Reversibility for Quantum Programming Language QML

Nely Plata-Cesar, José Raymundo Marcial-Romero, and José Antonio Hernández-Servín

***Abstract*—We present an extension of the denotational semantic model of the quantum programming language QML, to which computational reversibility is incorporated.**

The semantics of QML is defined in a functional setting which consider classical and quantum data, to which we add inverse functions. Additionally we incorporate into the semantics a history track which allows reversibility in QML.

From the generation and processing of the history track and the final result of a program, the rules for executing reversibility allow to compute the original input data.

This work contributes to the study of reversibility in quantum programming languages and considering that there is not yet a quantum computer in which the language can be implemented, this history and the proposed inverse functions are not trivial and allows us to determine that this language is reversible.

***Index Terms*—Programming language, QML, quantum computation, quantum programming, reversibility.**

I. Introducción

En años recientes ha incrementado el desarrollo y contribución a lenguajes de programación cuánticos explorando sus características, identificando y definiendo propiedades, comparando beneficios con respecto a los lenguajes clásicos, explotando su capacidad de realizar varios cálculos simultáneamente, entre otras investigaciones [1]–[7].

Hay lenguajes de programación cuánticos basados en paradigmas imperativo, funcional y otros. Entre los imperativos se encuentran: QCL [8], LanQ [9] y qGCL [10], los tres con mediciones; QCL y LanQ, tienen una implementación y reversibilidad implícita; qGCL tiene únicamente semántica operacional. Basado en otros paradigmas están pQCL [11], QPAlg [12], entre otros [5], [6], [13]–[17].

Respecto a los funcionales se considerarán dos debido a que tienen una estructura formal, particularmente respaldada por cálculo lambda, teoría de categorías y modelo de circuitos. Estos son: cálculo lambda cuántico (λ_q) [18], con una semántica en la cual define un conjunto de instrucciones que permiten determinar que su lenguaje es reversible, siendo esto una propiedad de un sistema cuántico. Y el lenguaje QML pionero en el control de datos clásicos y cuánticos, define una semántica operacional y denotacional, para posteriormente proponer una teoría ecuacional que es completa y válida (sin mediciones) [19], [20].

QML puede ser abordado desde dos perspectivas. La primera se enfoca en trabajar la semántica operacional desde circuitos cuánticos, considerando que estos tienen la propiedad de reversibilidad (implícita) y mediciones cuánticas [21]. La segunda es trabajar su semántica denotacional basada en funciones, que omite mediciones y reglas de reversibilidad [20]. Para este caso, se considera trabajar su semántica denotacional.

La reversibilidad es una área donde las operaciones involucradas deberán ser capaces de obtener el argumento de entrada a partir de la salida resultante, como es el caso del cómputo cuántico donde determinadas operaciones permiten hacer cómputo reversible [1]–[3], [22]–[25].

Actualmente debido a la ausencia de una computadora cuántica en la cual se pueda implementar cualquier programa, se debe considerar el cómo representar en general el comportamiento cuántico y la reversibilidad (sin circuitos).

Con referencia a lo anterior, se encuentra el trabajo de Van Tonder con cálculo lambda cuántico, quien propone reglas para reversibilidad computacional en dicho lenguaje [18]. De forma similar, otros autores abordan reversibilidad clásica y cuántica a través de un control de las operaciones realizadas [22], [26]–[29].

La contribución actual consiste en complementar la semántica del lenguaje de programación cuántico QML para incorporar reversibilidad computacional a través de funciones inversas que originarán una pila de historial, también se definen reglas para ejecutar tal reversibilidad. Esta propuesta de semántica no altera el significado de ningún programa, sin embargo, incrementa la capacidad del lenguaje al poder establecer reversibilidad explícitamente, sin requerir de circuitos cuánticos.

II. Preliminares

A. Computación cuántica

Hay cuatro principios o postulados que definen el comportamiento del cómputo cuántico [1].

Los estados puros se representan con vectores unitarios que almacenan en memoria la información utilizada. La evolución indica cómo el sistema cambia de un estado a otro, esto aplicando matrices unitarias las cuales tienen las instrucciones que la computadora llevará a cabo. La medición u observación se enfoca en la existencia de un observador que al ver o interactuar con el estado del sistema, determinará la probabilidad en la que puede ocurrir un resultado, lo que implica un colapso del sistema, es decir, pasar del contexto cuántico

Manuscript submitted October 18, 2020.

Nely Plata-Cesar. Universidad Autónoma del Estado de México, Facultad de Ingeniería. Toluca, México. e-mail: nplatac@uaemex.mx

José Raymundo Marcial-Romero. Universidad Autónoma del Estado de México, Facultad de Ingeniería. Toluca, México. e-mail: jrmarcialr@uaemex.mx

J. Antonio Hernández-Servín. Universidad Autónoma del Estado de México, Facultad de Ingeniería. Toluca, México. e-mail: xoseahernandez@uaemex.mx

TABLA I
SINTAXIS DE QML.

(Variables)	$x, y, \ldots$	$\in Vars$
(Amp. probabilidad)	$\kappa, \iota, \ldots$	$\in \mathbb{C}$
(Patrones)	p, q	$::= x \mid (x, y)$
(Términos)	t, u	$::= x \mid () \mid (t, u) \mid$
		$\mathbf{let}\ p = t\ \mathbf{in}\ u \mid \vec{0} \mid$
		$false \mid true \mid \kappa * t \mid$
		$\mathbf{if}^\circ\ t\ \mathbf{then}\ u\ \mathbf{else}\ u$'

TABLA II
REGLAS PARA TÉRMINOS CLÁSICOS BIEN FORMADOS.

$$\frac{}{\bullet \vdash false : \mathcal{Q}_2}\ f-intro \qquad \frac{}{\bullet \vdash true : \mathcal{Q}_2}\ t-intro$$

$$\frac{\Gamma \vdash t : \sigma \quad \Delta \vdash u : \tau}{\Gamma \otimes \Delta \vdash (t, u) : \sigma \otimes \tau}\ \otimes -intro \qquad \frac{}{x : \sigma \vdash x : \sigma}\ var$$

$$\frac{\Gamma \vdash t : \sigma \quad \Delta, x : \sigma \vdash u : \tau}{\Gamma \otimes \Delta \vdash \mathbf{let}\ x = t\ \mathbf{in}\ u : \tau}\ let$$

$$\frac{\Gamma \vdash c : \mathcal{Q}_2 \quad \Delta \vdash t, u : \sigma}{\Gamma \otimes \Delta \vdash \mathbf{if}^\circ\ c\ \mathbf{then}\ t\ \mathbf{else}\ u : \sigma}\ if^\circ$$

$$\frac{\Gamma \vdash t : \sigma \otimes \tau \quad \Delta, x : \sigma, y : \tau \vdash u : \rho}{\Gamma \otimes \Delta \vdash \mathbf{let}\ (x, y) = t\ \mathbf{in}\ u : \rho}\ \otimes -elim$$

$$\frac{\Gamma \vdash t : \sigma}{\Gamma, x : \mathcal{Q}_1 \vdash t : \sigma}\ wk-unit \qquad \frac{}{\bullet \vdash () : \mathcal{Q}_1}\ unit$$

al clásico. Finalmente, los sistemas compuestos se encargan de definir estados a partir de sistemas diferentes e investigar cómo interactúan entre ellos [1], [3], [24].

