PUBLISHED VERSION

Lagana, Antonio Alberto; Lohe, Max Adolph; von Smekal, Lorenz Johann Maria
Construction of a universal quantum computer Physical Review A, 2009; 79(5):2322

© 2009 American Physical Society

http://link.aps.org/doi/10.1103/PhysRevA.79.052322

PERMISSIONS

http://publish.aps.org/authors/transfer-of-copyright-agreement

“The author(s), and in the case of a Work Made For Hire, as defined in the U.S. Copyright Act, 17 U.S.C.

§101, the employer named [below], shall have the following rights (the “Author Rights”):

[...]

3. The right to use all or part of the Article, including the APS-prepared version without revision or modification, on the author(s)’ web home page or employer’s website and to make copies of all or part of the Article, including the APS-prepared version without revision or modification, for the author(s)’ and/or the employer’s use for educational or research purposes.”

19th March 2013

http://hdl.handle.net/2440/55035

Construction of a universal quantum computer

Antonio A. Lagana,* M. A. Lohe, and Lorenz von Smekal
Department of Physics, University of Adelaide, South Australia 5005, Australia
(Received 30 March 2009; published 18 May 2009)

We construct a universal quantum computer following Deutsch's original proposal of a universal quantum Turing machine (UQTM). Like Deutsch's UQTM, our machine can emulate any classical Turing machine and can execute any algorithm that can be implemented in the quantum gate array framework but under the control of a quantum program, and hence is universal. We present the architecture of the machine, which consists of a memory tape and a processor and describe the observables that comprise the registers of the processor and the instruction set, which includes a set of operations that can approximate any unitary operation to any desired accuracy and hence is quantum computationally universal. We present the unitary evolution operators that act on the machine to achieve universal computation and discuss each of them in detail and specify and discuss explicit program halting and concatenation schemes. We define and describe a set of primitive programs in order to demonstrate the universal nature of the machine. These primitive programs facilitate the implementation of more complex algorithms and we demonstrate their use by presenting a program that computes the NAND function, thereby also showing that the machine can compute any classically computable function.

DOI: 10.1103/PhysRevA.79.052322 PACS number(s): 03.67.Lx

I. INTRODUCTION

As best exemplified by Shor's factorization algorithm [1], quantum computing algorithms have the potential to achieve significant speed-ups over classical computing algorithms. In fact, it has even been suggested [2] that quantum computability can potentially surpass classical computability by solving problems such as the famous halting problem.

The gate array model of quantum computing that Deutsch [3] formulated has been shown to be universal in the sense that any unitary operation can be implemented using a set of single-qubit gates (H and the T or $\frac{\pi}{8}$ gate) and the two-qubit controlled-NOT (CNOT) gate much as the two-bit NAND gate is universal for classical logical circuits (see Nielsen and Chuang [4] chapter 4, for example). This means that a quantum computer can compute any function that can be computed by a classical computer and in certain cases, most notably unstructured database search [5] and integer factorization [1], with a speed-up over the best known classical algorithms. However, this universality is distinct from the notion of universality in the sense that every computable function can be computed by a universal Turing machine (UTM). According to the Church-Turing thesis, all finitely realized computing machines can be simulated by a *single* machine, the UTM. Modern computers are fundamentally UTM implementations. Thus, universality is important because programmability follows from it.

The quantum gate array computing model is not universal in this sense because it is not programmable. A single quantum gate array computer cannot simulate every other quantum computer. Each quantum gate array computer must be purpose built or configured to implement a particular algorithm. Even the recently proposed quantum adiabatic computing framework [6] is not strictly programmable in this sense because the time-dependent Hamiltonian must be individually tailored for each given problem. In this sense, currently envisioned quantum computers more closely resemble special purpose processors rather than general-purpose processors, to draw an analogy with classical computers.

In one of the founding papers of quantum computation, Deutsch [7] defined a quantum Turing machine (QTM) and further claimed that there exists a universal quantum Turing machine (UQTM) that is a natural quantum generalization of the classical UTM, both of which are quantum generalizations of their classical counterparts. The UQTM was defined to be a QTM for which there exists a program as part of its input state that has the effect of applying a unitary transformation on an arbitrary number of qubits arbitrarily close to any desired transformation. That is, the UQTM could simulate, up to arbitrary accuracy, the operation of any given QTM. As its classical counterpart, a QTM contains a halt qubit that is used to indicate whether the computation has been completed. Thus, the UQTM is universal in the sense in that it is programmable and able to simulate the operation of every possible QTM. The theoretical existence of such a machine is important because it would establish whether a programmable quantum computer can be constructed in principle.

Since the original proposal, several questions have been raised as to whether the UQTM, as defined by Deutsch, was indeed valid. In 1997, Myers [8] argued that the UQTM's halting scheme was invalid. In 2001 Shi [9] showed that the validity of the halting scheme ultimately rested on whether the program concatenation scheme was valid. If the concatenation scheme were valid, the halting scheme would be valid, and Myer's question could be resolved by Ozawa's [10] nondemolition measurements of the halting qubit subject to the requirement that the halting scheme be implemented in a way whereby the state of the memory tape ceases to change once the halt qubit is set. The question of whether the concatenation scheme is valid and hence whether a UQTM exists has remained an open question.

In the following sections, we first review the QTM and UQTM as defined by Deutsch and then present an explicit

*antonio.lagana@adelaide.edu.au

 ©2009 The American Physical Society

construction of a universal quantum computer to demonstrate that a universal (programmable) quantum computer exists and that program concatenation is valid. The machine supports programmatic execution basic instructions that include the universal set of unitary operations as well as a conditional branch instruction. Like Deutsch's UQTM, our machine consists of a memory tape and processor. The internal architecture of our machine is very similar to that of a classic microcontroller and contains a data address register, program counter, status flag, instruction fetch buffer register, and a memory read-and-write head. In addition, our machine contains a halt qubit that signifies whether program execution has completed and a flow control register and history buffer address register that are used to store program execution history information. The flow control and history buffer address registers are used to store a sufficient amount of information such that, in principle, the operation of any program can be reversed at any given time, consistent with unitarity. This theoretical construction will be useful to analyze other aspects of quantum computation, such as complexity analysis of algorithms, analysis of the halting problem (in the Church-Turing thesis sense), etc., in an analogous way that a UTM is used in classical computer science.

Sections II and III provide brief descriptions of Deutsch's QTM and UQTM, respectively. In Sec. IV we describe several problems that were raised about the QTM halting scheme and the fact that it relies on the validity of program concatenation, something that Deutsch did not prove. In Sec. V we present an explicit construction of a universal quantum computer and describe the internal architecture, instruction set, and the time evolution operator associated with the machine. In Sec. VI we define a set of basic programs in order to demonstrate the classical universal nature of our machine by constructing a program that computes the NAND function. In Sec. VII we discuss program concatenation and present a program concatenation operator for our machine thus demonstrating that program concatenation is valid for quantum computers.

II. QUANTUM TURING MACHINE

As defined by Deutsch, a QTM consists of two components: a finite processor and an infinite tape (external memory), of which only a finite portion is ever used. The finite processor consists of N qubits and the infinite tape consists of an infinite sequence of qubits, of which only a finite portion is ever used. The currently scanned tape location is specified by x which denotes the "address" number of the tape. Thus, the state of a QTM is a unit vector in the Hilbert space spanned by the basis states $|x\rangle|\mathbf{n}\rangle|\mathbf{m}\rangle$, where $|\mathbf{n}\rangle \overset{\text{def}}{=} |n_0,n_1,n_2,\ldots,n_{N-1}\rangle$, and $|\mathbf{m}\rangle \overset{\text{def}}{=} |\ldots,m_{-2},m_{-1},m_0,m_1,m_2\ldots\rangle$.

