c}_x} [\![A]\!]$, and so we can form $\phi \otimes x : I \otimes [\![A]\!]$ and use the unitor eliminator to remove the I.
- $[\![(M,N)]\!]$: This is almost immediate by induction and Proposition 3.2. The terms $[\![M]\!]$ and $[\![N]\!]$ are well-typed in contexts $[\![\Gamma_1]\!]$ and $[\![\Gamma_2]\!]$ respectively. These are weakened to

$$
\begin{aligned}
[\![\Gamma_1]\!]_{\mathsf{pal}} \mid [\![\Gamma_1]\!]_{\mathsf{vars}}, \underline{[\![\Gamma_2]\!]_{\mathsf{vars}}} &\vdash [\![M]\!] : A \\
[\![\Gamma_2]\!]_{\mathsf{pal}} \mid \underline{[\![\Gamma_1]\!]_{\mathsf{vars}}}, [\![\Gamma_2]\!]_{\mathsf{vars}} &\vdash [\![N]\!] : B
\end{aligned}
$$

by adding in the missing variables for each. These are very nearly of the correct form to apply $\otimes$-paring: the only issue is that $\phi :^{\mathfrak{c}_\phi} I$ appears in both contexts. This is corrected by applying the unit substitution to $[\![N]\!]$. Often, this substitution will reach an instance of the variable rule and cancel with the unitor eliminator, as we will see in the example later.

- $[\![\mathsf{let}\ (x,y) := M\ \mathsf{in}\ N]\!]$: This is more involved, to deal with the mismatch between the $\otimes$-eliminators of the two theories: in the bunched type theory the decomposed $\otimes$ is concatenated to the context with $\times$, whereas Quipper adds it using $\otimes$.

  For this we use a similar trick to [14, §1.6.1] to shuffle pieces of the context around. First $[\![N]\!]$ is used to produce a hom of shape $[\![A]\!] \multimap [\![B]\!] \multimap [\![C]\!]$ in context $[\![\Gamma_1]\!]$, which is paired with $[\![N]\!]$ (using the same unit manipulations as in $\otimes$-intro to deal with ϕ). Together this forms a term of type $([\![A]\!] \multimap [\![B]\!] \multimap [\![C]\!]) \otimes ([\![A]\!] \otimes [\![B]\!])$. After pattern matching this term as $h \otimes (x' \otimes y')$, we can simply (linearly) evaluate h at x' and y'.
- $[\![\lambda x.M]\!]$: This is straightforwardly translated to a linear lambda, assigning the new colour $\mathfrak{c}_x$ to the new variable x.
- $[\![MN]\!]$: This is straightforwardly translated to linear application, surrounded by the same unitor manipulations we have seen twice already.
- $[\![\mathbf{lift}\, M]\!]$: By the assumption that Φ is empty, the term $[\![M]\!]$ is in context $\mathfrak{c}_\phi \mid \phi : I\ \mathsf{ctx}$, so, after α-renaming ϕ to s, we may form the non-linear function $\lambda s.[\![M]\!][s/\phi] : I \to [\![A]\!]$. This is now a closed term, so we may use $\natural$-intro and (non-linearly) pair with the ϕ of the current context.
- $[\![\mathbf{force}\, M]\!]$: We can extract the term of I and the function, and apply them; this is the counit of Quipper's $p\sharp \dashv q\flat$ adjunction.

Now, we turn to handling a possibly non-empty parameter context. The reason this is a slight pain is that context structure used in Quipper does not distinguish between 'parameter variables' and 'linear variables': the choice of whether to consider a variable as a parameter is made independently every time a splitting rule is invoked. Our solution is to pack and unpack the chosen parameter context before and after a rule is used, using the following propositions.

