Programming with Quantum-Controlled Quantum Channels

KENGO HIRATA, The University of Edinburgh, UK and Kyoto University, Japan
TAKESHI TSUKADA, Chiba University, Japan

1 Introduction

The superpositions of $|0\rangle$ and $|1\rangle$ superpose data, and a natural question is whether superpositions of programs are also possible. In recent years, affirmative results on this question have been obtained and actively investigated, mainly by the physics community. A typical example is *quantum SWITCH*[1] [Chiribella et al. 2013]. This operation takes a control qubit x, two quantum operations F and G, and an input y. When $x = |0\rangle$, the SWITCH behaves as $F\,(G\,y)$; when $x = |1\rangle$, it behaves as $G\,(F\,y)$; and when $x = \alpha_0\,|0\rangle + \alpha_1\,|1\rangle$ (with $\alpha_0 \neq 0 \neq \alpha_1$), it behaves as a certain "mixture" of $F\,(G\,y)$ and $G\,(F\,y)$. Roughly speaking, x controls the order in which F and G are applied. Notably, when the control qubit x is in a superposition of $|0\rangle$ and $|1\rangle$, the quantum SWITCH exhibits peculiar behaviour: whether F or G is applied first becomes ill-defined. This phenomenon is referred to as *no causal order* or *indefinite causal order* and has been actively studied as an important topic in quantum computation since the work of Oreshkov et al. [2012].

A programming language capable of expressing operations such as the quantum SWITCH is important not only for describing and analysing procedures involving the quantum SWITCH, but also for exploring as-yet-unknown but potentially interesting operations of a similar kind. This paper aims to provide such a quantum programming language.

A natural idea would be to add a construct for *quantum control*, say qif, as a quantum analogue of the classical if statement, which branches on a classical Boolean value. We expect the following behaviour for qif x then P else Q. When $x = |1\rangle$, it behaves like P; when $x = |0\rangle$, it behaves like Q; and when x is in a superposition of $|0\rangle$ and $|1\rangle$, it behaves like a "mixture" of P and Q. If such a construct is available, one might try to express the quantum SWITCH as

$$SWITCH(x, y, F, G) \quad = \quad \text{qif } x \text{ then } F\,(G\,y) \text{ else } G\,(F\,y). \tag{1}$$

While such a construct and the above "definition" might appear natural, no existing language accepts it. Existing languages with quantum conditional branching either restrict F and G to the class of unitary operations [Altenkirch and Grattage 2005; Bichsel et al. 2020; Sabry et al. 2018; Svore et al. 2018], or, while allowing general quantum operations, exhibit behaviour on the right-hand side above that differs from the quantum SWITCH [Bădescu and Panangaden 2015; Ying 2016].

This paper develops a new quantum programming language with quantum control, enabling the quantum SWITCH to be described in the aforementioned natural and intuitive way.

2 Existing Language with Quantum Control and Measurements

The *quantum SWITCH* [Chiribella et al. 2013] is an operation $SWITCH(x, y, F, G)$ that takes a *control qubit* x, an operand qubit y and two quantum operations F and G and returns a pair (x', y') of the control and operand qubits after the operation. When F and G have Kraus decompositions $\{H_i\}_{i\in I}$ and $\{K_j\}_{j\in J}$, respectively, the quantum operation $SWITCH(-, -, F, G)$ on two qubits is defined by the following Kraus decomposition:

$$SWITCH(-, -, F, G) = \left\{\ |0\rangle\langle 0| \otimes H_i K_j + |1\rangle\langle 1| \otimes K_j H_i\ \right\}_{(i,j)\in I\times J}. \tag{2}$$

[1]The quantum SWITCH is unrelated to the switch statement (as in C language).

Authors' Contact Information: Kengo Hirata, The University of Edinburgh, Edinburgh, UK and Kyoto University, Kyoto, Japan, k.hirata@sms.ed.ac.uk; Takeshi Tsukada, Chiba University, Chiba, Japan.

At first glance, this definition appears to depend on the choice of Kraus decompositions. Interestingly, however, it is in fact independent of that choice. That is, for different Kraus decompositions $\{H'_{i'}\}_{i'\in I'}$ and $\{K'_{j'}\}_{j'\in J'}$ of F and G, even when the resulting Kraus decomposition $\{|0\rangle\langle 0| \otimes H'_{i'}K'_{j'} + |1\rangle\langle 1| \otimes K'_{j'}H'_{i'}\}_{(i',j')\in I'\times J'}$ differs from the one above, the two Kraus decompositions define the same quantum operation. Hence, the quantum SWITCH is well-defined on quantum operations.

Bădescu and Panangaden [2015] and Ying [2016] proposed languages with quantum conditional branching for general quantum operations. We review their proposals and show that (1) does not define the quantum SWITCH [Chiribella et al. 2013] in their languages.

In their languages, the semantics of a program P is a Kraus decomposition $[\![P]\!] = \{H_i\}_{i\in I}$.[2] The semantics of the quantum conditional branching by Bădescu and Panangaden [2015] is given by

$$[\![\lambda xy.\,\mathsf{qif}\ x\ \mathsf{then}\ P(y)\ \mathsf{else}\ Q(y)]\!] \quad := \quad \left\{ |0\rangle\langle 0| \otimes \frac{H_i}{\sqrt{|J|}} + |1\rangle\langle 1| \otimes \frac{K_j}{\sqrt{|I|}} \right\}_{(i,j)\in I\times J}, \tag{3}$$

where $[\![\lambda y.P(y)]\!] = \{H_i\}_{i\in I}$ and $[\![\lambda y.Q(y)]\!] = \{K_j\}_{j\in J}$ and $|I|$ is the number of elements in I. The coefficient $1/\sqrt{|J|}$ is needed for normalisation because H_i appears $|J|$ times in the above decomposition (as $|0\rangle\langle 0| \otimes H_i$). The interpretation by Ying [2016] is similar but uses coefficients different from $1/\sqrt{|J|}$ and $1/\sqrt{|I|}$. As the precise values of the coefficients are irrelevant to the following discussion, we focus on the semantics in Eq. (3).

