A History and Reversibility for Quantum Programming Language QML

Nely Plata César[1], J. Raymundo Marcial Romero[1], and J. Antonio Hernández Servín[1]

Facultad de Ingeniería
Universidad Autónoma del Estado de México

Abstract. We developed an operational model that incorporates a history track for the quantum programming language QML, considering classical and quantum data, and omitting measurements. With the history, the reversibility can be explicitly and naturally applied from the proposed rules. The language is worked first with classical data and extrapolated to quantum data.

Keywords: Functional quantum programming QML · History · Reversibility.

1 Introduction

There are several quantum programming languages, based on the imperative, functional or other paradigms. Only a few are described below: QML [1], nQML [2] and Quantum Lambda Calculus (λ_q) [3] with operational semantics and implicit reversibility, and QML and nQML have denotational semantic and implementation, but only λ_q have a history track and measurements.

Some languages based on imperative paradigm are: QCL [4], LanQ [5] and qGCL [6], all three with measurements; QCL and LanQ have an implementation and implicit reversibility, but only LanQ and qGCL have operational semantic. And based on other paradigms are pQCL [7], QPAlg [8], CQP [9,10], etc [11–13].

We will focus on QML developed by Grattage y Altenkirch, is a first order language and has finite data-types, the semantics of QML assigns to every well-typed program a quantum circuit, the computations have the property of being reversible and irreversible. Also implements the notion of quantum control *if*, which would lead to implicit measurements and quantum collapse [1, 7].

Several authors have considered this language, is the case of Díaz-Caro, Arrighi, Gadella y Grattage who give an idea to implement a history track and quantum measurements [14–16]. Altenkirch, Grattage, Vizzotto and Sabry not define operational semantics in terms of circuits and develop a sound and complete equational theory omitting measurements, they work QML subdividing the classical and quantum part of the language [17]. Also Lampis, Ginis, Papakyriakou and Papaspyrou define nQML with quantum data and control, an interpreter written in Haskell and being a more intuitive language [2].

Quantum languages must consider the implicit property of reversibility, if the operations are defined as unit matrix, and therefore, if the previous immediate matrix is applied to actual state, it can be returned to the previous step, and successively.

The contribution of this article is incorporate a history track to the operational model, the track will store the operations performed in a program. With this, reversibility can be explicitly executed. The above is added to QML language defined by Altenkirch, Grattage, Vizzotto and Sabry [17]. Our research guide is proposed by Díaz-Caro, Arrigui, Gadella and Grattage, who work a similar area but in the quantum lambda calculus language [14]. The resulting operational model considers classical and quantum data (superpositions), preserving orthogonality in programs that consider *if*. The rules to apply reversibility based on the history track are also defined.

2 Preliminaries

2.1 Quantum computing

There are four principles or postulates that define the behavior of quantum computing. The pure states are unitary vectors, which store in memory the information used. Evolution, this indicates how change from one state to another, for which unit matrices are applied, they have implicit instructions that the computer will carry out and that will be applied to the state to evolve. Measurement or observation focuses on the existence of an observer who, upon seeing the state of the system, will determine with what probability a result can happen, implying a collapse of the system. Composite systems, will be in charge of defining states conformed by states of different systems and how they interact among them [18, 19].

2.2 Quantum programming language QML

QML is a functional quantum language introduced by Grattage and Altenkirch, they apply quantum properties and develop a denotational and operational semantics [1, 20]. Subsequently, they research and develop a sound and complete equational theory, omitting recursive types and measurements [17]. This latest research will be retaken, which describes its syntax.

(Variables)	$x, y, \ldots \in Vars$	(Terms) $t, u ::= x \mid () \mid (t, u)$
(Prob. amplitudes)	$\kappa, \iota, \ldots \in \mathbb{C}$	$\mid$ **let** $p = t$ **in** $u \mid false$
(Patterns)	$p, q ::= x \mid (x, y)$	$\mid true \mid \mathbf{0} \mid \kappa * t \mid t + u$
		$\mid$ **if**$^\circ$ t **then** u **else** u'

From the syntax you can form conventional programs such as tuples, let, if, among others, and in turn append terms with a probability amplitude κ, that is, have a probability value that they happen, and also define superposition $t + u$, it means than a term can be in t and u at the same time.

The types are given by the grammar: $\sigma = \mathcal{Q}_1 | \mathcal{Q}_2 | \sigma \otimes \tau$, where $\mathcal{Q}_1$ is the type (), which carry no information and $\mathcal{Q}_2$ corresponds to qubits (0 and 1). The

types are finite and non-recursive, and the only types that can be represented are those in the collection of qubits.

Typing contexts (Γ, Δ) are given by: $\Gamma, x : \sigma = \bullet | \Gamma, x : \sigma$, where $\bullet$ stands for the empty context. We context correspond to functions from a finite set of variables to types. For this case, we assume that every variable appears at most once. To maps pairs of contexts to contexts, the $\otimes$ operator is incorporated, with the following operation:

$\Gamma, x : \sigma \otimes \Delta, x : \sigma = (\Gamma \otimes \Delta), x : \sigma$
$\Gamma, x : \sigma \otimes \Delta \quad = (\Gamma \otimes \Delta), x : \sigma$ si $x \notin \text{dom}\Delta$
$\bullet \otimes \Delta \quad = \Delta$

Interpret a judgement $\Gamma \vdash t : \sigma$ as a function is $[\![\Gamma \vdash t : \sigma]\!] \in [\![\Gamma]\!] \to [\![\sigma]\!]$, where, given a typing contexts, it returns a type of the collection of qubits.

To define well-formed programs $\Gamma \vdash t : \sigma$, the rules can be observed in Table 1. For example: $\Gamma \otimes \Delta \vdash \textbf{if}^\circ\ c\ \textbf{then}\ t\ \textbf{else}\ u : \sigma$, requires that $\Gamma \vdash c : Q_2$ and $\Delta \vdash t, u : \sigma$, that is, c has a value of qubit, and t and u are the same type.

$$\frac{}{x : \sigma \vdash x : \sigma}\text{var} \qquad \frac{\Gamma \vdash t : \sigma \qquad \Delta, x : \sigma \vdash t : \tau}{\Gamma \otimes \Delta \vdash \textbf{let}\ x = t\ \textbf{in}\ u : \tau}\text{let} \qquad \frac{\Gamma \vdash c : Q_2 \qquad \Delta \vdash t, u : \sigma}{\Gamma \otimes \Delta \vdash \textbf{if}^\circ\ c\ \textbf{then}\ t\ \textbf{else}\ u : \sigma}\textbf{if}^\circ$$

$$\frac{\Gamma \vdash t : \sigma \qquad \Delta \vdash u : \tau}{\Gamma \otimes \Delta \vdash (t, u) : \sigma \otimes \tau} \otimes\text{intro} \qquad \frac{}{\bullet \vdash \mathit{false} : Q_2}\text{f-intro} \qquad \frac{}{\bullet \vdash \mathit{true} : Q_2}\text{t-intro}$$

$$\frac{\Gamma \vdash t : \sigma \otimes \tau \qquad \Delta, x : \sigma, y : \tau \vdash u : \rho}{\Gamma \otimes \Delta \vdash \textbf{let}\ (x, y) = t\ \textbf{in}\ u : \rho} \otimes\text{-elim} \qquad \frac{\Gamma, x : Q_1 \vdash t : \sigma}{\Gamma \vdash t : \sigma}\text{wk-unit} \qquad \frac{}{\bullet \vdash () : Q_1}\text{unit}$$

Table 1: Typing classical terms (Source: [17], p. 29).

