

Quantum Recursive Programming with Quantum Case Statements

Mingsheng Ying and Zhicheng Zhang

Abstract—We introduce a novel scheme of quantum recursive programming, in which large unitary transformations, i.e. quantum gates, can be recursively defined using quantum case statements, which are quantum counterparts of conditionals and case statements extensively used in classical programming. A simple programming language for supporting this kind of quantum recursion is defined, and its semantics is formally described. A series of examples are presented to show that some quantum algorithms can be elegantly written as quantum recursive programs.

Index Terms—Quantum programming, recursive programming, quantum case statement, operational semantics.

I. INTRODUCTION

THE computer programming pioneers like Dijkstra, Hoare and many others had persuaded a high level of elegance in early programming research by introducing effective program constructs and programming schemes. In particular, iteration and recursion were employed to describe repetitive tasks without requiring a large number of steps to be specified individually. Typical examples include: (i) Quicksort can be elegantly expressed as a recursive program [6]; and (ii) Euclid’s algorithm that computes the greatest common divisor (gcd) of two positive integers can be elegantly written as a **do**-loop:

$$\begin{array}{l} \mathbf{do} \ x > y \rightarrow x := x - y \\ \quad \square \ x < y \rightarrow y := y - x \\ \mathbf{od} \end{array} \quad (1)$$

in the guarded commands language (GCL) [5].

How can we achieve the same level of elegance in quantum programming? Indeed, at this moment, the majority of quantum programming research focuses on relatively low-level features, and the higher-level elegance of quantum programming has not been seriously considered at all. This *short paper* presents an attempt toward the elegance in quantum programming by introducing a novel scheme of quantum recursion.

As is well-known, **if-then-else** conditionals, or more general case statements, are extensively used in recursive definitions of functions in classical programming. An example is the program (1) of the Euclid’s algorithm. It has been realised

Mingsheng Ying is with the State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, and the Department of Computer Science and Technology, Tsinghua University, China. E-mail: yingms@ios.ac.cn; yingmsh@tsinghua.edu.cn.

Zhicheng Zhang is with the Centre for Quantum Software and Information, University of Technology Sydney, Australia. E-mail: Zhicheng.Zhang@student.uts.edu.au.

that two fundamentally different kinds of case statements can be defined in quantum programming [13]:

- 1) *Measurement-based case statements* are usually given in the following form:

$$\mathbf{if} \ (\square i \cdot M[q] = m_i \rightarrow P_i) \ \mathbf{fi} \quad (2)$$

where q is a quantum variable and M a measurement performed on q with possible outcomes m_i ’s, and for each i , P_i is a subprogram. The statement (2) selects a command according to the outcome of measurement M : if the outcome is m_i , then the corresponding command P_i will be executed. It is worth noting that the control flow of (2) is classical because the selection of commands in it is based on classical information — the outcomes of a quantum measurement.

- 2) *Quantum case statements* are usually defined using a quantum “coin” in the following form:

$$\mathbf{qif}[c] \ (\square i \cdot |i\rangle \rightarrow P_i) \ \mathbf{fiq} \quad (3)$$

where $\{|i\rangle\}$ is an orthonormal basis of the state Hilbert space of an *external* “coin” system c , and the selection of subprograms P_i ’s is made according to the basis states $|i\rangle$ of the “coin” space. A fundamental difference between (2) and (3) is that the control flow of (3) is quantum because the basis states of quantum “coin” c can be superposed and thus c carries quantum information rather than classical information (for more about quantum control flow, see [13], Chapter 6 and [12]).

The scheme of recursion with measurement-based case statements (2) has already been studied in the literature, and was termed in [13] as *recursive quantum programming* for the recursion is executed along classical control flow. In this paper, we consider a new scheme of recursion with quantum case statements (3). An important difference between this scheme of quantum recursion and the previous one is as follows: in this scheme, procedure identifiers can occur in different branches of a quantum case statement of the form (3) and thus recursive calls to them may happen in the way of quantum parallelism as a superposition of execution paths. Thus, we call the new scheme *quantum recursive programming* for the execution is executed along quantum control flow. As will be shown in a series of examples, an important class of large quantum gates can be defined and quantum algorithms can be described in the new scheme of quantum recursion conveniently and elegantly.

This paper is organised as follows. As a basis for defining quantum recursion, we introduce quantum arrays in Section II. Our quantum recursive programs are then introduced in several

steps. We start from defining a quantum circuit description language **QC** in Section III. In Section IV, **QC** is embedded into a simple classical programming language. Then quantum recursive programs without parameters and their semantics are defined in Section V. The quantum recursive programs considered in V are generalised by equipped with parameters in Section VI, where actual parameters are described using the classical programming language presented in Section IV.

II. QUANTUM ARRAYS

In this section, we introduce the notions of quantum array and subscripted quantum variable, which will be needed in defining quantum recursive programs.

A. Quantum types

As a basis, let us first define the notion of quantum type. Recall that a basic classical type T denotes an intended set of values. Similarly, a basic quantum type \mathcal{H} denotes an intended Hilbert space. It will be considered as the state space of a simple quantum variable. We will also use higher quantum type of the form:

$$T_1 \times \dots \times T_n \rightarrow \mathcal{H} \quad (4)$$

where T_1, \dots, T_n are basic classical types, and \mathcal{H} is a basic quantum type. Mathematically, this type denotes the following tensor power of Hilbert space \mathcal{H} (i.e. tensor product of multiple copies of \mathcal{H}):

$$\mathcal{H}^{\otimes(T_1 \times \dots \times T_n)} = \bigotimes_{t_1 \in T_1, \dots, t_n \in T_n} \mathcal{H}_{t_1, \dots, t_n} \quad (5)$$

where $\mathcal{H}_{t_1, \dots, t_n} = \mathcal{H}$ for all $t_1 \in T_1, \dots, t_n \in T_n$. Intuitively, if \mathcal{H} is the state space of a quantum system A , then according to the basic postulates of quantum mechanics, the Hilbert space (5) is the state space of the composite system consisting of those quantum systems indexed by $(t_1, \dots, t_n) \in T_1 \times \dots \times T_n$, each of which is identical to A .