B. Lenguaje de programación cuántico QML

Las definiciones e información necesaria de QML se presentan a continuación. Comenzando con la sintaxis del lenguaje, que se encuentra en la Tabla I [19], [20].

A partir de esta sintaxis se pueden formar programas convencionales como tuplas, $\mathbf{let}\ p = t\ \mathbf{in}\ u$, $\mathbf{if}^\circ\ t\ \mathbf{then}\ u\ \mathbf{else}\ u$', términos cuánticos con amplitud de probabilidad $\kappa * t$ y superposiciones $t + u$. Las reglas para establecer programas bien tipados o formados están en la Tabla II [20].

Los tipos están dados por la gramática $\sigma = \mathcal{Q}_1 | \mathcal{Q}_2 | \sigma \otimes \tau$, donde $\mathcal{Q}_1$ corresponde al elemento $()$, éste no acarrea información y Q_2 corresponde al conjunto de valores 0 y 1. Los contextos de tipo se denotan como Γ o Δ, y están dados por la siguiente gramática $\Gamma = \bullet | \Gamma, x : \sigma$, donde $\bullet$ corresponde al contexto vacío. Los contextos corresponden a funciones que van de un conjunto finito de variables a tipos.

Se asume que cada variable definida se utiliza al menos una vez. También incorpora el operador $\otimes$ el cual, mapea contextos a contextos, dadas las siguientes condiciones.

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

Semánticamente, los tipos se establecen como: $[\![\mathcal{Q}_1]\!] = \{0\}$, $[\![\mathcal{Q}_2]\!] = \{0, 1\}$, $[\![\sigma \otimes \tau]\!] = [\![\sigma]\!] \times [\![\tau]\!]$, donde $\otimes$ es el producto de tipos, el tipo $\mathcal{Q}_2$ es asociado con estados cuánticos (qubits). En consecuencia, la sentencia $\Gamma \vdash t : \sigma$ corresponde a la función $[\![\Gamma \vdash t : \sigma]\!] \in [\![\Gamma]\!] \to [\![\sigma]\!]$, es decir, dado un contexto éste retorna un tipo de la colección de valores.

A continuación, se presenta una breve introducción acerca de reversibilidad computacional cuántica.

C. Reversibilidad computacional

Una computadora capaz de ser reversible evalúa una función invertible que va de n-bits a n-bits, donde para cada función invertible hay sólo una entrada para cada salida, entonces si se conoce el resultado es posible calcular la entrada inicial [30]–[33].

El cómputo cuántico hace uso de operaciones reversibles, dadas por compuertas cuánticas, que corresponden a matrices unitarias y hermitianas. Una matriz hermitiana es aquella que satisface ser igual a su transpuesta conjugada. Entonces, la reversibilidad en un modelo de circuito que utiliza compuertas cuánticas con las características mencionadas, podría resultar natural, porque hay un registro de las operaciones aplicadas y tal operación reversible ejecutada dos veces permitiría regresar al valor inicial.

Sin embargo, en un lenguaje de programación donde las operaciones realizadas no están almacenadas explícitamente, la reversibilidad no es trivial [22], [34]–[36]. Para esto, varios autores han trabajado la reversibilidad aplicada a la computación clásica y cuántica, modelando a través de diagramas de flujo reversibles [31], almacenando en una pila las operaciones aplicadas [18], definiendo funciones inversas, entre otras [22], [26]–[29], [36], [37].

Por ejemplo, Van Tonder propone en cálculo lambda cuántico una pila de historial que acumula las operaciones ejecutadas dado un programa, esta pila permitirá reversibilidad computacional, omitiendo mediciones intermedias [18]. Tal trabajo es una referencia importante para esta investigación.

III. DATOS CLÁSICOS Y CUÁNTICOS

En esta sección se agrega una pila de historial al modelo semántico de QML incorporando funciones inversas y condicionado operacionalmente a matrices unitarias hermitianas, esto permitirá definir reversibilidad. Las reglas presentadas complementan la definición original del lenguaje. La semántica propuesta se encuentra en la Tabla III.

Esto es, la regla $[\![\bullet \vdash false : \mathcal{Q}_2]\!]^{\mathcal{Q}} = (\boldsymbol{^{\mathcal{Q}}const\ 0})^{-1}_{\_}$; $^{\mathcal{Q}}const\ 0$, corresponde a la función constante 0, su inversa se define como $(\boldsymbol{^{\mathcal{Q}}const\ 0})^{-1}_{\_}$; las funciones inversas son las que generarán un historial. La notación $.^{\mathcal{Q}}$ denota que está manipulando datos cuánticos. Esto es similar en todas las sentencias en la Tabla III.

A. Funciones auxiliares

La semántica está definida en términos de funciones (Apéndice A), a partir de las cuales se incorporan sus inversas $.^{-1}$ respectivas. Los autores usan Tripletas de Kleisli para pasar de datos clásicos a cuánticos, es decir, tales funciones operan con vectores complejos [20], [36], [38], [39].

TABLA III
SEMÁNTICA DE QML.

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

Si bien, las funciones inversas no siempre se definen de manera natural, se debe considerar que se establecen con la estrategia llamada por nombre (*call-by-name*), ya que los argumentos no pueden reducirse hasta que sean requeridos, aunado a la dificultad de trabajar con datos clásicos y cuánticos.

Considerando que las funciones inversas van a manipular vectores y que para lograr esto, se utilizaron Tripletas de Kleisli, entonces ahora la sentencia $\Gamma \vdash t : \sigma$ se interpreta como la función $[\![\Gamma]\!] \to [\![\sigma]\!]^Q$, donde $[\![\sigma]\!]^Q = [\![\sigma]\!] \to \mathbb{C}$ representa vectores complejos sobre el conjunto base $[\![\sigma]\!]$. La notación $[\![\sigma]\!]^Q$ también se denota como $V[\![\sigma]\!]$. Brevemente se define Tripleta de Kleisli y se da información necesaria.

