A Physically Universal Quantum Cellular Automaton

Luke Schaeffer(✉)

Massachusetts Institute of Technology, Cambridge, Massachusetts, England
lrschaeffer@gmail.com

Abstract. We explore a quantum version of Janzing's "physical universality", a notion of computational universality for cellular automata which requires computations to be done directly on the cells. We discuss physical universality in general, the issues specific to the quantum setting, and give an example of a quantum cellular automaton achieving a quantum definition of physical universality.

1 Introduction

Many cellular automata are known to be *computationally universal*, in the sense that they can simulate Turing machines, and hence any other classical model of computation. Shortly after Bernstein and Vazirani introduced quantum Turing machines, Watrous [8] gave a definition for quantum cellular automata (QCA), and showed that QCA can simulate arbitrary quantum Turing machines. Raussendorf [4], van Dam [7], and others give QCA that achieve stricter notions of universality for quantum circuits.

Recently, Janzing [2] defined *physical universality* for cellular automata. A cellular automaton is *physically universal* if for any finite set of cells X, and any transformation f on the region X, there is some way to initialize the complement of X such that, whatever initial configuration x is in the region X, after some number of t timesteps, X contains $f(x)$. In other words, it is possible to perform any transformation on a region by initializing the surrounding cells and waiting for some prespecified time. We will discuss physical universality in general, the issues specific to the quantum setting, and finally give an example of a quantum cellular automaton which is physically universal in the quantum sense.

2 Cellular Automata

We will consider only layered cellular automata in the classical setting.

Definition 1. *A* layered cellular automaton *(LCA) is a 4-tuple* (L, V, Σ, ρ) *consisting of*

- *a finite dimensional lattice* $L = \mathbb{Z}^d$,
- *a list of shifts* $V = [v_1, \ldots, v_k]$ *for* k *different layers, where* $v_i \in \mathbb{Z}^d$,

J. Kari (Ed.): AUTOMATA 2015, LNCS 9099, pp. 46–58, 2015.
DOI: 10.1007/978-3-662-47221-7_4

– *a finite set of states, Σ, and*
– *a rule $\rho\colon \Sigma^k \to \Sigma^k$ for updating a cell.*

The points of in the lattice are called *cells*. Each *cell* in the lattice has k *layers*, and each layer of a cell has some state in Σ. Each time step, we apply the cell update rule ρ to every cell in the lattice, then shift each layer by the corresponding shift vector v_i. That is, the ith component of the state of a cell $x \in \mathbb{Z}^d$ is moved to the cell $x + v_i \in \mathbb{Z}^d$.

A *region* of cells is any finite subset of the lattice. A *configuration* of a region $X \subseteq L$ is mapping from cells in X to states in Σ^k, and we let $\mathcal{C}(X)$ denote the set of all configurations of X. Naturally, we can combine configurations $x \in \mathcal{C}(X)$ and $y \in \mathcal{C}(Y)$ of disjoint regions X and Y into a configuration $x \times y \in \mathcal{C}(X \cup Y)$ of $X \cup Y$.

A cellular automaton is *reversible* if every configuration of the automaton has a unique predecessor (according to the update rule of the automaton). One advantage of layered cellular automata is that it is trivial to check reversibility – a LCA is reversible if and only if the cell update rule ρ is bijective.

3 Quantum Cellular Automata

A reader unfamiliar with quantum computation will find quantum cellular automata analogous to probabilistic cellular automata (PCA). Probabilistic cellular automata (PCA) generalize (deterministic) cellular automata by letting the configuration of the lattice be a probability distribution over deterministic configurations, and letting the update rule map each classical state to a distribution of states.

If we think of quantum mechanics as a theory of probability with complex numbers [9], then quantum cellular automata (QCA) are very much like probabilistic cellular automata, except the probability distribution is replaced with a *quantum superposition* of classical configurations. That is, a quantum configuration is a map $\psi\colon \mathcal{C}(L) \to \mathbb{C}$ from classical configurations to complex number *amplitudes* such that

$$\ell_2(\psi) = \sum_{x \in L} |\psi(x)|^2 = 1.$$

The analogy between PCA and QCA does not end there. Given a distribution of deterministic configurations, each cell has a *marginal distribution*, but the marginal distributions for all cells do not give us a complete picture of a configuration because the states of cells may be *correlated*. Similarly, each cell of QCA has a quantum state, but cells may be *entangled*.

