12-8-2021

Computer Program Simulation of a Quantum Turing Machine with Circuit Model

Shixin Wu
Rose-Hulman Institute of Technology, WUS4@rose-hulman.edu

Computer Program Simulation of a Quantum Turing Machine with Circuit Model

Shixin Wu

December 8, 2021

Abstract

Molina and Watrous present a variation of the method to simulate a quantum Turing machine employed in Yao's 1995 publication "Quantum Circuit Complexity". We use a computer program to implement their method with linear algebra and an additional unitary operator defined to complete the details. Their method is verified to be correct on a quantum Turing machine.

1 Introduction

Alan Turing's paper in 1937 proposed the Turing machine [5]. It is seen as a mathematical abstraction of computation, and therefore used as a foundation to study quantum computing [1]. For example, Bernstein and Vazirani study quantum computation from a complexity theoretic viewpoint by going very in depth in the discussion of quantum Turing machines [4].

However, due to the cumbersome nature of the model and the complexity of the quantum theory, the Turing machine model is not the best tool to implement a quantum computer [1]. First described by Deutsch [3], the quantum circuit model has become a more useable tool compared to the quantum Turing machine after Yao's proof that they are equivalent by simulating the latter with the former [6]. While Yao's simulation method requires solving for a suitable circuit description with linear algebra, Molina and Watrous improved it by allowing us to directly reading an explicit description of the quantum circuits required to perform the simulation from a simple equation [1].

Due to the complexity of quantum superposition, one may find it difficult to trace the transitions of a quantum Turing machine by hand. A working simulation of a quantum Turing machine can serve as a tool for one to verify one's expected output given an input and a specific quantum Turing machine. In this paper, we implement the simulation method described by Molina and Watrous. Given this, our discussion does not go deep into the quantum circuits. Instead, we will use mathematical abstractions of quantum registers and quantum gates with column vectors and unitary matrices. As a result, certain operations which are easy to perform on the quantum circuits will require extra steps in our simulation. We have defined some operators to finish the details of Molina and Watrous' method to ease its implementation in programming.

We start off based on Bernstein and Vazirani's paper. In Section 2, we give the definition of a quantum Turing machine as well as its related concepts based on that of a deterministic Turing Machine. We will also introduce the Molina-Watrous algorithm proposed in their paper and the quantum Turing machine to be simulated in this section. Section 3 and Section 5 discuss our approach based on Molina and Watrous constructing a column vector representation of a configuration and the unitary matrix representation of the operator that maps one configuration to another. Both sections include subsections that describe the algorithms to compute a configuration vector and the operator matrix with pseudocode to avoid confusion about the syntax of any programming language. Section 5 will give our approach to the additional gate that completes the implementation of Molina and Watrous' method. We finish off in Section 6 by stating our future plans.

2 Preliminaries

2.1 Definition of a Quantum Turing Machine

In this section, we will give the definition of a quantum Turing machine, which is highly based upon the definition of a deterministic Turing machine.

2.1.1 Definition of a Deterministic Turing Machine

A *deterministic Turing machine* (DTM) is defined by a triplet (Σ, Q, δ) where Σ is a finite alphabet with an identified blank symbol $\#$; Q is a finite set of states with an identified start state q_1 and a subset of final states; and δ, the deterministic transition function, is defined by:

$$\delta : Q \times \Sigma \to \Sigma \times Q \times \{L, R\}$$

where $\{L, R\}$ indicates the set of tape head movements (L: left; R: right) on an infinitely long two-way tape [4]. The tape head of a DTM writes down a character in Σ before moving to the left or right on each step. Its initial position is on the first character of the input.

A *configuration* of a DTM is an element of $\Sigma^* Q \Sigma^*$, which is a complete description of the contents of the tape, the location of the tape head, and the state of the DTM. Suppose $c_D = uapbv$ is a configuration of a DTM M_D where $u, v \in \Sigma^*, a, b \in \Sigma, p \in Q$, then we say the *content* on the tape of M_D is $uabv$, the *location* of the tape head is on b and M_D is in the *state* p.

An *initial configuration* c_{init} of a DTM is the configuration before the DTM reads any character. A *final configuration* c_{fin} of a DTM is the configuration when the DTM is at a final state.

2.1.2 Definition of a Quantum Turing Machine

A *quantum Turing machine* (QTM) has exactly the same definition as a DTM except for that its transition function is defined as

$$\delta : Q \times \Sigma \to \widetilde{\mathbf{C}}^{\Sigma \times Q \times \{L,R\}}$$

where $\widetilde{\mathbf{C}}$ is the set of $\alpha \in \mathbf{C}$ such that there is a deterministic algorithm that computes the real and imaginary parts of α to with in 2^{-n} in time polynomial in n [4]. In other words, every element in $\widetilde{\mathbf{C}}$ will be calculated no slower than the transition functions themselves. If $p, q \in Q, \sigma, \tau \in \Sigma, d \in \{L, R\}$, we use the notation $\delta(p, \sigma)[\tau, q, d]$ to mean that the output of $\delta(p, \sigma)$ in $\widetilde{\mathbf{C}}^{\Sigma \times Q \times \{L,R\}}$ takes (τ, q, d) as the input and outputs an element in $\widetilde{\mathbf{C}}$.

A *configuration* of a QTM is an element of a vector space $\widetilde{\mathbf{C}}^{\Sigma^* Q \Sigma^*}$ which is the linear combinations of $\Sigma^* Q \Sigma^*$ over $\widetilde{\mathbf{C}}$ such that

$$\widetilde{\mathbf{C}}^{\Sigma^* Q \Sigma^*} = \left\{ \sum_i \chi_i |c_i\rangle \middle| \chi_i \in \widetilde{\mathbf{C}}, c_i \in \Sigma^* Q \Sigma^* \right\}$$

where for all $c \in \Sigma^* Q \Sigma^*$,

$$|c_i\rangle = \begin{cases} f(c) = 1 & (c = c_i) \\ f(c) = 0 & (\text{Otherwise}) \end{cases}$$

The definition of the initial and final configuration of a QTM is the same as a DTM.

2.1.3 Definition of a Quantum Turing Machine with a Looped Tape

A *quantum Turing machine with a looped tape* (QTMLT) has the same definition as a QTM with modifications to the tape and tape head movement. If a QTMLT halts after k steps, let the tape be, instead of an infinitely long two-way tape, one that only has $(2k + 1)$ slots in which a slot records a character. Let $\Lambda \geq 2k + 1$ and index the slots from 1 to Λ. Let the $(k + 1)$th slot be the starting slot of the tape head.