Definición 1 (Tripleta de Kleisli): Una Tripleta de Kleisli sobre una categoría C, es definida por la tupla $(V, return, \_^*)$:

- $V : Obj(C) \to Obj(C)$.
- $return : S \to V\ S$, para $S \in C$.
- $f^* : V\ S \to V\ T$, para $f : S \to V\ T$.

Donde V, $return$, f^*, es una flecha y mapeos, respectivamente. Tal que $return$, es la inclusión de valores dentro de cálculos y f^* es una extensión de f que va de cómputos a cómputos y se nombra como *levantamiento* o *lift*, haciendo referencia a la elevación de la función f. Con lo anterior, los morfismos se interpretan como operaciones computacionales [38].

La implementación de esta definición, implica que cada función de tipo $S \to T$ cambia al mapeo $S \to V\ T$ usando $return$. Por ejemplo, considerar la interpretación de la función $const\ a : [\![\mathcal{Q}_1]\!] \to S$, entonces, extrapolando se obtiene: $return \circ const\ a : [\![\mathcal{Q}_1]\!] \to V\ S$. Esto se implementó para todas las funciones definidas por los autores (Apéndice A). Por simplificación se usará la notación ${}^{\mathcal{Q}}const\ a = return \circ const\ a$.

Ahora, sea una función $f : S \to V\ T$, tal que $f(a) = v_b$, donde $v_b \in V\ T$ es un vector y $b \in T$; esto permitirá llevar los valores de salida a cualquier otra función auxiliar.

Para comprender este proceso, considere cualquier función auxiliar f donde clásicamente se tiene $f : [\![\sigma]\!] \to [\![\tau]\!]$, tal que $f(a) = b$, ahora la función ${}^{\mathcal{Q}}f$ tiene el tipo $[\![\sigma]\!] \to [\![\tau]\!]^{\mathcal{Q}}$, equivalente a ${}^{\mathcal{Q}}f : [\![\sigma]\!] \to [\![\tau]\!] \to \mathbb{C}$ y se define como $f(a)(b) = \alpha \in \mathbb{C}$, donde $\alpha = 1.0$ si $a = b$ y 0.0 en otro caso. Reconsiderar que toda función ${}^{\mathcal{Q}}f$ tiene el mecanismo de evaluación llamada por nombre, por lo que su argumento no se evaluará hasta que las funciones no se puedan derivar más.

Las siguientes funciones son parte de nuestra contribución. Esto es, para cada función auxiliar f, se define su inversa f^{-1}, el lift f^* (asociada con Tripletas de Kleisli) y su inversa f^{*-1}.

Tipo	Función
${}^{\mathcal{Q}}id : S \to VS$	${}^{\mathcal{Q}}id(a) = v_a$
$v_a : S \to \mathbb{C}$	$v_a(b) = \begin{cases} 1.0, & \text{si } a = b \\ 0.0, & \text{en otro caso} \end{cases}$
${}^{\mathcal{Q}}id^* : VS \to VS$	${}^{\mathcal{Q}}id^*(v_x)(b) = \sum_a (v_x\ a) * ({}^{\mathcal{Q}}id\ a\ b)$
$id^{-1} : VS \to S$	$id^{-1}(v_a) = a$
$id^{*-1} : VS \to VS$	$id^{*-1} = id^*$
${}^{\mathcal{Q}}id^+ : S \to V(R)$, $R = [\![Q_1]\!] \times S$	${}^{\mathcal{Q}}id^+(a) = v_{(0,a)}$
$v_{(0,a)} : V([\![Q_1]\!] \times S$	$v_{(0,a)}(b, c) = \begin{cases} 1.0, & \text{si } (0, a) = (b, c) \\ 0.0, & \text{en otro caso} \end{cases}$
${}^{\mathcal{Q}}id^{+*} : VS \to V(R)$, $R = [\![Q_1]\!] \times S$	${}^{\mathcal{Q}}id^{+*}(v_x)(0, c) = \sum_a (v_x\ a) * ({}^{\mathcal{Q}}id^+(a)\ (0, c))$
${}^{\mathcal{Q}}id_+ : R_1 \to VS$, $R_1 = [\![Q_2]\!] \times S$	${}^{\mathcal{Q}}id_+(0, a) = v_a$, ${}^{\mathcal{Q}}id_+(1, a) = v_1$
${}^{\mathcal{Q}}id^*_+ : V(R) \to VS$, $R = [\![Q_1]\!] \times S$	${}^{\mathcal{Q}}id^*_+(v_{(x,y)})(b) = \sum_a (v_{(x,y)}\ (0, a)) * ({}^{\mathcal{Q}}id_+\ (0, a)\ b)$, ${}^{\mathcal{Q}}id^{-1}_+(v_a) = (0, a$, ${}^{\mathcal{Q}}id^{*-1}_+(v_a) = v_{(0,a)}$
${}^{\mathcal{Q}}const_a : R_2$, $R_2 = [\![Q_1]\!] \to VS$	${}^{\mathcal{Q}}const\ a(0) = v_a$
${}^{\mathcal{Q}}const^*_a : V[\![Q_1]\!] \to VS$	${}^{\mathcal{Q}}const\ a^*(v_x)(b) = \sum_{a' \in [\![\mathcal{Q}_1]\!]} (v_x\ a') * ({}^{\mathcal{Q}}const\ a(a')(b))$
${}^{\mathcal{Q}}\delta : S \to V\ (S \times S)$	${}^{\mathcal{Q}}\delta(a) = v_{(a,a)}$
$v_{(a,a)} : (S \times S) \to \mathbb{C}$	$v_{(a,a)}(b, c) = \begin{cases} 1.0, & \text{si } a = b = c \\ 0.0, & \text{en otro caso} \end{cases}$, $\delta^{-1}(v_{(a',a)}) = a$, $\delta^{*-1}(v_{(a',a)}) = (v_a)$
${}^{\mathcal{Q}}swap : S \times T \to V(T \times S)$	${}^{\mathcal{Q}}swap(a, b) = v_{(b,a)}$
$v_{(b,a)} : V\ (T \times S)$	$v_{(b,a)}(b', a') = \begin{cases} 1.0, & \text{si } (b, a) = (b', a') \\ 0.0, & \text{en otro caso} \end{cases}$
${}^{\mathcal{Q}}swap^* : V(S \times T) \to R_3$, $R_3 = (T \times S) \to \mathbb{C}$	${}^{\mathcal{Q}}swap^*(v_{(x,y)})(b, a) = \sum_{a,b} (v_{(x,y)}\ (a, b)) * ({}^{\mathcal{Q}}swap\ (a, b)\ (b, a))$, $swap^{-1}(v_{(b,a)}) = (a, b)$, $swap^{*-1}(v_{(b,a)}) = (v_{(a,b)})$
$({}^{\mathcal{Q}}f \times {}^{\mathcal{Q}}g) : S_1 \times S_2 \to R_4$, $R_4 = V\ (T_1 \times T_2)$	$({}^{\mathcal{Q}}f \times {}^{\mathcal{Q}}g)(a, b) = v_{(fa,gb)}$

TABLA IV
REGLAS PARA TÉRMINOS CUÁNTICOS BIEN FORMADOS (II) [20].