Proposition 3.3. *If P is a parameter type, then $[\![P]\!] \simeq I \times \natural[\![P]\!]$.*

Proof. By induction on the structure of P. This is an internal version of the observation in [17, Remark 3.9]. □

Proposition 3.4. *For any parameter context Φ and context Γ, there are admissible operations*

$$\textsc{pack}\ \frac{\mathbf{t} \prec \mathfrak{c}_\phi \otimes [\![\Phi]\!]_{\mathsf{cols}} \otimes [\![\Gamma]\!]_{\mathsf{cols}} \mid \phi :^{\mathfrak{c}_\phi} I, [\![\Phi]\!], [\![\Gamma]\!] \vdash a : A}{\mathbf{t} \prec \mathfrak{c}_\phi \otimes [\![\Gamma]\!]_{\mathsf{cols}} \mid \phi :^{\mathfrak{c}_\phi} I, \underline{[\![\Phi]\!]}, [\![\Gamma]\!] \vdash \mathsf{pack}_\Phi(a) : A}$$

$$\textsc{unpack}\ \frac{\mathbf{t} \prec \mathfrak{c}_\phi \otimes [\![\Gamma]\!]_{\mathsf{cols}} \mid \phi :^{\mathfrak{c}_\phi} I, \underline{[\![\Phi]\!]}, [\![\Gamma]\!] \vdash a : A}{\mathbf{t} \prec \mathfrak{c}_\phi \otimes [\![\Phi]\!]_{\mathsf{cols}} \otimes [\![\Gamma]\!]_{\mathsf{cols}} \mid \phi :^{\mathfrak{c}_\phi} I, [\![\Phi]\!], [\![\Gamma]\!] \vdash \mathsf{unpack}_\Phi(a) : A}$$

where $\underline{[\![\Phi]\!]}$ denotes replacing every variable in $[\![\Phi]\!]$ with a marked variable.

Proof. By induction on the length of Φ, using Proposition 3.3 to extract a term of I from each variable in $[\![\Phi]\!]$, and iteratively applying the unitors to pack/unpack these into the variable ϕ. □

Each of the translations in Figure 2 can be modified to pack and unpack the parameter context as necessary. For example, the variable rule becomes $[\![x]\!] :\equiv \mathsf{unpack}_\Phi(\mathsf{let}\ ¤ \otimes x' := \phi \otimes x\,\mathsf{in}\,x')$, and the λ rule becomes $[\![\lambda x.M]\!] :\equiv \mathsf{unpack}_\Phi(\partial^{\mathfrak{t}} x^{\mathfrak{e}_x}.\mathsf{pack}_\Phi([\![M]\!]))$.

Proto-Quipper-D. We believe that the dependent linear type theory of Proto-Quipper-D [2] can also be translated into the bunched type theory, although the translation will necessarily be much more complicated.

In Proto-Quipper-D, the 'underlying shape' operation $\mathrm{Sh}(-)$ that mediates dependency is defined inductively on the syntax of types, whereas in the bunched type theory, the indexing set of a type A is extracted by the modality $\natural A$. This $\natural A$ does not compute inductively on the structure of A, instead, the equalities defining Sh in Proto-Quipper-D become *theorems* of the bunched type theory.

For example, the definition $\mathrm{Sh}((x:A)\otimes B) = (x:\mathrm{Sh}(A))\otimes \mathrm{Sh}(B)$ corresponds to the internally provable equivalence $I \times \natural[(x:A)\otimes B] \simeq [(x : I\times\natural A)\otimes(I\times\natural B)]$. These equivalences would need to be inserted any time that a dependent type is formed.

In Proto-Quipper-D, the '$\to$' type former is introduced to represent underlying indexing set of a $\multimap$-type. Like $!A$, this is a construction that already exists in the bunched type theory: the translation would define $[\![A \to B]\!] :\equiv I \times (\natural A \to \natural B)$.

A version of Proposition 3.3 would continue to be true in this setting, and so the ω annotations could be handled in a similar way to the to the parameter contexts Φ of Proto-Quipper-M: the unitors would be used to duplicate the assumptions as necessary when using the splitting rules.

4 Verification

We now translate what is almost the simplest possible Quipper program and verify one of its properties. The following code:

```
object Qubit
gate H : Qubit -> Qubit
circuit : ! (Qubit * Qubit -> Qubit * Qubit)
circuit n p = let (x, y) = p
              in  (H x, y)
```

represents the circuit with two qubits that simply applies the **H** gate to the first qubit. Let us verify that this circuit is its own inverse, assuming that **H** is its own inverse.