A notable basic difference between higher classical types and quantum ones is that some states in the space (5) are entangled between the quantum systems indexed by different $(t_1, \dots, t_n) \in T_1 \times \dots \times T_n$.

Example II.1. Let \mathcal{H}_2 be the qubit type denoting the 2-dimensional Hilbert space. If q is a qubit array of type **integer** $\rightarrow \mathcal{H}_2$, where **integer** is the (classical) integer type, then for any two integers $k \leq l$, section $q[k : l]$ stands for the restriction of q to the interval $[k : l] = \{\text{integer } i \mid k \leq i \leq l\}$. For example,

$$\frac{\bigotimes_{i=k}^l |0\rangle_i + \bigotimes_{i=k}^l |1\rangle_i}{\sqrt{2}}$$

is an entangled state of the qubits labelled k through l .

B. Quantum variables

In this paper, we will use two sorts of quantum variables:

- simple quantum variables, of a basic quantum type, say \mathcal{H} ;

- array quantum variables, of a higher quantum type, say $T_1 \times \dots \times T_n \rightarrow \mathcal{H}$.

Definition II.1. Let q be an array quantum variable of the type $T_1 \times \dots \times T_n \rightarrow \mathcal{H}$, and for each $1 \leq i \leq n$, let s_i be a classical expression of type T_i . Then $q[s_1, \dots, s_n]$ is called a subscripted quantum variable of type \mathcal{H} .

Intuitively, array variable q denotes a quantum system composed of subsystems indexed by $(t_1, \dots, t_n) \in T_1 \times \dots \times T_n$. Thus, whenever expression s_i is evaluated to a value $t_i \in T_i$ for each i , then $q[s_1, \dots, s_n]$ indicates the system of index tuple (t_1, \dots, t_n) . For example, let q be a qubit array of type **integer** \times **integer** $\rightarrow \mathcal{H}_2$. Then $q[2x + y, 7 - 3y]$ is a subscripted qubit variable; in particular, if $x = 5$ and $y = -1$ in the current classical state, then it stands for the qubit $q[9, 10]$.

III. A QUANTUM CIRCUIT DESCRIPTION LANGUAGE

In this section, we introduce a quantum circuit description language **QC**. The major difference between **QC** and other languages in the previous literature for the same purpose is that the construct of quantum case statement is added into **QC**. As pointed out in Section I, this language will be expanded gradually in the subsequent sections for recursive definition of large quantum gates and algorithms.

A. Syntax

We assume that the alphabet of **QC** consists of:

- A set QV of simple or subscripted quantum variables;
- A set \mathcal{U} of unitary matrix constants.

The unitary matrix constants in \mathcal{U} will be instantiated in practical applications. We fix the following notations:

- As defined in Section II, each quantum variable $q \in QV$ assumes a type $T(q)$. This means that variable q stands for a quantum system with the Hilbert space denoted by $T(q)$ as its state space.
- A sequence $\bar{q} = q_1, \dots, q_n$ of distinct quantum variables is called a quantum register. It denotes a composite quantum system consisting of subsystems q_1, \dots, q_n . Its type is defined as the tensor product $T(\bar{q}) = T(q_1) \otimes \dots \otimes T(q_n)$ of the types of q_1, \dots, q_n . For simplicity of the presentation, we often identify \bar{q} with the set $\{q_1, \dots, q_n\}$ of quantum variables occurring in \bar{q} .
- Each unitary matrix constant $U \in \mathcal{U}$ assumes a type of the form $T(U) = \mathcal{H}_1 \otimes \dots \otimes \mathcal{H}_n$. This means that the unitary transformation denoted by U can be performed on a composite quantum system consisting of n subsystems with types $\mathcal{H}_1, \dots, \mathcal{H}_n$, respectively. Thus, if $\bar{q} = q_1, \dots, q_n$ is a quantum register with $T(q_i) = \mathcal{H}_i$ for $i = 1, \dots, n$; that is, the types of U and register \bar{q} match, then $U[\bar{q}]$ can be thought of as a quantum gate with quantum wires q_1, \dots, q_n .

Definition III.1. Quantum circuits $C \in \mathbf{QC}$ are defined by the syntax:

$$C ::= U[\bar{q}] \mid C_1; C_2 \mid \mathbf{qif}[\bar{q}] (\square_{i=1}^d |\psi_i\rangle \rightarrow C_i) \mathbf{fiq} \quad (6)$$

More precisely, they are inductively defined by the following clauses, where $qv(C)$ denotes the quantum variables in C :

- (1) **Basic gates:** If $U \in \mathcal{U}$ is a unitary matrix constant and \bar{q} is a quantum register such that their types match, then quantum gate $U[\bar{q}]$ is a circuit, and $qv(U[\bar{q}]) = \bar{q}$;
- (2) **Sequential composition:** If C_1 and C_2 are circuits, then $C \equiv C_1; C_2$ is a circuit too, and $qv(C) = qv(C_1) \cup qv(C_2)$;
- (3) **Quantum case statement:** If \bar{q} is a quantum register, $\{|\psi_i\rangle\}_{i=1}^d$ is an orthonormal basis of the Hilbert space denoted by the type $T(\bar{q})$, and C_i ($i = 1, \dots, d$) are circuits with

$$\bar{q} \cap \left(\bigcup_{i=1}^d qv(C_i) \right) = \emptyset, \quad (7)$$

then

$$C \equiv \mathbf{qif}[\bar{q}] \left(\square_{i=1}^d |\psi_i\rangle \rightarrow C_i \right) \mathbf{fiq} \quad (8)$$

is a circuit, and $qv(C) = \bar{q} \cup \left(\bigcup_{i=1}^d qv(C_i) \right)$.

Intuitively, each $C \in \mathbf{QC}$ represents a circuit with quantum wires $qv(C)$. The circuit constructs introduced in the above definition are explained as follows.