Let us define quantum cellular automata more formally.

Definition 2. *A* layered quantum cellular automaton *(LQCA) is a 5-tuple (L, V, Σ, q, ρ) consisting of*

– *a finite dimensional lattice $L = \mathbb{Z}^d$,*
– *a list of shifts $V = [v_1, \ldots, v_k]$ for k different layers, where $v_i \in \mathbb{Z}^d$,*

- *a finite set of states,* Σ,
- *a special* quiescent state $q \in \Sigma^k$, *and*
- *a unitary transformation* ρ *on the Hilbert space*

$$\mathcal{H} = \{f : \Sigma^k \to \mathbb{C}\}$$

of functions from Σ^k *to* $\mathbb{C}$.

In addition, we require that ρ *fixes the quiescent state. That is, if* $\psi \in H$ *is the map that sends* q *to* 1 *and all other inputs to* 0, *then* $\rho(\psi) = \psi$.

Intuitively, the state of a cell is a vector x in $\mathcal{H}$ such that $\|x\|_2 = 1$. The state of a region, X, containing k cells is a vector x in the tensor product $\mathcal{H}^{\otimes k}$ such that $\|x\|_2 = 1$, which makes sense because $H^{\otimes k}$ is isomorphic to the Hilbert space of functions from $\mathcal{C}(X)$ to $\mathbb{C}$. We would like a quantum configuration for the entire lattice to be a vector of

$$\bigotimes_{x \in L} \mathcal{H},$$

but this infinite tensor product is not well-defined. Instead, we say a *finite configuration* is a classical configuration $c \in \mathcal{C}(L)$ such that all but finitely many cells in the quiescent state, q. Let $\mathcal{C}^*(L)$ be the set of all finite (classical) configurations the lattice L. Then the *quantum configurations* of the LQCA are defined to be

$$\mathcal{Q}(L) := \{\psi : \mathcal{C}^*(L) \to \mathbb{C} \mid \ell_2(\psi(x)) = 1\}.$$

That is, functions from finite configurations to amplitudes such that the ℓ_2-norm is 1.

The evolution of the LQCA is defined by ρ, a unitary transformation on the quantum state of a cell, and V, the list of shifts. As before, we apply ρ to each cell, then shift each layer by the corresponding vector. Each step is *linear* in the sense that if $x, y \in \mathcal{Q}(L)$ are quantum configurations which evolve to $x', y' \in \mathcal{Q}(L)$ when we apply ρ to every cell, then $\alpha x + \beta y$ evolves to $\alpha x' + \beta y'$ (when we apply ρ) for all $\alpha, \beta \in \mathbb{C}$ such that $\ell_2(\alpha x + \beta y) = |\alpha|^2 + |\beta|^2 = 1$. Therefore it suffices to define the two steps for quantum configurations where for classical configurations (actually quantum configurations where one classical configuration has amplitude 1).

Recall that a finite configuration has only finitely many cells which are not in state q. Since ρ fixes state q, we can ignore all those cells (they remain in state q), and consider a finite quantum system composed of the remaining cells. The set of quantum states for the finite set of cells is in $H^{\otimes k}$ for some k, and we apply ρ to each cell in this finite dimensional space, i.e., apply $\rho \otimes \rho \otimes \cdots \otimes \rho$. We have already seen how to shift the layers of a classical configuration; it is the same in the quantum setting as it was in the classical setting.

4 Physical Universality

Computational universality is well studied in cellular automata. There are cellular automata which can simulate a wide variety of (formal) computational devices: circuits, Turing machines, quantum circuits, other cellular automata, etc. Almost all of these cellular automata require the "data" to be written in a special form, usually distinct from the "program".

- Conway's Life encodes information as gliders, but the program must be laid out as glider guns.
- Margolus' billiard ball machine uses balls to represent data, and the configuration of the "table" determines the computation.
- Raussendorf's universal quantum CA [4] puts quantum bits in even columns (moving left), and the description of quantum gates in odd columns (moving right). Computation occurs as the interleaving columns pass each other.
- Wim van Dam's CA operates directly on qubits, but the program cells are over a larger state space.