With the hidden instances of ! elaborated, the function is defined by

$$\begin{aligned}\mathbf{H} &: !(\mathbf{Qubit} \to \mathbf{Qubit})\\ \mathbf{circuit} &: !(\mathbf{Qubit}\otimes\mathbf{Qubit} \to \mathbf{Qubit}\otimes\mathbf{Qubit})\\ \mathbf{circuit} &:\equiv \mathbf{lift}\ \lambda p.\mathsf{let}\ (x,y) := p\,\mathsf{in}\,((\mathbf{force}\,\mathbf{H})(x),y)\end{aligned}$$

We postulate a type Qubit and define $[\![\mathbf{Qubit}]\!] :\equiv \mathsf{Qubit}$. Applying the translation to the types of **H** and $[\![\mathbf{circuit}]\!]$ gives

$$\begin{aligned}[\![\mathbf{H}]\!] &: I \times \natural(I \to (\mathsf{Qubit} \multimap \mathsf{Qubit}))\\ [\![\mathbf{circuit}]\!] &: I \times \natural(I \to (\mathsf{Qubit}\otimes\mathsf{Qubit} \multimap \mathsf{Qubit}\otimes\mathsf{Qubit}))\end{aligned}$$

Let us work mechanically through the translation of the term $[\![\mathbf{circuit}]\!]$, omitting most of the slice annotations for legibility.

$$\begin{aligned}
&[\![\mathbf{lift}\ \lambda p.\mathsf{let}\ (x,y) := p\,\mathsf{in}\,((\mathbf{force}\,\mathbf{H})(x),y)]\!]\\
&\equiv (\phi,(\lambda s.[\![\lambda p.\mathsf{let}\ (x,y) := p\,\mathsf{in}\,((\mathbf{force}\,\mathbf{H})(x),y)]\!][s/\phi])^\natural)\\
&\equiv (\phi,(\lambda s.\partial p.[\![\mathsf{let}\ (x,y) := p\,\mathsf{in}\,((\mathbf{force}\,\mathbf{H})(x),y)]\!][s/\phi])^\natural)\\
&\equiv (\phi,(\lambda s.\partial p.\\
&\qquad \mathsf{let}\ h\otimes(x'\otimes y') := (\partial x.\partial y.[\![((\mathbf{force}\,\mathbf{H})(x),y)]\!][s/\phi])\otimes([\![p]\!][\maltese/\phi])\,\mathsf{in}\,h\langle x'\rangle\langle y'\rangle)^\natural)
\end{aligned}$$

The first remaining translation becomes

$$\begin{aligned}
&[\![((\mathbf{force}\,\mathbf{H})(x),y)]\!]\\
&\equiv [\![(\mathbf{force}\,\mathbf{H})(x)]\!]\otimes[\![y]\!][\maltese/\phi]\\
&\equiv [\![(\mathbf{force}\,\mathbf{H})(x)]\!]\otimes(\mathsf{let}\ \maltese\otimes x' := \phi\otimes y\,\mathsf{in}\,x')[\maltese/\phi]\\
&\equiv [\![(\mathbf{force}\,\mathbf{H})(x)]\!]\otimes y\\
&\equiv ([\![\mathbf{force}\,\mathbf{H}]\!]\langle[\![x]\!][\maltese/\phi]\rangle)\otimes y\\
&\equiv ([\![\mathbf{force}\,\mathbf{H}]\!]\langle x\rangle)\otimes y
\end{aligned}$$

and similarly to show that $[\![p]\!][\maltese/\phi]\equiv p$. So in all,

$$[\![\mathbf{circuit}]\!]\equiv(\phi,(\lambda s.\partial p.\mathsf{let}\ h\otimes(x'\otimes y') := (\partial x.\partial y.([\![\mathbf{force}\,\mathbf{H}]\!]\langle x\rangle)\otimes y)\otimes p\,\mathsf{in}\,h\langle x'\rangle\langle y'\rangle)^\natural)$$

Now that we are working in the bunched type theory, we can perform some simplification. First, the interesting part of this term is the piece of type $I\to(\mathsf{Qubit}\otimes\mathsf{Qubit}\multimap\mathsf{Qubit}\otimes\mathsf{Qubit})$, so

$$\begin{aligned}
&[\![\mathbf{circuit}]\!]_2 : I\to(\mathsf{Qubit}\otimes\mathsf{Qubit}\multimap\mathsf{Qubit}\otimes\mathsf{Qubit})\\
&[\![\mathbf{circuit}]\!]_2 :\equiv \lambda s.\partial p.\mathsf{let}\ h\otimes(x'\otimes y') := (\partial x.\partial y.([\![\mathbf{force}\,\mathbf{H}]\!]\langle x\rangle)\otimes y)\otimes p\,\mathsf{in}\,h\langle x'\rangle\langle y'\rangle)
\end{aligned}$$