- (i) The circuit $C_1; C_2$ in clause (2) stands for the sequential composition of circuits C_1 and C_2 . Indeed, if C_1 and C_2 do not share quantum variables; that is, $qv(C_1) \cap qv(C_2) = \emptyset$, we can also define their parallel composition $C_1 \otimes C_2$. But $C_1; C_2$ and $C_1 \otimes C_2$ are semantically equivalent whenever $qv(C_1) \cap qv(C_2) = \emptyset$. So, the parallel composition is not included in the above definition.
- (ii) The quantum case statement defined in equation (8) is a straightforward generalisation of (3) with a quantum ‘‘coin’’ c being replaced by a sequence \bar{q} of quantum variables. The condition (7) means that \bar{q} is a system external to all C_i ($i = 1, \dots, d$). Semantically, quantum case statement (8) is a quantum multiplexor (i.e. a multi-way generalisation of conditional) [11]. A multiplexor can be understood as a switch that passes one of its data inputs through to the output, as a function of a set of select inputs. Here, \bar{q} is the select register, and if \bar{q} is in state $|\psi_i\rangle$, then a quantum datum $|\varphi\rangle$ is inputted to the corresponding circuit C_i and an output $|\varphi_i\rangle$ is obtained at the end of C_i . A basic difference between classical and quantum multiplexors is that the quantum select register \bar{q} can be in a superposition of $|\psi_i\rangle$ ($i = 1, \dots, d$), say $\sum_{i=1}^d \alpha_i |\psi_i\rangle$. In this case, the output is then $\sum_{i=1}^d \alpha_i |\varphi_i\rangle$, a superposition of the outputs of different circuits C_i ($i = 1, \dots, d$). This point will be seen more clearly from the operational semantics defined below.

Remark III.1. A much more general notion of quantum case statement in which quantum measurements may occur was introduced in [13], Chapter 6. In this paper, however, the notion of quantum case statement is restricted to quantum circuits (thus, unitary transformations) so that its semantics can be more clearly defined.

B. Operational semantics

For each quantum variable $q \in QV$, assume its type $T(q)$ denotes Hilbert space \mathcal{H}_q . Then for any set $X \subseteq QV$ of quantum variables, the state space of the quantum variables in X is the tensor product $\mathcal{H}_X = \bigotimes_{q \in X} \mathcal{H}_q$. In particular, the state space of all quantum variables is \mathcal{H}_{QV} .

A configuration is defined as a pair $(C, |\psi\rangle)$, where $C \in \mathbf{QC}$ is a quantum circuit or $C = \downarrow$ stands for termination, and $|\psi\rangle$ is a pure quantum state in Hilbert space \mathcal{H}_X for some $qv(C) \subseteq X \subseteq QV$. We write \mathcal{C} for the set of all configurations with circuits in \mathbf{QC} .

Definition III.2. The operational semantics of quantum circuits in \mathbf{QC} is the transition relation $\rightarrow \subseteq \mathcal{C} \times \mathcal{C}$ between configurations that is defined by the transitional rules given in Table I.

The transition rules (GA) and (SC) are self-explanatory. But the rule (QC) needs an careful explanation. First, for any state $|\psi\rangle$ in a Hilbert space \mathcal{H}_X with $X \supseteq \bar{q}$, we can always write it in the form of $|\psi\rangle = \sum_{i=1}^d \alpha_i |\psi_i\rangle_{\bar{q}} |\theta_i\rangle$, as done in the premise of the rule (QC), because $\mathcal{H}_X = \mathcal{H}_{\bar{q}} \otimes \mathcal{H}_{X \setminus \bar{q}}$ and $\{|\psi_i\rangle\}$ is an orthonormal basis of $\mathcal{H}_{\bar{q}}$. Second, let us write \rightarrow^n for the composition of n copies of \rightarrow . If $i_1 \neq i_2$, then it is possible that

$$(C_{i_1}, |\theta_{i_1}\rangle) \rightarrow^{n_1} (\downarrow, |\theta'_{i_1}\rangle) \text{ and } (C_{i_2}, |\theta_{i_2}\rangle) \rightarrow^{n_2} (\downarrow, |\theta'_{i_2}\rangle)$$

for $n_1 \neq n_2$. This is why the reflexive and transitive closure \rightarrow^* of \rightarrow is used in the premise of the rule (QC). Finally, let $C \in \mathbf{QC}$ be a quantum circuit and $X \supseteq qv(C)$. Then the denotational semantics of C over X can be defined as operator $\llbracket C \rrbracket$ on \mathcal{H}_X as follows:

$$\llbracket C \rrbracket |\varphi\rangle = |\psi\rangle \text{ if and only if } (C, |\varphi\rangle) \rightarrow^* (\downarrow, |\psi\rangle).$$

It is easy to show that

$$\llbracket \mathbf{qif}[\bar{q}] \left(\square_{i=1}^d |\psi_i\rangle \rightarrow C_i \right) \mathbf{fiq} \rrbracket = \sum_{i=1}^d |\psi_i\rangle \langle \psi_i| \otimes \llbracket C_i \rrbracket. \quad (9)$$

Furthermore, if we adopt the matrix representation of operators in the orthonormal basis $\{|\psi_i\rangle\}$, then (9) can be written as the diagonal matrix $\text{diag}(\llbracket C_1 \rrbracket, \dots, \llbracket C_d \rrbracket)$. This confirms that semantically, quantum case statement is exactly the same as quantum multiplexor [11].

C. Illustrative Examples

To illustrate their applicability, let us present several examples showing that the circuit constructs introduced in Definition III.1; in particular quantum case statement, can be used to define some commonly used quantum gates conveniently.

Example III.1. Assume that the single-qubit identity matrix I , the NOT matrix X , and $\bar{R}_x(\theta) = iR_x(2\theta)$ are unitary matrix constants in \mathcal{U} , where $R_x(\theta)$ is the rotation about the x axis:

$$R_x(\theta) = \begin{pmatrix} i \cos \frac{\theta}{2} & \sin \frac{\theta}{2} \\ \sin \frac{\theta}{2} & i \cos \frac{\theta}{2} \end{pmatrix}$$