However, the "defining equation" of the quantum SWITCH (1) is not valid in their semantics. Assume that $[\![\lambda y.F(y)]\!] = \{H_i\}_{i\in I}$ and $[\![\lambda y.G(y)]\!] = \{K_j\}_{j\in J}$. Then $[\![F \circ G]\!] = \{H_iK_j\}_{(i,j)\in I\times J}$ and $[\![G \circ F]\!] = \{K_jH_i\}_{(i,j)\in I\times J}$, so

$$[\![\lambda xy.\,\mathsf{qif}\ x\ \mathsf{then}\ F(G(y))\ \mathsf{else}\ G(F(y))]\!] = \left\{ |0\rangle\langle 0| \otimes \frac{H_iK_j}{\sqrt{|I\times J|}} + |1\rangle\langle 1| \otimes \frac{K_{j'}H_{i'}}{\sqrt{|I\times J|}} \right\}_{(i,j,i',j')\in I\times J\times I\times J}$$

whereas $SWITCH(-,-,F,G) = \left\{|0\rangle\langle 0| \otimes H_iK_j + |1\rangle\langle 1| \otimes K_jH_i\right\}_{(i,j)\in I\times J}$.

Another issue is the ill-definedness of the semantics on completely positive maps. A completely positive map has many different Kraus decompositions: for example, $F = \{I_2\}$ and $F' = \{-I_2\}$ (where $I_2 \in \mathrm{Mat}_2(\mathbb{C})$ is the unit matrix) define the same completely positive map $F(\varrho) = F'(\varrho) = \varrho$. The semantics of a quantum conditional branching by Bădescu and Panangaden [2015] and Ying [2016] depends on the choice of Kraus decompositions, as they observed. For example,

$$[\![\lambda xy.\,\mathsf{qif}\ x\ \mathsf{then}\ F(y)\ \mathsf{else}\ y]\!] = \{I_2 \otimes I_2\} \quad \text{and} \quad [\![\lambda xy.\,\mathsf{qif}\ x\ \mathsf{then}\ F'(y)\ \mathsf{else}\ y]\!] = \{Z \otimes I_2\}.$$

3 Correspondence Problem

The key difference between the quantum SWITCH and the semantics in Bădescu and Panangaden [2015] and Ying [2016] is the existence of a *correspondence of indices* between then- and else-branches. In the quantum SWITCH (Eq. (2)), common indices (i, j) are used in both the then- and else-cases, whereas in the semantics of quantum conditional branching (Section 2), the indices (i, j) and (i', j') for the then- and else-branches are selected independently.

This correspondence issue is also necessary for defining a Kraus-decomposition-independent semantics. If F is a completely positive map and $\{H_i\}_{i\in I}$ is one Kraus decomposition of it, any Kraus decomposition $\{H'_j\}_{j\in I}$ can be represented as $\{H'_j := \sum_i u_{ij}H_i\}_{j\in I}$ for each $|I| \times |I|$ unitary map $U = (u_{ij})_{i,j\in I}$. The semantics of $\lambda xy.\,\mathsf{qif}\ x\ \mathsf{then}\ F(y)\ \mathsf{else}\ y$ can be calculated in the following

[2] Precisely speaking, the semantics by Ying [2016] is a function Ψ from a finite set I to operators. Here we identify such an operator-valued function with the Kraus decomposition $\{\Psi(i)\}_{i\in I}$.

two non-equivalent ways:

$$\left\{|0\rangle\langle 0| \otimes \frac{I_2}{\sqrt{|I|}} + |1\rangle\langle 1| \otimes H_i\right\}_{i\in I} \quad \not\simeq \quad \left\{|0\rangle\langle 0| \otimes \frac{I_2}{\sqrt{|I|}} + |1\rangle\langle 1| \otimes \sum_i u_{ij} H_i\right\}_{j\in I}$$

However, we need to consider not only the correspondence of indices, but also the *correspondence of implementations*. Let us now consider that we have two completely positive maps F and G and we have chosen Kraus decompositions of them with the same indices as $\{H_i\}_{i\in I}$ and $\{K_i\}_{i\in I}$. Even in this case, if we just replace H_i with H'_i as above, we end up getting non-equivalent Kraus decompositions. Instead, if we consider *coherently reindexing* the Kraus decompositions of $\{H_i\}_{i\in I}$ and $\{K_i\}_{i\in I}$, *i.e.*, take other Kraus decompositions of F and G to be $\{H'_j := \sum_i u_{ij} H_i\}_{j\in I}$ and $\{K'_j := \sum_i u_{ij} K_i\}_{j\in I}$ using a common unitary U, then the following two Kraus decompositions define the same completely positive map, because $|0\rangle\langle 0| \otimes H'_j + |1\rangle\langle 1| \otimes K'_j = \sum_i u_{ij}(|0\rangle\langle 0| \otimes H_j + |1\rangle\langle 1| \otimes K_j)$.

$$\{|0\rangle\langle 0| \otimes H_i + |1\rangle\langle 1| \otimes K_i\}_{i\in I} \quad \simeq \quad \left\{|0\rangle\langle 0| \otimes H'_j + |1\rangle\langle 1| \otimes K'_j\right\}_{j\in I}$$

