ISSN 1870-4069

Historial y reversibilidad en el sublenguaje clásico QML

Nely Plata César, José Raymundo Marcial Romero

Universidad Autónoma del Estado de México, México

Resumen. La presente investigación incorpora un historial de cálculos, ajusta el modelo operacional y define las reglas para aplicar reversibilidad en el lenguaje de programación cuántico QML. Sólo se abordarán las instrucciones clásicas del lenguaje mencionado. Esto será el preámbulo para incorporar historial y reversibilidad contemplando datos y control cuántico.

Palabras clave: lenguaje de programación cuántico, QML, reversibilidad, historial.

History and Reversibility in the Classic Sublenguage QML

Abstract. The present investigation incorporates a history of calculations, adjusts the operational model and of ne the rules to apply reversibility in the quantum programming language QML. Only the classic instructions of the aforementioned language will be addressed. This will be the preamble to incorporate history and reversibility contemplating data and quantum control.

Keywords: quantum programming language, QML, reversibility, history.

1. Introducción

El cómputo cuántico se enfoca en aplicar los principios de la mecánica cuántica, de tal manera que sus propiedades se hereden a la computación y al desarrollo de la misma. Las aplicaciones y cálculos que se pueden realizar bajo estos sistemas son múltiples; por ejemplo, el estudio y desarrollo de lenguajes de programación cuánticos. Actualmente, debido a que no se cuenta con una computadora cuántica, estos lenguajes están siendo implementados en máquinas clásicas [14,7,18,3,20]. Existen cuatro principios o postulados que rigen el comportamiento del cómputo cuántico:

El primero, son estados puros; es decir, utilizar vectores unitarios, los cuales se encargarán de almacenar la información en memoria. La evolución del sistema indica cómo cambiar de un estado a otro, para lo cual se aplican matrices unitarias, éstas tienen implícitas las instrucciones que se desean que efectúe la

computadora y serán aplicadas al estado que se desea evolucionar. El tercero es el axioma de medición u observación, se centra en que existe un observador que al ver el estado del sistema éste se colapsa a ese valor y se determina con qué probabilidad sucede. Por último, el cuarto principio es el de sistemas compuestos, encargado de definir estados conformados por distintos sistemas y el cómo interactúan entre ellos [13,15].

Una propiedad que debe tener un sistema cuántico, es la reversibilidad, es decir, si se está en un punto de algún cálculo, se puede regresar al paso anterior y así sucesivamente (sin mediciones intermedias). Esto se logra debido a que las operaciones están representadas por matrices unitarias y éstas tienen implícita tal propiedad; entonces, al aplicar dos veces la misma matriz que llevó a un estado actual de la computadora, se puede regresar al paso anterior.

Hay investigaciones que estudian cómo convertir cómputos clásicos o cuánticos en reversibles, así como técnicas para su construcción [16,1,10,8,22]. El definir explícitamente la reversibilidad permitirá reconstruir un programa o saber que sucedió antes de cualquier cálculo.

Una de las posibles ideas para poder aplicar explícitamente reversibilidad, es almacenar ordenadamente las operaciones ejecutadas durante las derivaciones de un programa, al resultado de lo que se guardó se le puede llamar historial [21]. Partiendo de lo anterior, el vínculo entre la reversibilidad e historial de cálculos es el área que se abordará, esto enfocado en el lenguaje de programación cuántico QML.

De manera general, en la literatura existen lenguajes de programación cuánticos basados en los paradigmas imperativo y funcional [14]. Dentro de los funcionales, se encuentran algunos como QPL, QML, cálculo lambda cuántico, nQML, entre otros [4,11,21,18,19].

El lenguaje QML, es propuesto por Grattage y Altenkirch, quienes desarrollan una semántica denotacional y operacional, aplican propiedades cuánticas, teoría de categorías y lógica [4,9]. Para abordar el cómo modelar la reversibilidad en QML, algunos autores han propuesto alternativas que no estrictamente van enfocadas a este lenguaje, pero resultan funcionales para nuestro propósito.