The operation or dynamics of the machine is defined by a *fixed* unitary operator U whose only nontrivial matrix elements are $\langle x\pm 1;\mathbf{n}';m'_x,m_{y\neq x}|U|x;\mathbf{n};m_x,m_{y\neq x}\rangle$. That is, only one tape qubit, the xth, participates in any given computational step and at each step, the position of the head cannot change by more than one unit, forward or backward, or both in the case that the position of the tape is a superposition of $|x\pm 1\rangle$. Each different U corresponds to a *different* QTM. Stated differently, each QTM corresponds to a specific algorithm in the same way that each quantum gate array circuit is an implementation of a specific algorithm. To signal whether the computation has been completed, the processor contains a special internal qubit, $|n_0 \overset{\text{def}}{=} h\rangle$, known as the halt qubit, that is initialized to 0 and is set to 1 upon completion of the computation. Thus, an external operator (or classical computer) may periodically observe $|h\rangle$ to determine whether the computation has been completed. The evolution of the QTM can thus be described as

$$|\psi(s\Delta T)\rangle = U^s|\psi(0)\rangle,$$

where $|\psi(0)\rangle$ is the initial state, s is the number of computation steps, and ΔT is the time duration of each computational step.

III. UNIVERSAL QUANTUM TURING MACHINE

As Shi [9] pointed out, a UQTM state may be defined as $|Q,D,P,\Sigma\rangle$, where Q is the state of the processor, including the head position x, D is the state of the data register, and P is the program state. D and P are each parts of the tape and Σ is the remaining part of the tape that is not used during the computation. Note that this does not deviate from the original definition of the UQTM by Deutsch in [7], as the corresponding basis elements of $|\mathbf{m}\rangle$ can be appropriately mapped to the corresponding basis elements of D, P, and Σ.

Deutsch claimed that there is a UQTM with which is associated a special unitary transformation U that when applied a positive integer number of times can come arbitrarily close to applying any desired unitary transformation on a finite number of data qubits. Stated differently, the claim was that there exists a UQTM, i.e., a special U, so that for an arbitrary accuracy ϵ and arbitrary unitary transformation $\mathcal{U}$ which changes D to $\mathcal{U}D$, there is always a program state $P(D,\mathcal{U},\epsilon)$ and a positive integer $s=s(D,\mathcal{U},\epsilon)$, such that

$$U^s|Q,D,P,\Sigma\rangle = |Q',D',P',\Sigma\rangle,$$

where D' is arbitrarily close to $\mathcal{U}$D, i.e., $\|D'-\mathcal{U}D\|^2<\epsilon$. Finally, like the QTM, the UQTM contains a special internal halt qubit $|h\rangle$ that is monitored to determine whether the computation has completed.

IV. IS THE HALTING SCHEME VALID?

In 1997 Myers [8] suggested that the UQTM's halting scheme was invalid. He argued that an entanglement between the halt qubit and other qubits could occur, thereby making it impossible to determine whether the machine has halted or not. His reasoning was as follows: suppose that two computations, A and B, halt after N_A and N_B steps, respectively, and without loss of generality, that $N_B>N_A$. Then for a computation that is a superposition of computations A and B, after N steps of the UQTM with $N_A<N<N_B$, the halt qubit will be in a superposition of halted and not halted states due to the linearity of the quantum evolution.

Because the computation time is unknown *a priori*, measurement of the halt qubit would collapse the state of the machine to that corresponding to the intermediate computation state of B (with $|h\rangle = |0\rangle$) or to the completed computation state of A (with $|h\rangle = |1\rangle$). Myers argued that this was a conflict between being universal and "being fully quantum," i.e., that the UQTM halting scheme was incompatible with superposition and hence the machine would need to operate on classic states. Conceptually, one could argue that this is not really a problem because any program will ultimately generate a single result. The case of superposed programs corresponds to the classical case of running a program with random data. The computation result depends on the data. In the superposed quantum computer case, the result obtained depends on the final measurement probabilities for obtaining each of the superposed program results.

In 1998 Ozawa [10] showed that monitoring $|h\rangle$ is a quantum nondemolition measurement, that is, periodic measurement of $|h\rangle$ while the computation is in progress does not alter the final measurement of the memory tape contents, which store the result of the computation. This is true even if $|h\rangle$ becomes entangled with other qubits during the computation. The crucial aspect of this proof is that the probabilities of obtaining each of the possible superposed results are not altered by periodic measurement of the halt qubit. The periodic measurement could be said to collapse the machine to one of the many superposed branches of computation as Myers aptly highlighted, but the probability of measuring that particular computational branch is no different than if the measurement is postponed until after the program has completed execution. The key assumption or requirement in Ozawa's proof is that the state of the memory tape remains unchanged once the halt qubit is set.

Furthermore, in 2001 Shi [9] also highlighted that universality and "being fully quantum" does not require the *entire* UQTM to evolve from a superposition. The superposition need only be on the *data* state. For example, if the data state is $|D\rangle = |A\rangle + |B\rangle$, the state of the total system starts at $|Q, A+B, P(A+B, \mathcal{U}, \epsilon), \Sigma\rangle$, rather than at $|Q, A, P(A, \mathcal{U}, \epsilon), \Sigma\rangle + |Q, B, P(B, \mathcal{U}, \epsilon), \Sigma\rangle$.

However, the scenario highlighted by Myer would arise if one were to require that the program be only dependent on the desired transformation $\mathcal{U}$ and the accuracy ϵ, but independent of the initial data state. In this case, a computation on data state $D = A + B$ would need to start at $|Q, A+B, P(\mathcal{U}, \epsilon), \Sigma\rangle$, or $|Q, A, P(\mathcal{U}, \epsilon), \Sigma\rangle + |Q, B, P(\mathcal{U}, \epsilon), \Sigma\rangle$. Hence in this case entanglement between the halt qubit and the rest of the system would occur if the execution times for A and B were different, which would be generally the case. However, the requirement for a data state independent program is unnecessary and the halt qubit entanglement problem could thus be avoided. Also if we require the programs to be data state independent and the halt qubit becomes entangled, Ozawa's proof applies and periodic measurements of the halt qubit do not affect the outcome of the computation.

However, Shi also pointed out that the halting scheme is a special case of the program *concatenation* scheme that was assumed to be valid in the original UQTM proposal. The original definition of the UQTM is based on the assumption that if there is a program whose effect is to apply $\mathcal{U}$ on the data state $|D\rangle$, then there exists a unitary operator whose effect is $|h=1\rangle\langle h=0| \otimes \mathcal{U}$ on $|h=0\rangle|D\rangle$. This assumption was not proven and the validity of program concatenation has not been addressed in other work (see Sec. 8.3 in [11], for example) that relies upon the QTM defined by Bernstein and Vazirani [12] in 1997; this version of the QTM not only requires a halting scheme like Deutsch's but also requires that every computational path reach a final configuration simultaneously, and thus every computational path must be somehow synchronized.

The problem with synchronizing every computational path is that, in general, it is not known *a priori* how long a program will take to halt or if it will halt at all because program execution times can depend on the data that the program operates upon. This problem was highlighted by several authors, including Iriyama, Miyadera, and Ohya as recently as 2008 [13]. Thus, it is not always possible to find an upper bound T on the time needed for all branches to halt and thereby equip each branch of a computation with a counter that increments at each time step and halts once it reaches some upper bound T. In essence, such a synchronization scheme is well suited for dealing with sequential programs that are guaranteed to halt but not for programs that may never halt due to conditional branches or loops.

We address these open questions by constructing a theoretical universal quantum computer with valid and explicit halting and program concatenation schemes, and which also supports conditional branching and does not require synchronization of all computational paths. This machine serves as a prototypical model for a general-purpose programmable quantum computer that will be useful in the development and analysis of new quantum algorithms, complexity analysis of quantum algorithms, and investigation of the physical basis of the Turing halting problem.

V. UNIVERSAL QUANTUM COMPUTER

Our goal is to devise a quantum computer that can compute any computable function. The machine itself is to be fixed and each different function is to be computed by providing the machine with a suitable set of input data and program. Any unitary operation can be approximated to any desired accuracy using the set of {H,CNOT,T} gates (see Nielsen and Chuang [4], Chap. 4, for example), where $\mathrm{H} \overset{\text{def}}{=} \frac{1}{\sqrt{2}}\{(|0\rangle + |1\rangle)\langle 0| + (|0\rangle - |1\rangle)\langle 1|\}$, $\mathrm{CNOT} \overset{\text{def}}{=} |00\rangle\langle 00| + |01\rangle\langle 01| + |11\rangle\langle 10| + |10\rangle\langle 11|$, and $\mathrm{T} \overset{\text{def}}{=} |0\rangle\langle 0| + e^{i\pi/4}|1\rangle\langle 1|$. This set is universal in the sense that any function (i.e., unitary operation) that can be computed by a quantum computer can be implemented using a combination of these gates. Thus, to create a universal quantum computer in the programmable sense, it suffices to devise one that can implement these operations on a specified set of qubits under the control of a quantum program. The quantum computer described below and illustrated in Fig. 1 is an instance of such a machine.