The modification to the tape head movement is as follows, if the tape head is on the ith slot:

$$\text{the tape head} \begin{cases} \text{goes to the } \Lambda \text{ slot} & (i = 1 \text{ and } \delta \text{ yields } L \text{ for the tape head movement}) \\ \text{goes to the first slot} & (i = \Lambda \text{ and } \delta \text{ yields } R \text{ for the tape head movement}) \\ \text{follows the result of } \delta & (\text{Otherwise}) \end{cases}$$

Equivalently, the tape head movement in terms of indexing can be calculated modulo Λ. Let θ, θ' be the index of the tape head and the index of the tape head after one step of transition.

$$\theta' = \begin{cases} (\theta - 2 \mod \Lambda) + 1 & (\delta \text{ yields } L \text{ for the tape head movement}) \\ (\theta \mod \Lambda) + 1 & (\delta \text{ yields } R \text{ for the tape head movement}) \end{cases}$$

A *configuration* of a QTMLT that takes k steps to halt is an element of a vector space $\widetilde{\mathbf{C}}^{\Sigma^a Q \Sigma^b}$ which is the linear combinations of $\Sigma^a Q \Sigma^b$ over $\widetilde{\mathbf{C}}$, for $a + b = \Lambda = 2k + 1$, such that

$$\widetilde{\mathbf{C}}^{\Sigma^a Q \Sigma^b} = \left\{ \sum_i \chi_i |c_i\rangle \middle| \chi_i \in \widetilde{\mathbf{C}}, c_i \in \Sigma^a Q \Sigma^b \right\}$$

where for all $c \in \Sigma^* Q \Sigma^*$,

$$|c_i\rangle = \begin{cases} f(c) = 1 & (c = c_i) \\ f(c) = 0 & (\text{Otherwise}) \end{cases}$$

The definition of the initial and final configuration of a QTMLT is the same as a QTM.

2.2 Linear Transformation of Superposed Configurations

The transition function of a QTM M defines a linear operator $U_M : S \to S$, called the *time evolution operator* of M [4]. Suppose $\alpha = \delta(p,\sigma)[\tau,q,L]$ and $\beta = \delta(p,\sigma)[\tau,q,R]$, and, for $c_i, c_{i_L}, c_{i_R} \in \Sigma^a Q \Sigma^b$, if $|c_i\rangle = |uap\sigma bv\rangle$, then $|c_{i_L}\rangle = |upa\tau bv\rangle$ and $|c_{i_R}\rangle = |ua\tau qbv\rangle$. For $s \in S$, define U_M:

$$\begin{aligned} U_M(s) &= U_M\left(\sum_i \chi_i |c_i\rangle\right) \\ &= \sum_i \chi_i U_M |c_i\rangle \\ &= \sum_{i,\tau,q} \chi_i(\alpha |c_{i_L}\rangle + \beta |c_{i_R}\rangle) \\ &= \sum_{i,\tau,q} \chi_i \left(\delta(p,\sigma)[\tau,q,L] |uap\sigma bv\rangle + \delta(p,\sigma)[\tau,q,R] |ua\tau qbv\rangle\right) \end{aligned}$$

We have already defined a QTMLT's configuration and its time evolutionary operator, and now we want to represent them with a column vector and a square matrix to allow numerical computation by a computer program.

2.3 The Molina-Watrous Algorithm

Molina and Watrous use a Boolean circuit to simulate an arbitrary QTMLT $\mathcal{L}$ that takes at least two steps for L to halt. Suppose $\mathcal{L}$ was given an input that takes k steps to halt. Notice that, according to Molina and Watrous, the simulation of a QTMLT will not be compromised for any $\Lambda \geq 2k+1 = 5$. The Boolean circuit is a concatenation of k identical subcircuits. Each subcircuit, referred to as Ω_g, for $g \in [1,k]$ performs one step of the transition (see Figure 1). An Ω_g is a circuit implementation of U_M, so it similarly inputs and outputs superposed configurations, but in their circuit implementations.

Molina and Watrous implement a superposed configuration in a quantum circuit as follows. For every slot within k steps from the starting slot of M, we use a register that stores an element in $Q' \times \Sigma$ where Q' is the "expanded" set of states. Besides the elements in Q, Q' also has a *zero state* $q^{(0)}$ and a set of *negative states* $\{q^{(-i)} | q^{(i)} \in Q\}$. The interpretations of the three kinds of states are, for the register of slot i,

$$q^{(i)}\text{means} \begin{cases} \text{the tape head is on slot } i\text{, and the head has already written a character} & (i<0) \\ \text{the tape head is not on slot } i & (i=0) \\ \text{the tape head is on slot } i\text{, and the head has not written a character} & (i>0) \end{cases}$$

This alternative configuration representation, different from representing a configuration as an element in $\widetilde{\mathbf{C}}^{\Sigma^* Q \Sigma^*}$, works because it stores the contents of the tape with the second register of every register and the state and tape head location with the first. However, Molina and Watrous did not specify the exact detail of how to store elements in Q' and Σ which will be discussed in section 3. Recall that we are only concerned with configurations in $\widetilde{\mathbf{C}}^{\Sigma^a Q \Sigma^b}$. Therefore, if a register takes ι wires to represent, then a configuration needs $\iota\Lambda$ wires, which is also the input and output of Ω. Let $R = \{r_i | i = 1 \ldots \Lambda\}$ be the set of every register r_i.