El artículo se estructura de la siguiente forma: En la sección 2 están las investigaciones de interés acerca de reversibilidad e historial, también se encontrará la definición del lenguaje de programación cuántico QML, puntualizando en el sublenguaje clásico del mismo. La sección 3, presenta nuestra aportación respecto al historial de cálculos y reversibilidad en QML, definiendo sus reglas de funcionamiento. En el bloque 4, está la propuesta del modelo operacional y la incorporación de las funciones inversas relacionadas al mismo, y, finalmente, la sección 5, presenta las conclusiones y trabajo a futuro.

2. Trabajo relacionado

En esta sección mencionaremos la investigación base para trabajar una pila de historial y con ésta aplicar reversibilidad, así como lo relacionado con QML, esto es, su sintaxis, tipos, reglas, etc.

2.1. Historial y reversibilidad

Uno de los pioneros en abordar la reversibilidad es Landauer [12], quien aporta la idea de que una computadora puede efectuar la reversibilidad si se guarda su historial de cómputos [17]. El historial, expresa o plasma los antecedentes de determinados procesos computacionales.

Otros investigadores sobresalientes son Abramsky y Bennett, quienes mostraron que el cómputo clásico puede ser transformado en cómputo reversible [2,6]. Partiendo de tales autores, Van Tonder [21] retoma sus ideas, en particular de Bennett [6], esto para incorporar el historial de operaciones en el lenguaje cálculo lambda cuántico. De esto, se define que la investigación de Van Tonder se considerará base para nuestra propuesta.

Posteriormente, Grattage, Altenkirch, Vizzotto y Sabry, retoman QML y desarrollan un modelo operacional y teoría ecuacional (omitiendo mediciones) abordando primero su parte clásica y los resultados extrapolados a lo cuántico [5]. Esta investigación es sobresaliente, porque determina una forma de trabajar un lenguaje cuántico; es decir, primero modelar cualquier propiedad en el entorno clásico, y una vez concluido, extenderlo a lo cuántico.

Descrito lo anterior, se establece que este artículo se enfocará en las dos investigaciones previas; es decir, Van Tonder y Grattage, Altenkirch, Vizzotto y Sabry, fusionándose para implementar una pila de historial en el sublenguaje clásico de QML, lo cual, será nuestro aporte central.

2.2. Sublenguaje clásico QML

Lenguaje que modela operaciones reversibles e irreversibles (a través de circuitos), utiliza datos cuánticos y control clásico, emplea teoría de categorías para modelar circuitos cuánticos, define una semántica operacional y denotacional, y desarrolla un simulador elemental en Haskell [9,4,20].

Para este caso se considera sólo el subconjunto de términos que excluyen elementos y superposiciones cuánticas. Este sublenguaje de forma precisa tiene su sintaxis, reglas para términos bien formados y su modelo operacional [5].

Su sintaxis se observa en la Tabla 1, los programas que se pueden formar son los convencionales como let, if, valores true, false, variables y combinación de estos. Los elementos de esta sintaxis tienen asociado un tipo, y los tipos están dados por la siguiente gramática $\sigma : \mathcal{Q}_1 | \mathcal{Q}_2 | \sigma \otimes \tau$. $\mathcal{Q}_2$ corresponde al qubit, es decir, el conjunto de 0 y 1, mientras $\mathcal{Q}_1$ no acarrea información y es el valor unitario denotado como $()$. Los tipos son finitos y no recursivos.

Los contextos de tipos (Γ, Δ) están dados por $\Gamma, x : \sigma = \bullet \mid \Gamma, x : \sigma$, donde, $\bullet$ es el conjunto vacío. Tales contextos se conciben como funciones que dado un conjunto finito de variables devuelven tipos; dentro de un programa se asume que cada variable definida aparece al menos una vez. Para mapear pares de contextos a contextos, se agrega el operador $\otimes$, realizando lo que se presenta en la Tabla 2.

Se define una expresión de la forma $\Gamma \vdash t : \sigma$, esto es, que dado Γ se deriva el término t del tipo σ. Ésta es interpretada por la función: $[\![\Gamma \vdash t : \sigma]\!] \in$

Tabla 1. Sintaxis del sublenguaje clásico QML.