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

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

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

TABLA V
REGLAS PARA DATOS CUÁNTICOS.

$[\![\bullet \vdash \vec{0} : \sigma]\!]^{\mathcal{Q}} = \boldsymbol{\mathcal{Q}const\ a_{-}^{-1}}; {}^{\mathcal{Q}}const\ a$	donde $\forall a \in [\![\sigma]\!].v_a = 0.0$
$[\![\Gamma \vdash \kappa * t : \sigma]\!]^{Q} = (\kappa) * f$	donde $\forall b \in [\![\Gamma]\!]$ $f = [\![\Gamma \vdash t : \sigma]\!]^{\mathcal{Q}}$
$[\![\Gamma \vdash t + u : \sigma]\!]^{\mathcal{Q}} = \frac{1}{\sqrt{2}}(f + g)$	donde $\forall a \in [\![\Gamma]\!]$ $f = [\![\Gamma \vdash t : \sigma]\!]^{\mathcal{Q}}$ $g = [\![\Gamma \vdash u : \sigma]\!]^{\mathcal{Q}}$

Tipo	Función
$({}^{\mathcal{Q}}f \times^{\mathcal{Q}} g) : (S_1 \times S_2) \to R_5$ $R_5 = (T_1 \times T_2) \to \mathbb{C}$	$({}^{\mathcal{Q}}f \times^{\mathcal{Q}} g)(a,b)(x,y) = ({}^{\mathcal{Q}}f\ a\ x *^{\mathcal{Q}} g\ b\ y)$
$({}^{\mathcal{Q}}f \times^{\mathcal{Q}} g)^* : R_6 \to R_5$	$({}^{\mathcal{Q}}f \times^{\mathcal{Q}} g)^*(v_{(a',b')})\ (x,y) =$
$R_6 = V\ (S_1 \times S_2)$	$\sum_{a,b} \left(v_{(a',b')}\ (a,b)\right) * \left(({}^{\mathcal{Q}}f \times^{\mathcal{Q}} g)\ (a,b)\ (x,y)\right)$
	$({}^{\mathcal{Q}}f^{-1} \times^{\mathcal{Q}} g^{-1})\ (v_{(a,b)}) = (f^{-1}\ a, g^{-1}\ b)$
	$({}^{\mathcal{Q}}f^{-1} \times^{\mathcal{Q}} g^{-1})^*\ (v_{(a,b)}) = v_{(f^{-1}a, g^{-1}b)}$
$({}^{\mathcal{Q}}f |^{\mathcal{Q}} g) : R_7 \to VT$	$({}^{\mathcal{Q}}f |^{\mathcal{Q}} g)(1,a) = ({}^{\mathcal{Q}}fa)$
$R_7 = ([\![Q_2]\!] \times S)$	$({}^{\mathcal{Q}}f |^{\mathcal{Q}} g)(0,a) = ({}^{\mathcal{Q}}g\ a)$
$({}^{\mathcal{Q}}f |^{\mathcal{Q}} g)^* : V(R_7) \to R_8$	$({}^{\mathcal{Q}}f |^{\mathcal{Q}} g)^*\ (v_{(x',a')}(y)) = \sum_{x,a} (v_{(x',a')}(x,a)) *$
$R_8 = T \to \mathbb{C}$	$\left(({}^{\mathcal{Q}}f |^{\mathcal{Q}} g)(x,a)\right)$
$({}^{\mathcal{Q}}f^{-1} |^{\mathcal{Q}} g^{-1}) : VT \to R_7$	$(f^{-1}|g^{-1})(v_a) = \begin{cases}(a, f^{-1}\ a), \text{ si } a = 1\\(a, g^{-1}\ a), \text{ si } a = 0\end{cases}$
$({}^{\mathcal{Q}}f^{-1} |^{\mathcal{Q}} g^{-1})^* : R_9$	
$R_9 = VT \to V(R_7)$	$({}^{\mathcal{Q}}f^{-1} |^{\mathcal{Q}} g^{-1})({}_{v_a}) = \begin{cases}(v_{(a,f^{-1}a)}, \text{ si } a = 0\\v_{(a,g^{-1}a)}, \text{ si } a = 1\end{cases}$

- ${}^{\mathcal{Q}}\delta_{\Gamma,\Delta}$, separa los contextos de tipo, ${}^{\mathcal{Q}}\delta_{\Gamma,\Delta} : [\![\Gamma \otimes \Delta]\!] \to V\ ([\![\Gamma]\!] \times [\![\Delta]\!])$ definido como: ${}^{\mathcal{Q}}\delta_{\Gamma,\Delta}(a \otimes a') = V_{(a,a')}$. Si uno de los contextos es vacío $\bullet$, entonces se aplica ${}^{\mathcal{Q}}\delta_{\Gamma,\Delta}(a) = v_a$. Con lift ${}^{\mathcal{Q}}\delta^*_{\Gamma,\Delta}(v_{(a\otimes a')} = v_{(a',a')}$ o ${}^{\mathcal{Q}}\delta^*_{\Gamma,\Delta}(v_a) = v_a$, según sea el caso; las funciones inversas son obtenidas similarmente a los casos previos.

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

Con las funciones anteriores se pueden establecer ciertos programas, sin embargo, aún se requiere anexar las reglas semánticas para términos cuánticos. Las reglas para formar términos bien tipados y la semántica para datos cuánticos se encuentran en la Tabla IV y la Tabla V, respectivamente. Con base en lo anterior, se procede a describir la estrategia para generar una pila de historial.

B. Semántica operacional para control cuántico e historial

La semántica ahora considera el vector cero $\overrightarrow{0}$, $\kappa * t$ y la superposición $t + u$.

TABLA VI
PRODUCTO INTERNO Y ORTOGONALIDAD [20].