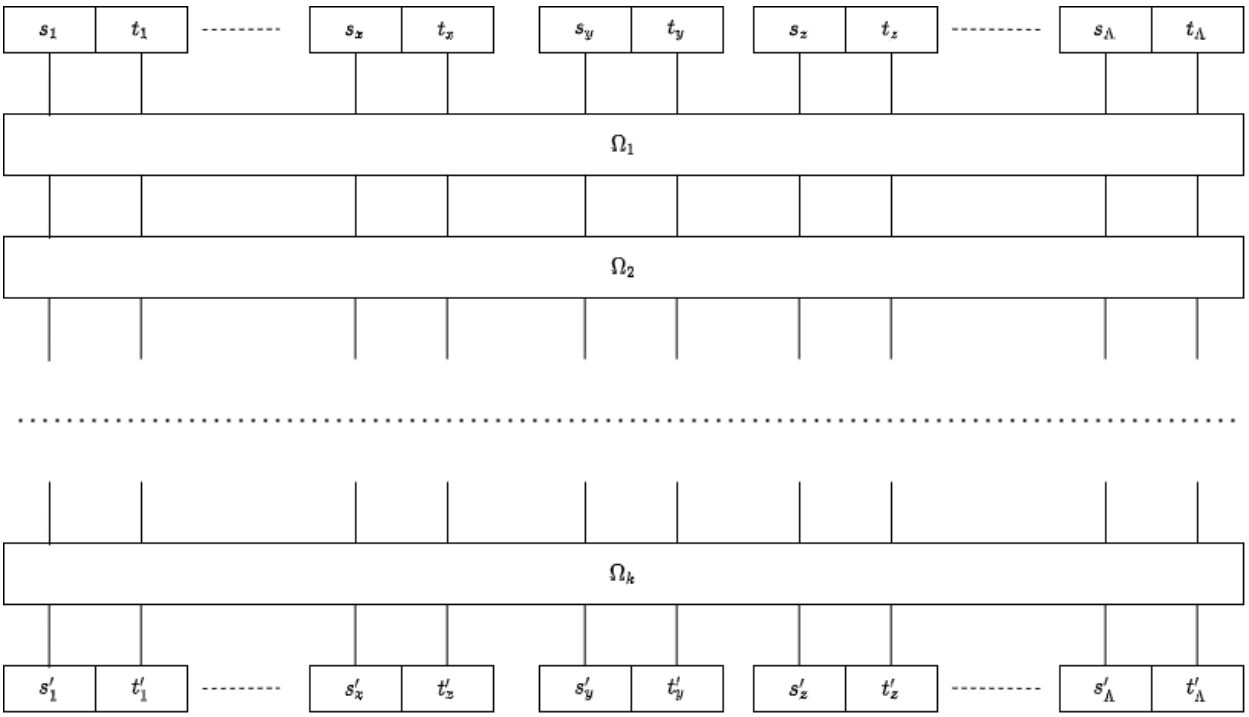

Figure 1: Boolean circuit to simulate a QTM
s_i and t_i are superposed states and characters in the initial configuration

Molina and Watrous construct Ω_g as follows. Let Ω_g be 3 sub-circuits concatenated, and name them $\omega_1, \omega_2, \omega_3$ (see Figure 2). An ω is made up of $\frac{\Lambda}{3}$ parallel circuits, and every circuit, named G, takes the input from registers $r_{(i-1 \mod \Lambda)+1}$, $r_{(i \mod \Lambda)+1}$ and $r_{(i+1 \mod \Lambda)+1}$. Finally, $\omega_{\Lambda-1}$ consists of Λ parallel F gates applied to the first register of every register.

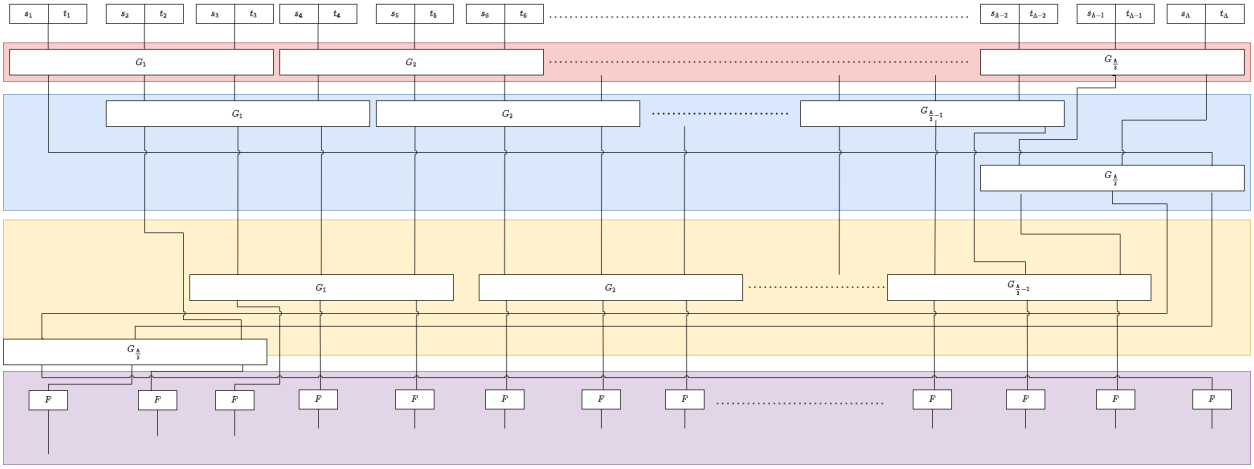

Figure 2: An arbitrary Ω_i

2.4 A QTM to Simulate

Given the limited available computing resources, we decided to simulate a rather trivial QTM $\mathcal{M} = (Q, \Sigma, \delta)$ where $Q = \{q_1, q_2\}$, with a start state q_1 and a finish state q_2, $\Sigma = \{0, 1, \#\}$, and δ which is defined as

follows:

$$\delta(q_1, 0) = f \qquad \delta(q_1, 1) = g \qquad \delta(q_1, \#) = h$$

where $f, g, h \in \widetilde{\mathbf{C}}^{\Sigma \times Q \times \{L,R\}}$ are defined as follows, for $q \in Q, \tau \in \Sigma, e \in \{L, R\}$:

$$f(q,\tau,e) = \begin{cases} \frac{1}{\sqrt{2}} & (q = q_1, \tau = 0, e = R) \\ \frac{1}{\sqrt{2}} & (q = q_1, \tau = 1, e = R) \\ 0 & (\text{Otherwise}) \end{cases}$$

$$g(q,\tau,e) = \begin{cases} \frac{1}{\sqrt{2}} & (q = q_1, \tau = 0, e = R) \\ -\frac{1}{\sqrt{2}} & (q = q_1, \tau = 1, e = R) \\ 0 & (\text{Otherwise}) \end{cases}$$

$$h(q,\tau,e) = \begin{cases} 1 & (q = q_2, \tau = \#, e = R) \\ 0 & (\text{Otherwise}) \end{cases}$$

We can also show the definition with a traditional graph representation of a Turing machine. Recall that the graph representation includes a set of vertices Q and directed edges $E = Q \times Q$. A vertex represents a state, and an edge $e = (p, q)$ for $p, q \in Q$ shows that the Turing machine leaves state p and goes into state q. To fully show the transition of our QTM, we also label the edges. If e is labeled by $\sigma \to \tilde{c}(\tau, d)$ for $\sigma, \tau \in \Sigma$, $d \in \{L, R\}$ and $\tilde{c} = \delta(p, \sigma)[\tau, q, d]$, we say if the QTM is at state p and reads σ, the amplitude that it goes to state q, writes σ and moves the tape head to direction d is $\tilde{c}$. Notice that $\tilde{c}(\tau, d)$ means that the amplitude of the situation is $\frac{1}{\sqrt{2}}$ for the QTM to be in state q while outputting character τ and moving the tape head to the direction d. The graph representation of $\mathcal{M}$ is in Figure 3.