Janzing [2] defined physical universality as a stronger notion of universality for cellular automata. Informally, a cellular automaton is *physically universal* if one can implement any transformation on any finite set of cells by "programming" the other cells. To state it more formally, we first need the following definition.

Definition 3. *Let M be a CA on a lattice L. Let X be a region of the lattice. We say a configuration $y \in \mathcal{C}(L\backslash X)$* implements a transformation $f\colon \mathcal{C}(X) \to \mathcal{C}(X)$ in t time steps *if for every configuration $x \in \mathcal{C}(X)$, there exists a configuration $y' \in \mathcal{C}(L\backslash X)$ such that $x \times y$ evolves to $f(x) \times y'$ in t timesteps.*

Then physical universality (in the classical setting) is defined as follows.

Definition 4. *Let M be a CA on a lattice L. Then M is* physically universal *if for every finite set of cells $X \subseteq L$ and for every function $f\colon \mathcal{C}(X) \to \mathcal{C}(X)$, there exists a configuration y of $L\backslash X$ and a time $t \in \mathbb{Z}$ such that y implements the transformation f on X in t timesteps.*

There were no examples of physically universal CAs in Janzing's original paper. We now know that (classical) physically universal CAs exist, with relatively simple examples due to Schaeffer [6], Salo and Törmä [5]. These examples are all layered cellular automata, and the construction used to show physical universality has the same general structure in each case:

1. First, show that any finite configuration eventually becomes inactive.
2. Allow the input configuration to become inactive, and collect whatever information remains.
3. Use the reversibility and computational universality of the automaton to forensically reconstruct the original configuration of the input region.
4. Use computational universality again to apply the given transformation on the input.

5. Find a way to put the desired output configuration in the output region, usually by appealing to reversibility and computational universality yet again.

We will follow exactly the same approach to show that a layered quantum cellular automaton is physically universal, but first let us define quantum physical universality.

5 Quantum Physical Universality

Before we introduce quantum physical universality in cellular automata, let us discuss the inherent limitations of programmable quantum devices in the context of quantum circuits. Nielsen and Chuang [3] call such circuits *programmable quantum gate arrays* (PQGA).

Definition 5. *Let G be a quantum circuit with a* program register $\mathcal{P}$*, and a* data register $\mathcal{D}$*, each consisting of many qubits. Then G is a* programmable quantum gate array *if there exist program states $\{P_i\}_{i\in I}$, and unitary transformations $\{U_i\}_{i\in I}$ of the data register such that for all i and d,*

$$G(P_i \otimes d) = P_i'(d) \otimes U_i(d)$$

where $P_i'(d)$, the garbage left in the program register, may depend on i and d.

In other words, there are $|I|$ settings of the program register, which effect unitary transformations $\{U_i\}_{i\in I}$ on a data register. Nielsen and Chuang make a number of observations about PQGAs in [3], which we summarize in the following theorem.

Theorem 1. *Let G be a PQGA, with P_i, P_i' and U_i as above. Then*

1. *the garbage in the program register, $P_i'(d) = P_i'$, does not depend on d, and*
2. *if U_i and U_j are not the same (up to multiplication by $e^{i\theta}$) then P_i and P_j are orthogonal.*

This has several interesting consequences for quantum physical universality.

1. Since distinct programs have orthogonal program states, the number of unitary operations is bounded by $|\mathcal{C}^*(\mathcal{P})|$, the number of classical configurations of the program register. It is natural to use classical program states (i.e., configurations in $\mathcal{C}^*(\mathcal{P})$), because there is apparently nothing to gain by making them quantum superpositions.
2. The program register *after* the computation cannot depend on the input in the data register. This is purely a side-effect of unitary evolution. Compare this to the notion of reversible physical universality in the classical setting [6], where the final value of the program register needs to be *defined* to be independent of the data.

3. If the program register is finite then the PQGA has finitely many distinct programs. In our cellular automaton, the program register is technically infinite, but since only finitely many bits can interact with the data in any given time, there are still only finitely many distinct programs. Hence, we must abandon the idea of implementing all unitary transformations, and confine ourselves to *approximations* of arbitary unitaries. Fortunately, this problem is shared by quantum circuits, so there are procedures [1] for approximating unitary transformations with a finite set of quantum gates.