(Variables)	$x, y, \ldots \in$	$Vars$
(Amplitudes de prob.)	$\kappa, \iota, \ldots \in$	$\mathbb{C}$
(Patrones)	$p, q ::=$	$x \mid (x, y)$
(Términos)	$t, u ::=$	x
		$\mid () \mid (t, u)$
		$\mid$ **let** $p = t$ **in** u
		$\mid$ **false** $\mid$ **true** $\mid$ **0**

Tabla 2. Mapeo de contextos a contextos.

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

$[\![\Gamma]\!] \to [\![\sigma]\!]$, tal que, un contexto devuelve un tipo de la coleccion de qubit. Y semánticamente los tipos se definen como:

$$[\![Q_1]\!] = \{0\},$$
$$[\![Q_2]\!] = \{0, 1\},$$
$$[\![\sigma \otimes \tau]\!] = [\![\sigma]\!] \times [\![\tau\ - NoValue -]\!],$$

donde el 0 se asocia con falso y 1 con verdadero. Una vez que se tiene la sintaxis y tipos, se pueden generar programas; sin embargo, hay reglas para definirlos adecuadamente y éstas se observan en la Figura 1, página 29 en [5].

2.3. Modelo operacional

Se mencionará el modelo operacional del sublenguaje clásico QML propuesto por los autores respectivos, éste describe las reglas de derivación para cada unos de los términos, está basado en composiciones y funciones auxiliares (página 30 en [5]) que permitirán ir evolucionando un programa hasta llegar a un resultado. Se presenta en la Tabla 3.

Considerar que a este modelo incorporaremos el historial de operaciones, implicando ajustar las funciones relacionadas con el mismo. Para observar su funcionamiento original se presenta el siguiente ejemplo (1) relacionado a la operación clásica *not*.

Ejemplo 1 $not\ false = \textbf{\textit{if}}^{\circ}\ false\ \textbf{\textit{then}}\ false\ \textbf{\textit{else}}\ true$
$[\![\bullet \otimes \bullet \vdash \textbf{\textit{if}}^{\circ} false\ \textbf{\textit{then}}\ false\ \textbf{\textit{else}}\ true : Q_2]\!] = (g|h) \circ (f \times id) \circ \delta_{\Gamma,\Delta}$

Esto es:

$$\begin{aligned}
[\![\bullet \otimes \bullet \vdash not\ false : Q_2]\!] &= \big(const\ 0 \mid const\ 1\big) \circ \big(const\ 0 \times id\big) \circ \big(id^*\big) \\
&= \big(const\ 0 \mid const\ 1\big) \circ \big(const\ 0 \times id\big) \circ id^*(0) \\
&= \big(const\ 0 \mid const\ 1\big) \circ \big(const\ 0 \times id\big)(0,0) \\
&= \big(const\ 0 \mid const\ 1\big) \circ \big(const\ 0\ (0),\ id(0)\big) \\
&= \big(const\ 0 \mid const\ 1\big)(0,0) \\
&= const\ 1\ 0 \\
&= 1
\end{aligned}$$

Devuelve el valor 1 (true) e invierte exitosamente el argumento inicial 0 (false).

Partiendo de lo anterior, procedemos con nuestra investigación, la cual consistirá en: agregar una pila de historial que se obtendrá a partir de modificar el modelo operacional, las reglas para aplicar reversibilidad y la incorporación de funciones inversas.

Tabla 3. Modelo operacional.

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

3. Historial y reversibilidad

En esta sección se encuentra una de nuestras aportaciones al implementar una pila de historial en el sublenguaje clásico QML.

Considerando que cada derivación está dada a partir de funciones auxiliares que se encargan de realizar ciertas operaciones, entonces, sus funciones inversas serán almacenadas en un historial. Esto se irá formalizando a continuación.

Definición 1 (Estado computacional) *Un* estado computacional *es una secuencia de la forma:*

$$\mathbf{h_{t_n}\_}; \cdots ; \mathbf{h_{t_2}\_}; \mathbf{h_{t_1}\_}; t_1 \circ t_2 \circ \cdots \circ t_n$$

donde, $\mathbf{h_{t_n}\_}; \cdots ; \mathbf{h_{t_2}\_}; \mathbf{h_{t_1}\_}$ *es la* pila del historial *y* $t_1 \circ t_2; \circ \cdots \circ t_n$ *el* registro computacional.