The next simplification is applying the chain of equivalences

$$(I\to(A\multimap B))\simeq(I\otimes A\to B)\simeq(A\to B)$$

to remove the need for the variable s: this gives

$$\begin{aligned}
&[\![\mathbf{circuit}]\!]_3 : \mathsf{Qubit}\otimes\mathsf{Qubit}\to\mathsf{Qubit}\otimes\mathsf{Qubit}\\
&[\![\mathbf{circuit}]\!]_3 :\equiv \lambda p.\mathsf{let}\ h\otimes(x'\otimes y') := (\partial x.\partial y.([\![\mathbf{force}\,\mathbf{H}]\!]\langle x\rangle)\otimes y)\otimes p\,\mathsf{in}\,h\langle x'\rangle\langle y'\rangle)
\end{aligned}$$

We can now apply the uniqueness principle for $\otimes$ on p, and the term mostly collapses:

$$\begin{aligned}
&[\![\mathbf{circuit}]\!]_3 : \mathsf{Qubit}\otimes\mathsf{Qubit}\to\mathsf{Qubit}\otimes\mathsf{Qubit}\\
&[\![\mathbf{circuit}]\!]_3 \equiv \lambda p.\mathsf{let}\ h\otimes(x'\otimes y') := (\partial x.\partial y.([\![\mathbf{force}\,\mathbf{H}]\!]\langle x\rangle)\otimes y)\otimes p\,\mathsf{in}\,h\langle x'\rangle\langle y'\rangle)\\
&\qquad = \lambda p.\mathsf{let}\ r\otimes s := p\,\mathsf{in}\,\mathsf{let}\ h\otimes(x'\otimes y') := (\partial x.\partial y.([\![\mathbf{force}\,\mathbf{H}]\!]\langle x\rangle)\otimes y)\otimes(r\otimes s)\,\mathsf{in}\,h\langle x'\rangle\langle y'\rangle)\\
&\qquad \equiv \lambda p.\mathsf{let}\ r\otimes s := p\,\mathsf{in}\,(\partial x.\partial y.([\![\mathbf{force}\,\mathbf{H}]\!]\langle x\rangle)\otimes y)\langle r\rangle\langle s\rangle\\
&\qquad \equiv \lambda p.\mathsf{let}\ r\otimes s := p\,\mathsf{in}\,[\![\mathbf{force}\,\mathbf{H}]\!]\langle r\rangle\otimes s
\end{aligned}$$

We have finally reached how we would have written the circuit by hand, if we were working directly in the bunched type theory.

It is easy to verify that this is self-inverse, i.e. that $[\![\mathbf{circuit}]\!]_3 \circ [\![\mathbf{circuit}]\!]_3 = \mathsf{id}_{\mathsf{Qubit}\otimes\mathsf{Qubit}}$. By function extensionality, it is sufficient to check pointwise. By the same uniqueness principle as above, it is sufficient to check on terms $x \otimes y$, and then:

$$\begin{aligned}
&[\![\mathbf{circuit}]\!]_3([\![\mathbf{circuit}]\!]_3(x \otimes y)) \\
&\equiv [\![\mathbf{circuit}]\!]_3([\![\mathbf{force}\,\mathbf{H}]\!]\langle x\rangle \otimes y \\
&\equiv [\![\mathbf{force}\,\mathbf{H}]\!]\langle [\![\mathbf{force}\,\mathbf{H}]\!]\langle x\rangle\rangle \otimes y
\end{aligned}$$

We assumed that $\mathbf{H}$ was self inverse, i.e., $[\![\mathbf{force}\,\mathbf{H}]\!]\langle [\![\mathbf{force}\,\mathbf{H}]\!]\langle x\rangle\rangle = x$, so congruence of equality on $\otimes$ lets us conclude that $[\![\mathbf{circuit}]\!]_3([\![\mathbf{circuit}]\!]_3(x \otimes y)) = x \otimes y$ as desired.