This informs our definition of quantum physical universality.

Definition 6. *Let M be a QCA on a lattice L, and let $X \subseteq L$ be a region. We say a configuration $y \in \mathcal{Q}(L\backslash X)$* implements a transformation $U\colon \mathcal{Q}(X) \to \mathcal{Q}(X)$ in t timesteps *if for every configuration $x \in \mathcal{Q}(X)$, there exists a configuration $y' \in \mathcal{Q}(L\backslash X)$ such that $x \otimes y$ evolves to $U(x) \otimes y'$ in t timesteps.*

Similarly, we say a configuration y of Y ϵ-approximately implements a transformation U in t timesteps *if y implements some transformation U' such that $\|U - U'\|_{tr} \leq \epsilon$, where*

$$\|\cdot\|_{tr} = \mathrm{trace}(\sqrt{A^*A})$$

is the trace norm.

Definition 7. *Let M be a QCA on a lattice L. Then M is* physically universal *if for every finite set of cells $X \subseteq L$, every unitary transformation $U\colon \mathcal{Q}(X) \to \mathcal{Q}(X)$ and every $\epsilon > 0$, there exists a configuration $y \in \mathcal{Q}(L\backslash X)$ and a time $t \in \mathbb{Z}$ such that y ϵ-approximately implements the transformation U on X in time t.*

We will see an alternative definition later, once we have an example of a physically universal quantum cellular automaton.

6 A Physically Universal LQCA

Our physically universal LQCA is on the lattice $L = \mathbb{Z}$, and has six layers of qubits (i.e., classical state 0 or 1) with speeds $-3, -2, -1, 1, 2, 3$. Like the reversibly physically universal CA of Salo and Törmä, we program a universal set of gates into the update rule (see below) for automaton. Specifically, we use the controlled-NOT CNOT, the $\pi/8$ gate T and the Hadamard gate H, described briefly below.

CNOT**:** A two-bit classical gate common in reversible computation. If the control bit is 1 then the other bit is negated, otherwise neither bit changes.

T**:** The $pi/8$ gate is a single qubit gate which changes the *phase* if the input is 1, but does nothing.

H**:** The Hadamard gate is a single qubit represented by the matrix

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

In other words, $H(0)$ is the superposition where 0 and 1 have weight $\frac{1}{\sqrt{2}}$, and $H(1)$ has the sign of 1 reversed.

These three gates are known to be universal, and there exist algorithms for approximating arbitrary unitary transformations [1] with this gate set .

We define ρ by defining it for classical inputs and extending linearly to quantum superpositions thereof. Given a classical cell $(x_{-3}, x_{-2}, x_{-1}, x_1, x_2, x_3)$, we define $\rho(x_{-3}, x_{-2}, x_{-1}, x_1, x_2, x_3)$ by the following list of rules. Use the first rule that applies.

- If $x_{-3} = x_{-2} = x_{-1} = 1$ then cyclically permute $x_1 \to x_2 \to x_3 \to x_1$. Likewise, if $x_1 = x_2 = x_3 = 1$ then cyclically permute $x_{-1} \to x_{-3} \to x_{-2} \to x_{-1}$.
- If $x_2 + x_3 = 1 = x_{-2} + x_{-3}$ then either swap x_1 and x_{-1}, or perform a controlled-NOT on x_1 and x_{-1} as follows.
  - If $x_2 = x_{-2} = 1$ or $x_3 = x_{-3} = 1$ then swap x_1 and x_{-1}.
  - If $x_2 = x_{-3} = 1$ then apply CNOT to x_1 and x_{-1} with x_1 as the control bit.
  - If $x_{-2} = x_3 = 1$ then apply CNOT to x_1 and x_{-1} with x_{-1} as the control bit.
- If $x_2 = x_3 = x_{-3} = 1$ and $x_{-2} = 0$ then apply H to x_{-1}. Similarly, if $x_3 = x_{-2} = x_{-3} = 1$ and $x_2 = 0$ then apply H to x_1.
- If $x_2 = x_3 = x_{-2} = 1$ and $x_{-3} = 0$ then apply T to x_{-1}. Similarly, if $x_2 = x_{-2} = x_{-3} = 1$ and $x_3 = 0$ then apply T to x_1.
- Otherwise, leave the cell unchanged.