Then:

$$\begin{array}{ll}
 \text{(GA)} & (U[\bar{q}], |\psi\rangle) \rightarrow (\downarrow, (U \otimes I_{QV\setminus\bar{q}})|\psi\rangle) \\
 \text{(SC)} & \frac{(C_1, |\psi\rangle) \rightarrow (C'_1, |\psi'\rangle)}{(C_1; C_2, |\psi\rangle) \rightarrow (C'_1; C_2, |\psi'\rangle)} \\
 \text{(QC)} & \frac{|\psi\rangle = \sum_{i=1}^d \alpha_i |\psi_i\rangle_{\bar{q}} |\theta_i\rangle \quad (C_i, |\theta_i\rangle) \rightarrow^* (\downarrow, |\theta'_i\rangle) \quad (i = 1, \dots, n)}{(\mathbf{qif}[\bar{q}] (\square_{i=1}^d |\psi_i\rangle \rightarrow C_i) \mathbf{fiq}, |\psi\rangle) \rightarrow (\downarrow, \sum_{i=1}^d \alpha_i |\psi_i\rangle_{\bar{q}} |\theta'_i\rangle)}
 \end{array}$$

TABLE I: Transition Rules for Quantum Circuits. In rule (GA), $I_{QV\setminus\bar{q}}$ stands for the identity operator on the Hilbert space $\mathcal{H}_{QV\setminus\bar{q}}$. In rule (SC), we make the convention that $\downarrow; C_2 = C_2$. In rule (QC), \rightarrow^* denotes the reflexive and transitive closure of relation \rightarrow .

- 1) The *CNOT (Controlled-NOT) gate* with q_1 as its control qubit can be defined by

$$\begin{array}{l}
 \text{CNOT}[q_1, q_2] := \mathbf{qif}[q_1] |0\rangle \rightarrow I[q_2] \\
 \quad \square |1\rangle \rightarrow X[q_2] \\
 \mathbf{fiq}
 \end{array}$$

- 2) The *Toffoli gate* with q_1, q_2 as its control qubits is defined by

$$\begin{array}{l}
 \text{Toffoli}[q_1, q_2, q_3] := \mathbf{qif}[q_1, q_2] |00\rangle \rightarrow I[q_3] \\
 \quad \square |01\rangle \rightarrow I[q_3] \\
 \quad \square |10\rangle \rightarrow I[q_3] \\
 \quad \square |11\rangle \rightarrow X[q_3] \\
 \mathbf{fiq}
 \end{array}$$

- 3) The *Deutsch gate* with q_1, q_2 as its control qubits is defined by

$$\begin{array}{l}
 \text{Deutsch}(\theta)[q_1, q_2, q_3] := \mathbf{qif}[q_1, q_2] |00\rangle \rightarrow I[q_3] \\
 \quad \square |01\rangle \rightarrow I[q_3] \\
 \quad \square |10\rangle \rightarrow I[q_3] \\
 \quad \square |11\rangle \rightarrow \bar{R}_x(\theta)[q_3] \\
 \mathbf{fiq}
 \end{array}$$

Note that $\text{Deutsch}(\frac{\pi}{2}) = \text{Toffoli}$.

- 4) The *Fredkin gate* with q_1 as its control qubit is defined by

$$\begin{array}{l}
 \text{Fredkin}[q_1, q_2, q_3] := \mathbf{qif}[q_1] |0\rangle \rightarrow I[q_2]; I[q_3] \\
 \quad \square |1\rangle \rightarrow \text{SWAP}[q_2, q_3] \\
 \mathbf{fiq}
 \end{array}$$

where the swap gate:

$$\text{SWAP}[q_2, q_3] := \text{CNOT}[q_2, q_3]; \text{CNOT}[q_3, q_2]; \\
 \quad \text{CNOT}[q_2, q_3].$$

Two quantum circuits $C_1, C_2 \in \mathbf{QC}$ are said to be equivalent, written $C_1 \equiv C_2$, if for any $|\psi\rangle, |\psi'\rangle$,

$$(C_1, |\psi\rangle) \rightarrow (\downarrow, |\psi'\rangle) \text{ if and only if } (C_2, |\psi\rangle) \rightarrow (\downarrow, |\psi'\rangle).$$

Then we have:

Example III.2. It is easy to verify that

$$\begin{array}{l}
 \mathbf{qif}[q_1] |+\rangle \rightarrow I[q_2] \square |-\rangle \rightarrow Z[q_2] \mathbf{fiq} \\
 \equiv \mathbf{qif}[q_2] |0\rangle \rightarrow I[q_1] \square |1\rangle \rightarrow X[q_2] \mathbf{fiq}.
 \end{array}$$

IV. QUANTUM CIRCUITS DEFINED WITH CLASSICAL VARIABLES

To increase the expressive power, in this section we embed **QC** into a classical programming language. Our aim for this embedding is to allow us to use classical expressions as parameters in quantum recursive programs.

A. Syntax

For simplicity, let us choose classical **while**-language as the host language. Thus, we obtain:

Definition IV.1. Quantum circuits $C \in \mathbf{QC}^+$ with classical variables are defined by the syntax:

$$\begin{array}{l}
 C ::= \text{skip} \mid \bar{x} := \bar{t} \mid U[\bar{q}] \mid C_1; C_2 \\
 \quad \mid \text{if } b \text{ then } C_1 \text{ else } C_2 \mathbf{fi} \\
 \quad \mid \text{while } b \text{ do } C \mathbf{od} \\
 \quad \mid \mathbf{qif}[\bar{q}] (\square_{i=1}^d |\psi_i\rangle \rightarrow C_i) \mathbf{fiq}
 \end{array} \tag{10}$$

where \bar{x} is a string of classical simple or subscripted variables, \bar{t} is a string of classical expression, b is a Boolean expression, and other condition are the same as in Definition III.1.

The quantum variables $qv(C)$ in $C \in \mathbf{QC}^+$ are defined inductively as follows:

- 1) $qv(\text{skip}) = qv(\bar{x} := \bar{t}) = \emptyset$;
- 2) $qv(U[\bar{q}]) = \bar{q}$;
- 3) $qv(C_1; C_2) = qv(\text{if } b \text{ then } C_1 \text{ else } C_2 \mathbf{fi}) = qv(C_1) \cup qv(C_2)$;
- 4) $qv(\text{while } b \text{ do } C \mathbf{od}) = qv(C)$;
- 5) $qv(\mathbf{qif}[\bar{q}] (\square_{i=1}^d |\psi_i\rangle \rightarrow C_i) \mathbf{fiq}) = \bar{q} \cup \left(\bigcup_{i=1}^d qv(C_i) \right)$.

