"Hamiltonian Quantum Cellular Automata in 1D" by Daniel Nagaj and Pawel Wocjan

Maris Ozols
(ID 20286921)

April 14, 2008

1 Introduction

Initially classical computers were build for a specific (of course military) purpose, thus the program was built in into the hardware. To run a different program one virtually had to build a new computer. However, after a while the first computers with stored programs appeared (they were built according to *von Neumann architecture*). Nowadays the development of *quantum computers* has taken a different direction, since we already have classical computers that can take over lots of functions. Usually a quantum computer is designed as follows: the data is stored on a quantum computer, but the program (a sequence of gates that must be applied) – on a classical computer attached to it. Thus it would be fair to call this beast "a quantum system with a classical computer attached to it" rather than "a quantum computer", since the execution of the program is fully controlled by a classical computer that sends a specific signal to the quantum system each time a gate must be applied. It suffices to implement a few types of quantum gates to be able to perform any quantum computation (i.e., to simulate any quantum circuit) in this way.

One might wonder if it is possible to build a quantum computer that stores and executes the program itself without any assistance of a classical computer? It turns out that it is possible. Such design of a quantum computer is called *Hamiltonian Quantum cellular automaton* or HQCA. A computation using HQCA consists of several steps:

1. prepare a basis state containing both – the *input* and the *program* itself,
2. let this state evolve undisturbed according to some *fixed* Hamiltonian for some amount of time,
3. measure a small subsystem in the computational basis to get the answer.

This approach is somewhat similar to *adiabatic quantum computing* and also resembles the *Feynman quantum computer*. It differs from the first one, since a

time-independent Hamiltonian is used. However, the most important difference from both models is that the Hamiltonian does not depend on the problem that has to be solved as well – it depends only on the problem size, because the problem is encoded into the input state. Therefore it is much closer to a classical model of computation called *universal Turing machine*.

Since our Hamiltonian depends only on the size of the problem, we can have a family of Hamiltonians of different sizes that lets us simulate any quantum circuit. We would like these Hamiltonians to be similar and as simple as possible. Natural restrictions to put on each Hamiltonian are as follows:

- it acts on a 1D chain of qudits,
- it is local – only nearest-neighbor interactions,
- it is translationally invariant.

It turns out that it is possible to satisfy these conditions simultaneously and in addition to have a reasonably small dimension d of each qudit. Two constructions of HQCA with such Hamiltonians will be given – in the first case $d = 10$, but in the second case $d = 20$.

2 Circuit simulation

In this section we will consider some ideas that will be relevant for both constructions. In particular, we will discuss how an arbitrary quantum circuit on N qubits can be encoded into a chain of qudits that is only polynomially longer than the length of the circuit.

2.1 Universal set of quantum gates

To be able to perform any quantum computation we need a universal set of quantum gates. Many such sets are known, however, we want our set to be as simple and small as possible. Moreover, we want the resulting Hamiltonian to satisfy the restrictions mentioned above. First, all gates must act locally on a chain of qubits. Second, if we have a certain k-qubit gate in our repertoire, we can apply it to any k *adjacent* qubits in only one way (the same as the original gate) – this will give us translational invariance. For example, if we have N qubits in total, then there are $N - k + 1$ ways to apply a k-qubit gate.

It is clear that we need a gate that acts on at least two-qubits. Since we might have to apply it to pairs of non-adjacent qubits or to adjacent qubits but in different order, we also need a gate that swaps two qubits:

$$S = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

It turns out that these two gates are enough.