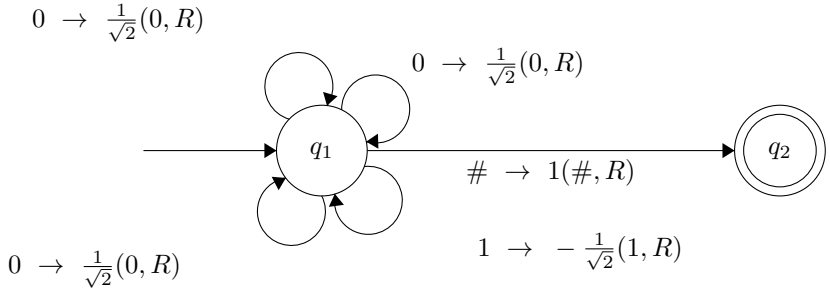

Figure 3: The QTM $\mathcal{M}$

3 Representation of Superposed States and Characters

In order for states and characters to be superposed, we use column vectors to represent states and characters. Let $|q\rangle$ and $|\sigma\rangle$ be the column vector representations of $q \in Q'$ and $\sigma \in \Sigma$. Then, in order to write every superposed state and character in a unique way, let $B_{\mathcal{S}} = \{|q\rangle\}_{q \in Q'}$ and $B_{\mathcal{T}} = \{|\sigma\rangle\}_{\sigma \in \Sigma}$ be orthonormal sets such that every $|q\rangle \in B_{\mathcal{S}}$ is equal to a different computational basis vector in $\widetilde{\mathbf{C}}^{|Q'|}$ and that every $|\sigma\rangle \in B_{\mathcal{T}}$ is equal to a different computational basis vector in $\widetilde{\mathbf{C}}^{|\Sigma|}$. Finally, let $\mathcal{S} = L(B_{\mathcal{S}})$ and $\mathcal{T} = L(B_{\mathcal{T}})$ be the Hilbert spaces in $\widetilde{\mathbf{C}}^{|Q'|}$ and $\widetilde{\mathbf{C}}^{|\Sigma|}$ of the vector representations of the superposed states and characters.

There are two states in the state set of Section 2.4, so our state set Q' is $\{q^{(-2)}, q^{(-1)}, q^{(0)}, q^{(1)}, q^{(2)}\}$. $B_{\mathcal{S}}$ is obtained by assigning every $|q\rangle$ to a computational basis vector in $\widetilde{\mathbf{C}}^5$:

$$|q^{(-2)}\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} |q^{(-1)}\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} |q^{(0)}\rangle = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} |q^{(1)}\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} |q^{(2)}\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Similarly, our character set Σ is $\{0, 1, \#\}$, and our character column vector basis B_T is correspondingly $\{|0\rangle, |1\rangle, |\#\rangle\}$ such that:

$$|0\rangle = \begin{pmatrix}0\\0\\1\end{pmatrix} |1\rangle = \begin{pmatrix}0\\1\\0\end{pmatrix} |\#\rangle = \begin{pmatrix}1\\0\\0\end{pmatrix}$$

4 Representation of Superposed Configurations

Now we have finished the representation of a register. Recall that Molina and Watrous use Λ registers to represent a superposed configuration. Thus, index $\mathcal{S}$ and $\mathcal{T}$ to be $\mathcal{S}_i$ and $\mathcal{T}_i$ for $i \in [1, \Lambda]$, and let $\mathcal{K}_\Lambda$ be the combined Hilbert space of $(\mathcal{S}_1, \mathcal{T}_1), \ldots, (\mathcal{S}_\Lambda, \mathcal{T}_\Lambda)$ such that:

$$\mathcal{K}_\Lambda = \bigotimes_{i=1}^{\Lambda} (\mathcal{S}_i \otimes \mathcal{T}_i)$$

Let $B_{\mathcal{K}_\Lambda}$ be the basis set of $\mathcal{K}_\Lambda$ such that, for $q_i \in Q', \sigma_i \in \Sigma$ and $b_\Lambda \in B_{\mathcal{K}_\Lambda}$

$$b_\Lambda = \bigotimes_{i=1}^{\Lambda} |q_i, \sigma_i\rangle$$

Let the input to M from Section 2.4 be 0. Since the input 0 only requires 2 steps for $\mathcal{M}$ to halt, without loss of generality, let the tape of M have length 6, a multiple of 3, as recommended by Molina and Watrous. Therefore, c_{init} and c_{fin} of M on input 0 are in $\widetilde{\mathbf{C}}^{\Sigma^a Q \Sigma^b}$ for $a + b = 6$ and defined to be:

$$c_{init} = |\#\#q_1 0\#\#\rangle \qquad c_{fin} = \frac{1}{\sqrt{2}} |\#\#\#q_2 0\#\#\rangle + \frac{1}{\sqrt{2}} |\#\#\#q_2 1\#\#\rangle$$

Let $k_{\Lambda_{init}}$ and $k_{\Lambda_{fin}}$ be elements in K_Λ such that they record the same configuration respectively as c_{init} and c_{fin}:

$$\begin{aligned} k_{\Lambda_{init}} &= |q_0, \#\rangle \otimes |q_0, \#\rangle \otimes |q_1, 0\rangle \otimes |q_0, \#\rangle \otimes |q_0, \#\rangle \otimes |q_0, \#\rangle \\ k_{\Lambda_{fin}} &= \frac{1}{\sqrt{2}} |q_0, \#\rangle \otimes |q_2, \#\rangle \otimes |q_0, 0\rangle \otimes |q_2, \#\rangle \otimes |q_0, \#\rangle \otimes |q_0, \#\rangle \\ &\quad + \frac{1}{\sqrt{2}} |q_0, \#\rangle \otimes |q_2, \#\rangle \otimes |q_0, 1\rangle \otimes |q_2, \#\rangle \otimes |q_0, \#\rangle \otimes |q_0, \#\rangle \end{aligned}$$

Notice that, due to entanglement, the vector representation of a configuration should only be written as a linear combination of elements in $B_{\mathcal{K}_\Lambda}$. When preparing the initial and final configuration in the simulation, it is convenient to use a 2D array of dimension 2Λ to store an element in $\mathcal{K}_\Lambda$, for $\alpha_j \in \widetilde{\mathbf{C}}$, $q_{ij} \in Q'$ and $\sigma_{ij} \in \Sigma$

$$A(k_\Lambda) = \left[\begin{bmatrix} \alpha_1, \\ |q_{1,1}\rangle, \\ |\sigma_{1,1}\rangle, \\ \vdots \\ |q_{\Lambda,1}\rangle, \\ |\sigma_{\Lambda,1}\rangle, \end{bmatrix}, \ldots, \begin{bmatrix} \alpha_\Lambda, \\ |q_{1,\Lambda}\rangle, \\ |\sigma_{1,\Lambda}\rangle \\ \vdots \\ |q_{\Lambda,\Lambda}\rangle, \\ |\sigma_{\Lambda,\Lambda}\rangle \end{bmatrix} \right]$$

In our case, for example, the array $A(k_{\Lambda_{init}})$ that stores $k_{\Lambda_{init}}$ can be written as:

$$A(k_{\Lambda_{init}}) = [[1, |q_0\rangle, |\#\rangle, |q_0\rangle, |\#\rangle, |q_1\rangle, |0\rangle, |q_0\rangle, |\#\rangle, |q_0\rangle, |\#\rangle, |q_0\rangle, |\#\rangle]]$$

Now we define an algorithm in Algorithm 1 that computes the column vector representation of a configuration given its representation as an array.