$\langle t|t\rangle = 1$ if $t \neq \overrightarrow{0}$	$\langle \lambda * t + \lambda' * t'|u\rangle = \overline{\lambda} * \langle t|u\rangle + \overline{\lambda'} * \langle t'|u\rangle$
$\langle t|\kappa * u + \kappa' * u'\rangle = \kappa * \langle t|u\rangle + \kappa' * \langle t|u'\rangle$	$\langle true|false\rangle = 0$
$\langle \lambda * t|u\rangle = \overline{\lambda}\,\langle t|u\rangle$	
$\langle \overrightarrow{0}|true\rangle = 0 = \langle true|\overrightarrow{0}\rangle$	$\langle t|\lambda * u\rangle = \lambda * \langle t|u\rangle$
$\langle \overrightarrow{0}|false\rangle = 0 = \langle false|\overrightarrow{0}\rangle$	$\langle t + t'|u\rangle = \langle t|u\rangle + \langle t'|u\rangle$
$\langle \overrightarrow{0}|x\rangle = 0 = \langle x|\overrightarrow{0}\rangle$	$\langle t|u + u'\rangle = \langle t|u\rangle + \langle t|u'\rangle$
$\langle (t,t')|(u,u')\rangle = \langle t|u\rangle * \langle t'|u'\rangle$	$\langle t|u\rangle =?$ otro caso

En los programas que contienen las expresiones $\textbf{if}^{\circ}$ o $t + u$, los términos que los conforman deben ser ortogonales entre sí, por ejemplo, en la superposición t debe ser ortogonal con u, denotado como $t \perp u$; para indicar que la norma de un término cuántico es 1 y que es libre de mediciones, se utiliza el símbolo $\vdash^{\circ}$. En la Tabla VI, se encuentran las reglas de ortogonalidad, donde el producto interno se denota como $\langle\ |\ \rangle$.

A partir de lo anterior, ahora se puede aplicar el modelo a un programa simple, por ejemplo: $[\![\bullet \otimes \bullet \vdash \textbf{if}^{\circ}\ true\ \textbf{then}\ (-1) * true + false\ \textbf{else}\ true + false : Q_2]\!]^{\mathrm{Q}} = (g|h)^* \circ (f \times \boldsymbol{\mathcal{Q}id_{-}^{-1}}; {}^{\mathcal{Q}}id)^* \circ (\boldsymbol{\mathcal{Q}\delta_{\Gamma,\Delta}^{-1}}; {}^{\mathcal{Q}}\delta_{\Gamma,\Delta})$. En este punto, las funciones f se mezclan con sus inversas f^{-1}, lo cual, para nuestros propósitos estas deben estar separadas. El cómo lograr esto, se encuentra en la sección IV.

IV. HISTORIAL Y REVERSIBILIDAD

Para aplicar reversibilidad, varios puntos son importantes: un concepto llamado pila de historial y retomar las funciones inversas definidas en esta propuesta.

Teniendo en cuenta que cada derivación se da a partir de funciones y si sus inversas se almacenan en una pila, entonces éstas darán como resultado reversibilidad, permitiendo retornar al argumento inicial del programa. Formalmente, los términos y la pila del historial crean un estado computacional, es decir [18]:

Definición 2 (Estado computacional): El *estado computacional* es una secuencia de la forma:

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

donde $\boldsymbol{h_{t_n}-; \cdots ; h_{t_2}-; h_{t_1}-}$, es llamada *pila o pista de historial* y $t_1 \circ t_2; \circ \cdots \circ t_n$, se denomina *registro computacional*.

Con respecto a $\boldsymbol{h_{t_i}}-;t_i$, la parte $\boldsymbol{h_{t_i}}$ representa la función inversa de la función t_i.

Definición 3: H, denota la pista de historial simplificada.

A partir de la semántica propuesta, al formar un programa se obtiene una expresión y para que ésta tenga la forma de un estado computacional, los términos deben ser *factorizados*, es decir, separar y organizar las funciones inversas de las directas.

A. Factorización

Sea la expresión $\boldsymbol{h_{t_1}}-;t_1\circ\boldsymbol{h_{t_2}}-;t_2\circ\cdots\circ\boldsymbol{h_{t_n}}-;t_n$, donde el punto y coma (**;**) denota la separación de funciones. Las expresiones $(h_{t_i}-;t_i)$ pueden ser para los casos clásico y cuántico, y tener diferentes formas dependiendo del programa, para lo cual, se consideran los siguientes casos:

1) $H;\big(\boldsymbol{h_t}-;t\big)(a)$ factorizado como: $H;\boldsymbol{h_t}-;\big(t(a)\big)$
2) $H;\big(\boldsymbol{h_{t_1}}-;t_1\ \times\ \boldsymbol{h_{t_2}}-;t_2\big)(a,b)$ factorizado como: $H;\boldsymbol{h_{t_1}}-\times\boldsymbol{h_{t_2}}-;\big(t_1\times\ t_2\big)(a,b)$
3) $H;\big(\boldsymbol{h_{t_1}}-;t_1\ +\ \boldsymbol{h_{t_2}}-;t_2\big)(a)$ factorizado como: $H;\boldsymbol{h_{t_1}}-+\boldsymbol{h_{t_2}}-;\big(t_1(a)+t_2(a)\big)$
4) $H;\big(\kappa*\boldsymbol{h_{t_1}}-;t_1\big)(a)$ factorizado como: $H;\boldsymbol{h_{t_1}}-;(\kappa*t_1(a))$, donde $\kappa\in\mathbb{C}$
5) Si t,t' son superposiciones o tuplas, entonces
$$H;\big(\boldsymbol{h_t}-;t|\boldsymbol{h_{t'}}-;t'\big)(a,b)=\begin{cases}H;\boldsymbol{h_t}-;t(b),\text{si } a=1\\ H;\boldsymbol{h_{t'}}-;t'(b),\text{si } a=0\end{cases}$$
sino $H;\big(\boldsymbol{h_t}-;t|\boldsymbol{h_{t'}}-;t'\big)(a,b)\ =\ H;(\boldsymbol{h_t}-|\boldsymbol{h_{t'}}-);\big((t|t')(a,b)\big)$, y la definición del condicional es aplicado.