Definición 2 (Factorización) *Dada una expresión de la forma:*

$$\mathbf{h_{t_1}\_}; t_1 \circ \mathbf{h_{t_2}\_}; t_2 \circ \cdots \circ \mathbf{h_{t_n}\_}; t_n$$

el símbolo ***;*** *denota concatenación de expresiones. Las expresiones* h_{t_i}*, son historiales correspondientes a cada término* t_i*, respectivamente. Estos deben ser factorizados cómo:*

$$\mathbf{h_{t_1}\_}; t_1 \circ \mathbf{h_{t_2}\_}; t_2 \circ \cdots \circ \mathbf{h_{t_n}\_}; t_n = \mathbf{h_{t_n}\_}; \cdots ; \mathbf{h_{t_2}\_}; \mathbf{h_{t_1}\_}; t_1 \circ t_2 \circ \cdots \circ t_n$$

Es decir, los historiales h_{t_i} se colocan del lado izquierdo de la expresión, de manera inversa y consecutiva respecto al orden de aparición. Y respecto a $h_{t_i}\_; t_i$, la parte h_{t_i} representa y almacena la operación inversa que ejecuta t_i.

Propiedad 1 *Sea un estado computacional (resultado de derivaciones) con la forma:*

$$\mathbf{h_{t_n}\_}; \cdots ; \mathbf{h_{t_2}\_}; \mathbf{h_{t_1}\_}; q,$$

donde q *es un término irreducible (por el modelo operacional).*

A una expresión con esta forma, se puede aplicar *reversibilidad* a partir de la pila del historial $(h_{t_1}\_; h_{t_2}\_; \cdots ; h_{t_n}\_)$. Considerando los casos:

I) $h_{t_i}\_$. Pasar como argumento q al historial inmediato anterior, remplazandolo por el símbolo $\_$, y aplicando la función respectiva. Es decir:

$$h_{t_i}\_; q = h_{t_i}(q)$$

II) $(h_{t_i}\_ \times h_{t_j}\_)$. El argumento q debe ser una tupla (q_1, q_2), y se aplica al historial en la forma:

$$(h_{t_i}\_ \times h_{t_j}\_)(q_1, q_2) = (h_{t_i}\ q_1, h_{t_j}\ q_2)$$

Esto se ejemplifica de forma general a continuación.

Ejemplo 2 *Sea un estado computacional con el argumento inicial (p); posterior a la ejecución de todas las funciones t_i se tiene:*

$$\begin{aligned}\mathbf{h_{t_n}}-;\cdots;\mathbf{h_{t_2}}-;\mathbf{h_{t_1}}-;t_1\circ t_2\circ\cdots\circ t_n(p) &= \mathbf{h_{t_n}}-;\cdots;\mathbf{h_{t_2}}-;\mathbf{h_{t_1}}-;t_1\circ t_2;\circ\cdots\circ(p')\\ &= \vdots\\ &= \mathbf{h_{t_n}}-;\cdots;\mathbf{h_{t_2}}-;\mathbf{h_{t_1}}-;(p''')\end{aligned}$$

A partir de la pila del historial se puede aplicar reversibilidad, como:

$$\begin{aligned}\mathbf{h_{t_n}}-;\cdots;\mathbf{h_{t_2}}-;\mathbf{h_{t_1}}-;(p''') &= \mathbf{h_{t_n}}-;\cdots;\mathbf{h_{t_2}}-;\mathbf{h_{t_1}}(p''')\\ &= \mathbf{h_{t_n}}-;\cdots;\mathbf{h_{t_2}}-;(p'')\\ &= \mathbf{h_{t_n}}-;\cdots;\mathbf{h_{t_2}}(p'')\\ &= \mathbf{h_{t_n}}-;\cdots;(p')\\ &= \mathbf{h_{t_n}}-;(p')\\ &= \mathbf{h_{t_n}}(p')\\ &= (p)\end{aligned}$$

Retornando al valor inicial dado.