Algorithm 1 Compute the column vector of a given configuration in an array

```
function getConfigVector(array A)
    initialize subArrayResult to 1
    initialize result to 0
    for every k ∈ A do
        for every a ∈ k do                          ▷ by the order of the array
            subArrayResult ← subArrayResult ⊗ a
        end for
        result ← result+subArrayResult
    end for
    return result
end function
```

5 Time Evolutionary Operator as Matrix

As we have discussed in section 2.3, Molina and Watrous represent $U_{\mathcal{K}}$ with G and F. We call their unitary matrix representations in the space of $(\mathcal{S}\otimes\mathcal{T})^{\otimes 3}$ and $\mathcal{S}$ respectively by the same name given their equivalence. In this section, we will first define F and G, expand them into global gates, define ROT_{ij} and finally construct $U_{\mathcal{K}}$ with F, G, ROT_{ij}.

5.1 The Definition of F

If $q_l \in Q'$, then

$$F\left|q^l\right\rangle = \left|q^{-l}\right\rangle$$

The exact unitary matrix representation of F of course varies depending on the column vector representation of $|q\rangle$ for some $q \in Q'$, but in our suggested way of representation, F behaves very similarly to the Pauli-X gate $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$. They therefore share the similar shape such that, for F of dimension $|Q'|$

$$F = \begin{pmatrix} 0 & 0 & \dots & 0 & 1 \\ 0 & 0 & \dots & 1 & 0 \\ \vdots & \vdots & \iddots & \vdots & \vdots \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

In the case of $\mathcal{M}$, since $|Q'| = 5$:

$$F = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

5.2 The Definition of G

We define G based on its transformations of basis vectors in the space $(\mathcal{S}\otimes\mathcal{T})^{\otimes 3}$. We know $|q_x, \sigma_x\rangle\,|q_y, \sigma_y\rangle\,|q_z, \sigma_z\rangle$ is a basis vector of $(\mathcal{S}\otimes\mathcal{T})^{\otimes 3}$ because $|q_x\rangle, |q_y\rangle, |q_z\rangle$ and $|\sigma_x\rangle, |\sigma_y\rangle, |\sigma_z\rangle$ are the basis vectors of $\mathcal{S}$ and $\mathcal{T}$ respectively.

Separate Q' into three mutually exclusive subsets such that $Q = P \cup N \cup \{q_0\}$ where P and N are the sets of positive and negative states. The transformations caused by applying G differ according to the subsets that each q_x, q_y, q_z is a member of. In summary, there are 27 possibilities for (q_x, q_y, q_z), and we summarize these possibilities into 4 cases such that in each case in each case G transforms a basis vector in $(\mathcal{S}\otimes\mathcal{T})^{\otimes 3}$ differently. The 4 cases are:

1. (a) When the tape head is on the yth slot and the transition of the machine has not been conducted: $q_x = q_z = q_0$, $q_y \in P$

(b) When the tape head is on the xth slot and the transition of the machine has been conducted: $q_y = q_z = q_0$, $q_x \in N$

(c) When the tape head is on the zth slot and the transition of the machine has been conducted: $q_x = q_y = q_0$, $q_z \in N$

(d) When the tape head is on none of the x, y, zth slot: $q_x = q_y = q_z = q_0$

(e) When the tape head is on more than one of the slots: at most one of $\{q_x, q_y, q_z\}$ is q_0

2. When the tape head is on the yth slot and the transition of the machine has been conducted: $q_x = q_z = q_0$, $q_y \in N$

3. When the tape head is on the xth slot and the transition of the machine has not been conducted: $q_y = q_z = q_0$, $q_x \in P$

4. When the tape head is on the zth slot and the transition of the machine has not been conducted: $q_x = q_y = q_0$, $q_z \in P$

For Case 1, G acts trivially:

$$G\left|q_x, \sigma_x\right\rangle \left|q_y, \sigma_y\right\rangle \left|q_z, \sigma_z\right\rangle = \left|q_x, \sigma_x\right\rangle \left|q_y, \sigma_y\right\rangle \left|q_z, \sigma_z\right\rangle$$

For Case 2, the result of G is the summation over all the vectors of $\overline{\alpha}\left|q_x, \sigma_x\right\rangle \left|q_0, \sigma_y\right\rangle \left|q_0, \sigma_z\right\rangle$ such that $\alpha = \delta(q_x, \sigma_x)[q_y, \sigma_y, R]$ and all the vectors of $\overline{\beta}\left|q_0, \sigma_x\right\rangle \left|q_0, \sigma_y\right\rangle \left|q_z, \sigma_z\right\rangle$ such that $\beta = \delta(q_z, \sigma_z)[q_y, \sigma_y, L]$

$$\begin{aligned}
&G\left|q_0, \sigma_x\right\rangle \left|q_y, \sigma_y\right\rangle \left|q_0, \sigma_z\right\rangle \\
=& \sum_{q_x \in P, \sigma_x \in \Sigma} \overline{\delta(q_x, \sigma_x)[q_{-y}, \sigma_x, R]} \left|q_x, \sigma_x\right\rangle \left|q_0, \sigma_y\right\rangle \left|q_0, \sigma_z\right\rangle \\
+& \sum_{q_z \in P, \sigma_z \in \Sigma} \overline{\delta(q_z, \sigma_z)[q_{-y}, \sigma_z, L]} \left|q_0, \sigma_x\right\rangle \left|q_0, \sigma_y\right\rangle \left|q_z, \sigma_z\right\rangle
\end{aligned}$$