Semantically the types are described as $[\![\mathcal{Q}_1]\!] = \{0\}$, $[\![\mathcal{Q}_2]\!] = \{0, 1\}$ and $[\![\sigma \otimes \tau]\!] = [\![\sigma]\!] \times [\![\tau]\!]$, where tensor product returns a types tuple. This information will be taken as basis for QML; next, a brief introduction about the history and reversibility.

2.3 Quantum history

The quantum computation implicitly has the property of reversibility, for this, several authors have made contributions in classical and quantum computation [21–24]. For example Van Tonder proposes in quantum lambda calculus a history stack that stores the executed operations in a program [3]. For reversibility, we propose that is enough sufficient to execute operations in reverse order and return to initial value.

Our proposal will be described below, which is based on modifying the operational model, to which are added inverse functions that allow storing the operation executed in a certain step; also the functions used in the model are modified and appended.

3 Quantum Data and Control

In this section the history stack is added to operational model, as well as the functions and rules to execute a program and reversibility. The base model is

located at Appendix 1. Our operational model stores the inverse operations executed during a certain derivation. The rules are observed in Table 2.

$$
\begin{aligned}
[\![\bullet \vdash () : \mathcal{Q}_1]\!]^{\mathrm{Q}} &= (\boldsymbol{const\ 0})^{-\mathbf{1}}_{-}; const\ 0 \\
[\![\bullet \vdash false : \mathcal{Q}_2]\!]^{\mathrm{Q}} &= (\boldsymbol{const\ 0})^{-\mathbf{1}}_{-}; const\ 0 \\
[\![\bullet \vdash true : \mathcal{Q}_2]\!]^{\mathrm{Q}} &= (\boldsymbol{const\ 1})^{-\mathbf{1}}_{-}; const\ 1 \\
[\![x : \sigma \vdash x : \sigma]\!]^{\mathrm{Q}} &= \boldsymbol{id}^{-\mathbf{1}}_{+-}; id_{+} \\
[\![\Gamma \otimes \Delta \vdash \mathbf{let}\ x = t\ \mathbf{in}\ u : \sigma]\!]^{\mathrm{Q}} &= g^{*} \circ (f^{*} \times \boldsymbol{id}^{*-\mathbf{1}}_{-}; id^{*}) \circ (\boldsymbol{\delta}^{-\mathbf{1}}_{\Gamma,\Delta-}; \delta_{\Gamma,\Delta}) \\
&\quad \text{where} \quad f = [\![\Gamma \vdash t : \sigma]\!]^{\mathrm{Q}} \\
&\qquad\quad\ \ g = [\![\Delta,\ x : \sigma \vdash u : \tau]\!]^{\mathrm{Q}} \\
[\![\Gamma \otimes \Delta \vdash (t,u) : \sigma \otimes \tau]\!]^{\mathrm{Q}} &= (f^{*} \times g^{*}) \circ ((\boldsymbol{\delta}_{\Gamma,\Delta})^{-\mathbf{1}}_{-}; \delta_{\Gamma,\Delta}) \\
&\quad \text{where} \quad f = [\![\Gamma \vdash t : \sigma]\!]^{\mathrm{Q}} \\
&\qquad\quad\ \ g = [\![\Delta \vdash u : \tau]\!]^{\mathrm{Q}} \\
[\![\Gamma \otimes \Delta \vdash \mathbf{let}\ (x,y) = t\ \mathbf{in}\ u : \rho]\!]^{\mathrm{Q}} &= g^{*} \circ (f^{*} \times \boldsymbol{id}^{*-\mathbf{1}}_{-}; id^{*}) \circ (\boldsymbol{\delta}^{-\mathbf{1}}_{\Gamma,\Delta-}; \delta_{\Gamma,\Delta}) \\
&\quad \text{where} \quad f = [\![\Gamma \vdash t : \sigma \otimes \sigma]\!]^{\mathrm{Q}} \\
&\qquad\quad\ \ g = [\![\Delta,\ x : \sigma,\ y : \tau \vdash u : \rho]\!]^{\mathrm{Q}} \\
[\![\Gamma \otimes \Delta \vdash \mathbf{if}^{\circ}\ c\ \mathbf{then}\ t\ \mathbf{else}\ u : \sigma]\!]^{\mathrm{Q}} &= (g^{*} | h^{*}) \circ (f^{*} \times \boldsymbol{id}^{*-\mathbf{1}}_{-}; id^{*}) \\
&\qquad\qquad \circ ((\boldsymbol{\delta}_{\Gamma,\Delta})^{-\mathbf{1}}_{-}; \delta_{\Gamma,\Delta}) \\
&\quad \text{where} \quad f = [\![\Gamma \vdash c : \mathcal{Q}_2]\!] \\
&\qquad\quad\ \ g = [\![\Delta \vdash t : \sigma]\!] \\
&\qquad\quad\ \ h = [\![\Delta \vdash u : \sigma]\!] \\
[\![\Gamma \vdash t : \sigma]\!]^{\mathrm{Q}} &= (f \times \boldsymbol{id}^{+-\mathbf{1}}_{-}; id^{+}) \\
&\quad \text{where} \quad f = [\![\Gamma,\ x : \mathcal{Q}_1 \vdash t : \sigma]\!]^{\mathrm{Q}}
\end{aligned}
$$

Table 2: Operational rules.

Rules like $[\![\bullet \vdash false : \mathcal{Q}_2]\!]^{\mathrm{Q}} = (\boldsymbol{const\ 0})^{-\mathbf{1}}_{-}; const\ 0$, that is, if you have a $false$ term, then the function $const\ 0$ is applied, and its inverse is stored. This is similar for all statements.

3.1 Auxiliary functions

Considering the auxiliary functions (Appendix 2) defined in the operational model (with quantum operations), the authors extend the functions using a Kleisli Triple, where each derivation returns a complex vector [17, 25, 26].

We develop the inverse functions $(.^{-1})$ and the r function applied to all, such that later allow the reversibility. The r function is responsible for storing the argument received respectively.