Observe that the CA is almost entirely classical. If a cell is in a classical state initially then, in most cases, ρ maps it to another classical state. The only exceptions are a handful of cases where change the phase (for T) or introduce a superposition (for H).

Our goal is to show that this LQCA is physically universal. Assume we are given a finite region X, and a unitary transformation U on the cells in X. By the Solovay-Kitaev theorem [1], for any $\epsilon > 0$ there exists a circuit C (of H, T and CNOT gates) which implements a unitary within ϵ of U. The problem is then to implement the circuit C.

We follow the pattern used in Schaeffer [6], and Salo and Törmä [5], so we start by showing that information contained in a bounded region will escape in a recoverable format. To this end, we define the notion of a depleted configuration.

Definition 8. *A quantum configuration $x \in \mathcal{Q}(\mathbb{Z})$ is* depleted *if, for every classical configuration with nonzero amplitude in x, there is at most one particle per cell and no particle of speed v_i occurs to the right of a particle of speed $v_j > v_i$, for all i and j.*

We show that any finite configuration quickly becomes depleted.

Theorem 2. *Let $X \subseteq \mathbb{Z}$ be the region $X = \{1, \ldots, n\}$ and let Y be the complement. Suppose $0_Y \in \mathcal{Q}(Y)$ is the configuration with all cells in the 0 state. Then there is a time $t = \frac{5}{2}n + O(1)$ such that for any $x \in \mathcal{Q}(X)$, the configuration $x \otimes 0_Y$ evolves to a depleted configuration within time t.*

Proof. The update rule does nothing to a cell if either $x_{-3} = x_{-2} = x_{-1} = 0$ or $x_3 = x_2 = x_1 = 0$. In particular, if there are no right-moving particles in cells $(-\infty, j]$ at some time, then there are no right-moving particles in the range $(-\infty, j+1]$ on the following step, because all the right-moving particles have moved right by at least one cell. Similarly for left-moving particles in $[j, \infty)$.

There are initially no right-moving particles (or particles of any kind) $(-\infty, 0]$, and no left-moving particles in $[n+1, \infty)$. By the observation above, there will be no right-moving particles in $(-\infty, \lceil \frac{n+1}{2} \rceil)$ in $\lceil \frac{n+1}{2} \rceil$ steps, nor left-moving particles in $[\lfloor \frac{n+1}{2} \rfloor, \infty)$. It follows that every cell contains only left-moving or right-moving particles (or neither), and therefore particles move at constant speed without interacting.

In $\lceil \frac{n+1}{2} \rceil$ time steps, a particle could move as far as $\frac{3}{2}n+1$ cells from its initial position. The right-moving particles, for instance, could be spread over a range of $2n + O(1)$ cells between $\frac{n}{2}$ and $\frac{5}{2}n + 2$. So it will take at most $2n + O(1)$ steps for the left-most speed 2 particles to overtake the right-most speed 1 particles, or for the left-most speed 3 particles to overtake the right-most speed 2 particles. Hence, the right-moving particles will be ordered by speed in at most $\frac{5}{2}n + O(1)$ time steps. Similarly for the left-moving particles, so the configuration becomes depleted in $t = \frac{5}{2}n + O(1)$ time steps. □

Suppose we start with a finite configuration. Over time, it becomes depleted according to the theorem above. At that point, whatever computation the configuration may have performed is over, since no interactions can occur in a depleted configuration. Furthermore, the remains of the computation are readily available in six groups of particles. Remember that each of these particles is present or absent in each classical configuration, representing a bit, but since we are in a quantum configuration, each particle is a qubit, and the qubits may be entangled. We will collect these quantum particles, gather them together, perform a quantum computation, and then place the result back into X.

Let us discuss how to manipulate these quantum particles. The way we manipulate these particles is by placing purely classical particles, which we call *manipulators* to distinguish them from the particles that come out of X, in the complement of X. The manipulators will interact with the quantum particles in such a way that quantum operations (H, T and CNOT) are performed on the particles, yet the manipulators remain purely classical particles, and do not change speed or direction.