For the case of quantum SWITCH (2), when different Kraus decompositions are chosen for F or G, then both branches change coherently as above because $\{H_i\}$ and $\{K_j\}$ appear in both:

$$\left\{|0\rangle\langle 0| \otimes H_i K_j + |1\rangle\langle 1| \otimes K_j H_i\right\}_{(i,j)\in I\times J} \quad \simeq \quad \left\{|0\rangle\langle 0| \otimes H'_i K'_j + |1\rangle\langle 1| \otimes K'_j H'_i\right\}_{(i,j)\in I\times J}$$

This is our explanation of why the quantum SWITCH defines a map that does not depend on the choice of Kraus operations. If each Kraus decomposition is regarded as roughly providing one implementation of the map, this phenomenon effectively amounts to requiring a correspondence between the implementations of the then- and else-clauses.

4 Correspondence Ensured by Linear Type

We now discuss our language with quantum conditional branching and measurement, in which the quantum SWITCH can be defined.

When designing a programming language with a unique natural semantics, we would like to ensure that every well-typed program has the canonical correspondence explained above. This can be achieved by using the resource-aware type system, namely *linear types*, which comes from linear logic [Girard 1987]. With this linearity, we can ensure that *each quantum operation appears exactly once in both branches*.

We divide the language into two sublanguages: a *classical sublanguage* in which measurements are available, and a *quantum sublanguage* in which quantum conditional branching is available. In the classical sublanguage, quantum branching is not directly permitted; in the quantum sublanguage, quantum measurement is not directly permitted. However, through the following form of interoperability, each sublanguage can indirectly make use of them. First, every program P in the quantum sublanguage can be used directly in the classical sublanguage. There is no embedding in the opposite direction. Nevertheless, we allow a variable defined in the classical sublanguage to be used within the quantum sublanguage. That is, an expression of the form $\mathsf{let}\, x = (\cdots \mathsf{meas} \cdots)\ \mathsf{in}\ [\mathsf{qif} \cdots \mathsf{then}\ (\cdots x \cdots)]$ is permitted, enabling us to handle measurements indirectly within the quantum sublanguage. For the reasons discussed above, every variable appearing in the quantum sublanguage must be used linearly, *i.e.*, must be used exactly once in the program. This linearity constraint also applies to variables defined in the classical sublanguage but used inside the quantum sublanguage.

Syntax. The syntax is defined in Fig. 1. We do not distinguish those two sublanguages.

The language provides three base types: the qubit type qbit, the boolean type bool, and the unit type unit. It also includes two type constructors, namely the tensor type $A \otimes B$ and the (linear)

$$
\begin{array}{llll}
\textit{Types} & A, B & ::= & \mathsf{qbit} \mid \mathsf{bool} \mid \mathsf{unit} \mid A \otimes B \mid A \multimap B \\
\textit{Terms} & M, N & ::= & x \mid () \mid M; N \mid \lambda x.M \mid M\,N \mid M \otimes N \\
 & & \mid & \mathsf{let}\, x = M \,\mathsf{in}\, N \mid \mathsf{let}\, x \otimes y = M \,\mathsf{in}\, N \mid \mathsf{let}\, x = \mathscr{F} \,\mathsf{in}\, M \\
 & & \mid & \mathsf{qif}\, M \,\mathsf{then}\, N_1 \,\mathsf{else}\, N_2 \mid \mathsf{if}\, M \,\mathsf{then}\, N_1 \,\mathsf{else}\, N_2 \\
 & & \mid & |1\rangle \mid U \mid \mathsf{meas} \mid \mathsf{true} \mid \mathsf{not} \\
\textit{Glob. Def.} & \mathcal{D} & ::= & \cdot \mid \mathcal{D}, \mathsf{def}\, \mathscr{F} = M
\end{array}
$$

Fig. 1. Syntax. Here, U is an n-qubit unitary operator, and $\mathscr{F}$ is a name for a globally defined term.

$$
\frac{;\, \Delta \vdash_{\mathbf{Hilb}} M : A}{\Gamma;\, \Delta \vdash_{\mathbf{CPM}} M : A} \qquad \frac{\mathit{Quantum}(A)}{;\, x : A \vdash_{\mathbf{Hilb}} x : A} \qquad \Gamma;\, x : A \vdash_{\mathbf{CPM}} x : A \qquad \frac{U : \mathbb{C}^{2^n} \longrightarrow \mathbb{C}^{2^n}, \mathrm{unitary}}{;\ \vdash_{\mathbf{Hilb}} U : \mathsf{qbit}^{\otimes n} \multimap \mathsf{qbit}^{\otimes n}}
$$

$$
\Gamma;\ \vdash_{\mathbf{CPM}} |1\rangle : \mathsf{qbit} \qquad \Gamma;\ \vdash_{\mathbf{CPM}} \mathsf{true} : \mathsf{bool} \qquad \Gamma;\ \vdash_{\mathbf{CPM}} \mathsf{meas} : \mathsf{qbit} \multimap \mathsf{bool}
$$

$$
\frac{;\, \Delta \vdash_{\mathbf{Hilb}} M : \mathsf{qbit} \qquad ;\, \Delta' \vdash_{\mathbf{Hilb}} N_1 : A \qquad ;\, \Delta' \vdash_{\mathbf{Hilb}} N_2 : A \qquad \mathit{FO}(A)}{;\, \Delta, \Delta' \vdash_{\mathbf{Hilb}} \mathsf{qif}\, M \,\mathsf{then}\, N_1 \,\mathsf{else}\, N_2 : \mathsf{qbit} \otimes A}
$$

Fig. 2. Selected typing rules. A program typed under $\vdash_{\mathbf{Hilb}}$ defines the quantum sublanguage, and a program typed under $\vdash_{\mathbf{CPM}}$ defines the classical sublanguage.

function type $A \multimap B$. A type A is *first order*, written $\mathit{FO}(A)$, when it has no function type $\multimap$. A type is *purely quantum* if it does not involve bool, and we write $\mathit{Quantum}(A)$ to indicate that A is purely quantum. The boolean type bool is not available in the quantum sublanguage.