In quantum environment the states are complex unit vectors and can be in superposition, to achieve this, Kleisli Triple are used [17, 25]. With this, we

interpret judgements $\Gamma \vdash t : \sigma$ as a function $[\![\Gamma]\!] \to [\![\sigma]\!]^Q$, where $[\![\sigma]\!]^Q = [\![\sigma]\!] \to \mathbb{C}$ represents complex vectors over the base set $[\![\sigma]\!]$; $[\![\sigma]\!]^Q$ is denoted as $V[\![\sigma]\!]$. Remember that σ is a type, defined from $\sigma = \mathcal{Q}_1|\mathcal{Q}_2|\sigma \otimes \tau$, interpreted as: $[\![\mathcal{Q}_2]\!] = \{0, 1\}$ (associated with qubits) and $[\![\mathcal{Q}_1]\!]$ (no information).

Definition 1 (Triple Kleisli). *A Triple Kleisli on a category $\mathcal{C}$, is defined by tuple $(V, return, \_^*)$, where:*

- $V : \mathcal{C} \to \mathcal{C}$
- $return : S \to V\ S$, *for* $S \in \mathcal{C}$
- $f^* : V\ S \to V\ T$, *for* $f : S \to V\ T$

To work with complex vectors, it is required modify the functions of the operational model by implementing Kleisli Triple. Every function of type $S \to T$ turns to a function of type $S \to V\ T$ using the $return$ function; this must be done for all the classical auxiliary functions (Appendix 2). For example, consider the function $const\ a : [\![Q_1]\!] \to S$, then, adjusting Kleisli Triple, you get: $return \circ const\ a : [\![Q_1]\!] \to V\ S$. We use $\overline{const}\ a = return \circ const\ a$.

Now, we define an r function which allows to carry the input arguments and output for any auxiliary function f.

Let f be any auxiliary function where classically $f : [\![\sigma]\!] \to [\![\tau]\!]$, such that $f(a) = b$. Then, the function r has type $[\![\sigma]\!] \to [\![\tau]\!]^Q$, e.g. $r : [\![\sigma]\!] \to [\![\tau]\!] \to \mathbb{C}$ (by definition of $[\![\tau]\!]^Q$), defined as $r(a\ f(a)) = \alpha$, $\alpha \in \mathbb{C}$, where $(r\ a\ f(a)) = 1.0$ if $a = f(a)$ and 0.0 otherwise. So $\overline{f}\ a = r(a\ f(a))$. Our function r has a call-by-name evaluation mechanism, so its argument will not be evaluated until needed.

The following functions are part of our contribution. For each auxiliary functions f we consider r to define the inverse f^{-1}, the lift f^* (associated with a Kleisli Triple) and its inverse f^{*-1}. Consider $S \in [\![Q_2]\!]$, the modified functions are:

- $id : S \to (S \to S \to \mathbb{C})$, this function returns the argument it receives, and is defined as: $id(a) = (r\ a\ a)$.
  $id^* : (S \to V\ S) \to (S \to S \to \mathbb{C})$ defined as: $id^*(r\ a'\ a) = (r\ a'\ a)$
  $id^{-1} : (S \to V\ S) \to S$ defined as: $id^{-1}(r\ a'\ a) = a$
  $id^{*-1} : (S \to V\ S) \to (S \to S \to \mathbb{C})$ defined as: $id^{*-1}(r\ a'\ a) = (r\ a'\ a)$
- $id^+ : S \to (S \to [\![Q_1]\!] \to \mathbb{C})$, concatenates the received argument with the value 0 and is defined as: $id^+(a) = (r\ a\ 0, r\ 0\ a)$
  $id^{+*} : (S \to V\ S) \to (S \to [\![Q_1]\!] \to \mathbb{C}) \times (S \to S \to \mathbb{C})$ defined as: $id^{+*}(r\ a'\ a) = (r\ a\ 0, r\ 0\ a)$, and the other options are defined as:
  $id^{+-1}(r\ a'\ b, r\ a'\ a) = \begin{cases} a, & \text{if } b{=}0 \\ b, & \text{if } b{=}1 \end{cases}$
  $id^{+*-1}(r\ a'\ 0, r\ a'\ a) = (r\ a'\ a)$
- $id_+ : ([\![Q_1]\!] \times S) \to (S \to S \to \mathbb{C})$ defined as: $id_+(0, a) = (r\ a\ a)$
  $id_+^* : (S \to V\ [\![Q_1]\!]) \times (S \to V\ S) \to (S \to S \to \mathbb{C})$ defined with the conditions: $id_+^*(r\ a'\ 0, r\ a'\ a) = (r\ a'\ a)$ and $id_+^*(r\ a''\ 1, r\ a'\ a) = (r\ a'\ 1)$
  $id_+^{-1}(r\ a'\ a) = (0, a)$
  $id_+^{*-1}(r\ a'\ a) = (r\ a'\ a, r\ 0\ a)$

– $const\ a : [\![Q_1]\!] \to ([\![Q_1]\!] \to S \to \mathbb{C})$ is defined as: $const\ a(0) = (r\ 0\ a)$
$const\ a^* : (S \to V\ S) \to ([\![Q_1]\!] \to S \to \mathbb{C})$ defined as: $const\ a^*(r\ a'\ a'') = (r\ 0\ a)$, and its inverse as:
$const\ 0^{-1}(r\ a'\ a) = 0$, and $const\ 1^{-1}(r\ a'\ a) = \begin{cases} 1, & \text{if } a{=}0 \\ 0, & \text{if } a{=}1 \end{cases}$
$const\ 0^{*-1}(r\ a'\ a) = (r\ a\ 0)$, and $const\ 1^{*-1}(r\ a'\ a) = \begin{cases} r\ a\ 1, & \text{if } a{=}0 \\ r\ a\ 0, & \text{if } a{=}1 \end{cases}$

– $\delta : S \to ((S \to S \to \mathbb{C}) \times (S \to S \to \mathbb{C}))$ double the argument received, defined as: $\delta(a) = (r\ a\ a, r\ a\ a)$
$\delta^* : (S \to V\ S) \to ((S \to S \to \mathbb{C}) \times (S \to S \to \mathbb{C}))$, defined as: $\delta(r\ a'\ a) = (r\ a'\ a, r\ a'\ a)$, and
$\delta^{-1}(r\ a'\ a, r\ a'\ a) = a$
$\delta^{*-1}(r\ a''\ a, r\ a'\ a) = (r\ a'\ a)$

– $swap : (S \times T) \to (S \to T \to \mathbb{C}) \times (T \to S \to \mathbb{C})$, function that exchanges the arguments, defined as: $swap(a, b) = (r\ a\ b, r\ b\ a)$
$swap^* : ((S \to V\ S) \times (S \to V\ T)) \to ((S \to T \to \mathbb{C})) \times (T \to S \to \mathbb{C}))$ defined as: $swap^*(r\ a' a, r\ b'\ b) = (r\ a\ b, r\ b\ a)$
$swap^{-1}(r\ a\ b, r\ b\ a) = (a, b)$
$swap^{*-1}(r\ a'\ b, r\ b'\ a) = (r\ b\ a, r\ a\ b)$