As usual in classical programming, a conditional of the form **if** b **then** C_1 **else** **skip** **fi** will be simply written as **if** b **then** C_1 **fi**.

Remark IV.1. Our aim of introducing classical computation in \mathbf{QC}^+ is to enable quantum circuits be defined using classical expressions as their parameters. So, in a sense, the connection between classical and quantum variables in \mathbf{QC}^+ is unidirectional from classical ones to quantum ones. But a connection from quantum variables to classical ones can also be introduced by adding statements of the form $x := M[\bar{q}]$, meaning that the outcome of measurement M on quantum variables \bar{q} is stored in classical variable x . For simplicity of presentation, however, we choose not to consider it in this paper.

$$\begin{array}{ll}
 \text{(SK)} & (\mathbf{skip}, \sigma, |\psi\rangle) \rightarrow (\downarrow, \sigma, |\psi\rangle) \\
 \text{(AS)} & (\bar{x} := \bar{t}, \sigma, |\psi\rangle) \rightarrow (\downarrow, \sigma[\bar{x} \leftarrow \sigma(\bar{t})], |\psi_\sigma\rangle) \\
 \text{(QC)} & \frac{|\psi\rangle = \sum_{i=1}^d \alpha_i |\psi_i\rangle_{\bar{q}} |\theta_i\rangle \quad (C_i, \sigma, |\theta_i\rangle) \rightarrow^* (\downarrow, \sigma', |\theta'_i\rangle) \quad (i = 1, \dots, n)}{(\mathbf{qif}[\bar{q}] (\square_{i=1}^d |\psi_i\rangle \rightarrow C_i) \mathbf{fiq}, \sigma, |\psi\rangle) \rightarrow (\downarrow, \sigma', \sum_{i=1}^d \alpha_i |\psi_i\rangle_{\bar{q}} |\theta'_i\rangle)} \\
 \text{(SC)} & \frac{(C_1, \sigma, |\psi\rangle) \rightarrow (C'_1, \sigma', |\psi'\rangle)}{(C_1; C_2, \sigma, |\psi\rangle) \rightarrow (C'_1; C_2, \sigma', |\psi'\rangle)} \\
 \text{(GA)} & (U[\bar{q}], \sigma, |\psi\rangle) \rightarrow (\downarrow, \sigma, (U \otimes I_{QV \setminus \bar{q}}) |\psi\rangle)
 \end{array}$$

TABLE II: Transition Rules for Quantum Circuits with Classical Variables. In rule (AS), let $\bar{x} = x_1, \dots, x_n$ and $\bar{t} = t_1, \dots, t_n$. Then $\sigma(t_i)$ denotes the value of expression t_i in state σ , and $\sigma[\bar{x} \leftarrow \sigma(\bar{t})]$ is the state of classical variables obtained by replacing the value of x_i in σ with $\sigma(t_i)$ simultaneously for all $1 \leq i \leq n$.

$$\text{(BS)} \quad (\mathbf{begin\ local} \ \bar{x} := \bar{t}, C \ \mathbf{end}, \sigma, |\psi\rangle) \rightarrow (\bar{x} := \bar{t}; C; \bar{x} := \sigma(\bar{x}), \sigma, |\psi\rangle)$$

TABLE III: Transition Rule for Local Variables.

B. Operational semantics

To define operational semantics of \mathbf{QC}^+ , we need to modify the definition of configuration in order to accommodate classical variables. A configuration is now defined as a triple $(C, \sigma, |\psi\rangle)$, where $C \in \mathbf{QC}^+$ is a quantum variable with classical variables, σ is a state of classical variables, and $|\psi\rangle$ is a pure quantum state in \mathcal{H}_X for some $\mathbf{qv}(C) \subseteq X \subseteq QV$. For simplicity, we abuse a bit of notation and still use \mathcal{C} to denote the set of all configurations.

Definition IV.2. *The operational semantics of quantum circuits in \mathbf{QC}^+ is the transition relation $\rightarrow \subseteq \mathcal{C} \times \mathcal{C}$ between configurations defined by the transitional rules given in Table II.*

The rules (SK), (SC), (AS) and (GA) are easy to understand. One design decision in the rule (QC) needs an explanation. In its premise, when starting in the state classical state σ , the executions of all branches $(C_i, \sigma, |\theta_i\rangle)$ ($i = 1, \dots, n$) are required to terminate in the same classical state σ' . At the first glance, this is a very strong requirement and hard to meet in practical applications. Indeed, as we will see in the next subsection and Example V.1, it can be easily achieved by introducing local variables.

C. Local variables

In this subsection, we further introduce local classical variables into \mathbf{QC}^+ by extending its syntax with the following clause:

$$C ::= \mathbf{begin\ local} \ \bar{x} := \bar{t}; C \ \mathbf{end} \quad (11)$$

where \bar{x} is a sequence of classical variables and \bar{t} a sequence of classical expressions. A statement of the form (11) is called a block statement, and its operational semantics is defined by the rule (BS) in Table III.

The rule (BS) is very similar to the rule defining the operational semantics of local variables in classical programs (see for example [2], the rule (ix) on page 154). In the execution of block statement (11) starting in classical state σ , the *local variables* \bar{x} are first initialised by assignment $\bar{x} := \bar{t}$, then the circuit C within the statement is executed. After that, \bar{x} resume their original values in σ . It is easy to see that if \bar{x} is

an empty sequence, then the block statement can be identified with circuit C within it.

We will see in Example V.1 how local variables can help in describing quantum recursive programs.

V. QUANTUM RECURSIVE PROGRAMS WITHOUT PARAMETERS

Now we are ready to define a language \mathbf{RQC}^+ of recursively defined quantum circuits. In this section, we only consider quantum recursive circuits without parameters. More general quantum recursive circuits with parameters will be considered in the next section. As we will see shortly, at the level of syntax, quantum recursive circuits and classical recursive programs (see for example [2], Chapters 4 and 5) are similar to each other. The major difference between them appears at the level of semantics, where quantum case statements involved in the former will exhibit superposition of the executions of multiple circuits within recursive procedures.