For Case 3, the result of G is the summation over all the vectors of $\alpha\left|q_0, \sigma_x\right\rangle \left|q_{-y}, \sigma_y\right\rangle \left|q_0, \sigma_z\right\rangle$ such that $\delta(q_x, \sigma_x)[q_y, \sigma_y, R] = \alpha$ and all the vectors of $\beta\overline{\gamma}\left|q, b\right\rangle \left|q_{-y}, \sigma_y\right\rangle \left|q_0, \sigma_z\right\rangle$ for $q \in Q'$ and $b \in \Sigma$ such that, for $r \in Q'$ and $c \in \Sigma$, $\beta = \delta(q_x, \sigma_x)[r, c, L]$ and $\gamma = \delta(q, b)[r, c, L]$.

$$\begin{aligned}
&G\left|q_x, \sigma_x\right\rangle \left|q_0, \sigma_y\right\rangle \left|q_0, \sigma_z\right\rangle \\
=& \sum_{q_y \in P, \sigma_x \in \Sigma} \delta(q_x, \sigma_x)[q_y, \sigma_x, R] \left|q_0, \sigma_x\right\rangle \left|q_{-y}, \sigma_y\right\rangle \left|q_0, \sigma_z\right\rangle \\
+& \sum_{\substack{q \in P, b \in \Sigma \\ r \in P, c \in \Sigma}} \delta(q_x, \sigma_x)[r, c, L]\overline{\delta(q, b)[r, c, L]} \left|q, b\right\rangle \left|q_0, \sigma_y\right\rangle \left|q_0, \sigma_z\right\rangle
\end{aligned}$$

For Case 4, the result of G is the summation over all the vectors of $\alpha\left|q_0, \sigma_x\right\rangle \left|q_{-y}, \sigma_y\right\rangle \left|q_0, \sigma_z\right\rangle$ such that $\delta(q_z, \sigma_z)[q_y, \sigma_y, L] = \alpha$ and all the vectors of $\beta\overline{\gamma}\left|q_0, \sigma_z\right\rangle \left|q_{-y}, \sigma_y\right\rangle \left|q, b\right\rangle$ for $q \in Q'$ and $b \in \Sigma$ such that, for $r \in Q'$ and $c \in \Sigma$, $\beta = \delta(q_z, \sigma_z)[r, c, R]$ and $\gamma = \delta(q, b)[r, c, R]$.

$$\begin{aligned}
&G\left|q_0, \sigma_x\right\rangle \left|q_0, \sigma_y\right\rangle \left|q_z, \sigma_z\right\rangle \\
=& \sum_{q_y \in P, \sigma_x \in \Sigma} \delta(q_z, \sigma_z)[q_y, \sigma_z, L] \left|q_0, \sigma_x\right\rangle \left|q_{-y}, \sigma_y\right\rangle \left|q_0, \sigma_z\right\rangle \\
+& \sum_{\substack{q \in P, b \in \Sigma \\ r \in P, c \in \Sigma}} \delta(q_z, \sigma_z)[r, c, R]\overline{\delta(q, b)[r, c, R]} \left|q_0, \sigma_x\right\rangle \left|q_0, \sigma_y\right\rangle \left|q, b\right\rangle
\end{aligned}$$

5.2.1 Implementation of G

As the linear transformation of basis vectors by G is case by case, our computation of the linear transformation is done in the same way. However, every case shares a considerable number of similarities with one another, so we use Case 2 as an example to illustrate the algorithm of the code implementation.

To enhance the implementation of Case 2, we define a subset Δ of $Q' \times \Sigma \times \Sigma \times Q' \times \{L, R\} \times (\widetilde{\mathbf{C}} \setminus \{0\})$ such that, for $(p, \sigma, \tau, q, d, \tilde{c}) \in \Delta$, $\delta(p, \sigma)[\sigma_z, q_{-y}, L] = \tilde{c}$. $A(\Delta)$ is the array representation of Δ where every sextuple of Δ is stored as an array of length six where every element in the array has the same location as in the sextuple. In the pseudocode, we use f to represent an element in $A(\Delta)$ and indices from 1 to 6 based on f to represent elements in $(p, \sigma, \tau, q, d, \tilde{c})$ according to their positions in the sextuple.

In Case 2, the result of G's linear transformation involves two summations which are calculated individually in the functions *searchLeft* and *searchRight*. The $left$ and *right* come from the tape head movements of $\delta(q_z, \sigma_z)[q_{-y}, \sigma_z, L]$ and $\delta(q_x, \sigma_x)[q_{-y}, \sigma_x, R]$ in two equations of summation.

In the function *searchLeft*, given the input $\sigma_x, \sigma_y, \sigma_z \in \Sigma$ and $q_y \in Q'$, after storing $|q_0\rangle \otimes |\sigma_x\rangle$ and $|q_0\rangle \otimes |\sigma_y\rangle$ to reg_1 and reg_2, we want to search in Δ for elements that $\tau = \sigma_z, q = q_{-y}$ and $d = L$. For any $e \in \Delta$ that satisfies this condition, we store the tensor product of p and σ in e in reg_3, and increase the incrementor *sum* by $\overline{\tilde{c}} \cdot reg_1 \otimes reg_2 \otimes reg_3$. We finally return *sum* after repeating this process for every e that satisfies the condition. The process for *searchRight* is identical to that of *searchLeft* except for that we search in Δ for elements that $\tau = \sigma_x, q = q_{-y}$ and $d = R$.

The function *searchResult* adds the results returned by *searchLeft* and *searchRight*, which in the equation for Case 2 is the addition of the two summations. Recall Case 2 is applicable to every basis vector in $(\mathcal{S} \otimes \mathcal{T})^{\otimes 3}$ that $q_x = q_z = q_0$, $q_y \in N$. Therefore, we use 4 nested for loops in *getCaseResult* to simulate this applicability, three of which are for characters $\sigma_x, \sigma_y, \sigma_z \in \Sigma$ and one is for q_y. For every iteration, we perform *searchResult* whose result is added to an array called *caseResult*. *caseResult* is a global array of column vectors that stores the *searchResult*s for every case. Finally, combining the column vectors in *caseResult* obtains G.