Theorem 1 (Rudolph & Grover, 2002)**.** If ϕ is an irrational multiple of π, then

$$G(\phi) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \cos\phi & -\sin\phi \\ 0 & 0 & \sin\phi & \cos\phi \end{pmatrix}$$

is universal for quantum computing. [RG02]

At first this result seems very surprising, because all entries of $G(\phi)$ are real and hence we clearly cannot approximate a unitary with imaginary entries. Moreover, since $|00\rangle$ is an eigenvector of both $G(\phi)$ and $SG(\phi)S$, the basis vector $|00\ldots0\rangle$ will always be an eigenvector of the resulting unitary, therefore we cannot approximate all rotations of the whole space. However, both apparent contradictions can be easily resolved. First, we have to use an additional qubit with basis states $\{|\mathrm{Re}\rangle, |\mathrm{Im}\rangle\}$ to simulate the real and imaginary parts separately as shown in [RG02]. Second, if we can approximate any rotation in some subspace, we can add more ancilla qubits to make this subspace arbitrary large.

The main idea of the proof of Theorem 1 is to show that we can simulate some set of quantum gates known to be universal, e.g., one-qubit gates and CNOT (which are universal according to [BBC+95]). For one-qubit gates it suffices to simulate any rotation around y and z axis. Rotation around y axis can be obtained using $G(\phi)$ with ancilla initialized to $|1\rangle$. For z axis we use the qubit that must be rotated as a control, but $\{|\mathrm{Re}\rangle, |\mathrm{Im}\rangle\}$ ancilla as the target. Finally, in [RG02] it is claimed that $G(\frac{\pi}{2})$ can be used instead of CNOT.

One more thing has to be clarified before we can use this theorem. Namely, in [RG02] it is assumed that $G(\phi)$ can be applied to any pair of qubits. This can be done by conjugating $G(\phi)$ with some permutation of qubits. Note that any permutation can be obtained using $O(N^2)$ gates S. Thus any circuit can be transformed into an equivalent one consisting only of $G(\phi)$ and S. According to *Solovay-Kitaev theorem* the length of the circuit will increase only by a polylogarithmic factor.

In [NW08] the gate $G(\phi)$ is used with ϕ a *rational* multiple of π.

Claim (Nagaj & Wocjan, 2008)**.** The gate

$$W = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ 0 & 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix} = G(\tfrac{\pi}{4})$$

is universal for quantum computing. [NW08]

It is argued that W is universal, because it can be used to simulate Toffoli and Hadamard gates, which are universal according to [Aha03]. However, it is not immediately obvious and it is also not explicitly demonstrated. Thus for convenience we will use W gate, but it can be replaced by $G(\phi)$ with ϕ an irrational multiple of π at any time.

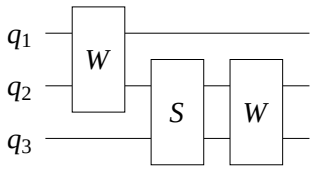


Figure 1: Original circuit.

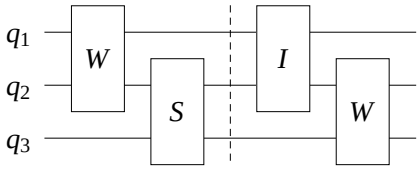


Figure 2: Circuit in the "ladder form".

2.2 Circuits in "ladder form"

In the previous section we discussed how an arbitrary circuit can be transformed into one consisting only of W and S gates that are applied to adjacent qubits and the control qubit of W is always above the target qubit (an example is shown in Fig. 1). However, we will have to transform our circuit further, since we want to lay it out on a chain of qudits in a specific way.

We break down the circuit into several *sequences*, so that in each sequence the gates have strictly descending positions. We insert additional identity gates if necessary, so that the vertical position always decreases by 1. We say that such circuit is in the "*ladder form*". For example, the circuit shown in Fig. 1 transforms into one shown in Fig. 2.

Note that there will be exactly $N-1$ gates in each sequence and the number of sequences is at most the number of gates in the original circuit. Hence the length of the circuit after this transformation will increase at most $N-1$ times, where N is the number of qubits.

3 Universal HQCA with $d = 10$

3.1 Encoding the circuit

In this section we will discuss how the circuit and data are encoded into the qudit chain. We will think of the state space of each qudit as a tensor product of two subspaces:

- *program subspace* spanned by $\{\centerdot, \blacktriangleright, W, S, I\}$,
- *data subspace* spanned by $\{0, 1\}$.