Following Deutsch [7], our machine $\mathcal{UQC}$ consists of two primary parts: a processor $\mathcal{Q}$ that implements the universal

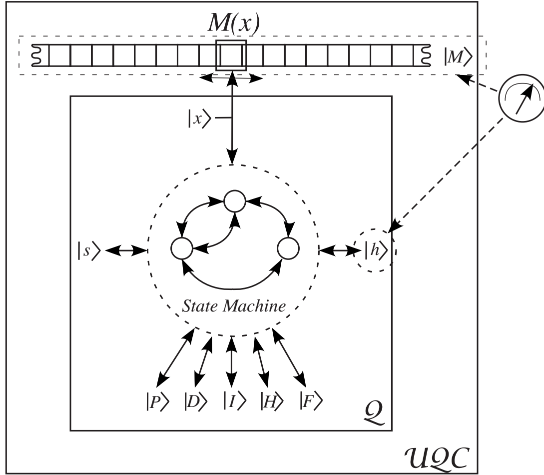

FIG. 1. Architecture of the universal quantum computer $\mathcal{UQC}$, showing the memory tape M, processor $\mathcal{Q}$, address of tape head $|x\rangle$, scratch qubit $|s\rangle$, instruction register $|I\rangle$, program address register $|P\rangle$, data address register $|D\rangle$, history address register $|H\rangle$, flow control register $|F\rangle$, halt qubit $|h\rangle$, and the qubits that are measured ($|M\rangle$ and $|h\rangle$).

set of unitary operations and an infinite tape that acts as the machine's external memory. The tape consists of an infinite sequence of qubits, $|M\rangle = \{|m_i\rangle\}, i \in \mathbb{Z}$, with only a finite subset of them ever being used for any given program. This corresponds to a classical computer's memory and external storage which, while finite, can be arbitrarily large. With the tape is associated an observable $\hat{x}$ in the processor that has the whole of $\mathbb{Z}$ as its spectrum and that acts as the address number of the currently scanned tape location. Addressing different tape locations can be realized either by a movable head over a static tape or by a movable tape under a static head. Since either scheme is identical for the purposes of constructing $\mathcal{UQC}$, we assume the latter as that allows for $\mathcal{Q}$ to be fixed in space, and movement of the tape is accomplished by a sliding "bin" of qubits that moves under $\mathcal{Q}$'s control.

As part of its internal state machine, $\mathcal{Q}$ also contains two additional observables, $\hat{D}$ and $\hat{P}$, that act as the data address and program counter, respectively. $\hat{D}$ is used to address individual data qubits on the tape and to specify the branch destination address and $\hat{P}$ is used to keep track of the program instruction that is to be executed. As with classical computers, $\hat{D}$ and $\hat{P}$ need not have an infinite spectrum as they need only be as "wide" as required to address the finite subset of the infinite tape that would ever be used. However, for the purpose of the most general construction, we do not restrict $\mathcal{UQC}$ to have a particular address range and thus treat $\hat{D}$ and $\hat{P}$ (and $\hat{x}$) as having an infinite spectrum.

$\mathcal{Q}$ also contains a four-qubit register $\hat{I}$ to load the instruction to be executed. In order to perform the two-qubit CNOT operation, $\mathcal{Q}$ contains a "scratch" qubit $|s\rangle$ that is used as the control qubit. Like Deutsch's UQTM, $\mathcal{UQC}$ also contains a dedicated observable qubit $|h\rangle$ that indicates whether the program execution has completed (i.e., the halt qubit). $\mathcal{Q}$ also contains a two-qubit register $\hat{F}$ that is used to control the execution flow (i.e., whether the program should loop on the current instruction, proceed to the next instruction, or branch to a new instruction). Finally, $\mathcal{UQC}$ contains a register $\hat{H}$ with the same spectrum as $\hat{x}$, $\hat{D}$, and $\hat{P}$. The purpose (and naming) of the $\hat{H}$ register is described later. For notational simplicity, we drop the $\hat{}$ notation hereafter when referring to $\mathcal{UQC}$ registers, e.g., D refers to the observable $\hat{D}$ whose corresponding state is $|D\rangle$.

The overall state of $\mathcal{UQC}$, then, is given by $|h,x,D,P,F,H,s,I,M\rangle$, where $|h,D,P,F,H,s,I\rangle$ corresponds to Deutsch's $|\mathbf{n}\rangle$ with $|h\rangle \overset{\text{def}}{=} |n_0\rangle$.

Each program consists of a finite sequence of four qubit instruction words. Self-modifying code is to be avoided because modifying program instructions during program execution can lead to unpredictable results. For example, the processor fetches instructions to be executed from the memory tape into the temporary internal buffer register I by swapping the contents of the memory tape and the I register (and swapping back the two when the instruction has been executed). Because I is initialized to $|0\rangle$, the swapped contents of the memory tape temporarily become $|0\rangle$ while the instruction is being executed. This means that if the program attempts to modify the location of the instruction being executed, it would be modifying $|0\rangle$ and not the actual instruction (that is temporarily held in the I register). This can lead to unintended and unpredictable behavior.

The instruction set of $\mathcal{UQC}$ is as follows. As mentioned earlier, we implement a universal set of unitary operations, namely {H,CNOT,T}, in order to ensure that $\mathcal{UQC}$ is universal. In order to enable the programmer to address any qubit on the memory tape and thus apply the universal set of operations to any qubit, we implement three instructions: an instruction to set D to 0, an instruction to increment D by 1, and an instruction to decrement D by 1. Because the CNOT operation requires two operands (control and data), we implement a swap instruction to enable the programmer to swap the qubit on the memory tape pointed to by D with the machine's s qubit, thereby enabling any qubit on the memory tape to be used as the control qubit. While not strictly necessary for universality, we implement a branching scheme in $\mathcal{UQC}$ because first, this is not explicitly possible in other popular quantum computing frameworks such as the gate array framework and second, because it is a common operation in classical computers. Branching is essentially implemented by allowing the programmer to swap the data register and program counter register contents, thereby allowing the program to branch to any instruction on the memory tape. We also implement an instruction to effectively clear s by swapping its contents with the next available 0 slot on the negative portion of the memory tape (pointed to by H). The clear s instruction provides for a simple and convenient way for the programmer to load s with 0 without having to hunt around the memory tape looking for a 0 data qubit slot. Finally, we implement an instruction to set the halt qubit to 1 but because we also want the memory tape to remain unchanged once the halt qubit is set, we implement an accompanying instruction (NOP) to follow the halt instruction that will accomplish this.

TABLE I. $\mathcal{UQC}$ instruction set.

Label	Encoding	Description
$\lvert\mathrm{NOP}\rangle$	$\lvert 0000\rangle$	No operation
$\lvert D\rightarrow 0\rangle$	$\lvert 0001\rangle$	$D\rightarrow 0$
$\lvert D+1\rangle$	$\lvert 0010\rangle$	$D\rightarrow D+1$
$\lvert D-1\rangle$	$\lvert 0011\rangle$	$D\rightarrow D-1$
$\lvert\mathrm{H}\rangle$	$\lvert 0100\rangle$	Apply Hadamard operation to $\lvert M\rangle_D$
$\lvert\mathrm{T}\rangle$	$\lvert 0101\rangle$	Apply T operation to $\lvert M\rangle_D$
$\lvert\mathrm{SWAP}\rangle$	$\lvert 0110\rangle$	$\lvert M\rangle_D\leftrightarrow\lvert s\rangle$
$\lvert\mathrm{CNOT}\rangle$	$\lvert 0111\rangle$	CNOT of $\lvert M\rangle_D$ and $\lvert s\rangle$ ($\lvert s\rangle$: control)
$\lvert D\leftrightarrow P\rangle$	$\lvert 1000\rangle$	$\lvert D\rangle\leftrightarrow\lvert P\rangle$ (branch) if $s=0$
$\lvert\mathrm{CLS}\rangle$	$\lvert 1001\rangle$	Clear s
$\lvert\mathrm{R}_0\rangle$	$\lvert 1010\rangle$	Unused
$\lvert\mathrm{R}_1\rangle$	$\lvert 1100\rangle$	Unused
$\lvert\mathrm{R}_2\rangle$	$\lvert 1101\rangle$	Unused
$\lvert\mathrm{R}_3\rangle$	$\lvert 1110\rangle$	Unused
$\lvert h\rightarrow 1\rangle$	$\lvert 1111\rangle$	$\lvert h\rangle\rightarrow\lvert 1\rangle$ (set halt qubit)

The instruction set of $\mathcal{UQC}$, then, consists of 11 instructions, whose operations and encodings are defined in Table I. The single qubit operations H and T act on the qubit at tape location $M(D)$, denoted $\lvert M\rangle_D$, and the two qubit operations SWAP and NAND act on $\lvert M\rangle_D$ and the scratch qubit $\lvert s\rangle$, the latter being used as the control qubit for the NAND operation.