La siguiente etapa es implementar la pila del historial en el modelo operacional; para esto, se agregan las funciones inversas, esperando obtener el registro computacional. Esto de muestra a continuación.

4. Modelo operacional incorporando historial

Dado el modelo de la Tabla 3, las reglas de cada uno de los términos donde la función es explícita, se concatena su función inversa, por ejemplo $false$, $true$, x y otros casos. Con esto, la combinación de las funciones y sus inversas darán pie a un estado computacional, donde el historial se distingue del término por el símbolo; y el tipo de letra negra (Tabla 4).

Tabla 4. Modelo operacional con historial.

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

Para poder derivar adecuadamente un término, también se agregan las funciones inversas, esto con la idea de aplicar la operación que previamente se efectuó.

Funciones auxiliares inversas Sea $S = \{0, 1\}$, donde $a, a', b \in S$

- $\boldsymbol{id^{-1}} : S \to S$, por lo tanto, $id^{-1} = id$
- $\boldsymbol{(id^*)^{-1}} : S' \times S \to S$, donde $(id^*)^{-1} = id_*$

$$(id^*)^{-1}(a', a) = \begin{cases} (id^*)^{-1}(a', a) = a, \text{ si a'=0} \\ (id^*)^{-1}(a', a) = a', \text{ si a'=1} \end{cases}$$

- $\boldsymbol{(id_*)^{-1}} : S \to Q_1 \times S$, donde $(id_*)^{-1} = id^*$
- $\boldsymbol{(const\ 0)^{-1}} : S \to Q_1$, tal que

$$(const\ 1)^{-1}(a) = \begin{cases} 1, \text{ si a=0} \\ 0, \text{ si a=1} \end{cases}$$

- $\boldsymbol{\delta^{-1}} : (S, S) \to S$, donde $\delta^{-1}(a, a) = a$.
- $\boldsymbol{swap^{-1}} : T \times S \to S \times T$, donde $swap^{-1}(b, a) = (a.b)$.

- $\boldsymbol{(f \times g)^{-1}} : (T_1 \times T_2) \to (S_1 \times S_2)$,
  donde $(f \times g)^{-1} = (f^{-1} \times g^{-1})$, por lo tanto

$$(f \times g)^{-1}(a, b) = (f^{-1}\ a, g^{-1}\ b)$$

- $\boldsymbol{(\delta_{\Gamma,\Delta})^{-1}} : [\![\Gamma]\!] \times [\![\Delta]\!] \to [\![\Gamma \otimes \Delta]\!]$

$$(\delta_{\Gamma,\Delta})^{-1} = \begin{cases} (\delta_{\Gamma',\Delta'})^{-1} \times \delta^{-1} & \text{si } \Gamma = \Gamma', x : \sigma \text{ y } \Delta = \Delta', x : \sigma \\ (\delta_{\Gamma',\Delta})^{-1} \times id & \text{si } \Gamma = \Gamma', x : \sigma \text{ y } x \notin dom(\Delta) \\ id_* & \text{si } \Gamma = \bullet \end{cases}$$

- Condicional:
  $\boldsymbol{(f^{-1}|g^{-1})} : T \to ([\![Q_2]\!] \times S)$, donde:

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

Con esto, se conluyen las definiciones respecto a historial y reversibilidad. Para poder observar su funcionamiento, se procede a desarrollar un programa con los resultados previamente descritos.

4.1. Aplicación

Sea el programa *not* con reversibilidad:

$$not\ false = if^{\circ}\ false\ \textbf{then}\ false\ \textbf{else}\ true$$

$$[\![\bullet \otimes \bullet \vdash \textbf{if}^{\circ} false\ \textbf{then}\ false\ \textbf{else}\ true : Q_2]\!] = (g|h) \circ (f \times \boldsymbol{id_{-}^{-1}}; id) \circ (\boldsymbol{\delta_{\Gamma,\Delta}^{-1}}_{-}; \delta_{\Gamma,\Delta})$$