A. Syntax

Let us first define the syntax of \mathbf{RQC}^+ . We add a set of *procedure identifiers*, ranged over by symbols P, P_1, P_2, \dots into the alphabet of \mathbf{QC}^+ . Then the syntax of \mathbf{RQC}^+ is defined by extending the syntax (10) and (11) of \mathbf{QC}^+ by adding the clause:

$$C ::= P \quad (12)$$

with quantum variables $\mathbf{qv}(P) = \emptyset$. As in classical recursive programming [2], an occurrence of a procedure identifier in a program is called a *procedure call*. We assume that each procedure identifier P is defined by a *declaration* of the form

$$P \Leftarrow C \quad (13)$$

where $C \in \mathbf{RQC}^+$ is called the procedure body. Note that in the declaration (13), P may appear in the procedure body C ; the occurrences of P in C are thus called *recursive calls*. We assume a fixed set \mathcal{D} of procedure declarations.

B. Operational semantics

To define the operational semantics of \mathbf{RQC}^+ , we first generalise the notion of configuration $(C, \sigma, |\psi\rangle)$ by allowing

$C \in \mathbf{RQC}^+$. Then the operational semantics is the transition relation \rightarrow between configurations defined by the transition rules given in Tables II and III together with the following *copy rule*:

$$(CR) \quad \frac{P \Leftarrow C \in \mathcal{D}}{(P, \sigma, |\psi\rangle) \rightarrow (C, \sigma, |\psi\rangle)} \quad (14)$$

Intuitively, the copy rule allows that a procedure call is dynamically replaced by the procedure body of its declaration given in declarations \mathcal{D} .

C. Illustrative Examples

The following two examples show how some large unitary transformations (i.e quantum gates) can be elegantly described as quantum recursive programs defined above.

1) *Controlled unitary transformations*: Controlled unitaries are a class of quantum gates widely used in quantum computing. Mathematically, let U be a unitary operator on a single qubit and n a positive integer. Then the controlled- U gate $C^{(n)}(U)$ with q_1, \dots, q_n as its control qubits and q_{n+1} as its target qubit is defined by

$$\begin{aligned} C^{(n)}(U)|i_1, \dots, i_n\rangle|\psi\rangle \\ = \begin{cases} |i_1, \dots, i_n\rangle U|\psi\rangle & \text{if } i_1 = \dots = i_n = 1; \\ |i_1, \dots, i_n\rangle|\psi\rangle & \text{otherwise} \end{cases} \end{aligned}$$

for any $i_1, \dots, i_n \in \{0, 1\}$ and $|\psi\rangle \in \mathcal{H}_2$.

Using a quantum programming language without recursion, one has to define $C^{(n)}$ for different integers n individually. Now within the recursion scheme introduced above, we can define $C^{(n)}$ in a uniform way:

Example V.1. *Let q be a qubit array of type $\mathbf{integer} \rightarrow \mathcal{H}_2$. Then the controlled- U gate on the section $q[\mathit{first} : \mathit{last}]$ with the first ($\mathit{last} - \mathit{first}$) qubits as its control qubits and the last one as its target qubit can be written as the recursive program:*

```

 $C^{(*)}(U) \Leftarrow$ 
if  $\mathit{first} = \mathit{last}$ 
  then  $U[q[\mathit{last}]]$ 
else qif $[q[\mathit{first}]]|0\rangle \rightarrow \mathbf{skip}$ 
   $\square |1\rangle \rightarrow \mathbf{begin\ local\ } \mathit{first} := \mathit{first} + 1;$ 
   $C^{(*)}(U) \mathbf{end}$ 
fiq
fi

```

It is worth noting that a block statement with local variables is employed in the above program $C^{(*)}(U)$ to guarantee that different branches of a quantum case statement terminate in the same classical state (see the explanation of the transition rule (QC) given after Definition IV.2).

2) *Quantum Fourier transforms*: As a key subroutine, quantum Fourier transforms appear in many important quantum algorithms, including Shor's factoring algorithm. The quantum Fourier transform $QFT(n)$ on n qubits is mathematically defined by

$$QFT(n)|j\rangle = \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} e^{2\pi i j k / 2^n} |k\rangle \quad (15)$$

for $j = 0, 1, \dots, 2^n - 1$. If we use the binary representation $j = j_1 j_2 \dots j_n = \sum_{l=1}^n j_l 2^{n-l}$ and binary fraction $0.k_1 k_2 \dots k_m = \sum_{l=1}^m k_l 2^{-l}$, then the defining equation (15) of $QFT(n)$ can be rewritten as

$$QFT(n)|j_1, \dots, j_n\rangle = \frac{1}{\sqrt{2^n}} \bigotimes_{l=1}^n (|0\rangle + e^{2\pi i 0.j_{n-l+1} \dots j_n} |1\rangle). \quad (16)$$

As shown in Table IV, $QFT(n)$ can be decomposed into a sequence of single-qubit and two-qubit basic gates, namely the Hadamard gate H and controlled-rotations $C(R_l)$ ($l = 1, \dots, n$), where

$$R_l = \begin{pmatrix} 1 & 0 \\ 0 & e^{2\pi i / 2^l} \end{pmatrix}$$

is a single-qubit gate.

In a quantum programming language that does not support recursion, one has to program $QFT(n)$ in a way similar to Table IV. For a large number n of qubits, the size of such a $QFT(n)$ program will be very large. Using recursion, however, we can write $QFT(n)$ as a program, of which the size is independent of the number n of qubits:

Example V.2. *Let q be a qubit array of type $\mathbf{integer} \rightarrow \mathcal{H}_2$. Then quantum Fourier transform on the section $q[m, n]$ can be written as the following recursive program:*

```

 $QFT(m, n) \Leftarrow H[q[m]];$  if  $m < n$  then  $Rotate(m, n);$ 
   $QFT[m + 1, n]$ 
fi;
   $Reverse(m, n)$ 
 $Rotate(m, n) \Leftarrow Rotate(m, n - 1);$ 
  qif $[q[n]]|0\rangle \rightarrow \mathbf{skip}$ 
   $\square |1\rangle \rightarrow R_n[q[m]]$ 
fiq
 $Reverse(m, n) \Leftarrow \mathbf{if\ } m < n \mathbf{\ then\ } SWAP[q[m], q[n]];$ 
  if  $m + 2 \leq n$  then
   $Reverse(m + 1, n - 1)$ 
fi
fi

```

VI. QUANTUM RECURSIVE PROGRAMS WITH PARAMETERS

In this section, we further expand the language \mathbf{RQC}^+ defined in the last section to \mathbf{RQC}^{++} of quantum recursive circuits with *classical* parameters.