The operation of $\mathcal{UQC}$ proceeds as follows:

(1) An external operator (or classical computer) initializes the state of M at $t=0$ with the desired data and program. Data qubit i, $i\in\mathbb{Z}^+$, is placed on tape location $M(5i-1)$ and program instruction j, $j\in\mathbb{Z}^+$, is placed on tape locations $M[5j-2:5(j-1)]$, i.e., data are placed at $M(4), M(9), M(14), \ldots$, and program instructions are placed at $M(3:0), M(8:5), M(18:15), \ldots$. The negative portions of the tape are initialized to $\lvert 0\rangle$, as illustrated in Fig. 2.

(2) The processor registers are all initialized to $\lvert 0\rangle$.

(3) An external operator starts $\mathcal{Q}$ by releasing it from the reset state.

(4) $\mathcal{Q}$ fetches the program instruction at tape location $M(P)$ into register I.

(5) $\mathcal{Q}$ executes the operation specified by I.

(6) If the halt qubit $\lvert h\rangle$ becomes set, $\mathcal{Q}$ halts execution (strictly speaking, because $\mathcal{UQC}$ is a quantum system, $\mathcal{Q}$ continues to evolve but the evolution of the memory tape becomes trivial—i.e., $U=\mathbb{1}$—after the halt qubit has been set) and awaits an external measurement of the results. Otherwise, $\mathcal{Q}$ continues execution of the program by loading the next program instruction.

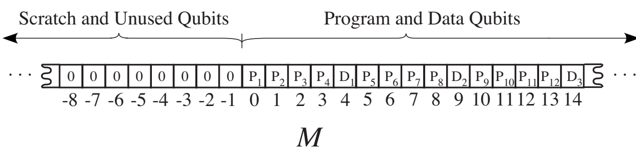

FIG. 2. Initial memory tape contents. The negative qubit slots are used as scratch qubits and the non-negative qubit slots are initialized with interleaved program instruction and data qubits.

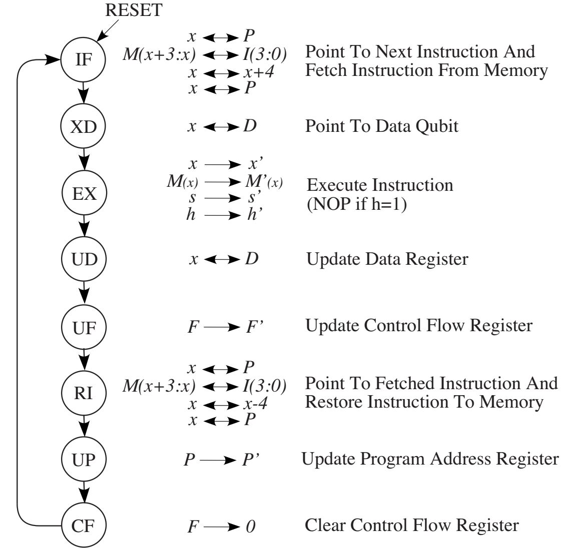

FIG. 3. $\mathcal{Q}$ state machine diagram that corresponds to the evolution of the universal quantum computer. The overall evolution is determined by eight unitary transformations.

(7) An external operator periodically performs a measurement on the halt qubit.

(8) If measurement of the halt qubit yields $\lvert 1\rangle$, the program has completed execution. The results are obtained by measuring the contents of M. Otherwise, $\mathcal{Q}$ is allowed to continue program execution.

The operation of $\mathcal{Q}$ is governed by the state machine depicted in Fig. 3.

A. Evolution of $\mathcal{Q}$

We now define the unitary evolution operators associated with the $\mathcal{Q}$ state transitions. In the equations below, subscripts on projectors denote the qubit(s) on which the projector acts, e.g., $\lvert i\rangle\langle i\rvert_k$ acts on qubit k, and unspecified qubits are understood to be operated on by an implicit identity operator, e.g., $\lvert i\rangle\langle j\rvert_k\otimes\lvert l\rangle\langle m\rvert_n$ is short hand for $\lvert i\rangle\langle j\rvert_k\otimes\lvert l\rangle\langle m\rvert_n\otimes\mathbb{1}_{\neq k,n}$ which acts on qubits k and n and leaves all other qubits unaffected. $\lvert\psi\rangle_R$ denotes $\Pi_i\lvert\psi(i)\rangle_{R(i)}$, where R is a multiple qubit register (e.g., D) with $R=\Pi_i R(i)$ and $\psi=\Pi_i\psi(i)$. Moreover, for notational simplicity in the rest of this paper, we define four primitive unitary operations, SWAP, DEC, INC, and NAND as follows:

(1) Swap contents of registers a and b,

$$\mathrm{SWAP}_{a,b}\overset{\text{def}}{=}\sum_{i,j}\lvert i\rangle\langle j\rvert_a\otimes\lvert j\rangle\langle i\rvert_b. \quad (1)$$

(2) Decrement the contents of register a,

$$\mathrm{DEC}_a\overset{\text{def}}{=}\sum_i\lvert i-1\rangle\langle i\rvert_a. \quad (2)$$

(3) Increment the contents of register a,

$$\mathrm{INC}_a\overset{\text{def}}{=}\sum_i\lvert i+1\rangle\langle i\rvert_a. \quad (3)$$

(4) CNOT operation using qubit a as the control and qubit b as the data,

$$\mathrm{CNOT}_{a,b} \stackrel{\mathrm{def}}{=} (|00\rangle\langle 00| + |01\rangle\langle 01| + |11\rangle\langle 10| + |10\rangle\langle 11|)_{a,b}. \tag{4}$$

The operators, U_{IF}, U_{XD}, U_{EX}, U_{UD}, U_{UF}, U_{RI}, U_{UP}, and U_{CF}, which govern the $\mathcal{Q}$ state transitions, then, are defined as follows.

(1) Fetch next instruction at $M(P)$,

$$U_{\mathrm{IF}} \stackrel{\mathrm{def}}{=} \mathrm{DEC}_P^4 \cdot \mathrm{SWAP}_{x,P} \cdot \left(\prod_{i=3}^{0} \mathrm{INC}_x \cdot \mathrm{SWAP}_{M(x),I(i)} \right) \mathrm{SWAP}_{x,P}. \tag{5}$$

This operator fetches the next program instruction by "swapping" the next program instruction qubits on the memory tape with the contents of the I register. As stated earlier, because I is initialized to $|0\rangle$, the instruction slot on the memory tape becomes temporarily $|0\rangle$ while the instruction is being executed but is restored to its original state once the instruction has been executed. Note that P will be pointing back to the fetched instruction address after this operator is applied because the update of the program counter is deferred until U_{UP} is applied.

(2) Move tape head to $M(D)$,

$$U_{\mathrm{XD}} \stackrel{\mathrm{def}}{=} \mathrm{SWAP}_{x,D}. \tag{6}$$

U_{XD} points the memory tape head to the qubit addressed by the data register D.