Most term constructors are from the standard linear lambda calculus or from quantum λ-calculi as in Selinger and Valiron [2008]. The first and second lines are standard constructs in the linear lambda calculus. An exception is the binding $\mathsf{let}\, x = \mathscr{F} \,\mathsf{in}\, M$, which assigns a globally defined constant $\mathscr{F}$ to a local variable x. We distinguish between a variable x and a globally defined constant $\mathscr{F}$ because the latter is duplicable and discardable.[3] The third line contains two conditionals, qif and if. The final line enumerates the primitive constants: qubit initialization $|1\rangle$, all unitary operators U, measurement meas, and the boolean primitives true and not.

Type System. A *typing context* Δ (resp. Γ) is a finite sequence of $x : A$ (resp. $\mathscr{F} : A$).

This language features a linear type system with two derivation modes, $\vdash_{\mathbf{Hilb}}$ and $\vdash_{\mathbf{CPM}}$. The judgment $;\, \Delta \vdash_{\mathbf{Hilb}} M : A$ is for the quantum sublanguage. Both the context Δ and the type A must be purely quantum, preventing copying or discarding. The quantum conditional branching qif is available only in this fragment. The boolean type bool, the classical conditional branching if, and the measurement meas are not available. The judgment $\Gamma;\, \Delta \vdash_{\mathbf{CPM}} M : A$ is for the classical sublanguage. It augments the linear context Δ with a *non-linear* context Γ for globally defined constants that may be duplicated. Booleans, classical conditional branching and measurement are available in this fragment, but qif is not.

The typing rules appear in the Appendix. We explain some selected rules in Fig. 2.

Most rules are inherited from the standard linear lambda calculus or the quantum lambda calculus; some constructs are available in both judgments, whereas others are available only in **Hilb** or in **CPM**, following the policy described above.

[3] A constant can be seen as a value of type $!A$. The distinction between variables and globally defined constants can be avoided by introducing the !-type into the classical sublanguage. In this paper, however, we do not permit general !-types, simplifying the exposition by restricting duplication to globally defined constants.

The most important aspect is the mechanism enabling interoperability between **Hilb** and **CPM**. The *embedding rule* at the top left derives a **CPM**-judgment from a **Hilb**-judgment, bridging the two derivation modes. Conversely, the way in which **CPM** constructs (such as quantum measurement) become available inside **Hilb** is subtly hidden and thus requires attention. The underlying trick is that the type environment Δ does not record whether a variable x has been bound in **CPM** or in **Hilb**. Consequently, variables defined in **CPM** can be used in **Hilb**, which indirectly enables the use of quantum measurement and other **CPM** constructs within **Hilb**. For example, this allows us to write a term of the form

$$\mathsf{let}\ f = \big(\lambda q.\,\mathsf{if\ meas}\ q\ \mathsf{then}\ \mathsf{X}\,|1\rangle\ \mathsf{else}\ |1\rangle\big)\ \mathsf{in}\ \big(\,\mathsf{qif}\ x\ \mathsf{then}\ (\cdots f\cdots)\ \mathsf{else}\ (\cdots f\cdots)\big).$$

Variable rules in $\vdash_{\mathbf{Hilb}}$ and $\vdash_{\mathbf{CPM}}$ are distinct for restricting global definitions and booleans from the **Hilb** fragment. Each conditional is confined to its respective fragment: Quantum conditional qif is available only in **Hilb**, where strict linearity is enforced, while if is available only in **CPM**. The result types also differ: qif returns the conditional qubit (as a qubit is not discardable), whereas if does not return the controlling boolean (following the convention).

Example 4.1. Now, the program for quantum SWITCH can be written as follows:

$$\mathsf{let}\ f = (\cdots)\ \mathsf{in}\ \mathsf{let}\ g = (\cdots)\ \mathsf{in}\ \lambda x, y.\big(\,\mathsf{qif}\ x\ \mathsf{then}\ g(f(y))\ \mathsf{else}\ f(g(y))\big).$$

while allowing the variables f and g to be defined in the classical sublanguage where arbitrary completely positive maps, including measurements, can be defined. □

5 Categorical Semantics

Here, we provide our language with a natural categorical semantics. The fact that there is one clear canonical choice of the semantics demonstrates that the correspondence problem identified does not remain in our language with linear types.

Our model of the language consists of two compact closed categories: **Hilb**, the category of finite-dimensional Hilbert spaces and linear maps, for modelling the quantum sublanguage, and **CPM**, (the biproduct completion of) the category of completely positive maps, for the classical sublanguage. The semantics $[\![-]\!]_{\mathbf{Hilb}}$ and $[\![-]\!]_{\mathbf{CPM}}$ can be defined in the categories **Hilb** and **CPM**, respectively. These two semantics are often identified through the canonical functor **Hilb** $\longrightarrow$ **CPM** that maps a vector state $|\varphi\rangle$ to the corresponding density matrix $|\varphi\rangle\langle\varphi|$.