Cuando la expresión tiene la forma $(h_{t_i}-;t_i)^*$, tener en cuenta los siguientes casos:

1) $H;\big(\boldsymbol{h_t}-;t\big)^*(v_a)$ factorizado como: $H;\boldsymbol{h_t}-^*;\big(t(v_a)\big)^*$
2) $H;\big(\boldsymbol{h_{t_1}}-;t_1\ \times\ \boldsymbol{h_{t_2}}-;t_2\big)^*(v_{(a,b)})$ factorizado como: $H;(\boldsymbol{h_{t_1}}-\times\boldsymbol{h_{t_2}}-)^*;\big(t_1\times\ t_2\big)^*(v_{(a,b)})$
3) $H;\big(\boldsymbol{h_{t_1}}-;t_1\ +\ \boldsymbol{h_{t_2}}-;t_2\big)^*(v_a)$ factorizado como: $H;\boldsymbol{h^*_{t_1}}-+\boldsymbol{h^*_{t_2}}-;\big(t_1+t_2\big)^*(v_a)$
4) $H;\big(\kappa*\boldsymbol{h_{t_1}}-;t_1\big)^*(v_a)$ factorizado como: $H;\boldsymbol{h_{t_1}}-^*;\big(\kappa*t_1\big)^*(v_a)$, donde $\kappa\in\mathbb{C}$
5) Para el condicional $(f|g)^*$: Si t,t' son superposiciones o tuplas, implica:
$$(\boldsymbol{h_t}-;t|\boldsymbol{h_{t'}}-;t')^*(v_{(a,b)})=\begin{cases}H;\boldsymbol{h_t}-^*;t(b),\ \text{si } a=1\\ H;\boldsymbol{h_{t'}}-^*;t'(b),\text{si } a=0\end{cases}$$

Para cualquier término con o sin lift $(.^*)$, la factorización se inicia de derecha a izquierda:

$$\begin{aligned}\boldsymbol{h_{t_1}}-;t_1\circ\boldsymbol{h_{t_2}}-;t_2\circ\cdots\circ\boldsymbol{h_{t_n}}-;t_n=&\boldsymbol{h_{t_n}}-;\big(\boldsymbol{h_{t_1}}-;t_1\circ\\ &\boldsymbol{h_{t_2}}-;t_2\circ\cdots\circ t_n\big)\\ =&\boldsymbol{h_{t_n}}-;\boldsymbol{\cdots};\boldsymbol{h_{t_2}}-;\\ &\big(\boldsymbol{h_{t_1}}-;t_1\circ t_2\circ\cdots\circ t_n\big)\\ =&\underbrace{\boldsymbol{h_{t_n}}-;\boldsymbol{\cdots};\boldsymbol{h_{t_2}}-;\boldsymbol{h_{t_1}}-;}_{\text{Pila de historial}(H)}\\ &\underbrace{\big(t_1\circ t_2\circ\cdots\circ t_n\big)}_{\text{Registro computacional}}\end{aligned}$$

Si existe un condicional $(t|t')$, donde t y t', son superposiciones; sus funciones inversas correspondientes incorporarán a la pista una vez que se haya evaluado el condicional. Con lo anterior tenemos el estado computacional y se continua con las reglas propuestas para la reversibilidad.

B. Reversibilidad

La reversibilidad puede ejecutarse a partir del hecho de que si tenemos un estado y ejecutamos todas las funciones del registro computacional obtendremos un valor final q, es decir, $\boldsymbol{h_{t_n}}-;\boldsymbol{\cdots};\boldsymbol{h_{t_2}}-;\boldsymbol{h_{t_1}}-;\big(t_1\circ t_2\circ\cdots\circ t_n\big)(a)=\cdots=\boldsymbol{h_{t_n}}-;\boldsymbol{\cdots};\boldsymbol{h_{t_2}}-;\boldsymbol{h_{t_1}}-;q$, donde q es irreducible y también podría ser un vector (para el modelo semántico).

Dado un historial $h_{t_i}-$, se aplica como argumento q al historial inmediato anterior, remplazándolo por el símbolo $\_$ y aplicando la función inversa respectiva, esto es, $h_{t_i}-;a=h_{t_i}(a)$. La reversibilidad se puede aplicar considerando los siguientes casos:

1) $h_{t_i}-;q=h_{t_i}(q)$.
2) $(h_{t_i}-\times h_{t_j}-)\ (q_1,q_2)=(h_{t_i}\ q_1,h_{t_j}\ q_2)$
3) $({}^{\mathcal{Q}}h_{t_i}-\times^{\mathcal{Q}}h_{t_j}-)^*\ v_{(x,y)}=v_{({}^{\mathcal{Q}}h_{t_i(x)},{}^{\mathcal{Q}}h_{t_j(y)})}$
4) $(h_{t_i}-+h_{t_j}-)\ \alpha*(q_1+q_2)=(h_{t_i}\ q_1,h_{t_j}\ q_2)$
5) $({}^{\mathcal{Q}}h_{i}-+^{\mathcal{Q}}h_{j}-)^*\ \alpha*(v_x+v_y)=v_{({}^{\mathcal{Q}}h_{i(v_x)},\ {}^{\mathcal{Q}}h_{j(v_y)})}$
6) $h_{t_1}-;(\alpha*q)=h_{t_1}(q)$

donde $\alpha\in\mathbb{C}$ se descartan.

Con las propuestas anteriores, se muestra un ejemplo.

V. Ejemplo: Compuerta de Hadamard

$had\ true=$
$\mathbf{if}^{\circ}\ true\ \mathbf{then}\ (-1)*true+false\ \mathbf{else}\ true+false.$

$[\![\bullet\otimes\bullet\vdash had\ true:Q_2]\!]^{\mathcal{Q}}=(g|h)^*\circ(f\times{}^{\mathcal{Q}}\boldsymbol{id}_-^{-1};{}^{\mathcal{Q}}id)^*\circ({}^{\mathcal{Q}}\boldsymbol{\delta_{\Gamma,\Delta}})_-^{-1};{}^{\mathcal{Q}}\delta_{\Gamma,\Delta})$

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

$$
\begin{aligned}
&= H; \Big(\frac{1}{\sqrt{2}}\big((-1) * \mathbb{Q}\boldsymbol{const}\ \mathbf{1}_{-}^{*-1}; {}^{\mathbb{Q}}const\ 1 + \\
&\quad \mathbb{Q}\boldsymbol{const}\ \mathbf{0}_{-}^{*-1}; {}^{\mathbb{Q}}const\ 0\big)\Big)(0) \\
&= H; (\mathbb{Q}\boldsymbol{const}\ \mathbf{1}_{-}^{*-1} + \mathbb{Q}\boldsymbol{const}\ \mathbf{0}_{-}^{*-1}); \\
&\quad \Big(\frac{1}{\sqrt{2}}\big((-1) * {}^{\mathbb{Q}}const\ 1 + {}^{\mathbb{Q}}const\ 0\big)\Big)(0) \\
&= H; \frac{1}{\sqrt{2}}\Big((-1) * {}^{\mathbb{Q}}const\ 1\ (0) + {}^{\mathbb{Q}}const\ 0\ (0)\Big) \\
&= H; \frac{1}{\sqrt{2}}\Big((-1) * v_1 + v_0\Big)
\end{aligned}
$$