(3) Execute instruction

$$\begin{aligned} U_{\mathrm{EX}} \stackrel{\mathrm{def}}{=} & |\mathrm{NOP}\rangle\langle\mathrm{NOP}|_I + |D \to 0\rangle\langle D \to 0|_I \otimes \mathrm{DEC}_x \\ & + |D+1\rangle\langle D+1|_I \otimes \mathrm{INC}_x + |D-1\rangle\langle D-1|_I \otimes \mathrm{DEC}_x \\ & + |\mathrm{H}\rangle\langle\mathrm{H}|_I \otimes \mathrm{H}_{M(x)} + |\mathrm{T}\rangle\langle\mathrm{T}|_I \otimes \mathrm{T}_{M(x)} \\ & + |\mathrm{SWAP}\rangle\langle\mathrm{SWAP}|_I \otimes \mathrm{SWAP}_{M(x),s} \\ & + |\mathrm{CNOT}\rangle\langle\mathrm{CNOT}|_I \otimes \mathrm{CNOT}_{s,M(x)} \\ & + |D \leftrightarrow P\rangle\langle D \leftrightarrow P|_I + |\mathrm{CLS}\rangle\langle\mathrm{CLS}|_I \\ & \otimes \mathrm{SWAP}_{x,H} \cdot \mathrm{DEC}_x \cdot \mathrm{SWAP}_{M(x),s} \cdot \mathrm{SWAP}_{x,H} \\ & + |h \to 1\rangle\langle h \to 1|_I \otimes (|1\rangle\langle 0| + |0\rangle\langle 1|)_h + \sum_{i=0}^{4} |\mathrm{R_i}\rangle\langle\mathrm{R_i}|_I. \end{aligned} \tag{7}$$

U_{EX} applies the appropriate transformation associated with the instruction being executed. The transformations associated with the NOP, $D+1$, $D-1$, H, T, SWAP, CNOT, and reserved instructions are self-evident but those associated with the $D\to 0$, $D \leftrightarrow P$, CLS, and $h\to 1$ instructions warrant some explanation.

The $D\to 0$ instruction works as follows. D is decremented by one by U_{EX} and P is left unchanged by U_{UP} [see Eq. (11)] until $D=0$. Leaving the program counter unchanged has the effect of keeping P pointing to the $D\to 0$ instruction such that it is refetched in the next iteration. Thus, $\mathcal{Q}$ continues to fetch and execute the same $D\to 0$ instruction until $D=0$. In other words, it will loop on the $D\to 0$ instruction, decrementing D until it reaches 0. Once $D=0$, P is incremented by 5 such that it points to the next instruction, thus completing the loop.

It is important to note that this scheme relies on the assumption that $D>0$ when the $D\to 0$ instruction is encountered. Therefore, the programmer must ensure that $D>0$ when $\mathcal{Q}$ fetches the $D\to 0$ instruction. This can be accomplished by preceding the $D\to 0$ instruction with a $D+1$ instruction since, in the absence of programming error, D will always be positive. If the programmer fails to meet this requirement $\mathcal{Q}$ could loop forever stepping through the negative portions of the memory tape.

The transformation associated with the $D \leftrightarrow P$ operation is the identity operation here because its execution is deferred until later. Deferring the actual swapping of the D and P register contents is necessary in order to keep the address of the $D \leftrightarrow P$ instruction unchanged so that we can restore the branch instruction back to its original slot on the memory tape and only then update the program counter to point to the next instruction in the program execution flow.

The CLS instruction first points the memory tape head to the address contained in the H register (the next slot on the negative portion of the memory tape that contains $|0\rangle$), swaps the contents of the s qubit with the contents of the memory tape slot ($|0\rangle$) thereby clearing s (but leaving the previous value of s on the memory tape making the operation reversible in principle), decrements H such that it points to the next $|0\rangle$ slot on the memory tape, and then points the memory tape head back to where it was.

The NOP instruction plays a key role in the $\mathcal{UQC}$ halting scheme. In our implementation, the halting scheme requires the halt instruction $|h\to 1\rangle$ to be followed by a $|\mathrm{NOP}\rangle$ instruction. In other words, the "true" halt instruction is effectively $|h\to 1\rangle|\mathrm{NOP}\rangle$ or $|11110000\rangle$ using the presently defined instruction encodings. This is such that after the halt qubit is set, $\mathcal{Q}$ will continue to fetch the next instruction following $|h\to 1\rangle$ which being $|\mathrm{NOP}\rangle$ will guarantee that $\mathcal{Q}$ loops forever doing nothing, thereby effectively halting program execution (but not quantum evolution). The fact that the encoding of the NOP instruction is $|0000\rangle$ is also intentional. This ensures that the contents of the memory tape remain unchanged after the halt qubit is set because swapping the instruction slot on the memory tape with the contents of the I register leaves the state of the memory tape unchanged. The halting scheme relies on all halt instructions in any given program being followed by a NOP instruction and stopping the program counter from changing when a NOP instruction is executed such that P will continue to point at the NOP instruction following the instruction that caused the halt qubit to be set.

The halting scheme is thus effectively a two step process: the first step is to set the halt qubit using the $h\to 1$ instruction to alert an external observer that the program has halted and the second step is to loop forever on the NOP instruction. In this sense, the NOP instruction is really a "loop forever" trap instruction. As such, the NOP instruction must only be used following a halt instruction. If it is inadvertently placed

TABLE II. F encodings.

Label	Encoding	Description
$\lvert\mathrm{LOOP}\rangle$	$\lvert 00\rangle$	Loop ($P \to P$)
$\lvert\mathrm{NEXT}\rangle$	$\lvert 01\rangle$	Next sequential instruction ($P \to P+5$)
$\lvert\mathrm{BR2D}\rangle$	$\lvert 10\rangle$	Branch to D ($P \to D$)
$\lvert\mathrm{R}_0\rangle$	$\lvert 11\rangle$	Unused

anywhere else in the program, program execution will halt but the halt qubit will not be set so the external observer will not know that the program has halted.

An improved halting scheme that does not require all instances of the halt instruction in a program to be followed by a NOP instruction may be possible and is an area for future investigation.

(4) Update contents of D register,

$$U_{\mathrm{UD}} \overset{\mathrm{def}}{=} \mathrm{SWAP}_{x,D}\,. \tag{8}$$

U_{UD} updates the D register with the results of executing the instruction since x will contain any changes to D after U_{EX} has been applied.

(5) Update control flow register with instruction flow information

$$\begin{aligned} U_{\mathrm{UF}} \overset{\mathrm{def}}{=} & [\lvert D \to 0\rangle\langle D \to 0\rvert_I \otimes (\mathbb{1} - \lvert 0\rangle\langle 0\rvert)_D + \lvert \mathrm{NOP}\rangle\langle \mathrm{NOP}\rvert_I] \\ & + \lvert D \leftrightarrow P\rangle\langle D \leftrightarrow P\rvert_I \otimes \lvert 0\rangle\langle 0\rvert_s \otimes \mathrm{INC}_F^2 \\ & + [\mathbb{1} - \lvert D \to 0\rangle\langle D \to 0\rvert_I \otimes (\mathbb{1} - \lvert 0\rangle\langle 0\rvert)_D - \lvert \mathrm{NOP}\rangle\langle \mathrm{NOP}\rvert_I \\ & - \lvert D \leftrightarrow P\rangle\langle D \leftrightarrow P\rvert_I \otimes \lvert 0\rangle\langle 0\rvert_s] \otimes \mathrm{INC}_F \end{aligned} \tag{9}$$

U_{UF} updates F whose value is later used to update P to point to the address of the next instruction to be executed. Note that F is initialized to $\lvert 0\rangle$ and the evolution of $\mathcal{UQC}$ is designed to ensure that $F=0$ when U_{UF} is applied (F is effectively "cleared" by U_{CF} by swapping its contents with the infinite supply of $\lvert 0\rangle$ slots on the negative portion of the memory tape as we describe later). As explained earlier, if the instruction is $D \to 0$ and $D \neq 0$ or if the instruction is NOP, P will be left unchanged to effectively loop on the instruction. If the instruction is $D \leftrightarrow P$ and $s=0$ then P will be swapped with D to effectively branch to D. Otherwise, P is set to point to the instruction following the instruction that was just executed (i.e., $P \to P+5$). The encodings of F are defined in Table II.