The categorical semantics of types A is given by $[\![A]\!]$ as follows.

$$[\![\mathsf{unit}]\!] = \mathbb{C},\quad [\![\mathsf{qbit}]\!] = \mathbb{C}^2,\quad [\![A \otimes B]\!] = [\![A]\!] \otimes [\![B]\!],\quad [\![A \multimap B]\!] = [\![A]\!]^* \otimes [\![B]\!],\quad [\![\mathsf{bool}]\!] = \mathbb{C} \oplus_{\mathbf{CPM}} \mathbb{C} \in \mathbf{CPM}.$$

Note that, since bool is considered only in the **CPM**-context, its semantics is defined only in **CPM**.

We define the categorical semantics of a derivation $;\ \Delta \vdash_{\mathbf{Hilb}} M\colon A$ as a map $[\![M]\!]_{\mathbf{Hilb}}\colon [\![\Delta]\!]_{\mathbf{Hilb}} \longrightarrow [\![A]\!]_{\mathbf{Hilb}}$, where $[\![\Delta]\!]$ is the tensor product of the interpretations of the types in Δ, following the convention of categorical semantics for linear lambda calculus. For **CPM**-terms, since a derivation may have a non-linear context Γ, we define the categorical semantics of a derivation $\Gamma;\ \Delta \vdash_{\mathbf{CPM}} M\colon A$ parameterised by the valuation $\varrho \in \prod_{(\mathscr{F}\colon B)\in\Gamma} [\![B]\!]_{\mathbf{CPM}}$ of non-linear context, *i.e.*, a map $[\![M]\!]_{\mathbf{CPM},\varrho}\colon [\![\Delta]\!]_{\mathbf{CPM}} \longrightarrow [\![A]\!]_{\mathbf{CPM}}$.

The semantics of each derivation is defined in the Appendix. The rule that promotes a **Hilb**-term into a **CPM**-term is interpreted via the embedding functor **Hilb** $\longrightarrow$ **CPM**. Note that $\iota\big([\![\Delta]\!]_{\mathbf{Hilb}}\big) = [\![\Delta]\!]_{\mathbf{CPM}}$ uses the fact that ι preserves the compact closed structures. The two conditional terms qif and if utilise the biproducts in each category. It is important that these terms are interpreted in different categories as the biproducts differ in **Hilb** and **CPM**.

Theorem 5.1. *The categorical semantics of SWITCH (1) coincides with the quantum SWITCH.* □

References

Thorsten Altenkirch and Jonathan Grattage. 2005. A Functional Quantum Programming Language. In *20th IEEE Symposium on Logic in Computer Science (LICS 2005), 26-29 June 2005, Chicago, IL, USA, Proceedings*. IEEE Computer Society, 249–258. doi:10.1109/LICS.2005.1

Costin Bădescu and Prakash Panangaden. 2015. Quantum Alternation: Prospects and Problems. *Electronic Proceedings in Theoretical Computer Science* 195 (nov 2015), 33–42. doi:10.4204/eptcs.195.3

Benjamin Bichsel, Maximilian Baader, Timon Gehr, and Martin T. Vechev. 2020. Silq: a high-level quantum language with safe uncomputation and intuitive semantics. In *Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2020, London, UK, June 15-20, 2020*, Alastair F. Donaldson and Emina Torlak (Eds.). ACM, 286–300. doi:10.1145/3385412.3386007

Giulio Chiribella, Giacomo Mauro D'Ariano, Paolo Perinotti, and Benoit Valiron. 2013. Quantum computations without definite causal structure. *Physical Review A* 88, 2 (aug 2013), 022318. doi:10.1103/physreva.88.022318

Jean-Yves Girard. 1987. Linear logic. *Theoretical Computer Science* 50, 1 (1987), 1–101. doi:10.1016/0304-3975(87)90045-4

Ognyan Oreshkov, Fabio Costa, and Časlav Brukner. 2012. Quantum correlations with no causal order. *Nature Communications* 3, 1 (Oct. 2012). doi:10.1038/ncomms2076

Amr Sabry, Benoît Valiron, and Juliana Kaizer Vizzotto. 2018. From Symmetric Pattern-Matching to Quantum Control. In *Foundations of Software Science and Computation Structures - 21st International Conference, FOSSACS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings (Lecture Notes in Computer Science, Vol. 10803)*, Christel Baier and Ugo Dal Lago (Eds.). Springer, 348–364. doi:10.1007/978-3-319-89366-2_19

Peter Selinger. 2004a. Towards a quantum programming language. *Mathematical Structures in Computer Science* 14, 4 (aug 2004), 527–586. doi:10.1017/s0960129504004256

Peter Selinger. 2004b. Towards a semantics for higher-order quantum computation. In *Proceedings of the 2nd International Workshop on Quantum Programming Languages*. 127–143.