– $f \times g$; for any two functions:
$f : S_1 \to (S_1 \to V\ T_1)$, and $f^* : (S_1 \to V\ S_1') \to (S_1' \to T_1 \to \mathbb{C})$
$g : S_2 \to (S_2 \to V\ T_2)$, and $g^* : (S_2 \to V\ S_2') \to (S_2' \to T_2 \to \mathbb{C})$, we have:
$f \times g : (S_1 \times S_2) \to (S_1 \to T_1 \to \mathbb{C}) \times (S_2 \to T_2 \to \mathbb{C})$ is defined as: $f \times g\ (a, b) = (f\ a, g\ b)$
$f^* \times g^* : (S_1 \to V\ S_1') \times (S_2 \to V\ S_2') \to (S_1' \to T_1 \to \mathbb{C}) \times (S_2' \to T_2 \to \mathbb{C})$, defined as: $f^* \times g^*\ (r\ a'\ a, r\ b'\ b) = (f^*\ (r\ a'\ a), g^*\ (r\ b'\ b))$
$(f^{-1} \times g^{-1})\ (r\ a'\ a, r\ b'\ b) = (f^{-1}\ (r\ a'\ a), g^{-1}\ (r\ b'\ b))$
$(f^{*-1} \times g^{*-1})\ (r\ a'\ a, r\ b'\ b) = (f^{*-1}\ (r\ a'\ a), g^{*-1}\ (r\ b'\ b))$

– $\delta_{\Gamma,\Delta}$, separate contexts depending on the program formed, the conditions are below.
$\delta_{\Gamma,\Delta} : [\![\Gamma \otimes \Delta]\!] \to (S \to [\![\Gamma]\!] \to \mathbb{C}) \times (S \to [\![\Delta]\!] \to \mathbb{C})$ defined as: $\delta_{\Gamma,\Delta}(a \otimes a') = ((r\ a\ a), (r\ a'\ a'))$, if one of the contexts is empty $\bullet$, then it applies: $\delta_{\Gamma,\Delta}(a) = (r\ a\ a)$. The function with * is defined as:
$\delta^*_{\Gamma,\Delta}((r\ a\ a') \otimes (r\ b\ b')) = ((r\ a\ a'), (r\ b\ b'))$ or $\delta^*_{\Gamma,\Delta}(r\ a\ a') = (r\ a\ a')$, the inverse functions are obtained similarly to the previous cases.
The conditions are the following:
$$(\delta_{\Gamma,\Delta}) = \begin{cases} \delta_{\Gamma',\Delta'} \times \delta & \text{if } \Gamma = \Gamma', x : \sigma \text{ y } \Delta = \Delta', x : \sigma \\ \delta_{\Gamma',\Delta} \times id & \text{if } \Gamma = \Gamma', x : \sigma \text{ y } x \notin dom(\Delta) \\ id^+ & \text{if } \Gamma = \bullet \end{cases}$$

– $(f|g) : ([\![Q_2]\!] \times S) \to (S \to T \to \mathbb{C})$, is associated with **if**$^\circ$ conditional, defined as: $(f|g)\ (1, a) = (f\ a)$ and $(f|g)\ (0, a) = (g\ a)$.
$(f^{-1}|g^{-1}) : (S \to V\ T) \to ((S \to S \to \mathbb{C}) \times (S \to S \to \mathbb{C}))$, defined as:

$$(f^{-1}|g^{-1})(r\ a'\ a) = \begin{cases}(a, f^{-1}\ a) & \text{if } a{=}1\\ (a, g^{-1}\ a) & \text{if } a{=}0\end{cases}$$

$(f^*|g^*) : (S \to V\ S) \times (S \to V\ S) \to (S \to T \to \mathbb{C})$, defined as:
$(f^*|g^*)\ (r\ a'\ 1, r\ a'\ a) = f^*(r\ a'\ a)$ or $(f^*|g^*)\ (r\ a'\ 0, r\ a'\ a) = g^*(r\ a'\ a)$
$(f^{*-1}|g^{*-1}) : (S \to V\ T) \to (S \to S \to \mathbb{C}) \times (S \to S \to \mathbb{C})$

$$(f^{*-1}|g^{*-1})\ (r\ a'\ a) = \begin{cases}((r\ a'\ a), f^{*-1}(r\ a'\ a)) & \text{if } a{=}0\\ ((r\ a'\ a), g^{*-1}(r\ a'\ a)) & \text{if } a{=}1\end{cases}$$

Consider the following: $(f \times g)^* = f^* \times g^*$ and $(f \times g)^{-1} = f^{-1} \times g^{-1}$.

With the previous rules and functions, certain programs can be derived, however, we still need define the modified rules for terms with probability amplitudes and superpositions. The new rules to form well-typed judgements and for deriving terms (quantum and classic) are in Table 3 and Table 4, respectively. Based on these definitions, we propose the modification that also allows storing operations in the history stack.

$$\frac{}{\bullet \vdash \overrightarrow{0} : \sigma}\text{z-intro} \qquad \frac{\Gamma \vdash^\circ c : Q_2 \quad \Delta \vdash^\circ t, u : \sigma}{\Gamma \otimes \Delta \vdash^\circ \mathbf{if}^\circ\ c\ \mathbf{then}\ t\ \mathbf{else}\ u : \sigma}\mathbf{if}^\circ$$

$$\frac{\Gamma \vdash t : \sigma}{\Gamma \vdash \kappa + t : \sigma}\text{prob} \qquad \frac{\Gamma \vdash^\circ t, u : \sigma \quad t \perp u \quad |\lambda|^2 + |\kappa|^2 = 1}{\Gamma \vdash^\circ \lambda * t + \kappa * u : \sigma}\text{sup}^\circ$$

$$\frac{\Gamma \vdash t, u : \sigma}{\Gamma \vdash t + u : \sigma}\text{sup} \qquad \frac{\Gamma \vdash^\circ t : \sigma \quad \Gamma \vdash t \equiv u : \sigma}{\Gamma \vdash^\circ u : \sigma}\text{subst}$$

Table 3: Typing quantum data (II) (Source: [17], p. 36,38).

3.2 Operational model for quantum control with history

Three rules are attached that consider: the vector zero, a term with a certain probability amplitude and superposition.

If we have quantum data and control, consider the programs that contain $\mathbf{if}^\circ$ or $t + u$, its terms must be orthogonal to each other. This is denoted in the derivations with the symbol $\vdash^\circ$, the orthogonality with $\perp$ and internal product with $\langle\ |\ \rangle$. To satisfy orthogonality we can see Table 5, and for a quantum terms well formed see Table 3.