Thus the basis states of qudits are labeled by elements of $\{\centerdot, \blacktriangleright, W, S, I\} \times \{0, 1\}$, hence the state space is 10-dimensional.

Once the circuit is transformed into the "ladder form", we prepend an identity gate I to each sequence of gates and encode the obtained sequence as a string of length N over the alphabeth $\{W, S, I\}$. The reason for prepending these identity gates will become clear later. For example, the circuit shown in Fig. 2 is encoded as (IWS, IIW). Then for each gate sequence we create the corresponding *marker sequence* that consists of $N-1$ dots "$\centerdot$" and the *pointer symbol* "$\blacktriangleright$". For example, the marker and gate sequences for the circuit in

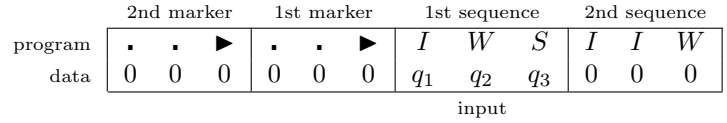


Figure 3: Initial configuration of HQCA with $d = 10$.

Fig. 2 are shown in Fig. 3 (see the first row of the qudit chain). Note that we number the marker sequences in reversed order, since the first marker sequence will interact with the first gate sequence.

Finally, we initialize the data qubits below the first gate sequence in the basis state $|q_1, q_2, \ldots, q_N\rangle$ that contains the input of the circuit and reset all other qubits to $|0\rangle$ (see the second row of the qudit chain in Fig. 3).

The length of the qudit chain is $\frac{2N}{N-1}$ times the number of sequences in the "ladder form" circuit. Thus the qudit chain is at most $2N$ times longer than the original circuit.

3.2 Transition rules

In this section we will define the rules of a discrete-time cellular automaton whose evolution mimics the execution of the circuit. Surprisingly, our cellular automaton has only two transition rules:

where $A \in \{W, S, I\}$. The first rule corresponds to moving a gate to the left, but the second – to applying the gate to the two qubits below it. Note that usually there will be many ways how these rules can be applied, thus our cellular automaton is *non-deterministic*.

3.3 Discrete-time evolution

In this section we will discuss the evolution of a discrete-time cellular automaton according to the rules defined above. We will see that all gates will be applied in the right order and to the right qubits, no matter how we choose to apply the transition rules.

Both rules move the gates from right to left, but the second rule applies the gate as well. For each gate there is a unique pair of adjacent qubits on which the gate must be applied. However, we see that each gate will be applied several times on different pairs of qubits (the number of times equals the number of pointer symbols "▶"). More precisely, the ith pointer symbol will apply the gates from the ith sequence at correct positions no matter in which order the rules are used (the reason for this is that the pointer symbol "▶" cannot move

through "."). Unfortunately all other gates will be applied to wrong qubits, so we have to make sure that this does not destroy the result computed so far.

Luckily this is the case, since the only qubits in the chain that get changed during the computation are the ones that contain the input (we will call them *data qubits*). They get changed only when the corresponding gates are applied to them. All other qubits remain in the state $|0\rangle$ throughout the computation. To see this, we will consider two cases.

First, let us consider the gates from the circuit in the "ladder form" (i.e., all gates except the identity gates prepended to each sequence). Remember that all pointer symbols "▶" and sequences of gates are numbered. Note that a pointer symbol can never meet a gate from a sequence with a different number above the data qubits. It will meet the gate on the left of the data qubits if the pointer symbol has a larger number, or on the right, otherwise. In any case the qubits below the pointer symbol and the gate will be blank, i.e., have value $|00\rangle$. Since this is a common eigenvector with eigenvalue 1 for all three gates W, I, S, it will not be affected when the gate is applied.

Second, if the pointer symbol meets a gate above $|0\rangle\,|q\rangle$ or $|q\rangle\,|0\rangle$, where $|q\rangle$ is one of the data qubits, it has actually met one of the identity gates prepended to each sequence. Thus the data will not get corrupted in this case as well. This is the reason why the identity gates were added.