References

[1] Peng Fu, Kohei Kishida, Neil J. Ross & Peter Selinger (2020): *A Tutorial Introduction to Quantum Circuit Programming in Dependently Typed Proto-Quipper*. In: *Reversible Computation*, Springer International Publishing, Cham, pp. 153–168, doi:10.1007/978-3-030-52482-1_9.

[2] Peng Fu, Kohei Kishida & Peter Selinger (2020): *Linear Dependent Type Theory for Quantum Programming Languages: Extended Abstract*. In: *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '20, Association for Computing Machinery, Saarbrücken, Germany, pp. 440–453, doi:10.1145/3373718.3394765.

[3] Alexander S. Green, Peter LeFanu Lumsdaine, Neil J. Ross, Peter Selinger & Benoît Valiron (2013): *An introduction to quantum programming in Quipper*. In: *Reversible computation*, *Lecture Notes in Comput. Sci.* 7948, Springer, Heidelberg, pp. 110–124, doi:10.1007/978-3-642-38986-3_10.

[4] Alexander S. Green, Peter LeFanu Lumsdaine, Neil J. Ross, Peter Selinger & Benoît Valiron (2013): *Quipper: A Scalable Quantum Programming Language*. *SIGPLAN Not.* 48(6), pp. 333–342, doi:10.1145/2499370.2462177.

[5] Per Martin-Löf (1984): *Intuitionistic Type Theory*. *Studies in Proof Theory. Lecture Notes* 1, Bibliopolis, Naples. Available at `https://www.csie.ntu.edu.tw/~b94087/ITT.pdf`. Notes by Giovanni Sambin.

[6] David J. Myers, Mitchell Riley, Hisham Sati & Urs Schreiber (2023): *Effective Quantum Certification via Linear Homotopy Types*. Submitted.

[7] David Jaz Myers, Hisham Sati & Urs Schreiber (2023): *Topological Quantum Gates in Homotopy Type Theory*. arXiv:2303.02382.

[8] Peter O'Hearn (2003): *On bunched typing*. *Journal of Functional Programming* 13(4), pp. 747–796, doi:10.1017/S0956796802004495.

[9] Peter W. O'Hearn & David J. Pym (1999): *The Logic of Bunched Implications*. *Bulletin of Symbolic Logic* 5(2), pp. 215–244, doi:10.2307/421090.

[10] Jennifer Paykin, Robert Rand & Steve Zdancewic (2017): *QWIRE: A Core Language for Quantum Circuits*. In: *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, POPL '17, Association for Computing Machinery, Paris, France, pp. 846–858, doi:10.1145/3009837.3009894.

[11] Jennifer Paykin & Steve Zdancewic (2019): *A HoTT Quantum Equational Theory (Extended Version)*. arXiv:1904.04371.

[12] Robert Rand (2018): *Formally Verified Quantum Programming*. Ph.D. thesis, University of Pennsylvania. Available at `https://repository.upenn.edu/edissertations/3175`.

[13] Robert Rand, Jennifer Paykin & Steve Zdancewic (2018): *QWIRE Practice: Formal Verification of Quantum Circuits in Coq*. *Electronic Proceedings in Theoretical Computer Science*. arXiv:1803.00699.

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

[15] Mitchell Riley, Eric Finster & Daniel R. Licata (2021): *Synthetic Spectra via a Monadic and Comonadic Modality*. arXiv:2102.04099.

[16] Mitchell Riley & Daniel R. Licata (2021): *A Dependent Bunched Type Theory for Synthetic Stable Homotopy Theory*.

[17] Francisco Rios & Peter Selinger (2018): *A Categorical Model for a Quantum Circuit Description Language (Extended Abstract)*. In: *Proceedings 14th International Conference on Quantum Physics and Logic, Electron. Proc. Theor. Comput. Sci. (EPTCS)* 266, pp. 164–178, doi:10.4204/EPTCS.266.11.