Peter Selinger and Benoît Valiron. 2008. On a Fully Abstract Model for a Quantum Linear Functional Language. *Electronic Notes in Theoretical Computer Science* 210 (jul 2008), 123–137. doi:10.1016/j.entcs.2008.04.022

Krysta M. Svore, Alan Geller, Matthias Troyer, John Azariah, Christopher E. Granade, Bettina Heim, Vadym Kliuchnikov, Mariia Mykhailova, Andres Paz, and Martin Roetteler. 2018. Q#: Enabling Scalable Quantum Computing and Development with a High-level DSL. In *Proceedings of the Real World Domain Specific Languages Workshop, RWDSL@CGO 2018, Vienna, Austria, February 24-24, 2018*. ACM, 7:1–7:10. doi:10.1145/3183895.3183901

Mingsheng Ying. 2016. Foundations of Quantum Programming. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. doi:10.1016/C2014-0-02660-3

$$\frac{;\ \Delta \vdash_{\mathbf{Hilb}} M : A}{\Gamma;\ \Delta \vdash_{\mathbf{CPM}} M : A} \qquad \frac{\mathit{Quantum}(A)}{;\ x : A \vdash_{\mathbf{Hilb}} x : A} \qquad \Gamma;\ x : A \vdash_{\mathbf{CPM}} x : A \qquad \frac{\Gamma;\ \Delta, x : A \vdash_{\mathcal{E}} M : B}{\Gamma;\ \Delta \vdash_{\mathcal{E}} \lambda x.M : A \multimap B}$$

$$\frac{\Gamma;\ \Delta \vdash_{\mathcal{E}} M : \mathsf{unit} \qquad \Gamma;\ \Delta' \vdash_{\mathcal{E}} N : A}{\Gamma;\ \Delta, \Delta' \vdash_{\mathcal{E}} (M; N) : A} \qquad \frac{\Gamma;\ \Delta \vdash_{\mathcal{E}} M : A \multimap B \qquad \Gamma;\ \Delta' \vdash_{\mathcal{E}} N : A}{\Gamma;\ \Delta, \Delta' \vdash_{\mathcal{E}} M\,N : B}$$

$$\frac{\Gamma;\ \Delta \vdash_{\mathcal{E}} M : A \qquad \Gamma;\ \Delta' \vdash_{\mathcal{E}} N : B}{\Gamma;\ \Delta, \Delta' \vdash_{\mathcal{E}} M \otimes N : A \otimes B} \qquad \frac{\Gamma;\ \Delta \vdash_{\mathcal{E}} M : A \qquad \Gamma;\ \Delta', x : A \vdash_{\mathcal{E}} N : B}{\Gamma;\ \Delta, \Delta' \vdash_{\mathcal{E}} \mathsf{let}\, x = M \,\mathsf{in}\, N : B}$$

$$\frac{\Gamma;\ \Delta \vdash_{\mathcal{E}} M : A \otimes B \qquad \Gamma;\ \Delta', x : A, y : B \vdash_{\mathcal{E}} N : C}{\Gamma;\ \Delta, \Delta' \vdash_{\mathcal{E}} \mathsf{let}\, x \otimes y = M \,\mathsf{in}\, N : C} \qquad \frac{U : \mathbb{C}^{2^n} \longrightarrow \mathbb{C}^{2^n}, \text{unitary}}{;\ \vdash_{\mathbf{Hilb}} U : \mathsf{qbit}^{\otimes n} \multimap \mathsf{qbit}^{\otimes n}}$$

$$;\ \vdash_{\mathbf{Hilb}} () : \mathsf{unit} \qquad \Gamma;\ \vdash_{\mathbf{CPM}} |1\rangle : \mathsf{qbit} \qquad \Gamma;\ \vdash_{\mathbf{CPM}} \mathsf{true} : \mathsf{bool}$$

$$\Gamma;\ \vdash_{\mathbf{CPM}} \mathsf{meas} : \mathsf{qbit} \multimap \mathsf{bool} \qquad \Gamma;\ \vdash_{\mathbf{CPM}} \mathsf{not} : \mathsf{bool} \multimap \mathsf{bool}$$

$$\frac{;\ \Delta \vdash_{\mathbf{Hilb}} M : \mathsf{qbit} \qquad ;\ \Delta' \vdash_{\mathbf{Hilb}} N_1 : A \qquad ;\ \Delta' \vdash_{\mathbf{Hilb}} N_2 : A \qquad \mathit{FO}(A)}{;\ \Delta, \Delta' \vdash_{\mathbf{Hilb}} \mathsf{qif}\, M \,\mathsf{then}\, N_1 \,\mathsf{else}\, N_2 : \mathsf{qbit} \otimes A}$$

$$\frac{\Gamma;\ \Delta \vdash_{\mathbf{CPM}} M : \mathsf{bool} \qquad \Gamma;\ \Delta' \vdash_{\mathbf{CPM}} N_1 : A \qquad \Gamma;\ \Delta' \vdash_{\mathbf{CPM}} N_2 : A \qquad \mathit{FO}(A)}{\Gamma;\ \Delta, \Delta' \vdash_{\mathbf{CPM}} \mathsf{if}\, M \,\mathsf{then}\, N_1 \,\mathsf{else}\, N_2 : A}$$

$$\frac{(\mathscr{F} : A) \in \Gamma \qquad \Gamma;\ \Delta, x : A \vdash_{\mathbf{CPM}} M : B}{\Gamma;\ \Delta \vdash_{\mathbf{CPM}} \mathsf{let}\, x = \mathscr{F} \,\mathsf{in}\, M : B} \qquad \frac{\vdash \mathcal{D} : \Gamma \qquad \Gamma;\ \vdash_{\mathbf{CPM}} M : A}{\vdash (\mathcal{D}, \mathsf{def}\, \mathscr{F} = M) : (\Gamma, \mathscr{F} : A)}$$

Fig. 3. Typing rules. The rules labelled $\vdash_{\mathcal{E}}$ are shared between $\vdash_{\mathbf{Hilb}}$ and $\vdash_{\mathbf{CPM}}$. We omit the exchange rule, which permutes the order of elements in Δ.

A Syntax and Type System

The typing rules appear in Fig. 3. Most rules are inherited from the standard linear lambda calculus or the quantum lambda calculus; some constructs are available in both judgments, whereas others are available only in **Hilb** or in **CPM**, following the policy described above.