	1st sequence			2nd sequence			2nd marker			1st marker		
program	I	W	S	I	I	W	.	.	▶	.	.	▶
data	0	0	0	0	0	0	|	s_{123}	$\rangle$	0	0	0
							output					

Figure 4: Final configuration of HQCA with $d = 10$.

When a polynomial amount of time (in the number of gates of the circuit) has passed, all gates will be moved to the left of the chain, no matter in which order the rules were applied. The final configuration, is shown in Fig. 4.

3.4 The Hamiltonian

The transition matrix for two adjacent qubits of a discrete cellular automaton that evolves according to the rules defined above can be written as follows:

$$R = \sum_{A\in\{W,S,I\}} \Big(|A.\rangle\,\langle .A| \otimes I + |A\,\blacktriangleright\rangle\,\langle \blacktriangleright\, A| \otimes A \Big).$$

Since these rules apply to any two adjacent qubits, the Hamiltonian is:

$$H_{10} = -\sum_{j=1}^{L-1} (R + R^\dagger)_{(j,j+1)},$$

where L is the length of the qudit chain. Unfortunately we have to include the term $R^\dagger$ that allows backward evolution, since the Hamiltonian must be

Hermitian. Thus we simultaneously compute and uncompute. However, this will not cause problems.

3.5 Hamiltonian evolution

The analysis of the evolution of the HQCA is more complicated than that of a discrete-time cellular automaton, since the evolution is continuous and happens in both directions simultaneously.

Consider how H_{10} acts on the program register. If we map $\{\blacktriangleright, \centerdot\}$ to $|\bar{0}\rangle$ and all gates $\{W, S, I\}$ to $|\bar{1}\rangle$, then we get

$$H_{\text{fermion}} = -\sum_{j=1}^{L-1} \left(|\bar{1}\bar{0}\rangle \langle \bar{0}\bar{1}| + |\bar{0}\bar{1}\rangle \langle \bar{1}\bar{0}| \right)_{(j,j+1)}.$$

If we think of $|\bar{1}\rangle$ as a fermion, but $|\bar{0}\rangle$ as no fermion, then this Hamiltonian describes a diffusion of a discrete free fermion gas on a line. One can see that the initial state will remain in the subspace spanned by the basis vectors of the same Hamming weight. Thus we can interpret this process as a certain number fermions that are moving around.

The computation is done when all fermions (gates) have moved to the left. However, the probability of such event is small, when L is large and it also depends on time. To deal with the first problem we can increase the length of the qudit chain by prepending a large number of qudits in the state $\left[\begin{smallmatrix} \vdots \\ 0 \end{smallmatrix}\right]$ and appending a few qudits $\left[\begin{smallmatrix} I \\ 0 \end{smallmatrix}\right]$. Then it will be more likely to find the all non-identity gates on the left of the data qubits. To deal with the time-dependence, we simply choose the evolution time uniformly at random from 0 to some τ_{10}. It is possible to adjust the number of gates padded and choose the maximum evolution time τ_{10} to be polynomial, so that the computation is completed with high probability.

4 Universal HQCA with $d = 20$

4.1 Encoding the circuit

As before, we prepend the identity gate I to all gate sequences. However, we also append it to all sequences except the last one. The last sequence is appended with the turn-around symbol "$\circlearrowleft$". For example, the circuit shown in Fig. 2 is encoded as $(IWSI, IIW \circlearrowleft)$. The rest of the first row is filled with the empty spot symbol "$\centerdot$" (see Fig. 5).

Similarly we put the input data $|q_1, q_2, \ldots, q_N\rangle$ below the first identity gate and the first gate sequence (see the second row of Fig. 5). This time we do not fill the rest of the second line with zeros, but instead we put *boundary markers* "1", so that there are exactly N zeros between any two consecutive boundary markers (see Fig. 5). We put the same number of boundary markers on both sides of the input qubits.