A. Syntax

We add a set of procedure identifiers P, P_1, P_2, \dots into the alphabet of \mathbf{QC}^+ . Each identifier P is given an arity $ar(P)$. Then the syntax of \mathbf{RQC}^{++} is defined by the syntax (10) and (11) of \mathbf{QC}^+ together with the following clause:

$$C ::= P(t_1, \dots, t_n) \quad (17)$$

$$\begin{aligned}
 QFT(n)[q_1, \dots, q_n] ::= & H[q_1]; C(R_2)[q_2, q_1]; C(R_3)[q_3, q_1]; \dots; C(R_{n-1})[q_{n-1}, q_1]; C(R_n)[q_n, q_1]; \\
 & H[q_2]; C(R_2)[q_3, q_2]; C(R_3)[q_4, q_2]; \dots; C(R_{n-1})[q_n, q_2]; \\
 & \dots\dots\dots \\
 & H[q_{n-1}]; C(R_2)[q_n, q_{n-1}]; \\
 & H[q_n]; \\
 & Reverse[q_1, \dots, q_n]
 \end{aligned}$$

TABLE IV: A quantum circuit for quantum Fourier transform. Here, $C(R_l)[q_i, q_j]$ stands for the controlled- R_l with q_i as its control qubit and q_j as its target qubit, and $Reverse[q_1, \dots, q_n]$ is the quantum gate that reverses the order of qubits q_1, \dots, q_n .

$$(RC) \frac{P(u_1, \dots, u_n) \Leftarrow C \in \mathcal{D}}{(P(t_1, \dots, t_n), \sigma, |\psi\rangle) \rightarrow (\mathbf{begin} \text{ local } \bar{u} := \bar{t}; C \text{ end}, \sigma, |\psi\rangle)}$$

TABLE V: Transition Rule for Quantum Recursive Circuits with Parameters.

The above syntax of \mathbf{RQC}^{++} is very similar to that of classical recursive programs with parameters given in [2]. In procedure call (17), P is a procedure identifier with $ar(P) = n$, and t_1, \dots, t_n are classical expressions, called *actual parameters*. Whenever $n = 0$, procedure $P(t_1, \dots, t_n)$ degenerates to a procedure without parameters considered in the last section.

Each procedure identifier P is defined by a declaration of the form:

$$P(u_1, \dots, u_n) \Leftarrow C \quad (18)$$

where u_1, \dots, u_n are classical simple variables, called *formal parameters*; and the procedure body $C \in \mathbf{RQC}^{++}$. We assume a fixed set \mathcal{D} of procedure declarations.

B. Operational semantics

Now configurations are triples $(C, \sigma, |\psi\rangle)$ with $C \in \mathbf{RQC}^{++}$ and $\sigma, |\psi\rangle$ being as in Section V. The semantics of \mathbf{RQC}^{++} is then a transition relation between configurations defined by the rules in Tables II and III together with the recursive rule (RC) in Table V.

C. Illustrative examples

1) *Controlled unitary transformations revisited*: Using recursion with parameters introduced in this section, controlled unitaries can be programmed in a more compact way than Example V.1:

Example VI.1. *The controlled- U gate on the section $q[m : n]$ with the first $(n - m)$ qubits as its control qubits and the last one as its target qubit can be written as the recursive program:*

```

 $C^{(*)}(U)(m, n) \Leftarrow \mathbf{if} \ m = n$ 
     $\mathbf{then} \ U[q[n]]$ 
     $\mathbf{else} \ \mathbf{qif}[q[m]]|0\rangle \rightarrow \mathbf{skip}$ 
     $\square \ |1\rangle \rightarrow C^{(*)}(U)[m + 1, n]$ 
     $\mathbf{fiq}$ 
 $\mathbf{fi}$ 

```

In particular, it is interesting to note that different from Example V.1, the $C^{(*)}(U)$ program in the above example does not use any block statement and local variable.

Remark VI.1. *The idea of the above example can be easily generalised to give a recursive definition of a quantum case statement with multiple quantum coins of the form:*

$$\mathbf{qif}[q[1 : k]](\square_{x \in \{0,1\}^k} |x\rangle \rightarrow U_x[q[k + 1]]) \mathbf{fiq} \quad (19)$$

in terms of quantum case statements with a single quantum coin of which the number of branches is fixed.

2) *Quantum state preparation*: Quantum state preparation (QSP) is a basic procedure employed in many quantum algorithms; in particular, in quantum simulation and quantum machine learning. The problem is as follows. Given an N -dimensional complex vector $\mathbf{a} = (a_j)_{j=0}^{N-1} \in \mathbb{C}^N$, where $N = 2^n$. Our goal is to generate the n -qubit state:

$$\frac{1}{\sqrt{a}} \sum_{j=0}^{N-1} \sqrt{a_j} |j\rangle$$

from the basis state $|0\rangle^n$, where $a = \sum_{j=0}^{N-1} |a_j|$.

For each $0 \leq j < N$, and for any $0 \leq l < r \leq N$, define θ_j and $S_{l,r}$ such that $a_j = e^{i\theta_j} |a_j|$ and $S_{l,r} = \sum_{j=l}^{r-1} |a_j|$. Then the QSP algorithm (which slightly generalises the one in [8]) consists of n steps. For $0 \leq k < n$, in the k th step, it performs the transformation:

$$\begin{cases} |0\rangle^n \mapsto U_{0,0} |0\rangle |0\rangle^{\otimes(n-1)} & k = 0; \\ |x\rangle |0\rangle^{\otimes(n-k)} \mapsto |x\rangle U_{k,x} |0\rangle |0\rangle^{\otimes(n-k-1)} & 1 \leq k < n \end{cases}$$

for all $0 \leq x < 2^k - 1$, where $U_{k,x}$ is a single qubit gate such that:

$$U_{k,x} |0\rangle = \sqrt{\gamma_x} |0\rangle + e^{i\beta_x/2} \sqrt{1 - \gamma_x} |1\rangle,$$

and $\gamma_x = \frac{S_{u,w}}{S_{u,v}}$, $\beta_x = \theta_w - \theta_u$, $u = 2^{n-k}x$, $v = 2^{n-k}x + 2^{n-k}$ and $w = \frac{u+v}{2}$. Using the language \mathbf{RQC}^+ , the QSP algorithm can be elegantly rewritten as a quantum recursive program:

Example VI.2. *Let*

```

QSP(k, n) ← if k = 0 then U0,0[q[1]] else
    if 1 ≤ k < n then
        qif[q[1 : k]](□|x) → Uk,x[q[k + 1]];
        QSP(k + 1, n)
    fiq
fi
fi

```

Then one calls QSP(0, n) for the quantum state preparation.