Explícitamente H es: $(\mathbb{Q}\boldsymbol{id}_{-}^{+-1}); (\mathbb{Q}\boldsymbol{const}\ \mathbf{1}_{-}^{-1} \times^{\mathbb{Q}} \boldsymbol{id}_{-}^{-1})^*;$ $(\mathbb{Q}\boldsymbol{const}\ \mathbf{1}_{-}^{*-1} + \mathbb{Q}\boldsymbol{const}\ \mathbf{0}_{-}^{*-1})$; y aplicando reversibilidad se tiene:

$$
\begin{aligned}
&(\mathbb{Q}\boldsymbol{id}_{-}^{+-1}); (\mathbb{Q}\boldsymbol{const}\ \mathbf{1}_{-}^{-1} \times \mathbb{Q}\boldsymbol{id}_{-}^{-1})^*; \\
&\qquad (\mathbb{Q}\boldsymbol{const}\ \mathbf{1}_{-}^{*-1} + \mathbb{Q}\boldsymbol{const}\ \mathbf{0}_{-}^{*-1}) \frac{1}{\sqrt{2}}\big((-1) * v_1 + v_0\big) \\
&= (\mathbb{Q}\boldsymbol{id}_{-}^{+-1}); (\mathbb{Q}\boldsymbol{const}\mathbf{1}_{-}^{-1} \times \mathbb{Q}\boldsymbol{id}_{-}^{-1})^*; (\mathbb{Q}\boldsymbol{const}\ \mathbf{1}_{-}^{*-1} \\
&\qquad + \mathbb{Q}\boldsymbol{const}\ \mathbf{0}_{-}^{*-1})\ \ (v_1 + v_0) \\
&= (\mathbb{Q}\boldsymbol{id}_{-}^{+-1}); (\mathbb{Q}\boldsymbol{const}\ \mathbf{1}_{-}^{-1} \times \mathbb{Q}\boldsymbol{id}_{-}^{-1})^* \\
&\qquad v_{({}^{\mathbb{Q}}const\ 1^{-1}(v_1), {}^{\mathbb{Q}}const\ 0^{-1}(v_0))} \\
&= (\mathbb{Q}\boldsymbol{id}_{-}^{+-1}); (\mathbb{Q}\boldsymbol{const}\ \mathbf{1}_{-}^{-1} \times \mathbb{Q}\boldsymbol{id}_{-}^{-1})^*\ v_{(0,0)} \\
&= (\mathbb{Q}\boldsymbol{id}_{-}^{+-1}); v_{(const\ 1^{-1}\ 0,\ id_{-}^{-1}\ 0)} \\
&= {}^{\mathbb{Q}}id^{+-1}\ v_{(1,0)} = 1
\end{aligned}
$$

A través de este ejemplo, se observa cómo se ejecuta la reversibilidad y la aplicación de las reglas definidas. A continuación se procede con las conclusiones finales.

VI. Conclusiones

En este artículo se presenta un mecanismo que permite reversibilidad computacional en el lenguaje de programación cuántico QML. Los fundamentos del lenguaje se retoman y partiendo de su semántica (dada por funciones), se modificaron sus reglas tal que para cada función ejecutada en un programa, se vincule y guarde la operación inversa respectiva.

Las funciones inversas al almacenarse generan una pila de historial, esta pila requirió de reglas para determinar cómo trabajar y recibir los argumentos, para gradualmente regresar al valor inicial dado cualquier programa, es decir, si se conoce la salida final de un programa, entonces semánticamente resulta posible obtener la entrada y por lo tanto este proceso es reversible.

Las funciones y reglas para reversibilidad siguen involucrando Tripletas de Kleisli, las cuales permiten operar sobre datos cuánticos; destacando que parte de las dificultades de la propuesta fue definirlas adecuadamente. Finalmente, se concluye que con las definiciones establecidas, el lenguaje QML tiene explícitamente reversibilidad computacional para datos clásicos y cuánticos, omitiendo mediciones. Considerando que la motivación de este trabajo fue el lenguaje cálculo lambda cuántico.

Apéndice A
Funciones auxiliares

Sea $a, a', b \in S$, $S = \{0, 1\}$ [20].

- $id : S \to V\ S$, definido como: $id(a) = a$
- $id^+ : S \to [\![Q_1]\!] \times S$, definido como: $id^+(a) = (0, a)$
- $id_+ : [\![Q_1]\!] \times S \to S$, definido como: $id_+(0, a) = a$
- $const\ a : [\![Q_1]\!] \to S$, definido como: $const\ a(0) = a$
- $\delta : S \to S \times S$, definido como: $\delta(a) = (a, a)$
- $swap : S \times T \to T \times S$, definido como: $swap(a, b) = (b, a)$
- Sean las funciones $f : S_1 \to T_1$ y $g : S_2 \to T_2$, por lo tanto, $f \times g : S_1 \times S_2 \to T_1 \times T_2$, definidas como: $f \times g\ (a, b) = (f\ a, g\ b)$
- $\delta_{\Gamma,\Delta} : [\![\Gamma \otimes \Delta]\!] \to [\![\Gamma]\!] \times [\![\Delta]\!]$ definidas por inducción como:

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

- Para dos funciones cualquiera $f, g \in S \to T$, $(f|g) \in S \to T$, se define:

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

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

Referencias

[1] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.

[2] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," 1994.

[3] M. Ying, *Foundations of Quantum Programming*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2016.

[4] R. J. Lipton and K. W. Regan, *Quantum Algorithms via Linear Algebra: A Primer*. The MIT Press, 2014.

[5] J. A. Miszczak, "High-level structures for quantum computing," *Synthesis Lectures on Quantum Computing*, vol. 4, 05 2012.

[6] J. Palsberg, "Toward a universal quantum programming language," *XRDS: Crossroads, The ACM Magazine for Students*, vol. 26, pp. 14–17, 09 2019.

[7] W.-L. Chang and A. V. Vasilakos, *Fundaments of Quantum Programming in IBM's Quantum Computers*. Springer International Publishing, 01 2021.

[8] B. Ömer, "A procedural formalism for quantum computing," *AIP Conference Proceedings*, vol. 627, no. 1, p. 276, 2002.

[9] H. Mlnar'k, "Quantum programming language LanQ," September 2007.

[10] J. W. Sanders and P. Zuliani, "Quantum programming," in *Mathematics of Program Construction*, R. Backhouse and J. N. Oliveira, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 80–99.

[11] S. J. Gay, "Quantum programming languages: Survey and bibliography," *Mathematical. Structures in Comp. Sci.*, vol. 16, no. 4, pp. 581–600, 2006.

[12] P. Jorrand and M. Lalire, *From Quantum Physics to Programming Languages: A Process Algebraic Approach*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 1–16.

[13] P. Selinger, "A brief survey of quantum programming languages," in *In Proceedings of the 7th International Symposium on Functional and Logic Programming*. Springer, 2004.

[14] ——, "Towards a semantics for higher-order quantum computation," in *Proceedings of the 2nd International Workshop on Quantum Programming Languages*, 2004, pp. 127–143.