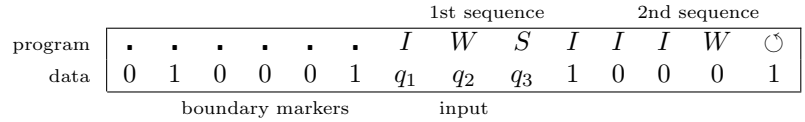


Figure 5: Initial configuration of HQCA with $d = 20$.

This time the dimension of each qudit is $d = 20$, because the basis states are marked by elements of $\{W, S, I, \textcircled{W}, \textcircled{S}, \textcircled{I}, \blacktriangleright, \triangleright, \circlearrowleft, .\} \times \{0, 1\}$. These symbols have the following meanings:

- $\textcircled{W}$ $\textcircled{S}$ $\textcircled{I}$ – *marked gates* used to propagate the active spot to the left,
- $\blacktriangleright$ – *apply gate symbol* has the same meaning as before,
- $\triangleright$ – *shift gate symbol* has the same function as the dot symbol "." before, since it is used to move the gates away after they have been applied,
- $\circlearrowleft$ – *turn-around symbol* appears each time the active spot changes its direction of motion,
- . – *empty spot symbol* has a slightly different meaning, since it does not move, but is transported indirectly via the apply or shift gate symbols.

The length of the qudit chain will be smaller than $2(N+1)$ times the number of the gates in the original circuit.

4.2 Transition rules

To move the active spot from right to left:

$$\boxed{A}\boxed{\circlearrowleft} \to \boxed{\textcircled{A}}\boxed{.} \quad \text{(R1.1)}$$

$$\boxed{A}\boxed{\textcircled{B}} \to \boxed{\textcircled{A}}\boxed{B} \quad \text{(R1.2)}$$

$$\boxed{.}\boxed{\textcircled{A}} \to \boxed{\circlearrowleft}\boxed{A} \quad \text{(R1.3)}$$

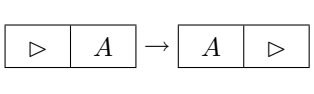
To turn around at the left end:

$$\begin{array}{|c|c|}\hline . & \circlearrowleft \\ \hline & 1 \\ \hline\end{array} \to \begin{array}{|c|c|}\hline . & \blacktriangleright \\ \hline & 1 \\ \hline\end{array} \quad \text{(R2.1)} \qquad \begin{array}{|c|c|}\hline . & \circlearrowleft \\ \hline & 0 \\ \hline\end{array} \to \begin{array}{|c|c|}\hline . & \triangleright \\ \hline & 0 \\ \hline\end{array} \quad \text{(R2.2)}$$

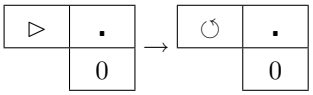
To apply the gates and turn around at the right end:

$$\begin{array}{|c|c|}\hline \blacktriangleright & A \\ \hline x & y \\ \hline\end{array} \to \begin{array}{|c|c|}\hline A & \blacktriangleright \\ \hline \multicolumn{2}{|c|}{A(x,y)} \\ \hline\end{array} \quad \text{(R3.1)} \qquad \begin{array}{|c|c|}\hline \blacktriangleright & . \\ \hline & 1 \\ \hline\end{array} \to \begin{array}{|c|c|}\hline \circlearrowleft & . \\ \hline & 1 \\ \hline\end{array} \quad \text{(R3.2)}$$

To shift the gates and turn around at the right end:



(R4.1)



(R4.2)

In all rules A and B stand for any gate from $\{W, I, S\}$.

Both rules for turning around at the right end slightly differ from those in the original paper [NW08], since I believe there is a mistake. For example, the original rule for turning "▶" into "↺" will clearly fail for the configuration shown in equation (33) in [NW08].

4.3 Discrete-time evolution

The most important feature of this construction is that the discrete-time evolution of the cellular automaton is *deterministic*, i.e., at each moment of time there is exactly one rule that can be applied. Moreover, it is designed so that the backward evolution is also deterministic, thus this automaton is *reversible*. To make sure that during the backward evolution no more than one rule can be applied at each moment, it is important that the presence of the boundary marker is checked in rules (R3.2) and (R4.2). During the forward evolution this might seem strange, since the empty spot symbol "▪" already indicates the end of the program sequence.