By inspection of the cell update rule, we need at least three particles in a cell for an interaction to occur. Two manipulators are required to perform a swap or apply CNOT, and three are required for T and H. In the cases where only two manipulators are required, the operation does nothing unless there is a third particle present (usually in the speed 1 or speed -1 layer). We rely on the fact that two particles or manipulators have at most one point of intersection. This ensures that any pair of manipulators can only affect one particle, at a time and place predetermined by the initial locations of the manipulators.

We need to show how to do three things with manipulators:

1. Take the six groups of particles in a depleted configuration and redirect them such that they all move in the same direction at speed 1, in a format suitable for computation.
2. Approximate an arbitrary quantum computation. We do this by implementing an arbitrary quantum circuit (with gates H, T, and CNOT) *exactly*.
3. Convert a single group of speed 1 particles back into six groups of different speed, aimed towards the output region X.

The first step is a poor introduction to the manipulation of particles, so we begin with the second step – computation – and prove Theorem 3. We will then return to the problem of redirecting particles and prove Theorem 4. Finally we argue that the third step is the reverse of the first step, and therefore follows from the Theorem 4.

Theorem 3. *Suppose X is a region of size $2n$, containing particles $x_1, \ldots, x_n$ of speed 1 in the even cells. Let C be a circuit on n inputs, composed of* CNOT*, H, and T gates. Then there exists a time t (polynomial in n and the size of C), such that we can implement the transformation defined by C on $x_1, \ldots, x_n$ in time t, leaving the result in the even cells of a region Y of size $2n$.*

Proof. Let us call the area containing the particles the *workspace*. The workspace is initially X, but moves right as the particles move right, and may grow as we move particles around.

We need to show how to implement four operations: we must be able to apply T, H and CNOT to quantum particles, and be able to move or swap particles around. It suffices to be able to move a particle left relative to its peers, since this allows us to completely reorder the particles if we need to. Given these four operations, it is clear we can implement an arbitrary circuit C.

T and H: The easiest operation is T. We arrange three manipulators (of speed 2, -2, -3) to intercept the desired particle x_i at some time. The update rule causes an T gate to be applied to the speed 1 layer, containing x_i. Similarly, three manipulators (of speed 3, -2, -3) will apply a Hadamard gate H to a speed 1 particle. Since none of the manipulators have the same speed as the particles, they spend at most $O(n)$ time steps in the workspace. After that, the workspace is clear for the next operation.

Move left: If we meet a particle with two manipulators of speed 3 and -3, it swaps the speed 1 and speed -1 particles. The speed -1 layer is kept empty, so this effectively reverses the direction of the particle. After the particle has travelled in the opposite direction for some time, we may reverse it again, as long as there is no speed 1 particle already in this cell. This allows us to move particles to the left. There are, however, a few limitations:

- A particle with speed -1 moves 2 cells per timestep relative to the particles of speed 1. Hence, the distance between the initial and final position is a multiple of two. This is why the particles are assumed to be in even cells.

– We must be careful not to let the manipulators from the two swaps meet. If they do, they would perform another swap, potentially sending one of the x_i's off in the wrong direction. The manipulators will meet if and only if the time between the two reversals is a multiple of three, which we can easily avoid. If we need to move a particle by a multiple of three cells, we simply split the move into two parts.

As before, we wait until the manipulators have cleared the workspace before performing another operation.

CNOT: For CNOT, the idea is to reverse the direction of one input, and then at the moment it meets the other input, have two manipulators (speed -2 and 3) induce a CNOT operation (where the speed -1 particle controls the speed 1 particle). We have already seen how to move particles, and how to apply gates, so the only new problem is how to avoid interference between the manipulators of the two swaps and the manipulators which implement the CNOT.

If a speed -2 manipulator meets a speed 3 manipulator, it will implement a CNOT on the speed 1 and speed -1 layers of that cell. However, the CNOT does nothing unless there is a speed -1 particle, and the only speed -1 particle is the one we intend to use in the CNOT operation. Hence, we can ignore speed -2 manipulators.