(6) Restore executed instruction back to the memory tape location from where it was fetched,

$$U_{\mathrm{RI}} \overset{\mathrm{def}}{=} U_{\mathrm{IF}}^{\dagger}\,. \tag{10}$$

U_{RI} restores the instruction that was just executed back to its original slot on the memory tape. Recall that the P update has been deferred and will be controlled by the state of the F register. Thus, the only operator that has affected P thus far has been U_{IF} so $U_{\mathrm{IF}}^{\dagger}$ suffices to undo the fetch. In essence, F is a temporary place holder to store the information necessary to determine the next instruction location after restoring the instruction back to the memory tape and hence losing knowledge of how to update P otherwise.

(7) Update program counter to the address of the next instruction to be executed,

$$\begin{aligned} U_{\mathrm{UP}} \overset{\mathrm{def}}{=} & \lvert \mathrm{LOOP}\rangle\langle \mathrm{LOOP}\rvert_F + \lvert \mathrm{NEXT}\rangle\langle \mathrm{NEXT}\rvert_F \otimes \mathrm{INC}_P^5 \\ & + \lvert \mathrm{BR2D}\rangle\langle \mathrm{BR2D}\rvert_F \otimes \mathrm{SWAP}_{D,P} + \lvert \mathrm{R}_0\rangle\langle \mathrm{R}_0\rvert_F. \end{aligned} \tag{11}$$

U_{UP} updates P to the address of the next instruction to be executed according to the state of F.

(8) Clear flow control register such that it can be used again in the next cycle,

$$U_{\mathrm{CF}} \overset{\mathrm{def}}{=} \mathrm{SWAP}_{x,H}\left(\prod_{i=1}^{0} \mathrm{DEC}_x \cdot \mathrm{SWAP}_{M(x),F(i)}\right)\mathrm{SWAP}_{x,H}\,. \tag{12}$$

U_{CF} first swaps the contents of the x and H registers. The H register contains the address of the next slot on the negative portion of the tape that contains $\lvert 00\rangle$. These "0" slots are used to clear the F register back to $\lvert 0\rangle$ each cycle. Since the sequence of F values effectively contains the information about the program execution flow, in essence the negative portion of the tape contains the "history" of instructions that $\mathcal{UQC}$ has executed and is a side effect of the need for all $\mathcal{UQC}$ programs to be reversible.

In other words, the negative portion of M is used to store the ancillary garbage data that would be required to reverse the operation of the program. The number of $\lvert 0\rangle$ slots required for any given program is equal to the number of instructions that are executed by the program. U_{CF} clears F by swapping its contents with the contents of the next $\lvert 0\rangle$ slot on the negative portion of the tape. After application of U_{CF}, H points to the next $\lvert 0\rangle$ slot on the tape and the previous F value is contained on the slot to the right of the first $\lvert 0\rangle$ slot on the negative portion of the tape. At this point, $\mathcal{Q}$ has completed processing the instruction and is ready to fetch the next instruction in the execution flow.

These operators are all readily verified to be unitary and to have the desired effects of implementing the state machine shown in Fig. 3. The overall evolution of $\mathcal{UQC}$, then, is governed by the unitary operator,

$$U = U_{\mathrm{CF}} U_{\mathrm{UP}} U_{\mathrm{RI}} U_{\mathrm{UF}} U_{\mathrm{UD}} U_{\mathrm{EX}} U_{\mathrm{XD}} U_{\mathrm{IF}}. \tag{13}$$

Unlike Deutsch's original UQTM, the memory tape (or tape head) of $\mathcal{UQC}$ is not restricted to move at most one position to the left or to the right ($x \to x \pm 1$) in any given step. This is most obvious in the case of the branch instruction where the tape head will jump by an arbitrarily large amount in a single step. However, the evolution of $\mathcal{UQC}$ is still unitary and hence physically possible in principle.

VI. SOME PRIMITIVE PROGRAMS

In this section we describe a set of primitive programs or operations to demonstrate the universal nature of $\mathcal{UQC}$. These routines serve as building blocks for devising and analyzing more complicated and useful programs.

The first set of primitive programs, $\{|D_{+i}\rangle, |D_{-i}\rangle, |D_i\rangle, |S_{i,s}\rangle, |S_{i,j}\rangle, |B_i\rangle\}$, that we define perform basic functions to manipulate the data address register, swap qubits, and conditionally branch to an arbitrary address. The superscripts on the programs denote the operation performed by the program and the subscripts indicate the qubits on which the program operates. For notational simplicity, $|P_h\rangle$ denotes the program that causes $\mathcal{UQC}$ to halt, i.e., $|P_h\rangle \overset{\text{def}}{=} |h \to 1\rangle|\text{NOP}\rangle$.

(1) $|D_{+i}\rangle$: Increment D by i,

$$|D_{+i}\rangle \overset{\text{def}}{=} \begin{cases} \prod_{k=1}^{i} |D+1\rangle, & \text{if } i \geq 1 \\ \mathbb{1}, & \text{otherwise.} \end{cases} \tag{14}$$

(2) $|D_{-i}\rangle$: Decrement D by i,

$$|D_{-i}\rangle \overset{\text{def}}{=} \begin{cases} \prod_{k=1}^{i} |D-1\rangle, & \text{if } i \geq 1 \\ \mathbb{1}, & \text{otherwise.} \end{cases} \tag{15}$$

(3) $|D_i\rangle$: Set D to i, $i>0$,

$$|D_i\rangle \overset{\text{def}}{=} |D+1\rangle|D \to 0\rangle|D_{+i}\rangle. \tag{16}$$

Recall from the discussion of U_{EX} that we are preceding the $D \to 0$ instruction with a $D+1$ instruction to ensure that $D > 0$ when the $D \to 0$ instruction is executed.

(4) $|S_{i,s}\rangle$: Swap data qubits $D(i)$ and s,

$$|S_{i,s}\rangle \overset{\text{def}}{=} |D_{5i-1}\rangle|\text{SWAP}\rangle. \tag{17}$$

(5) $|S_{i,j}\rangle$: Swap data qubits $D(i)$ and $D(j)$,

$$|S_{i,j}\rangle \overset{\text{def}}{=} |S_{5i-1,s}\rangle|S_{5j-1,s}\rangle|S_{5i-1,s}\rangle. \tag{18}$$

(6) Branch to the ith instruction [i.e., instruction at $M(5(i-1))$], where $i \in \mathbb{Z}^+$,

$$|B_i\rangle \overset{\text{def}}{=} |D_{5(i-1)}\rangle|D \leftrightarrow P\rangle. \tag{19}$$

Note that, as defined, this instruction will have no effect unless $|s\rangle = |0\rangle$ so this operation is only useful following nontrivial operations on the $|s\rangle$ qubit.

Next we describe a set of programs, $\{|P_i^{\text{H}}\rangle, |P_{i,j}^{\text{H}}\rangle, |P_i^{\text{T}}\rangle, |P_{i,j}^{\text{C}}\rangle\}$, to apply the H, T, and CNOT operations on arbitrary qubits i and j on the memory tape, where i and $j \in \mathbb{Z}$. These comprise a universal set of unitary operations from which any arbitrary unitary operation can be constructed.

(1) $|P_i^{\text{H}}\rangle$: Apply H to data qubit $D(i)$,

$$|P_i^{\text{H}}\rangle \overset{\text{def}}{=} |D_{5i-1}\rangle|\text{H}\rangle. \tag{20}$$

(2) $|P_{i,j}^{\text{H}}\rangle$: Apply H to data qubits $D(i:j)$, where $i \geq j$,

$$|P_{i,j}^{\text{H}}\rangle \overset{\text{def}}{=} \prod_{k=j}^{i} |P_k^{\text{H}}\rangle. \tag{21}$$

One could implement this program using a loop but that would require first implementing binary addition of M qubits. Binary addition is possible because one can implement a binary adder such as a Carry Lookahead Adder [14] using the NAND program that we define later in this section. However, since we are only interested in a polynomial order (in the number of qubits) multiple qubit Hadamard transformation program, we define $|P_{i,j}^{\text{H}}\rangle$ as a sequential "unrolled" loop program.