First we apply rules (R1.1), (R1.2), and (R1.3) to move the active spot to the front of the program sequence. Then we have to apply either (R2.1) or (R2.2) depending on the value of the qubit below the turn-around symbol "↺". In our case we have to apply (R2.1) to obtain the apply gate symbol "▶". Then we use (R3.1) several times to apply the first sequence of gates. Once the apply gate symbol reaches the end of the program sequence, we use (R3.2) to get back the turn-around symbol "↺". Again, we use (R1.1), (R1.2), and (R1.3) to move it to the front. This time we have to use (R2.2) to obtain the shift gate symbol "▷". Then we use (R4.1) several times to shift all gates to the left by one position. When the shift gate symbol reaches the end of the program sequence, we apply (R4.2) to get back the turn-around symbol. This process is repeated several times until the gates are shifted far enough so that the next sequence of gates can be applied.

The reason why the gates keep moving to the left is that all rules preserve the positions of the gates, except (R3.1) and (R4.1) that both move a gate one position to the left. However, it is not so obvious how new empty spot symbols "▪" appear on the right of the program sequence. Notice that (R1.1) and (R1.3) are the only rules that affect the number of the empty spot symbols. We can think of rule (R1.3) as grabbing "▪" from the front and (R1.1) as dropping it at the end of the program sequence.

The evolution stops when all gates have moved to the left. This will happen after a polynomial (in the length of the original circuit) number of steps. The final configuration is shown in Fig. 6.

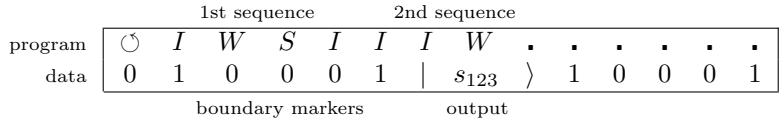


Figure 6: Final configuration of HQCA with $d = 20$.

4.4 The Hamiltonian

From rules (R1.1) to (R4.2) we obtain the following Hamiltonian:

$$H_{20} = -\sum_{j=1}^{L-1} \sum_{r \in \text{rules}} (R_r + R_r^\dagger)_{(j,j+1)},$$

where R_r for each rule r is defined as before.

4.5 Hamiltonian evolution

Since the discrete-time cellular automaton is deterministic and reversible, we can change the basis so that H_{20} transforms to

$$H_{\text{line}} = -\sum_{t=0}^{T-1} \big(|t\rangle \langle t+1| + |t+1\rangle \langle t| \big).$$

This Hamiltonian corresponds to a continuous time quantum walk on a line of length $T + 1$, where T is the number of steps it takes the discrete-time cellular automaton to finish the computation. The computation is done when the state of the system (in the new basis) is $|T\rangle$.

To increase the success probability, we use the same tricks as before – we append a large number of qudits in the state $\left[\begin{smallmatrix} I \\ 0 \end{smallmatrix}\right]$ to the chain and choose the evolution time uniformly at random from 0 to a polynomially large τ_{20}.

References

[NW08] Daniel Nagaj, Pawel Wocjan, Hamiltonian Quantum Cellular Automata in 1D, `arXiv:0802.0886v1`. Pawel's talk at PI is available at `pirsa-08010008`.

[RG02] Terry Rudolph, Lov Grover, A 2 rebit gate universal for quantum computing, `arXiv:quant-ph/0210187`.

[BBC+95] A. Barenco, C.H. Bennett, R. Cleve, D.P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. Smolin, H. Weinfurter, Elementary gates for quantum computation, `arXiv:quant-ph/9503016`.

[Aha03] Dorit Aharonov, A Simple Proof that Toffoli and Hadamard are Quantum Universal, `arXiv:quant-ph/0301040`.