Having the above, now we can apply the operational model with a program, for example: $[\![\bullet \otimes \bullet \vdash \mathbf{if}^\circ\ 1\ \mathbf{then}\ (-1) * true + false\ \mathbf{else}\ true + false : \mathcal{Q}_2\,]\!]^{\mathrm{Q}} = (g^*|h^*)\ \circ\ (f^* \times \boldsymbol{id}_-^{*-1}; id^*) \circ ((\boldsymbol{\delta_{\Gamma,\Delta}})_-^{-1}; \delta_{\Gamma,\Delta})$.
At this point, the functions are mixed with their inverses, which must be separated, how to achieve this we will define immediately.

$[\![\bullet \vdash \mathbf{0} : \sigma]\!]^{\mathbb{Q}} = \boldsymbol{const}\ \boldsymbol{v}_{-}^{-1}; const\ v$	where $\forall a \in [\![\sigma]\!].v\ a = 0.0$
$[\![\Gamma \vdash \kappa * t : \sigma]\!]^{\mathbb{Q}} = (\kappa) * f$	where $\forall b \in [\![\Gamma]\!]$ $f = [\![\Gamma \vdash t : \sigma]\!]^{\mathbb{Q}}$
$[\![\Gamma \vdash t + u : \sigma]\!]^{\mathbb{Q}} = \frac{1}{\sqrt{2}}(f + g)$	where $\forall a \in [\![\Gamma]\!]$ $f = [\![\Gamma \vdash t : \sigma]\!]^{\mathbb{Q}}$ $g = [\![\Gamma \vdash u : \sigma]\!]^{\mathbb{Q}}$

Table 4: Operational rules for quantum data.

$\langle t \vert t \rangle = 1$ if $t \neq \vec{0}$	$\langle \lambda * t + \lambda' * t' \vert u \rangle = \overline{\lambda} * \langle t \vert u \rangle + \overline{\lambda'} * \langle t' \vert u \rangle$
$\langle false \vert true \rangle = 0$	$\langle t \vert \kappa * u + \kappa' * u' \rangle = \kappa * \langle t \vert u \rangle + \kappa' * \langle t \vert u' \rangle$
$\langle true \vert false \rangle = 0$	
	$\langle \lambda * t \vert u \rangle = \overline{\lambda}\, \langle t \vert u \rangle$
$\langle \vec{0} \vert true \rangle = 0 = \langle true \vert \vec{0} \rangle$	$\langle t \vert \lambda * u \rangle = \lambda * \langle t \vert u \rangle$
$\langle \vec{0} \vert false \rangle = 0 = \langle false \vert \vec{0} \rangle$	$\langle t + t' \vert u \rangle = \langle t \vert u \rangle + \langle t' \vert u \rangle$
$\langle \vec{0} \vert x \rangle = 0 = \langle x \vert \vec{0} \rangle$	$\langle t \vert u + u' \rangle = \langle t \vert u \rangle + \langle t \vert u' \rangle$
$\langle (t, t') \vert (u, u') \rangle = \langle t \vert u \rangle * \langle t' \vert u' \rangle$	$\langle t \vert u \rangle =?$ otherwise

Table 5: Inner products and orthogonality (Source: [17], p. 38).

4 History and Reversibility

To apply reversibility, several points are important: A concept called a history stack and auxiliary functions defined in our proposal of the operational model. Considering that each derivation is given from functions, and these are stored in a stack, then, the inverse functions will result in reversibility, allowing returns to the initial argument of the program. Formally, the terms and the history stack make a computational state, that is:

Definition 2 (Computational state). *The* computational state *is a sequence of the form:*

$$\boldsymbol{h_{t_n}-;\cdots;h_{t_2}-;h_{t_1}-}; t_1 \circ t_2 \circ \cdots \circ t_n$$

where $\boldsymbol{h_{t_n}-;\cdots;h_{t_2}-;h_{t_1}-}$*, is called the* history track *and* $t_1 \circ t_2; \circ \cdots \circ t_n$*, the* computational register*.*

Regarding $\boldsymbol{h_{t_i}-}; t_i$, the part $\boldsymbol{h_{t_i}}$ represents and stores the inverse operation that executes t_i.

Definition 3. $\mathcal{H}$*, denotes the (possibly empty) history track.*

To arrive at the computational state, the terms must be *factored*, considering different cases.

4.1 Factorization

Let the expression $\boldsymbol{h_{t_1}}-;t_1\circ\boldsymbol{h_{t_2}}-;t_2\circ\cdots\circ\boldsymbol{h_{t_n}}-;t_n$, where the semicolon (**;**) denotes string concatenation. The h_{t_i} expressions are histories corresponding to each term t_i, respectively. The terms t_i are associated with functions of operational model. therefore, they will receive arguments.

The expressions $(h_{t_i}-;t_i)$ can have different forms, for wich, to factor consider the following cases:

i) $\mathcal{H};\big(\boldsymbol{h_t}-;t(a)\big)$ factored as: $\mathcal{H};\boldsymbol{h_t}-;\big(t(a)\big)$
ii) $\mathcal{H};\big(\boldsymbol{h_{t_1}}-;t_1\times\boldsymbol{h_{t_2}}-;t_2\big)(a,b)$ is factored as: $\mathcal{H};\boldsymbol{h_{t_1}}-\times\boldsymbol{h_{t_2}}-;\big(t_1\times,\ t_2\big)(a,b)$
iii) $\mathcal{H};\big(\boldsymbol{h_{t_1}}-;t_1+\boldsymbol{h_{t_2}}-;t_2\big)(a)$ factored as: $\mathcal{H};\boldsymbol{h_{t_1}}-+\boldsymbol{h_{t_2}}-;\big(t_1(a)+t_2(a)\big)$
iv) $\mathcal{H};\big(\kappa*\boldsymbol{h_{t_1}}-;t_1\big)(a)$ factored as: $\mathcal{H};\boldsymbol{h_{t_1}}-;\big(\kappa*t_1(a)\big)$, where $\kappa\in\mathbb{C}$
v) If t,t' are superpositions or tuples then

$$\big(\boldsymbol{h_t}-;t|\boldsymbol{h_{t'}}-;t'\big)(a,b)=\begin{cases}\mathcal{H};\boldsymbol{h_t}-;\big(t(b)\big) & \text{if } a=1\\ \mathcal{H};\boldsymbol{h_{t'}}-;\big(t'(b)\big) & \text{if } a=0\end{cases}$$

else
$\big(\boldsymbol{h_t}-;t|\boldsymbol{h_{t'}}-;t'\big)(a,b)=\mathcal{H};(\boldsymbol{h_t}-|\boldsymbol{h_{t'}}-);\ \big((t|t')(a,b)\big)$, and the conditional definition is applied.