The most important aspect is the mechanism enabling interoperability between **Hilb** and **CPM**. The *embedding rule* at the top left derives a **CPM**-judgment from a **Hilb**-judgment, bridging the two derivation modes. Conversely, the way in which **CPM** constructs (such as quantum measurement) become available inside **Hilb** is subtly hidden and thus requires attention. The underlying trick is that the type environment Δ does not record whether a variable x has been bound in **CPM** or in **Hilb**. Consequently, variables defined in **CPM** can be used in **Hilb**, which indirectly enables the use of quantum measurement and other **CPM** constructs within **Hilb**. For example, this allows us to write a term of the form

$$\mathsf{let}\, d = \big(\lambda q.\, \mathsf{if\ meas}\ q\ \mathsf{then}\ \mathsf{X}\,|1\rangle\ \mathsf{else}\ |1\rangle\big)\ \mathsf{in}\ \big(\,\mathsf{qif}\ x\ \mathsf{then}\ (\cdots d \cdots)\ \mathsf{else}\ (\cdots d \cdots)\big).$$

We discuss other aspects. Variable rules in $\vdash_{\mathbf{Hilb}}$ and $\vdash_{\mathbf{CPM}}$ are distinct for restricting global definitions and booleans from the **Hilb** fragment. Each conditional is confined to its respective fragment: qif is available only in **Hilb**, where strict linearity is enforced, while if is available only in **CPM**. The result types also differ: qif returns the conditional qubit (as a qubit is not discardable), whereas if does not return the controlling boolean (following the convention). We forbid function types to occur in the conditional results by a technical reason. Global constants $\mathscr{F}$ in the non-linear context Γ can be referenced only in **CPM**. We can bind $\mathscr{F}$ to a local variable x, and once bound, x

becomes linear even though $\mathscr{F}$ remains reusable. Any closed term M can be promoted to a global definition.

B Categorical Semantics

We briefly review the definitions and properties of two specific compact closed categories, **Hilb** and **CPM**: for details, see, e.g., [Selinger 2004a,b; Selinger and Valiron 2008].

The category **Hilb** is the category of finite dimensional $\mathbb{C}$-vector spaces and linear maps. More concretely, the objects of the category are natural numbers, each object n representing the n-dimensional $\mathbb{C}$-vector space $\mathbb{C}^n$ spanned by $|0\rangle, \ldots, |n-1\rangle$, and the morphisms are $\mathbb{C}$-linear maps $f\colon \mathbb{C}^n \longrightarrow \mathbb{C}^m$. The category **Hilb** has a standard monoidal product corresponding to the tensor product of the vector spaces, *i.e.*, $n \otimes m := n \times m$. The basis of $n \otimes m$ can be given by the product of basis as $|00\rangle, |01\rangle, \ldots, |0(m-1)\rangle, |10\rangle, \ldots, |(n-1)(m-1)\rangle$. The tensor unit is 1. The dual object n^* of n is given by the dual space, which is again n. The unit and counit morphism are defined by

$$\eta_n\colon 1 \longrightarrow n \otimes n^*;\ \alpha \longmapsto \alpha \textstyle\sum_{i=0}^{n-1} |i\rangle \otimes |i\rangle \qquad \varepsilon_n\colon n^* \otimes n \longrightarrow 1;\ \sum_{i,j=0}^{n-1} \alpha_{ij}\, |i\rangle \otimes |j\rangle \longmapsto \textstyle\sum_{i=0}^{n-1} \alpha_{ii}.$$

This category **Hilb** also admits biproducts defined by $n \oplus_{\mathbf{Hilb}} m := n + m$. In particular, $1 \oplus_{\mathbf{Hilb}} 1 = 2$ corresponds to $\mathbb{C}^2$, in which we interprets the qubit type.

The other category **CPM** of completely positive maps is defined as follows. An object of **CPM** is a finite sequence $\vec{n} = (n_0, \ldots, n_{k-1})$ of natural numbers. A morphism $f \in \mathbf{CPM}(\vec{n}, \vec{m})$ is a matrix of completely positive maps $(f_{ij}\colon \mathrm{Mat}_{n_i}(\mathbb{C}) \longrightarrow \mathrm{Mat}_{m_j}(\mathbb{C}))$. The monoidal product in **CPM** is defined by $(n_0, \ldots, n_{k-1}) \otimes (m_0, \ldots, m_{\ell-1}) := (n_0 m_0, \ldots, n_0 m_{\ell-1}, n_1 m_0, \ldots, n_{k-1} m_{\ell-1})$. The monoidal unit is (1), and the dual $\vec{n}^*$ of $\vec{n}$ is again $\vec{n}$. The unit and counit morphisms are defined by

$$(\eta_n)_{1,\ell}\colon r \longmapsto r \textstyle\sum_{i,j=0}^{n_\ell - 1} |i\rangle\langle j| \otimes |i\rangle\langle j| \qquad (\varepsilon_n)_{\ell,1}\colon \sum_{i,j,i',j'=0}^{n_\ell-1} \alpha_{iji'j'}\, |i\rangle\langle j| \otimes |i'\rangle\langle j'| \longmapsto \textstyle\sum_{ij} \alpha_{ijij}.$$

This category **CPM** also admits biproducs defined by the concatenation of vectors $\vec{n} \oplus_{\mathbf{CPM}} \vec{m} = \vec{n}\vec{m}$. The biproduct of the monoidal unit $(1) \oplus_{\mathbf{CPM}} (1) = (1, 1)$ will be the semantics of booleans.