Donde:

$$\begin{aligned} f &= [\![\bullet \vdash false : Q_2]\!] = \boldsymbol{const\ 0_{-}}; const\ 0 \\ g &= [\![\bullet \vdash false : Q_2]\!] = \boldsymbol{const\ 0_{-}}; const\ 0 \\ h &= [\![\bullet \vdash true : Q_2]\!] = \boldsymbol{const\ 1_{-}}; const\ 1 \\ \delta_{\Gamma,\Delta} &= id^* \end{aligned}$$

Por lo tanto:

$$[\![\bullet \otimes \bullet \vdash not\ false : Q_2]\!] = \Big((\boldsymbol{const\ 0_{-}}; const\ 0)|(\boldsymbol{const\ 1_{-}}; const\ 1)\Big) \circ \Big((\boldsymbol{const\ 0_{-}}; const\ 0) \times (\boldsymbol{id_{-}}; id)\Big) \circ \Big((\boldsymbol{id_{*-}}; id^*)\Big)$$

Factorizando:

$$
\begin{aligned}
&= \Big((\boldsymbol{const\ 0\_} | \boldsymbol{const\ 1\_}); (const\ 0 | const\ 1) \Big) \circ \Big((\boldsymbol{const\ 0\_} \times \boldsymbol{id\_}); \\
&\quad (const\ 0 \times id) \Big) \circ \Big(\boldsymbol{id_{*}\_}; id^{*} \Big) \\
&= \Big(\boldsymbol{id_{*}\_; (const\ 0\_} \times \boldsymbol{id\_); (const\ 0\_} | \boldsymbol{const\ 1\_) } \Big); \\
&\qquad \Big((const\ 0 | const\ 1) \circ (const\ 0 \times id) \circ id^{*}(0) \Big) \text{ se pasa el argumento } 0 \\
&= \Big(\boldsymbol{id_{*}\_; (const\ 0\_} \times \boldsymbol{id\_); (const\ 0\_} | \boldsymbol{const\ 1\_) } \Big); \\
&\qquad \Big((const\ 0 | const\ 1) \circ (const\ 0 \times id)(0,0) \Big) \\
&= \Big(\boldsymbol{id_{*}\_; (const\ 0\_} \times \boldsymbol{id\_); (const\ 0\_} | \boldsymbol{const\ 1\_) } \Big); \\
&\qquad \Big((const\ 0 | const\ 1) \circ (const\ 0\ 0, \times id\ 0) \Big) \\
&= \Big(\boldsymbol{id_{*}\_; (const\ 0\_} \times \boldsymbol{id\_); (const\ 0\_} | \boldsymbol{const\ 1\_) } \Big); \\
&\quad \Big((const\ 0 | const\ 1)(0,0) \Big) \\
&= \Big(\boldsymbol{id_{*}\_; (const\ c\_} \times \boldsymbol{id\_); (const\ 0\_} | \boldsymbol{const\ 1\_) } \Big); \\
&\quad \Big((const\ 0 | const\ 1)(0,0) \Big) \\
&= \Big(\boldsymbol{id_{*}\_; (const\ c\_} \times \boldsymbol{id\_); (const\ 0\_} | \boldsymbol{const\ 1\_) } \Big); \Big(const\ 1\ (0) \Big) \\
&= \Big(\boldsymbol{id_{*}\_; (const\ c\_} \times \boldsymbol{id\_); (const\ 0\_} | \boldsymbol{const\ 1\_) } \Big); \Big(1 \Big)
\end{aligned}
$$

Aplicando **reversibilidad**: Si el argumento inicial fue 0 y se obtuvo 1, entonces se espera regresar a 0.

$$
\begin{aligned}
(\mathbf{id}_{*}\_); (\mathbf{const\ 0}\_ \times \mathbf{id}\_); (\mathbf{const\ 0}\_ | \mathbf{const\ 1}\_); (1) = (\mathbf{id}_{*}\_); (\mathbf{const\ 0}\_ \times \mathbf{id}\_); \\
(\mathbf{const\ 0}\_ | \mathbf{const\ 1}\_)(1)
\end{aligned}
$$

$$
\begin{aligned}
&= (\mathbf{id}_{*\_}); (\mathbf{const}\ \mathbf{0}_{\_} \times \mathbf{id}_{\_}); \\
&\quad (1, const\ 0\ 1) \\
&= (\mathbf{id}_{*\_}); (const\ 0_{\_} \times id_{\_}) \\
&\quad (1, 0) \\
&= (\mathbf{id}_{*\_}); (const\ 0\ 1,\ id\ 0) \\
&= (\mathbf{id}_{*\_})(0, 0) \\
&= id_{*}(0, 0) \\
&= 0
\end{aligned}
$$

Hasta este punto, se tiene incorporado el historial para el sublenguaje clásico de QML, permitiendo aplicar reversibilidad exitosamente.