Algorithm 2 Compute the array of column vectors produced in Case 6

```
1:  function searchLeft(char σ_x, char σ_y, char σ_z, state q_y)
    ▷ search for transition rules that write σ_z, go into state q_{-y} and move the tape head to the left
2:      reg_1 ← |q_0⟩ ⊗ |σ_x⟩                          ▷ a reg is the vector of a state-character pair
3:      reg_2 ← |q_0⟩ ⊗ |σ_y⟩
4:      initialize sum
5:      for every f ∈ A(Δ) do
6:          if f[3] = |q_{-y}⟩ and f[4] = |σ_z⟩ and f[5] = −1 then      ▷ The index of f starts from 1
7:              reg_3 ← f[1] ⊗ f[2]
8:              sum ← sum + f[6] · reg_1 ⊗ reg_2 ⊗ reg_3
9:          end if
10:     end for
11:     return sum
12: end function
13:
14: function searchRight(char σ_x, char σ_y, char σ_z, state q_y)
    ▷ search for transition rules that write σ_x, go into state q_{-y} and move the tape head to the right
15:     reg_2 ← |q_0⟩ ⊗ |σ_y⟩
16:     reg_3 ← |q_0⟩ ⊗ |σ_z⟩
17:     initialize sum
18:     for every f ∈ A(Δ) do
19:         if f[3] = |q_{-y}⟩ and f[4] = |σ_x⟩ and f[5] = 1 then
20:             reg_1 ← f[1] ⊗ f[2]
21:             sum ← sum + f[6] · reg_1 ⊗ reg_2 ⊗ reg_3
22:         end if
23:     end for
24:     return sum
25: end function
```

```
26:
27: function searchResult(char σ_x, char σ_y, char σ_z, state q_y)
28:     return searchLeft(σ_x, σ_y, σ_z, q_y) + searchRight(σ_x, σ_y, σ_z, q_y)
29: end function
30: function getCaseResult
31:     initialize caseResult
32:     for every σ_x, σ_y, σ_z, every q_y ∈ Q' do        ▷ This is an abbreviation for 4 nested for loops
33:         Append searchResult(σ_x, σ_y, σ_z, q_y) to caseResult
34:     end for
35: end function
```

5.3 Local Gates to Global Gates

Recall that the circuit simulation uses $\iota\Lambda$ wires mentioned in Section 2.3. If a gate that operates on ϕ consecutive wires and $\phi < \iota\Lambda$, then we say this gate is a *local gate*; if $\phi = \iota\Lambda$, then we say this gate is a *global gate*. G and F in Section 2.3 are examples of local gates. If applying a local gate to some consecutive wires while applying no gates on other wires and applying a global gate to all wires give the same result on every wire on the set of $\iota\Lambda$ wires, we say the global gate is an *equivalent global gate* to the local gate. Because our simulation uses vectors for configuration and a unitary matrix for U_M and a unitary matrix cannot multiply a vector that has different dimensions, we must construct an equivalent global gate for every local gate we will use.

To mathematically construct an equivalent global gate in our simulation, if O is the matrix representation of a local gate that operates on the wires from the $(\alpha+1)$ to $\beta-1$th wire for $\alpha + \dim O + \beta = \iota\Lambda$, its equivalent global gate's matrix representation $O_{\alpha+1}$ is

$$O_{\alpha+1} = I_\alpha \otimes O \otimes I_\beta \tag{1}$$

5.4 Definition of ROT_{ij}

Recall that the circuit simulates a QTM on a looped tape, but it is impossible to loop around wires, vectors, or matrices. This presents a problem when applying a local gate G to, for example, the wires from the last, first, and second registers, because a local gate can only apply to consecutive wires. The solution is to apply a unitary operator that rotates the values in the wires as if the wires are looped around and in a consecutive order. Then, we may apply G to the consecutive wires and reverse the rotation operation.

For $i, j, \in [1, \Lambda]$ and $i < j$, define two unitary operators ROT_{ij} and ROT_{ji} over $\mathcal{K}_\Lambda$. Suppose $|k_\Lambda\rangle, |k_\Lambda\rangle', |k_\Lambda\rangle'' \in \mathcal{K}_\Lambda$ are the vector representation of the values in the circuit, such that, if $|k_\Lambda\rangle$ represents the original value in the circuit, then $|k_\Lambda\rangle'$ and $|k_\Lambda\rangle''$ respectively represent the values in the circuit as if the wires of registers of index l for $l \in [i, j]$ up and down by ι (the number of wires from a register) while not changing the values in other wires. Then, $ROT_{ij}|k_\Lambda\rangle = |k_\Lambda\rangle'$ and $ROT_{ij}|k_\Lambda\rangle = |k_\Lambda\rangle''$.

Notice that in Figure 2 that we mainly want to rotate the values of all wires up and down by ι. Therefore, the operators used for the simulation are $ROT_{1,\Lambda}$ and $ROT_{\Lambda,1}$.

Suppose $|\psi\rangle \in (\mathcal{S}_x \otimes \mathcal{T}_x) \otimes (\mathcal{S}_{x+1} \otimes \mathcal{T}_{x+1})$ and $|\psi'\rangle \in (\mathcal{S}_{x+1} \otimes \mathcal{T}_{x+1}) \otimes (\mathcal{S}_x \otimes \mathcal{T}_x)$. Then define an operator $SWAP$ over $(\mathcal{S} \otimes \mathcal{T})^{(\otimes 2)}$ such that

$$SWAP = \sum_{\substack{q,q' \in Q' \\ \sigma,\sigma' \in \Sigma}} |q,\sigma\rangle |q',\sigma'\rangle \langle q',\sigma'| \langle q,\sigma| \tag{2}$$

As a result, $SWAP(|\psi\rangle) = |\psi'\rangle$. Then, let $SWAP_x$ be the matrix representation of the equivalent global gate of $SWAP$. Therefore, by Equation 1,

$$SWAP_x = I_{\iota(x-1)} \otimes SWAP \otimes I_{\iota(\Lambda-x-1)} \tag{3}$$

We construct ROT_{ij} and ROT_{ji} by multiplying $SWAP_x$ operators such that

$$ROT_{ij} = SWAP_i \cdot SWAP_{i+1} \cdot \cdots \cdot SWAP_{j-1} \tag{4}$$

$$ROT_{ji} = SWAP_{j-1} \cdot SWAP_{j-2} \cdot \cdots \cdot SWAP_i \tag{5}$$

Algorithm 3 Generate ROT_{ij} and ROT_{ji}

function getSWAP
    initialize $regArray$
▷ Put the vector representations of all possible state-vector pairs into the array $regArray$
    for every $q \in Q'$, every $\sigma \in \Sigma$ **do**
        Append $|q\rangle \otimes |\sigma\rangle$ to $regArray$
    end for
▷ Take the summation as in Equation 2
    $d \leftarrow |Q'| \times |\Sigma|$
    initialize $sum = [0]_{d \times d}$
    for every $|q, \sigma\rangle, |q', \sigma'\rangle \in regSums$ **do**
        $sum \leftarrow sum + |q, \sigma\rangle \langle q', \sigma'|$
    end for
    return sum
end function