There are several important maps in this category. (a) For each object $\vec{n}$ in **CPM**, there is a *discarding map* $\overline{\top}_{\vec{n}}\colon \vec{n} \longrightarrow (1)$ defined by the componentwise trace operator $(\overline{\top}_{\vec{n}})_{\ell,1}\colon \sum_{ij=0}^{n_\ell-1} \alpha_{ij}\, |i\rangle\langle j| \longrightarrow \sum_i \alpha_{ii}$. (b) Each object $\vec{n} = (n_0, \ldots, n_{k-1})$ in **CPM** is a retraction of (N) where $N := \sum_{\ell=0}^{k-1} n_\ell$. The *section* $s_n\colon \vec{n} \longrightarrow (N)$ is the componentwise embedding of $n_\ell \times n_\ell$ matrices to $N \times N$ matrices defined by $|i\rangle\langle j| \mapsto |N_\ell + i\rangle\langle N_\ell + j|$ where $N_\ell := n_0 + \cdots + n_{\ell-1}$. The corresponding *retraction* map $r_{\vec{n}}\colon (N) \longrightarrow \vec{n}$ is the projection to the corresponding element of the matrix. For the case of $\vec{n} = (1, 1)$, we define $m := r_{(1,1)}$ and call it the *measurement map*.

There is a canonical functor in one direction $\iota\colon \mathbf{Hilb} \longrightarrow \mathbf{CPM}$, which we call the *embedding*. This functor is defined by $\iota(n) := (n)$ and $\iota(f)(\rho) := f\rho f^\dagger$. This functor preserves the compact closed structure on the nose. Equially important is that the biproduct is not preserved by this embedding.

The semantics of derivation is defined in Fig. 4.

$$[\![\ ;\Delta \vdash_{\mathbf{Hilb}} M\colon A]\!]_{\mathbf{CPM}} = \iota[\![\ ;\Delta \vdash_{\mathbf{Hilb}} M\colon A]\!]_{\mathbf{Hilb}} \qquad [\![x\colon A]\!] = \mathrm{id}_{[\![A]\!]} \qquad [\![M;N]\!] = [\![M]\!] \otimes [\![N]\!]$$

$$[\![\lambda x^A.M]\!] = \Lambda_A[\![M]\!] \qquad [\![M\,N]\!] = \mathrm{ev}([\![M]\!] \otimes [\![N]\!]) \qquad [\![M \otimes N]\!] = [\![M]\!] \otimes [\![N]\!]$$

$$[\![\mathsf{let}\ x = M\ \mathsf{in}\ N]\!] = [\![N]\!]\sigma_{A,\Delta'}([\![M]\!] \otimes \mathrm{id}_{\Delta'}) \qquad [\![\mathsf{let}\ x \otimes y = M\ \mathsf{in}\ N]\!] = [\![N]\!]\sigma_{AB,\Delta'}([\![M]\!] \otimes \mathrm{id}_{\Delta'})$$

$$[\![()]\!] = \mathrm{id}_1 \qquad [\![U]\!] = U \qquad [\![|1\rangle]\!]_{\mathbf{CPM}} = |1\rangle\langle 1| \qquad [\![\mathsf{meas}]\!]_{\mathbf{CPM}} = m \qquad [\![\mathsf{true}]\!]_{\mathbf{CPM}} = \mathrm{inr}_{I,I}$$

$$[\![\mathsf{not}]\!] = \mathrm{swap}_{\oplus_{\mathbf{CPM}}} \qquad [\![\mathsf{qif}\ M\ \mathsf{then}\ N_1\ \mathsf{else}\ N_2]\!]_{\mathbf{Hilb}} = d^{-1}_{\oplus_{\mathbf{Hilb}}}([\![N_2]\!] \oplus_{\mathbf{Hilb}} [\![N_1]\!])d_{\oplus_{\mathbf{Hilb}}}([\![M]\!] \otimes \mathrm{id}_{\Delta'})$$

$$[\![\mathsf{if}\ M\ \mathsf{then}\ N_1\ \mathsf{else}\ N_2]\!]_{\mathbf{CPM}} = (\bar{\top}_{(1,1)} \otimes \mathrm{id}_A)d^{-1}_{\oplus_{\mathbf{CPM}}}([\![N_2]\!] \oplus_{\mathbf{CPM}} [\![N_1]\!])d_{\oplus_{\mathbf{CPM}}}([\![M]\!] \otimes \mathrm{id}_{\Delta'})$$

$$[\![\mathsf{let}\ f = \mathscr{F}\ \mathsf{in}\ M]\!]_\varrho = [\![M]\!] \circ (\mathrm{id}_\Delta \otimes \varrho(\mathscr{F})) \qquad [\![\mathscr{D}, \mathsf{def}\ \mathscr{F}(x^A) = M]\!] = [\![\mathscr{D}]\!] \cup (\mathscr{F} \mapsto [\![M]\!])$$

Fig. 4. Categorical semantics for Qif. Here, $\mathrm{ev}\colon (A \multimap B)\otimes B \to B$ is the evaluation map, $\sigma_{A,B}\colon A\otimes B \cong B\otimes A$ is the braiding, $d_{\oplus_\mathcal{E}}\colon A\otimes(1\oplus_\mathcal{E} 1) \cong A\oplus_\mathcal{E} A$ is the distribution isomorphism, $m\colon (2) \to (1,1)$ is the measurement map, $\bar{\top}\colon I\oplus I \to I$ is the discard map, and $\Lambda_Y(-)$ maps a morphism $f\colon X\otimes Y \to Z$ to a morphism $X \to Y \multimap Z$.