5. Conclusiones y trabajo a futuro

Se consideró el sublenguaje clásico del lenguaje de programación cuántico QML, del cual, se retoma su modelo operacional respectivo, al cual, se incorporó un historial de cálculos.

El modelo operacional con historial, se desarrolla incorporando las funciones inversas que fueron aplicadas al derivar un programa. Las operaciones ejecutadas generaron un pila de historial, ésta permite la aplicación de reversibilidad.

La reversibilidad se ejecuta al aplicar cada uno de los historiales respectivos de la pila mencionada, permitiendo regresar a un paso anterior, y así sucesivamente, hasta el inicial.

Como trabajo a futuro, se sugiere agregar el historial al lenguaje completo de QML, considerando datos y control cuántico. Y definir la forma de modelar la reversibilidad a partir de operaciones cuánticas.

Referencias

1. Abdessaied, N., Amy, M., Drechsler, R., Soeken, M.: Complexity of reversible circuits and their quantum implementations. Theoretical Computer Science 618, 85–106 (2016)
2. Abramsky, S.: A structural approach to reversible computation. Theoretical Computer Science 347(3), 441–464 (2005)
3. Abramsky, S., Coecke, B.: Physical traces: Quantum vs. classical information processing. CoRR cs.CG/0207057 (2002)
4. 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)
5. 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), proceedings of the 3rd International Workshop on Quantum Programming Languages (QPL 2005)

6. Bennett, C.H.: Logical reversibility of computation. IBM J. Res. Dev. 17(6), 525–532 (Nov 1973)
7. Gay, S.J.: Quantum programming languages: Survey and bibliography. Mathematical. Structures in Comp. Sci. 16(4), 581–600 (Aug 2006)
8. Glück, R., Kaarsgaard, R.: A categorical foundation for structured reversible flowchart languages. Electronic Notes in Theoretical Computer Science 336, 155–171 (2018)
9. Grattage, J.J.: A functional quantum programming language
10. Heunen, C., Karvonen, M.: Reversible monadic computing. Electronic Notes in Theoretical Computer Science 319, 217–237 (2015)
11. 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 (2008)
12. Landauer, R.: Irreversibility and heat generation in the computing process. IBM Journal of Research and Development 5(3), 183–191 (July 1961)
13. McMahon, D.: Quantum Computing Explained (2007)
14. Miszczak, J.A.: High-level structures for quantum computing. Synthesis Lectures on Quantum Computing 4 (05 2012)
15. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information: 10th Anniversary Edition. Cambridge University Press (2010)
16. Nishida, N., Palacios, A., Vidal, G.: Reversible computation in term rewriting. Journal of Logical and Algebraic Methods in Programming 94, 128–149 (2018)
17. Nishida, N., Palacios, A., Vidal, G.: Reversible computation in term rewriting. Journal of Logical and Algebraic Methods in Programming 94, 128–149 (2018)
18. Selinger, P.: A brief survey of quantum programming languages. In: In Proceedings of the 7th International Symposium on Functional and Logic Programming. Springer (2004)
19. Selinger, P.: Towards a semantics for higher-order quantum computation (2004)
20. Sofge, D.A.: A survey of quantum programming languages: History, methods, and tools. In: Second International Conference on Quantum, Nano and Micro Technologies (ICQNM 2008). pp. 66–71 (Feb 2008)
21. van Tonder, A.: A lambda calculus for quantum computation. SIAM J. Comput. 33(5), 1109–1135 (May 2004)
22. Yokoyama, T.: Reversible computation and reversible programming languages. Electronic Notes in Theoretical Computer Science 253(6), 71–81 (2010)