At the first glance, there seems a bug in the above program: the number 2^k of branches of the **qif**-statement varies as the number k of coin qubits $q[1 : k]$. But it is actually not a bug because as pointed out in Remark VI.1, the **qif**-statement with coins $q[1 : k]$ can be recursively defined in terms of **qif**-statement with a single coin qubit.

3) *Quantum Random-Access Memory (QRAM)*: Many applications of quantum computing (from optimisation and machine learning to cryptanalysis) presume the existence of QRAM, a quantum counterpart of RAM (Random-Access Memory) in classical computing [7]. The strongest type of QRAM is called QRAQM (Quantum Random-Access Quantum Memory), which stores quantum data and access data based on addresses that are themselves a quantum state in a superposition. Among several equivalent forms, we consider a QRAQM that performs the following transformation: for any data set $D[0 : N]$ with $N = 2^n - 1$, and for any address $0 \leq j \leq N$,

$$|j\rangle|D[0 : N]\rangle \mapsto |j\rangle|D[j]\rangle|D[0 : j - 1]\rangle|D[j + 1 : N]\rangle. \quad (20)$$

Intuitively, given an address j , the desired data element $D[j]$ is swapped out. A simple (but not very efficient) implementation of QRAQM can be written as a quantum recursive program:

Example VI.3. *We use $q_a[1 : n]$ for the address register holding $|j\rangle$ and $q_D[0 : N]$ for the data register holding $|D[0 : N]\rangle$ on the LHS of (20). Let*

```

U(l, r, k) ← if k ≤ n then
    begin local m := ⌊(l + r)/2⌋;
        qif[qa[k]] |0⟩ → U(l, m, k + 1)
        □ |1⟩ → U(m + 1, r, k + 1);
        SWAP[qD[l], qD[m + 1]]
    fiq
end
fi

```

Then one calls U(0, N, 1) for the QRAQM operation.

It is interesting to note that a Divide-and-Conquer strategy was employed in the above example where QRAM is divided into two subproblems smaller than the original one that are then solved respectively in each of the branches of a quantum case statement.

VII. CONCLUSION

This *short paper* introduces a new scheme of quantum recursive programming. The basic ideas of this scheme of quantum recursion are illustrated through a series of interesting examples. It should be emphasised that this scheme of quantum recursion is defined based on the notion of quantum case statement; indeed, it cannot be realised without quantum case statements, as we can observe from the examples.

This paper is merely one of the first steps toward a theory of quantum recursive programming. Plenty of problems about quantum recursions remain unsolved. Here, we would like to mention the following two open problems:

- *Implementation of quantum recursion*: Classical recursive programs are usually implemented employing stack [4], [1]. How can we implement the kind of quantum recursion introduced in this paper? Indeed, a notion of quantum stack has still not been properly defined.
- *More sophisticated quantum recursive programming techniques*: A simple Divide-and-Conquer strategy was employed in Example VI.3. Several other quantum Divide-and-Conquer strategies have been proposed in the literature (see for example [3]). It is interesting to see whether or not and how quantum algorithms developed with these Divide-and-Conquer strategies can be recursively programmed. Furthermore, how can quantum recursive programming be combined with structural development techniques of quantum algorithms as recently proposed in [10].

ACKNOWLEDGMENTS

This work was partly supported by the National Natural Science Foundation of China (Grant No: 61832015). Zhicheng Zhang was supported by the Sydney Quantum Academy, NSW, Australia.

REFERENCES

- [1] H. Abelson, G. J. Sussman and J. Sussman, *Structure and Interpretation of Computer Programs* (2nd Edition), The MIT Press, 1996.
- [2] K. R. Apt, F. S. de Boer and E. -R. Olderog, *Verification of Sequential and Concurrent Programs*, Springer, London 2009.
- [3] A. M. Childs, R. Kothari, M. Kovacs-Deak, A. Sundaram and D. C. Wang, Quantum divide and conquer, *arXiv* 2210.06419.
- [4] E. W. Dijkstra, Recursive programming, *Numerische Mathematik* 2(1960)312-318.
- [5] E. W. Dijkstra, Guarded command, nondeterminacy and formal derivation of programs, *Communications of the ACM* 19(1975)453-457.
- [6] C. A. R. Hoare, Quicksort, *The Computer Journal* 5(1962)10-16.
- [7] S. Jaques and A. G. Rattew, QRAM: a survey and critique, *arXiv*: 2305.10310.
- [8] I. Kerenidis and A. Prakash, Quantum recommendation systems, in: *Proceedings of the 8th Innovations in Theoretical Computer Science Conference (ITCS)*, 2017, pp. 49:1-21.
- [9] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, Cambridge University Press, 2000.
- [10] Z. M. Rossi and I. L. Chuang, Semantic embedding for quantum algorithms, *arXiv*: 2304.14392.
- [11] V. V. Shende, S. S. Bullock and I. L. Markov, Synthesis of quantum-logic circuits, *IEEE Transactions on CAD of Integrated Circuits and Systems* 25(2006) 1000-1010.
- [12] C. Yuan, A. Villanyi and M. Carbin, Quantum control machine: The limits of control flow in quantum programming, *arXiv*: 2304.15000.
- [13] M. S. Ying, *Foundations of Quantum Programming*, Morgan Kaufmann, 2016.
- [14] M. S. Ying and Y. Feng, A flowchart language for quantum programming, *IEEE Transactions on Software Engineering* 37(2011) 466-485.