[15] M. Lampis, K. G. Ginis, M. A. Papakyriakou, and N. S. Papaspyrou, "Quantum data and control made easier," *Electron. Notes Theor. Comput. Sci.*, vol. 210, pp. 85–105, Jul. 2008.

[16] P. Fu, K. Kishida, and P. Selinger, "Linear dependent type theory for quantum programming languages," 2020.

[17] P. Sestoft, *Programming Language Concepts*. Springer, 01 2017.

[18] A. v. Tonder, "A lambda calculus for quantum computation," *SIAM J. Comput.*, vol. 33, no. 5, pp. 1109–1135, 2004.

[19] J. J. Grattage, "A functional quantum programming language," 2006.

[20] T. Altenkirch, J. Grattage, J. K. Vizzotto, and A. Sabry, "An algebra of pure quantum programming," *Electronic Notes in Theoretical Computer Science*, vol. 170, pp. 23 – 47, 2007, proceedings of the 3rd International Workshop on Quantum Programming Languages (QPL 2005).

[21] T. Altenkirch and J. Grattage, "A functional quantum programming language," in *20th Annual IEEE Symposium on Logic in Computer Science (LICS' 05)*, June 2005, pp. 249–258.

[22] J. Carette, C.-H. Chen, V. Choudhury, and A. Sabry, "From reversible programs to univalent universes and back," *Electronic Notes in Theoretical Computer Science*, vol. 336, pp. 5 – 25, 2018, the Thirty-third Conference on the Mathematical Foundations of Programming Semantics (MFPS XXXIII).

[23] R. Wille, E. Schönborn, M. Soeken, and R. Drechsler, "Syrec: A hardware description language for the specification and synthesis of reversible circuits," *Integration, the VLSI Journal*, vol. 53, 11 2015.

[24] D. McMahon, *Quantum Computing Explained*. John Wiley & Sons, 2007.

[25] P. Selinger, "A finite alternation result for reversible boolean circuits," *Science of Computer Programming*, vol. 151, p. 2–17, Jan 2018. [Online]. Available: http://dx.doi.org/10.1016/j.scico.2017.08.011

[26] R. Landauer, "Irreversibility and heat generation in the computing process," *IBM Journal of Research and Development*, vol. 5, no. 3, pp. 183–191, July 1961.

[27] N. Nishida, A. Palacios, and G. Vidal, "Reversible computation in term rewriting," *Journal of Logical and Algebraic Methods in Programming*, vol. 94, pp. 128 – 149, 2018.

[28] S. Abramsky, "A structural approach to reversible computation," *Theoretical Computer Science*, vol. 347, no. 3, pp. 441 – 464, 2005.

[29] C. H. Bennett, "Logical reversibility of computation," *IBM J. Res. Dev.*, vol. 17, no. 6, pp. 525–532, 1973.

[30] J. Preskill, *Lecture Notes for Physics 229:Quantum Information and Computation*. CreateSpace Independent Publishing Platform, 2015.

[31] R. Glück and R. Kaarsgaard, "A categorical foundation for structured reversible flowchart languages," *Electronic Notes in Theoretical Computer Science*, vol. 336, pp. 155 – 171, 2018, the Thirty-third Conference on the Mathematical Foundations of Programming Semantics (MFPS XXXIII).

[32] A. S. Green and T. Altenkirch, "From reversible to irreversible computations," *Electronic Notes in Theoretical Computer Science*, vol. 210, pp. 65 – 74, 2008, proceedings of the 4th International Workshop on Quantum Programming Languages (QPL 2006).

[33] A. D. Vos, "Reversible computer hardware," *Electronic Notes in Theoretical Computer Science*, vol. 253, no. 6, pp. 17 – 22, 2010, proceedings of the Workshop on Reversible Computation (RC 2009).

[34] A. B. Matos, "Linear programs in a simple reversible language," *Theoretical Computer Science*, vol. 290, no. 3, pp. 2063 – 2074, 2003.

[35] T. Yokoyama, "Reversible computation and reversible programming languages," *Electronic Notes in Theoretical Computer Science*, vol. 253, no. 6, pp. 71 – 81, 2010, proceedings of the Workshop on Reversible Computation (RC 2009).

[36] C. Heunen and M. Karvonen, "Reversible monadic computing," *Electronic Notes in Theoretical Computer Science*, vol. 319, pp. 217 – 237, 2015, the 31st Conference on the Mathematical Foundations of Programming Semantics (MFPS XXXI).

[37] L. Paolini, M. Piccolo, and L. Roversi, "A class of reversible primitive recursive functions," *Electronic Notes in Theoretical Computer Science*, vol. 322, pp. 227 – 242, 2016, proceedings of ICTCS 2015, the 16th Italian Conference on Theoretical Computer Science.

[38] E. Moggi, "Computational lambda-calculus and monads," in *Proceedings of the Fourth Annual Symposium on Logic in Computer Science*. IEEE Press, 1989, pp. 14–23.

[39] S. Awodey, *Category Theory*, ser. Oxford Logic Guides. Oxford University Press, 2006.

Nely Plata-Cesar Recibí el grado de Maestra en Ciencias de la Ingeniería por la Universidad Autónoma del Estado de México y actualmente estudiante de Doctorado y Profesora en la Facultad de Ingeniería de la Universidad Autónoma del Estado de México. Mi área de interés es el cómputo cuántico, previamente trabajé autómatas cuánticos y actualmente en lenguajes de programación cuánticos, reversibilidad e historial de cálculos.

José Raymundo Marcial-Romero Actualmente profesor investigador de tiempo completo en la Facultad de Ingeniería de la Universidad Autónoma del Estado de México. Miembro del Sistema Nacional de Investigadores del Consejo Nacional de Ciencia y Tecnología , Nivel 1. En el año 2000 obtuvo el título de Licenciatura en Ciencias de la Computación y en el año 2007 el grado de Doctor en Ciencias de la Computación por The University of Birmingham UK en The School of Computer Science.

José Antonio Hernández-Servín Actualmente profesor investigador de tiempo completo en la Facultad de Ingeniería de la Universidad Autónoma del Estado de México. Miembro del Sistema Nacional de Investigadores del Consejo Nacional de Ciencia y Tecnología, Nivel 1. En 1999 obtuvo el título de Licenciado en Ciencias Físico-Matématicas y termino la Maestría en Matemáticas Puras en 2001 en la UMSNH (Univ. Aut. San Nicolás de Hidalgo). Para el año 2005, obtiene el Doctorado en Ciencias (PhD) por The University of Nottingham, UK, en The School of Electrical and Electronic Engineering, en el departamento de Óptica.