The factorization is started from right to left, namely:

$$\begin{aligned}\boldsymbol{h_{t_1}}-;t_1\circ\boldsymbol{h_{t_2}}-;t_2\circ\cdots\circ\boldsymbol{h_{t_n}}-;t_n&=\boldsymbol{h_{t_n}}-;\big(\boldsymbol{h_{t_1}}-;t_1\circ\boldsymbol{h_{t_2}}-;t_2\circ\cdots\circ t_n\big)\\&=\boldsymbol{h_{t_n}}-;\cdots;\boldsymbol{h_{t_2}}-;\big(\boldsymbol{h_{t_1}}-;t_1\circ t_2\circ\cdots\circ t_n\big)\\&=\underbrace{\boldsymbol{h_{t_n}}-;\cdots;\boldsymbol{h_{t_2}}-;\boldsymbol{h_{t_1}}-;}_{\text{History track }(\mathcal{H})}\ \underbrace{\big(t_1\circ t_2\circ\cdots\circ t_n\big)}_{\text{Computational Register}}\end{aligned}$$

If there exists a conditional $(t|t')$, where t and t', are superpositions; will be incorporated into the history track once the conditional has been evaluated. With the above we have the computational state, and we continue with the rules developed for the reversibility.

4.2 Reversibility

The reversibility can be executed starting from the fact that if we have a state and execute all the functions of the computational register we will obtain a final value q, i.e., $\boldsymbol{h_{t_n}}-;\cdots;\boldsymbol{h_{t_2}}-;\boldsymbol{h_{t_1}}-;\big(t_1\circ t_2\circ\cdots\circ t_n\big)(a)=\cdots=\boldsymbol{h_{t_n}}-;\cdots;\boldsymbol{h_{t_2}}-;\boldsymbol{h_{t_1}}-;q$. Let a computational state (result of derivations) with the form:

$$\boldsymbol{h_{t_1}}-;\cdots;\boldsymbol{h_{t_2}}-;\boldsymbol{h_{t_1}}-;q,$$

where q it's an irreducible term (for the operational model). Given $h_{t_i}-$ a history, apply as argument q to the previous immediate history, replacing by the symbol $-$, and applying the respective function, for example: $h_{t_i}-;a=h_{t_i}(a)$. Reversibility can be applied to the value q, from the history stack $(h_{t_1}-;h_{t_2}-;\cdots;h_{t_n}-)$, considering the following cases:

i) $h_{t_i-}; q = h_{t_i}(q)$
ii) $(h_{t_i-} \times h_{t_j-})\ (q_1, q_2) = (h_{t_i}\ q_1, h_{t_j}\ q_2)$
iii) $(h_{t_i-} + h_{t_j-})\ \alpha(q_1 + q_2) = (h_{t_i}\ q_1, h_{t_j}\ q_2)$, where $\alpha \in \mathbb{C}$.
iv) $h_{t_1-}; \kappa * q = h_{t_1}(q)$

where $\kappa, \alpha \in \mathbb{C}$ are discarded.

With the previous proposals, we show an example with classic and quantum data, this is the Hadamard gate.

5 Example: Hadamard gate

$had\ 1 = \mathbf{if}^{\circ}\ 1\ \mathbf{then}\ (-1) * true + false\ \mathbf{else}\ true + false$
$[\![\bullet \otimes \bullet \vdash had\ 1 : \mathcal{Q}_2]\!]^{\mathbb{Q}} = (g^*|h^*) \circ (f^* \times \boldsymbol{id}_-^{*-1}; id^*) \circ ((\boldsymbol{\delta_{\Gamma,\Delta}})_-^{-1}; \delta_{\Gamma,\Delta})$

$$
\begin{aligned}
&= \Big(\overbrace{\frac{1}{\sqrt{2}}((-1) * \boldsymbol{const}\ \mathbf{1}_-^{*-1}; const\ 1^* + \boldsymbol{const}\ \mathbf{0}_-^{*-1}; const\ 0^*)}^{g^*} \ | \\
&\quad \overbrace{\frac{1}{\sqrt{2}}(\mathbf{const}\ \mathbf{1}_-^{*-1}; const\ 1^* + \boldsymbol{const}\ \mathbf{0}_-^{*-1}; const\ 0^*)}^{h^*} \Big) \\
&\quad \circ \Big(\boldsymbol{const}\ \mathbf{1}_-^{*-1}; const\ 1^* \times \boldsymbol{id}_-^{*-1}; id^* \Big) \circ \Big(\boldsymbol{id}_-^{+-1}; id^+ \Big) \\
&= \overbrace{(\boldsymbol{id}_-^{+-1}); (\boldsymbol{const}\ \mathbf{1}_-^{*-1} \times \boldsymbol{id}_-^{*-1})}^{\mathcal{H}}; \big(g^*|h^*\big) \circ (const\ 1^* \times id^*) \circ id^+(1) \\
&= \mathcal{H}; \big(g^*|h^*\big) \circ (const\ 1^* \times id^*)(r\ 1\ 0, r\ 0\ 1) \\
&= \mathcal{H}; \big(g^*|h^*\big) \circ (const\ 1^*(r\ 1\ 0),\ id^*(r\ 0\ 1)) \\
&= \mathcal{H}; \big(g^*|h^*\big)(r\ 0\ 1,\ r\ 0\ 1) = \mathcal{H}; g^*(r\ 0\ 1) \\
&= \mathcal{H}; \Big(\frac{1}{\sqrt{2}}((-1) * \boldsymbol{const}\ \mathbf{1}_-^{*-1}; const\ 1^* + \mathbf{const}\ \mathbf{0}_-^{*-1}; const\ 0^*) \Big)(r\ 0\ 1) \\
&= \mathcal{H}; (\boldsymbol{const}\ \mathbf{1}_-^{*-1} + \boldsymbol{const}\ \mathbf{0}_-^{*-1}); \Big(\frac{1}{\sqrt{2}}((-1) * const\ 1^* + const\ 0^*) \Big)(r\ 0\ 1) \\
&= \mathcal{H}; \frac{1}{\sqrt{2}} \Big((-1) * const\ 1^*(r\ 0\ 1) + const\ 0^*(r\ 0\ 1) \Big) \\
&= \mathcal{H}; \frac{1}{\sqrt{2}} \Big((-1) * (r\ 0\ 1) + (r\ 0\ 0) \Big)
\end{aligned}
$$

If evaluate r functions we get: $\frac{1}{\sqrt{2}}\big((-1) * (0.0) + (1.0)\big)$. Explicitly $\mathcal{H}$ is:
$\boldsymbol{(id_-^{+-1}); (const\ 1_-^{*-1} \times id_-^{*-1}); (const\ 1_-^{*-1} + const\ 0_-^{*-1})}$;