The speed 3 manipulators, as we have already discussed, will intersect if the operations they implement are separated in time by a multiple of three. Fortunately, we only have three operations here: two particle reversals and a CNOT, so we can schedule these operations such that they do not interfere. In particular, this means that the initial position (relative to the other particles) of the control particle (of the CNOT), the final position of the control particle, and the position of the target particle must be distinct modulo 3. We may be required to move some of the particles around to accommodate this condition, but we have already seen how to do that.

We separate all operations by $O(n)$ time steps to ensure that manipulators from one operation leave the workspace entirely before the next operation, to avoid collisions within the workspace. Manipulators will inevitably meet outside the workspace, but there is no interaction unless there are at least three. It is difficult, if not impossible, to avoid having two manipulators intersect, but whenever *three* manipulators intersect, we can always postpone the last of the three operations (corresponding to the manipulators) to avoid the collision. □

Next we consider the problem of capturing the remains of X, once it has reached a depleted configuration.

Theorem 4. *Let $X \subseteq \mathbb{Z}$ be the region $X = \{1, \ldots, n\}$ and let Y be the complement. There is some configuration $y \in \mathcal{Q}(Y)$ and some time t such that if we let $x \otimes y$ evolve for t time steps, only the speed 1 layer, in even cells, depends on x. In other words, the information from X is contained in the speed 1 layer of even cells.*

Proof. First, Theorem 2 tells us the particles in X will separate out into six groups by speed, with the fastest left-moving particles on the far left and the fastest right-moving particles on the far right, in time $O(n)$. We will show how to change the speed of each group to 1.

Consider the group of speed 3 particles. The way we change the speed (but not direction) of a particle is by intercepting it with three manipulators (one of each speed) moving in the opposite direction. Applied to a speed 3 particle, these manipulators will reduce its to speed 1. We manipulate particles from back to front, so that the speed 1 particles fall behind the speed 3 particles, instead of being overtaken. We also leave plenty of time between manipulations to ensure that two manipulators ever intercept a particle, and three manipulators never intersect, except at the planned times and locations of manipulations.

For speed 2 particles, we manipulate each particle to have speed 3, reducing the problem to one we have already seen. This time we order the manipulations from front to back so that the new speed 3 particles do not overtake the old speed 2 particles. Similarly, we can convert speed -3 or speed -2 particles to speed -1.

The final step is to reverse the speed -1 particles. We saw how to do this in Theorem 3: two manipulators (speed 3 and -3) collide to swap the speed -1 layer with the speed 1 layer. As before, we can avoid unintended collisions between manipulators if we perform the manipulations in the right order, and with sufficient time between them.

Now if some particles of speed 1 lie in odd cells, then use further manipulations to increase their speed back to 2, breaking parity, and allowing us to maneuver the particle to an even cell (more accurately, an even cell in even time steps, an odd cell in odd time steps). Then we increase its speed to 3, and back to 1 again, but in an even cell. □

To finish the proof of physical univesrality, we need to show how to output the computed configuration. This means taking a collection of particles of speed 1, and dividing them into six groups. Then we accelerate each group to a different speed, and aim them at the region X. We assume that the particles output by the computational phase (i.e., the collection of particles of speed 1) were computed to account for the interactions that occur when the particles come together in region X, so they produce the desired output, namely U (or a close approximation) applied to the initial contents of X.

Observe that our LQCA is reversible, so we can run it backwards. Furthermore, it is close to being the same automaton in reverse – the case which allows us to change speed 1 to speed 2 to speed 3 cycles in the opposite direction, gates are inverted (of course, H and CNOT are their own inverses), and particles move backwards, but the automaton is close enough that Theorem 2 and Theorem 4 go through. Hence, we can construct a configuration (in the reverse QCA) which takes the contents of the region X, reduces their speed, and collects them together. The manipulators do not depend on the contents of X, so we can compute their positions at the end (again, in the reverse QCA) of this construction. Now let us run these manipulators forward in the (forward) LQCA.

The manipulators take a group of speed 1 particles and force them into an output region, which is exactly what we want!

In summary, we claim that the LQCA defined earlier is physically universal. Given a region X and a unitary transformation U on X, we construct a circuit (of T, H and CNOT gates) to implement some U' such that $\|U - U'\|_{tr} \leq \epsilon$. Then we implement U' in the LQCA in three steps:

- We extract the data initially in X by letting it reach a depleted configuration, and rounding up the particles that escape.
- We decode the initial configuration from the particles, apply the circuit for U', and encode the desired configuration as a collection of particles.
- We aim the particles at region X, and wait for them to interact in X and produce the transformed output.