(3) $|P_i^{\text{T}}\rangle$: Apply T to data qubit $D(i)$,

$$|P_i^{\text{T}}\rangle \overset{\text{def}}{=} |D_{5i-1}\rangle|\text{T}\rangle. \tag{22}$$

(4) $|P_{i,j}^{\text{C}}\rangle$: Apply CNOT to data qubits $D(i)$ and $D(j)$ with $D(i)$ as the control qubit,

$$|P_{i,j}^{\text{C}}\rangle \overset{\text{def}}{=} |S_{i,s}\rangle|D_{5j-1}\rangle|\text{CNOT}\rangle|S_{i,s}\rangle. \tag{23}$$

Using the sets of primitive programs defined above, we can now define the set of programs, $\{|P_i^{\text{X}}\rangle, |P_i^{\text{S}}\rangle, |P_i^{\text{T}^\dagger}\rangle\}$, that apply the Pauli X, Phase (S), and $\text{T}^\dagger$ operations on data qubit $i \in \mathbb{Z}^+$. These operations are often used in quantum algorithms so it is useful to identify the programs that implement them. A constant subscript on a program denotes that some suitable qubit on the memory tape has been prepared with the appropriate value. For example, $|P_1\rangle$ is shorthand for $|P_k\rangle$ where $M(k)$, for some suitable k, has been prepared with the value $|1\rangle$.

(1) $|P_i^{\text{X}}\rangle$: Apply σ_x to qubit $M(i)$,

$$|P_i^{\text{X}}\rangle = |P_{1,i}^{\text{C}}\rangle,$$

$$|P_i^{\text{X}}\rangle: |1\rangle_j|\psi\rangle_i \to |1\rangle_j|1 \oplus \psi\rangle_i = |1\rangle_j|\bar{\psi}\rangle_i = |1\rangle_j \sigma_x|\psi\rangle_i. \tag{24}$$

(2) $|P_i^{\text{S}}\rangle$: Apply phase (S) to qubit $M(i)$.

Noting that $\text{S} = \text{T}^2$, the following program implements the phase operation.

$$|P_i^{\text{S}}\rangle = |P_i^{\text{T}}\rangle|P_i^{\text{T}}\rangle$$

$$|P_i^{\text{S}}\rangle: |\psi\rangle_i \to \text{T}^2|\psi\rangle_i = \text{S}|\psi\rangle_i. \tag{25}$$

(3) $|P_i^{\text{T}^\dagger}\rangle$: Apply $\text{T}^\dagger$ (reverse T) to qubit $M(i)$,

This operation is used to define the Toffoli operation in a later section so we define it here. Noting that $\text{S} = \text{T}^2$ and that $S^4 = \mathbb{1}$, $\text{T}^\dagger = \text{T}^\dagger \text{S}^4 = \text{T}^\dagger \text{T}^2 \text{S}^3 = \text{TS}^3$, the following program implements the $\text{T}^\dagger$ operation:

$$|P_i^{\text{T}^\dagger}\rangle = |P_i^{\text{S}}\rangle|P_i^{\text{S}}\rangle|P_i^{\text{S}}\rangle|P_i^{\text{T}}\rangle. \tag{26}$$

Although not specifically shown here, other useful quantum gates such as σ_y, σ_z, entanglement gate, etc. can be similarly implemented. These enable us to implement any algorithm from the quantum gate array framework on $\mathcal{UQC}$ by appropriate combinations of the programs we have just defined and adding $|P_h\rangle$ as the last step in the combined program to halt $\mathcal{UQC}$ upon completion. Since the quantum gate array framework is universal (see [4], Chap. 4, for example), this means that $\mathcal{UQC}$ is also quantum computationally universal with the additional advantage that $\mathcal{UQC}$ provides a fixed and programmable machine to implement the

algorithms unlike the quantum gate array framework.

The two-bit NAND operation is universal for classical computation. That is, the NAND operation can be used to implement any Boolean function. Hence it is useful to define a program that emulates the NAND operation on two qubits as this could be used as the basis for emulating classical functions on $\mathcal{UQC}$. For this purpose, we first define a program that implements the Toffoli operation which itself is a universal classical gate [4]. The Toffoli program, $|P^{\text{Toff}}_{i,j,k}\rangle$, applies the Toffoli operation to qubits $D(i)$, $D(j)$, and $D(k)$, where $D(i)$ and $D(j)$ are the control qubits and $D(k)$ is the target qubit.

Armed with the Toffoli program, implementing a program that takes the NAND of qubits $D(i)$ and $D(j)$ and storing the result in qubit $D(c)$ is a simple matter of executing the program $|P^{\text{NAND}}_{i,j,c}\rangle = |P^{\text{Toff}}_{i,j,1}\rangle$,

$$|P^{\text{Toff}}_{i,j,k}\rangle = |P^{\text{H}}_{k}\rangle|P^{\text{C}}_{j,k}\rangle|P^{\text{T}^\dagger}_{k}\rangle|P^{\text{C}}_{i,k}\rangle|P^{\text{T}}_{k}\rangle|P^{\text{C}}_{j,k}\rangle|P^{\text{T}^\dagger}_{k}\rangle|P^{\text{C}}_{i,k}\rangle|P^{\text{T}}_{k}\rangle|P^{\text{T}^\dagger}_{j}\rangle|P^{\text{C}}_{i,j}\rangle \times|P^{\text{H}}_{k}\rangle|P^{\text{T}^\dagger}_{j}\rangle|P^{\text{C}}_{i,j}\rangle|P^{\text{S}}_{j}\rangle|P^{\text{T}}_{i}\rangle. \tag{27}$$

The ability to perform a two-qubit NAND operation gives $\mathcal{UQC}$ the ability to compute any classically computable function thus demonstrating that it can emulate a classical universal Turing machine. This is in addition to being a universal quantum computer since it can also implement the set of universal quantum operations on arbitrary qubits on its memory tape as shown earlier. $\mathcal{UQC}$ can compute any classically computable function, it can compute any quantum computable function, and it is programmable. In short, $\mathcal{UQC}$ is computationally universal.

VII. PROGRAM CONCATENATION SCHEME

In the process of defining the primitive programs in the preceding section, we have implicitly used program concatenation whereby we sequentially combined separate programs to create larger programs. Strictly speaking, the programs that we have thus far defined are really subroutines since complete programs must include the halting program, $|P_h\rangle$, in order to signal program completion. However, it is readily seen that all of the primitive subroutines can be converted into full-fledged programs by adding $|P_h\rangle$ as the last instruction.

Sequential programs (programs without branch instructions) can thus be concatenated by simply removing the last $|P_h\rangle$ step from each constituent program, concatenating the resulting subroutines, and appending on $|P_h\rangle$ at the end. Suppose that we have two sequential programs, $|P_A\rangle$ and $|P_B\rangle$, that we wish to concatenate to create a program $|P_{AB}\rangle$ whose effect is to execute $|P_B\rangle$ followed by $|P_A\rangle$. Since $|P_A\rangle$ and $|P_B\rangle$ are sequential, this means that $|P_h\rangle$ is the last step in each program. That is, $|P_A\rangle = |P_{A'}\rangle|P_h\rangle$ and $|P_B\rangle = |P_{B'}\rangle|P_h\rangle$. Thus, to achieve the effect of running $|P_A\rangle$ followed by $|P_B\rangle$, we simply construct the program $P_{AB} \overset{\text{def}}{=} |P_{A'}\rangle|P_{B'}\rangle|P_h\rangle$.

The situation is quite different for branching programs. In general, without complete knowledge of the operations of the programs to be concatenated, it is not possible to concatenate them in the strictest sense of joining the individual programs into a single larger program. This is not a limitation of $\mathcal{UQC}$ but of any computer, be it classical or quantum. The problem is that the branch destinations in a branching program can be data dependent and the branching address may also be manipulated as data. Therefore, it is not sufficient to add an appropriate offset (the number of instructions of preceding concatenated programs) to all branch instructions because this would have the adverse effect of potentially adding an offset to the manipulated data and hence altering the intended computation results.