Applying reversibility we have:

$$(\boldsymbol{id}_{-}^{+-1}); (\boldsymbol{const}\ \boldsymbol{1}_{-}^{*-1} \times \boldsymbol{id}_{-}^{*-1}); (\boldsymbol{const}\ \boldsymbol{1}_{-}^{*-1} + \boldsymbol{const}\ \boldsymbol{0}_{-}^{*-1})(r\ 0\ 1 + r\ 0\ 0)$$
$$= (\boldsymbol{id}_{-}^{+-1}); (\boldsymbol{const}\boldsymbol{1}_{-}^{*-1} \times \boldsymbol{id}_{-}^{*-1}); (\boldsymbol{const}\ \boldsymbol{1}_{-}^{*-1} + \boldsymbol{const}\ \boldsymbol{0}_{-}^{*-1})(r\ 0\ 1 + r\ 0\ 0)$$
$$= (\boldsymbol{id}_{-}^{+-1}); (\boldsymbol{const}\ \boldsymbol{1}_{-}^{*-1} \times \boldsymbol{id}_{-}^{*-1}); (const\ 1^{*-1}(r\ 0\ 1), const\ 0^{*-1}(r\ 0\ 0))$$
$$= (\boldsymbol{id}_{-}^{+-1}); (\boldsymbol{const}\ \boldsymbol{1}_{-}^{*-1} \times \boldsymbol{id}_{-}^{*-1})(r\ 1\ 0,\ r\ 0\ 0)$$
$$= (\boldsymbol{id}_{-}^{+-1}); (const\ 1^{*-1}(r\ 1\ 0),\ id_{-}^{*-1}(r\ 0\ 0)) = id^{+-1}(r\ 0\ 1, r\ 0\ 0) = 1$$

Through this example, is observed how the reversibility is executed and that it is successfully achieved, that is, return to the initial argument, in this case, to value 1. We proceed with the conclusions and future work.

6 Conclusions and Further Work

The construction to add the history track was partially worked, the classical elements of the syntax and its respective operational model were considered first, immediately, the inverse functions were defined and added to the model, where the inverse operations are stored in the history stack. This is extrapolated to the complete syntax incorporating terms with probability amplitude and superpositions, allowing a model also with inverse functions.

If you have a value (resulting from a program) and this is applied to the respective history stack, then the inverse functions are applied, allowing you to return to the initial value of the program, this results in reversibility.

As a possible future work, a simple implementation of programs can be made than execute reversibility in a more direct and natural way.

References

1. Altenkirch, T., Grattage, J.: A functional quantum programming language. In: 20th Annual IEEE Symposium on Logic in Computer Science (LICS' 05). pp. 249 – 258 (2005). 10.1109/LICS.2005.1
2. Lampis, M., Ginis, K.G., Papakyriakou, M.A., Papaspyrou, N.S.: Quantum data and control made easier. Electron. Notes Theor. Comput. Sci. **210**, 85–105 (Jul 2008). 10.1016/j.entcs.2008.04.020
3. Tonder, A.v.: A lambda calculus for quantum computation. SIAM J. Comput. **33**(5), 1109–1135 (2004). 10.1137/S0097539703432165
4. Ömer, B.: A procedural formalism for quantum computing. AIP Conference Proceedings **627**(1), 276 (2002). 10.1063/1.1503695
5. Mlnar'k, H.: Quantum Programming Language LanQ. phdthesis, Faculty of Informatics (September 2007)
6. Sanders, J.W., Zuliani, P.: Quantum programming. In: Backhouse, R., Oliveira, J.N. (eds.) Mathematics of Program Construction. pp. 80–99. Springer Berlin Heidelberg, Berlin, Heidelberg (2000)
7. Gay, S.J.: Quantum programming languages: Survey and bibliography. Mathematical. Structures in Comp. Sci. **16**(4), 581–600 (2006). 10.1017/S0960129506005378

8. Jorrand, P., Lalire, M.: From Quantum Physics to Programming Languages: A Process Algebraic Approach, pp. 1–16. Springer Berlin Heidelberg, Berlin, Heidelberg (2005). 10.1007/11527800_1
9. Gay, S.J., Nagarajan, R.: Communicating quantum processes. In: Proceedings of the 32Nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 145–157. POPL '05, ACM, New York, NY, USA (2005). 10.1145/1040305.1040318
10. Gay, S.J., Nagarajan, R.: Communicating quantum processes. SIGPLAN Not. **40**(1), 145–157 (2005). 10.1145/1047659.1040318
11. Miszczak, J.A.: High-level structures for quantum computing. Synthesis Lectures on Quantum Computing **4** (05 2012). 10.2200/S00422ED1V01Y201205QMC006
12. Selinger, P.: A brief survey of quantum programming languages. In: In Proceedings of the 7th International Symposium on Functional and Logic Programming. Springer (2004). 10.1007/978-3-540-24754-8_1
13. Selinger, P.: Towards a quantum programming language. In: Mathematical Structures in Computer Science. pp. 527–586 (2004). 10.1017/S0960129504004256
14. Díaz-Caro, A., Arrighi, P., Gadella, M., Grattage, J.: Measurements and confluence in quantum lambda calculi with explicit qubits. Electronic Notes in Theoretical Computer Science **270**(1), 59 – 74 (2011). 10.1016/j.entcs.2011.01.006, proceedings of the Joint 5th International Workshop on Quantum Physics and Logic and 4th Workshop on Developments in Computational Models (QPL/DCM 2008)
15. Abramsky, S.: Computational interpretations of linear logic. Theoretical Computer Science **111**(1), 3 – 57 (1993). 10.1016/0304-3975(93)90181-R
16. Abramsky, S.: A structural approach to reversible computation. Theor. Comput. Sci. **347**(3), 441–464 (2005). 10.1016/j.tcs.2005.07.002
17. Altenkirch, T., Grattage, J., Vizzotto, J.K., Sabry, A.: An algebra of pure quantum programming. Electronic Notes in Theoretical Computer Science **170**, 23 – 47 (2007). 10.1016/j.entcs.2006.12.010, proceedings of the 3rd International Workshop on Quantum Programming Languages (QPL 2005)
18. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information: 10th Anniversary Edition. Cambridge University Press (2010). 10.1017/CBO9780511976667
19. McMahon, D.: Quantum Computing Explained
20. Grattage, J.J.: A functional quantum programming language (2006)
21. Landauer, R.: Irreversibility and heat generation in the computing process. IBM Journal of Research and Development **5**(3), 183–191 (July 1961). 10.1147/rd.53.0183
22. Reversible computation in term rewriting. Journal of Logical and Algebraic Methods in Programming **94**, 128 – 149 (2018). 10.1016/j.jlamp.2017.10.003
23. Abramsky, S.: A structural approach to reversible computation. Theoretical Computer Science **347**(3), 441 – 464 (2005). 10.1016/j.tcs.2005.07.002
24. Bennett, C.H.: Logical reversibility of computation. IBM J. Res. Dev. **17**(6), 525–532 (1973). 10.1147/rd.176.0525
25. Moggi, E.: Computational lambda-calculus and monads. In: Proceedings of the Fourth Annual Symposium on Logic in Computer Science. pp. 14–23. IEEE Press (1989)
26. Awodey, S.: Category Theory. Oxford Logic Guides (2006)