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

A Rules for Palettes, Slices and Splits

Palettes.

$$\textsc{pal-empty}\ \frac{}{1\ \mathsf{palette}} \qquad \textsc{pal-}\times\ \frac{\Phi_1\ \mathsf{palette} \qquad \Phi_2\ \mathsf{palette}}{\Phi_1, \Phi_2\ \mathsf{palette}}$$

$$\textsc{pal-unit}\ \frac{}{\varnothing\ \mathsf{palette}} \qquad \textsc{pal-}\otimes\ \frac{\Phi_1\ \mathsf{palette} \qquad \Phi_2\ \mathsf{palette}}{\Phi_1 \otimes \Phi_2\ \mathsf{palette}} \qquad \textsc{pal-col}\ \frac{\Phi\ \mathsf{palette}}{\mathfrak{c} \prec \Phi\ \mathsf{palette}}$$

Slices. We have a 'preslice' judgement, representing a valid list of colours from the palette, a 'preslice$_\varepsilon$' judgement that optionally adds $- \otimes 1$, and finally an actual 'slice' judgement, adding the fresh $\mathfrak{t}. \prec -$ top label if necessary.

$$\frac{}{\Phi \vdash \varnothing\ \mathsf{preslice}} \qquad \frac{}{\mathfrak{c} \prec \Phi \vdash \mathfrak{c}\ \mathsf{preslice}} \qquad \frac{\Phi \vdash s\ \mathsf{preslice}}{\mathfrak{c} \prec \Phi \vdash s\ \mathsf{preslice}}$$

$$\frac{\Phi_1 \vdash s\ \mathsf{preslice}}{\Phi_1, \Phi_2 \vdash s\ \mathsf{preslice}} \qquad \frac{\Phi_2 \vdash s\ \mathsf{preslice}}{\Phi_1, \Phi_2 \vdash s\ \mathsf{preslice}}$$

$$\frac{\Phi_L \vdash s_L\ \mathsf{preslice} \qquad \Phi_R \vdash s_R\ \mathsf{preslice}}{\Phi_L \otimes \Phi_R \vdash s_L \otimes s_R\ \mathsf{preslice}}$$

$$\frac{\Phi \vdash s\ \mathsf{preslice}}{\Phi \vdash s\ \mathsf{preslice}_\varepsilon} \qquad \frac{\Phi \vdash s\ \mathsf{preslice}}{\Phi \vdash s \otimes 1\ \mathsf{preslice}_\varepsilon}$$

$$\frac{\Phi \vdash \mathfrak{c}\ \mathsf{colour}}{\Phi \vdash \mathfrak{c}\ \mathsf{slice}} \qquad \frac{\Phi \vdash s\ \mathsf{preslice}_\varepsilon}{\Phi \vdash (\mathfrak{t}. \prec s)\ \mathsf{slice}}$$

For any slice s, the *underlying preslice* of s, written $\mathsf{u}(s)$, is defined by $\mathsf{u}(\mathfrak{c}) :\equiv \mathfrak{c}$ and $\mathsf{u}(\mathfrak{t}. \prec s) :\equiv s$; this is used when describing the rules for splits.

Splits.

$$\frac{}{\varnothing \vdash (\varnothing \otimes \varepsilon_L) \boxtimes (\varnothing \otimes \varepsilon_R)\ \mathsf{presplit}} \qquad \frac{\varepsilon_L \equiv \top \text{ or } \varepsilon_R \equiv \top}{\Phi \vdash (\varnothing \otimes \varepsilon_L) \boxtimes (\varnothing \otimes \varepsilon_R)\ \mathsf{presplit}}$$

$$\frac{}{\mathfrak{c} \prec \Phi \vdash (\mathfrak{c} \otimes \varepsilon_L) \boxtimes (\varnothing \otimes \varepsilon_R)\ \mathsf{presplit}} \qquad \frac{}{\mathfrak{c} \prec \Phi \vdash (\varnothing \otimes \varepsilon_L) \boxtimes (\mathfrak{c} \otimes \varepsilon_R)\ \mathsf{presplit}}$$

$$\frac{\Phi \vdash s_L \boxtimes s_R\ \mathsf{presplit}}{\mathfrak{c} \prec \Phi \vdash s_L \boxtimes s_R\ \mathsf{presplit}}$$

$$\frac{\Phi_1 \vdash s_{1L} \boxtimes s_{1R}\ \mathsf{presplit}}{\Phi_1, \Phi_2 \vdash s_{1L} \boxtimes s_{1R}\ \mathsf{presplit}} \qquad \frac{\Phi_2 \vdash s_{2L} \boxtimes s_{2R}\ \mathsf{presplit}}{\Phi_1, \Phi_2 \vdash s_{2L} \boxtimes s_{2R}\ \mathsf{presplit}}$$