The solution, of course, is to first run $|P_A\rangle$, wait for it to complete, replace $|P_A\rangle$ with $|P_B\rangle$, reset the program counter register to 0, leave all other internal registers and memory tape qubits unchanged, and resume execution to run $|P_B\rangle$. However, strictly speaking, this is not program concatenation *per se* because while the overall operation has the effect of running $|P_A\rangle$ followed by $|P_B\rangle$, the program that is run is not $|P_A\rangle|P_B\rangle$. There is the intermediate step of replacing $|P_A\rangle$ with $|P_B\rangle$ and restarting execution, which, strictly speaking, are not program operations. In the context of $\mathcal{UQC}$ this could be achieved by initializing M with $|P_A\rangle$, running $|P_A\rangle$ and once the halt qubit is measured as $|1\rangle$, replacing the program portion of M with $|P_B\rangle$, setting the program counter register to 0 while leaving all other $\mathcal{UQC}$ registers and memory qubits unchanged, and clearing the halt qubit to resume execution of $|P_B\rangle$ with the results of the preceding program(s). This scheme, of course, works not only for branching programs but also for sequential programs.

Formally, our $\mathcal{UQC}$ program concatenation operator, $\Pi^{(n)}$, is defined as

$$\Pi^{(n)} \overset{\text{def}}{=} \left(\sum_{i=1}^{n-1} |P_{i+1}\rangle\langle P_i|_{M(P)} \otimes |0\rangle\langle 1|_h \otimes \text{SWAP}_{P,P'(i)} \otimes |1\rangle\langle 0|_{S(i+1)} \right) + |P_1\rangle\langle 0|_{M(P)} \otimes |0\rangle\langle 0|_h \otimes |1\rangle\langle 0|_{S(1)} + \left(\sum_{i=1}^{n-1} |P_i\rangle\langle P_{i+1}|_{M(P)} \otimes |1\rangle\langle 0|_h \otimes |1\rangle\langle 0|_{S(i)} \right) + |P_n\rangle\langle P_n|_{M(P)} \otimes |1\rangle\langle 1|_h \otimes |1\rangle\langle 0|_{S(n)} + \sum_{i=1}^{n} |P_i\rangle\langle P_i|_{M(P)} \otimes |0\rangle\langle 1|_{S(i)} + \sum_{i=n+1}^{\infty} |P_i\rangle\langle P_i|_{M(P)}, \tag{28}$$

where n denotes the number of programs to be concatenated, $|P_i\rangle$ denotes the ith program in the concatenation sequence (we are assuming that the programs have been enumerated such that the first n programs are the ones that we wish to concatenate), $M(P)$ denotes the program qubits portion of M, h denotes the halt qubit, P denotes the program counter register, $P'(i)$ denotes the ith ancillary program counter register, and $S(i)$ is the ith flag denoting that program i has been swapped. Note that some suitable finite unused subset of M can be used for $P'(i)$ and $S(i)$ since these are initialized to 0 and are only used once in the program concatenation operation. The P' and S' arrays of qubits are required to save intermediate states during program swaps to ensure unitarity.

In order to concatenate n given programs, then, we simply modify the overall $\mathcal{UQC}$ evolution operator to

$$U \stackrel{\text{def}}{=} \Pi^{(n)} U_{\text{CF}} U_{\text{UP}} U_{\text{RI}} U_{\text{UF}} U_{\text{UD}} U_{\text{EX}} U_{\text{XD}} U_{\text{IF}}. \tag{29}$$

U then has the net effect of running each program until it halts, swapping each completed program with the next program in the concatenation sequence, swapping the program counter register with 0, flipping the halt qubit (and hence starting execution of the swapped program), and leaving the final result on the tape when the last program, $|P_n\rangle$, halts. Even if the individual programs are known to halt, the concatenated program will not necessarily halt because, in general, the input data to the individual programs will change when run as part of a concatenated program. Hence, whether or not a concatenated program will halt is independent of whether or not its constituent programs halt.

The famous Turing halting problem is only relevant in the context of executing programs that can branch. Nonbranching finite programs, by construction, will always halt so the halting problem is a moot point in that case. This raises a question about Deutsch's UQTM. Deutsch did not explicitly consider branching in his original UQTM proposal and thus it is unclear whether or not his program concatenation scheme rested on the assumption that UQTM programs were nonbranching. If UQTM programs could involve branching, then without guaranteed halting of the concatenated programs, the validity of Deutsch's program concatenation scheme is problematic. Deutsch's description of his program concatenation scheme suggests that it was an "appending" scheme rather than a "swapping" scheme as we have defined.

There is still one problem with the program concatenation scheme. As currently defined, the halt qubit will be flipped several times during the course of executing a concatenation of programs (assuming that each constituent program halts, of course). Thus, it may appear that there is no way for an external observer (or classical computer) to distinguish between the intermediate and final states of the halt qubit. However, this does not pose a problem so long as the measurement of the halt qubit does not affect the result that we will ultimately measure on the memory tape in which case we simply measure the halt qubit, wait the time associated with program swap operations (i.e., $|P_{i+1}\rangle\langle P_i|$) to be completed, and measure the halt qubit again. If the halt qubit was in an intermediate set state, we will then find it cleared. If, on the other hand, the halt qubit was in its final set state, then we will find it still set and we can then measure the memory tape to find the result. Thus, periodic measurements of the halt qubit suffice to identify whether the concatenated program has halted. The question, then, is whether periodic measurements of the halt qubit affect the final measurement of the memory tape. Ozawa [10] has already proven that periodic measurement of the halt qubit does not spoil the result of the computation (i.e., the final measurement of the memory tape contents). That is, the probability of finding the memory tape in state M_i after N iterations of U with periodic measurements (monitoring) of the halt qubit and the probability of finding the memory tape in the state M_i after N iterations of U without periodic measurements of the halt qubit (i.e., one single measurement of M after N iterations of U) are identical. Thus, periodic measurements of the halt qubit do not spoil the intermediate computation as Myers argued.

Therefore, we see that concatenation of $\mathcal{UQC}$ programs works in the same way as concatenation of classical computer programs. While the halting question for the resultant program still remains just as it does for classical computers, a valid unitary $\mathcal{UQC}$ program concatenation scheme exists. The programs to be concatenated are sequentially executed without changing the state of internal registers except for the program counter. Not surprisingly, the concatenation scheme is analogous to the classical case.

VIII. CONCLUSION

The quantum computer we have defined is universal in the sense that, under the control of quantum programs, it can first emulate any classical Turing machine by being able to compute the NAND function and second can approximate any unitary operation to any desired accuracy by being able to apply the set of {H,CNOT,T} operations on a specified set of qubits. The machine also supports conditional branching and hence conditional execution, a feature that is not directly possible in the quantum gate array circuit framework. The defined halting scheme works in a way that prevents changes to the memory tape once the program has halted thus satisfying Ozawa's proof requirement and allowing for a valid program concatenation scheme. Because of its universality, $\mathcal{UQC}$ serves as a prototypical model for general-purpose programmable quantum computation and should find uses in the development and analysis of quantum algorithms and complexity. Work in progress using the $\mathcal{UQC}$ includes a demonstration of how to implement oracle based algorithms such as the Grover search algorithm.

[1] *Proceedings of the 35th Annual Symposium on the Foundations of Computer Science*, edited by S. Goldwasser (IEEE Computer Society, Los Alamitos, CA, 1994).
[2] T. Kieu, Int. J. Theor. Phys. **42**, 1461 (2003).
[3] D. Deutsch, Proc. R. Soc. Lond. **425**, 73 (1989).
[4] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information* (Cambridge University Press, Cambridge, England, 2000).
[5] L. K. Grover, Phys. Rev. Lett. **79**, 325 (1997).
[6] E. Farhi, J. Goldstone, S. Gutmann, and M. Sipser, e-print arXiv:quant-ph/0001106.
[7] D. Deutsch, Proc. R. Soc. Lond. **400**, 97 (1985).
[8] J. M. Myers, Phys. Rev. Lett. **78**, 1823 (1997).
[9] Y. Shi, Phys. Lett. A **293**, 277 (2002).
[10] M. Ozawa, Phys. Rev. Lett. **80**, 631 (1998).
[11] N. S. Yanofski and M. A. Mannucci, *Quantum Computing for Computer Scientists* (Cambridge University Press, Cambridge, England, 2008).
[12] E. Bernstein and U. Vazirani, SIAM J. Comput. **26**, 1411 (1997).
[13] S. Iriyama, T. Miyadera, and M. Ohya, Phys. Lett. A **372**, 5120 (2008).
[14] S. Waser and M. J. Flynn, *Introduction to Arithmetic for Digital Systems Designers* (Holt, Rinehart and Winston, New York, 1982).