Appendix 1

$$
\begin{aligned}
[\![\bullet \vdash () : Q_1]\!] &= const\ 0 \\
[\![\bullet \vdash false : Q_2]\!] &= const\ 0 \\
[\![\bullet \vdash true : Q_2]\!] &= const\ 1 \\
[\![x : \sigma \vdash x : \sigma]\!] &= id_* \\
[\![\Gamma \otimes \Delta \vdash \mathbf{let}\ x = t\ \mathbf{in}\ u : \sigma]\!] &= g \circ (f \times id) \circ \delta_{\Gamma,\Delta} \\
&\quad \text{where:} \\
&\qquad f = [\![\Gamma \vdash t : \sigma]\!] \\
&\qquad g = [\![\Delta,\ x : \sigma \vdash u : \tau]\!] \\
[\![\Gamma \otimes \Delta \vdash (t,u) : \sigma \otimes \tau]\!] &= (f \times g) \circ \delta_{\Gamma,\Delta} \\
&\quad \text{where:} \\
&\qquad f = [\![\Gamma \vdash t : \sigma]\!] \\
&\qquad g = [\![\Delta \vdash u : \tau]\!] \\
[\![\Gamma \otimes \Delta \vdash \mathbf{let}\ (x,y) = t\ \mathbf{in}\ u : \rho]\!] &= g \circ (f \times id) \circ \delta_{\Gamma,\Delta} \\
&\quad \text{where:} \\
&\qquad f = [\![\Gamma \vdash t : \sigma \otimes \sigma]\!] \\
&\qquad g = [\![\Delta,\ x : \sigma,\ y : \tau \vdash u : \rho]\!] \\
[\![\Gamma \otimes \Delta \vdash \mathbf{if}^{\circ}\ c\ \mathbf{then}\ t\ \mathbf{else}\ u : \sigma]\!] &= (g|h) \circ (f \times id) \circ \delta_{\Gamma,\Delta} \\
&\quad \text{where:} \\
&\qquad f = [\![\Gamma \vdash c : Q_2]\!] \\
&\qquad g = [\![\Delta \vdash t : \sigma]\!] \\
&\qquad h = [\![\Delta \vdash u : \sigma]\!] \\
[\![\Gamma \vdash t : \sigma]\!] &= f \times id^* \\
&\quad \text{where:} \\
&\qquad f = [\![\Gamma,\ x : Q_1 \vdash t : \sigma]\!]
\end{aligned}
$$

Table 6: Meaning of classical derivations (Source: [17], p. 31).

Appendix 2. Auxiliary functions

Let $a, a', b \in S$, $S = \{0, 1\}$ (Source: [17], p. 30).

- $id : S \to V\ S$, defined by: $id(a) = a$
- $id^+ : S \to [\![Q_1]\!] \times S$, defined by: $id^+(a) = (0, a)$
- $id_+ : [\![Q_1]\!] \times S \to S$, defined by: $id_+(0, a) = a$
- $const\ a : [\![Q_1]\!] \to S$, defined by: $const\ a(0) = a$
- $\delta : S \to (S, S)$, defined by: $\delta(a) = (a, a)$
- $swap : S \times T \to T \times S$, defined by: $swap(a, b) = (b, a)$
- Let a functions $f : S_1 \to T_1$ and $g : S_2 \to T_2$, therefore $f \times g : S_1 \times S_2 \to T_1 \times T_2$, is defined by: $f \times g\ (a, b) = (f\ a, g\ b)$
- $\delta_{\Gamma,\Delta} : [\![\Gamma \otimes \Delta]\!] \to [\![\Gamma]\!] \times [\![\Delta]\!]$ defined by induction as follows:

$$\delta_{\Gamma,\Delta} = \begin{cases} \delta_{\Gamma',\Delta'} \times \delta & \text{if } \Gamma = \Gamma', x : \sigma \text{ y } \Delta = \Delta', x : \sigma \\ \delta_{\Gamma',\Delta} \times id & \text{if } \Gamma = \Gamma', x : \sigma \text{ y } x \notin dom(\Delta) \\ id^+ & \text{if } \Gamma = \bullet \end{cases}$$

- For any two functions $f, g \in S \to T$, $(f|g) \in S \to T$, defined as:

$$(f|g)\ (1, a) = (f\ a)$$

$$(f|g)\ (0, a) = (g\ a)$$

Appendix 3

Example 1. true + false

$$[\![\bullet \vdash true + false : \mathcal{Q}_2]\!]^{\mathrm{Q}} = \frac{1}{\sqrt{2}}(f + g)$$

where:

$$f = [\![\bullet \vdash true : \mathcal{Q}_2]\!]^{\mathrm{Q}} = \boldsymbol{const\ 1^{-1}_{-}}; const\ 1$$
$$g = [\![\bullet \vdash false : \mathcal{Q}_2]\!]^{\mathrm{Q}} = \boldsymbol{const\ 0^{-1}_{-}}; const\ 0$$

therefore:

$$\begin{aligned}
[\![\bullet \vdash true + false : \mathcal{Q}_2]\!]^{\mathrm{Q}} = \frac{1}{\sqrt{2}}\Big(&\boldsymbol{const\ 1^{-1}_{-}}; const\ 1\ (0) \\
&+ \boldsymbol{const\ 0^{-1}_{-}}; const\ 0\ (0)\Big) \\
= &\boldsymbol{const\ 1^{-1}_{-}} + \boldsymbol{const\ 0^{-1}_{-}}; \\
&\frac{1}{\sqrt{2}}\Big(const\ 1\ (0) + const\ 0\ (0)\Big) \\
= &\boldsymbol{const\ 1^{-1}_{-}} + \boldsymbol{const\ 0^{-1}_{-}}; \frac{1}{\sqrt{2}}\big(r\ 0\ 1 + r\ 0\ 0\big)
\end{aligned}$$

is equivalent to

$$= \boldsymbol{const\ 1^{-1}_{-}} + \boldsymbol{const\ 0^{-1}_{-}}; \frac{1}{\sqrt{2}}\big(0.0 + 1.0\big)$$

Applying reversibility:

$$\begin{aligned}
\boldsymbol{const\ 1^{-1}_{-}} + \boldsymbol{const\ 0^{-1}_{-}}; \frac{1}{\sqrt{2}}\big(r\ 0\ 1 + r\ 0\ 0\big) = &const\ 1^{-1}_{-}(r\ 0\ 1) \\
&+ const\ 0^{-1}_{-}(r\ 0\ 0) \\
= &(0 + 0)
\end{aligned}$$