$$\frac{\Phi_1 \vdash s_{1L} \boxtimes s_{1R}\ \mathsf{presplit} \quad \Phi_2 \vdash s_{2L} \boxtimes s_{2R}\ \mathsf{presplit}}{\Phi_1 \otimes \Phi_2 \vdash (s_{1L} \otimes s_{2L}) \boxtimes (s_{1R} \otimes s_{2R})\ \mathsf{presplit}}$$

$$\frac{\Phi \vdash \mathsf{u}(s_L) \boxtimes \mathsf{u}(s_R)\ \mathsf{presplit}}{\Phi \vdash s_L \boxtimes s_R\ \mathsf{split}}$$

B Admissible Rules

Marking contexts:

$$\textsc{ctx-mark}\ \frac{\mathfrak{t} \prec \Phi \mid \Gamma\ \mathsf{ctx}}{\mathfrak{t} \mid \underline{\Gamma}\ \mathsf{ctx}} \qquad \begin{array}{c} \underline{\cdot} :\equiv \cdot \\ \underline{\Gamma, x^{\mathfrak{c}} : A} :\equiv \underline{\Gamma}, \underline{x}^{\mathfrak{c}} : \underline{A} \\ \underline{\Gamma, \underline{x}^{\mathfrak{c}} : A} :\equiv \underline{\Gamma}, \underline{x}^{\mathfrak{c}} : A \end{array} \qquad \frac{\mathfrak{t} \prec \Phi \mid \Gamma\ \mathsf{ctx}}{\underline{\Gamma} \equiv \underline{\Gamma}}$$

Filtering contexts:

$$\textsc{pal-filter}\ \frac{\Phi \vdash \mathfrak{c}\ \mathsf{colour}}{\Phi^{\mathfrak{c}}\ \mathsf{palette}} \qquad \frac{\Phi \vdash \mathfrak{c}\ \mathsf{colour} \qquad \Phi^{\mathfrak{c}} \vdash \mathfrak{d}\ \mathsf{colour}}{(\Phi^{\mathfrak{c}})^{\mathfrak{d}} \equiv \Phi^{\mathfrak{d}}\ \mathsf{palette}}$$

$$\textsc{ctx-filter}\ \frac{\Phi \vdash \mathfrak{c}\ \mathsf{colour} \qquad \Phi \mid \Gamma\ \mathsf{ctx}}{\Phi^{\mathfrak{c}} \mid \Gamma^{\mathfrak{c}}\ \mathsf{ctx}} \qquad \begin{array}{l} (\cdot)^{\mathfrak{c}} :\equiv \cdot \\ (\Gamma, x^{\mathfrak{d}} : A)^{\mathfrak{c}} :\equiv \Gamma^{\mathfrak{c}}, x^{\mathfrak{d}} : \underline{A} \quad \text{if } \mathfrak{d} \in \Phi^{\mathfrak{c}} \\ (\Gamma, x^{\mathfrak{d}} : A)^{\mathfrak{c}} :\equiv \Gamma^{\mathfrak{c}}, \underline{x}^{\mathfrak{d}} : A \quad \text{if } \mathfrak{d} \notin \Phi^{\mathfrak{c}} \\ (\Gamma, \underline{x}^{\mathfrak{d}} : A)^{\mathfrak{c}} :\equiv \Gamma^{\mathfrak{c}}, \underline{x}^{\mathfrak{d}} : A \end{array}$$

Palette-weakening and mark-weakening:

$$\textsc{palwk}\ \frac{\mathfrak{t} \mid \Gamma\ \mathsf{ctx} \quad \mathfrak{t} \mid \Gamma \vdash \mathscr{J}}{\mathfrak{t} \prec \Phi \mid \Gamma \vdash \mathscr{J}} \qquad \textsc{markwk}\ \frac{\mathfrak{t} \prec \Phi \mid \Gamma\ \mathsf{ctx} \quad \mathfrak{t} \mid \underline{\Gamma} \vdash \mathscr{J}}{\mathfrak{t} \prec \Phi \mid \Gamma \vdash \mathscr{J}}$$