With sufficient time between these three phases, we can avoid collisions between the manipulators. This concludes the proof of our main result.

Theorem 5. *The LQCA described at the beginning of the section is physically universal.*

In fact, the LQCA achieves a stronger definition of physical universality where the program configuration is not allowed to depend on ϵ. In other words, a single configuration implements arbitrarily good approximations of U if we let it run longer.

Definition 9. *Let M be a QCA on a lattice L. Then M is* strongly physically universal *if for every finite set of cells $X \subseteq L$, and every unitary transformation $U \colon \mathcal{Q}(X) \to \mathcal{Q}(X)$ there exists $y \in \mathcal{Q}(L \backslash X)$ such that for any $\epsilon > 0$ there exists a time $t \in \mathbb{Z}$ such that y ϵ-approximately implements the transformation U on X in time t.*

Corollary 1. *The LQCA described at the beginning of the section is strongly physically universal.*

Proof. Suppose the input region is X_0 and the unitary is U_0. Let $(\epsilon_i)_{i=0}^{\infty}$ be a sequence of positive real numbers tending to zero. By physical universality, there is a configuration y_0 in $\mathcal{Q}(L \backslash X_0)$ and time t_0 such that y ϵ_0-approximately implements the transformation U_0 on X_0 in time t_0.

Now iteratively build programs on larger and larger regions. In general, let X_{i+1} be a region containing X_i plus $3t_i$ cells on either side, and all non-quiescent cells in y_i. This region is large enough that no particle outside it can possibly affect X_0 in t_0 steps. Apply physical universality to X_{i+1} with error $\epsilon_{i+1} > 0$ and unitary U_{i+1}, where U_{i+1} applies U_i to the region X_i, and the identity transformation to the cells in $X_{i+1} \backslash X_i$. Physical universality gives us a program $y_{i+1} \in \mathcal{Q}(L \backslash X_{i+1})$ and time t_{i+1}.

Finally, combine the y_i configurations into a single large configuration $y \in \mathcal{Q}(L \backslash X_0)$. Given an $\epsilon > 0$, find some $\epsilon_i < \epsilon$ and let y run for t_i time steps. This is just enough time for the program y_i to execute, but not enough time for the later programs to interfere, so we get an $\epsilon_i < \epsilon$ approximation of U applied to X_0. □

7 Future Work

- We leave the time and space complexity of this cellular automaton open for analysis. Can we quantify the performance of the construction in the proof of strong physical universality?
- The automaton is not as simple or aesthetically pleasing as its classical counterparts. Can we construct a less obviously artificial LQCA in more dimensions?
- Is there a notion of physical universality for unbounded computations on a quantum Turing machine?

References

1. Dawson, C.M., Nielsen, M.A.: The Solovay-Kitaev algorithm. Quantum Info. Comput. **6**(1), 81–95 (2006)
2. Janzing, D.: Is there a physically universal cellular automaton or Hamiltonian? (2010). http://arxiv.org/abs/1009.1720
3. Nielsen, M.A., Chuang, I.L.: Programmable quantum gate arrays. Phys. Rev. Lett. **79**, 321 (1997)
4. Raussendorf, R.: Quantum cellular automaton for universal quantum computation. Phys. Rev. A **72**, 022301 (2005)
5. Salo, V., Törmä, I.: A one-dimensional physically universal cellular automaton. Personal Communication (2014)
6. Schaeffer, L.: A physically universal cellular automaton. In: Roughgarden, T. (ed.) Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science, pp. 237–246. ACM, Rehovot (2015)
7. van Dam, W.: A universal quantum cellular automaton. In: Proceedings of PhysComp96, pp. 323–331. InterJournal (1996)
8. Watrous, J.: On one-dimensional quantum cellular automata. In: 36th Annual Symposium on Foundations of Computer Science, pp. 528–537. Society Press (1995)
9. Youssef, S.: Quantum Mechanics as Bayesian Complex Probability Theory. Modern Physics Letters A **9**, 2571–2586 (1994)