function getSwap$_{\mathbf{x}}$(integer x, integer Λ)
    $d \leftarrow |Q'| \times |\Sigma|$
    $SWAP \leftarrow getSWAP()$
▷ Based on Equation 3
    return $I_{(d(x-1))} \otimes SWAP \otimes I_{d(\Lambda - x - 1)}$
end function

function getROT$_{\mathbf{ij}}$(integer i, integer j, integer Λ)
    $l \leftarrow i$
    initialize $result = I_{\Lambda(|Q'|+|\Sigma|)}$
▷ Take the matrix products as in Equation 4
    for $l \leq j$ **do**
        $result \leftarrow result \cdot getSwap_x(l, \Lambda)$
        $l \leftarrow l + 1$
    end for
    return $result$
end function

function getROT$_{\mathbf{ji}}$(integer i, integer j, integer Λ)
    $l \leftarrow i$
    initialize $result = I_{\Lambda(|Q'|+|\Sigma|)}$
▷ Take the matrix products as in Equation 5
    for $l \geq i$ **do**
        $result \leftarrow result \cdot getSwap_x(l, \Lambda)$
        $l \leftarrow l - 1$
    end for
    return $result$
end function

5.5 Global Gates Based on G and F

For $i \in [1, \Lambda]$, let $j = (i \mod \Lambda) + 1$ and $k = (i + 1 \mod \Lambda) + 1$. Define G_{ijk}, the global unitary operator based on G to be:

$$G_{ijk} = \begin{cases} G_i & (j = i + 1 \wedge k = i + 2) \\ ROT_{1,\Lambda} \cdot G_i \cdot ROT_{\Lambda,1} & (j = i + 1 \wedge k \neq i + 2) \\ ROT_{\Lambda,1} \cdot G_i \cdot ROT_{1,\Lambda} & (\text{Otherwise}) \end{cases} \tag{6}$$

We then construct the matrix representation F_s of the circuits in the purple section in Figure 2.

$$F_s = \bigotimes_{i=1}^{\Lambda} F \otimes I_{|\Sigma|} \tag{7}$$

5.6 Formalization of the Molina-Watrous Algorithm

Now we have finally defined all matrices for the global operators we will use to construct $U_{\mathcal{K}}$. We construct $U_{\mathcal{K}}$ according to Figure 2. Observe that in Figure 2, every 3 registers are applied with a G_{ijk}. Therefore,

$$U_{\mathcal{K}} = F_s \cdot G_{\Lambda,jk} \cdot G_{\Lambda-1,jk} \cdot \cdots \cdot G_{1,jk} \tag{8}$$

In the pseudocode, we use $getF_s$ and $getG_{ijk}$ to obtain F_s and G_{ijk} respectively, according to Equation 6 and Equation 7. Finally, we use $getU_{\mathcal{K}}$ to obtain $U_{\mathcal{K}}$ according to Equation 8.

Algorithm 4 Generate $U_{\mathcal{K}}$

function getF$_s$
▷ Calculate F_s as in Equation 7
  initialize *result* to $[1]_{1\times 1}$
  for $i \in [1, \Lambda]$ **do**
    $result \leftarrow result \otimes (F \otimes I_{|\Sigma|})$
  end for
  return *result*
end function

function getG$_{ijk}$ (i)
▷ Calculate G_{ijk} as in Equation 6. Given i, first calculate j and k.
  $j \leftarrow (i \mod \Lambda) + 1$
  $k \leftarrow (i+1 \mod \Lambda) + 2$
  $\iota \leftarrow |Q'| + |\Sigma|$
  $G_i \leftarrow I_{\iota(i-1)} \otimes G \otimes I_{\iota(\Lambda-i-1)}$
  if $j = i+1 \wedge k = i+2$ **then**
    return G_i
  end if
  if $j = i+1 \wedge k \neq i+2$ **then**
    return $getROT_{ij}(1, \Lambda) \cdot G_i \cdot getROT_{ji}(\Lambda, 1)$
  end if
  return $getROT_{ji}(\Lambda, 1) \cdot G_i \cdot getROT_{ij}(1, \Lambda)$
end function

function getU$_{\mathcal{K}}$
▷ Calculate $U_{\mathcal{K}}$ as in Equation 8
  $U_{\mathcal{K}} \leftarrow getF_s()$
  $i \leftarrow \Lambda$
  for every $i \geq 1$ **do**
    $U_{\mathcal{K}} \leftarrow U_{\mathcal{K}} \cdot getG_{ijk}(i)$
    $\Lambda = \Lambda - 1$
  end for
  return $U_{\mathcal{K}}$
end function

6 Conclusion

Recall $k_{\Lambda_{init}}$ and $k_{\Lambda_{fin}}$ from Section 4, and that applying one $U_{\mathcal{K}}$ to an element in $\mathcal{K}_\Lambda$ is equivalent to one step of transition of M to the configuration represented by the element in $\mathcal{K}_\Lambda$. Since in the input in our

simulation takes two steps for M to halt, $U_{\mathcal{K}}$ must be applied twice. By confirming that $(U_{\mathcal{K}})^2 k_{init} = k_{fin}$, we have confirmed that the method of simulating a QTM proposed by Molina and Watrous works on $\mathcal{M}$.

Although complicated, this is only the start of QTM simulation. An immediate improvement to this paper is to simulate a less trivial QTM to determine whether there are any uncovered edge cases. We would also like to investigate the simulation of variations of QTMs, such as those with multiple tapes and/or tape heads, with tape heads allowed to remain stationary in one transition, and so on. Additionally, Bernstein and Vazirani have proposed many classifications of QTMs, such as unidirectional QTM, QTM of normal form, well-formed QTM, well-behaved QTM, etc. [4]. Through variations of our current work, we should be able to simulate all QTMs above, which would be very helpful in gaining more insight into the significance of these variations.

References

[1] Bernstein, Ethan, and Umesh Vazirani. "Quantum complexity theory." *SIAM Journal on computing* 26.5 (1997): 1411-1473.

[2] Deutsch, David. "Quantum theory, the Church–Turing principle, and the universal quantum computer." *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences* 400.1818 (1985): 97-117.

[3] Deutsch, David. "Quantum computational networks." *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences* 425.1868 (1989): 73-90.

[4] Molina, Abel, and John Watrous. "Revisiting the simulation of quantum Turing machines by quantum circuits." *Proceedings of the Royal Society A* 475.2226 (2019): 20180767.

[5] Turing, Alan Mathison. "On computable numbers, with an application to the Entscheidungsproblem." *Proceedings of the London Mathematical Society* 2.1 (1937): 230-265.

[6] Yao, A. Chi-Chih. "Quantum circuit complexity." *Proceedings of 1993 IEEE 34th Annual Foundations of Computer Science.* IEEE, (1993): 352-361.