

General Computation using Slidable Tiles with Deterministic Global Forces*

Alberto Avila-Jimenez¹, David Barreda¹, Sarah-Laurie Evans¹, Austin Luchsinger¹, Aiden Massie¹, Robert Schweller¹, Evan Tomai¹, and Tim Wylie¹

¹University of Texas Rio Grande Valley

Abstract

We study the computational power of the Full-Tilt model of motion planning, where slidable polyominoes are moved maximally around a board by way of a sequence of directional “tilts.” We focus on the deterministic scenario in which the tilts constitute a repeated clockwise rotation. We show that general-purpose computation is possible within this framework by providing a direct and efficient simulation of space-bounded Turing machines in which one computational step of the machine is simulated per $O(1)$ rotations. We further show that the initial tape of the machine can be programmed by an initial tilt-sequence preceding the rotations. This result immediately implies new PSPACE-completeness results for the well-studied problems of *occupancy* (deciding if a given board location can be occupied by a tile), *vacancy* (deciding if a location can be emptied), *relocation* (deciding if a tile can be moved from one location to another), and *reconfiguration* (can a given board configuration be reconfigured into a second given configuration) that hold even for deterministically repeating tilt cycles such as rotations. All of our PSPACE-completeness results hold even when there is only a single domino in the system beyond singleton tiles. Following, we show that these results work in the Single-Step tilt model for larger constant cycles. We then investigate computational efficiency by showing a modification to implement a two-tape Turing machine in the Full-Tilt model and Systolic Arrays in the Single-Step model. Finally, we show a cyclic implementation for tilt-efficient Threshold Circuits.

1 Introduction

The “tilt” motion-planning model [8] is an elegant and simple model of multi-agent motion planning where polyominoes are moved based on external global controls such as gravity, magnetic fields, or light. The model consists of a 2D grid board with a mix of fixed immovable walls and a collection of slidable polyominoes that move maximally based on directional *tilt* commands. By applying sequences of such directional tilts, polyominoes can be rearranged into different permutations, possibly simulating computation. If glues are added, the initial polyominoes can combine into larger shapes, and the tilt framework serves as a model of self-assembly. In this paper, we focus on one of the simplest variants of this model where the tilt sequence is a simple repeated clockwise rotation (a *rotational* tilt sequence).

Motivation and Previous Work. The study of Tilt is motivated by the fact that this extremely simple and natural model of global control has extensive depth. For example, in terms of computation, dual-rail logic devices and binary counters have been designed [6]. In terms of complexity, PSPACE-completeness has been shown for various natural problems such as finding the minimum number of tilts to reconfigure a board [6], or deciding if a configuration is reachable [4, 5] at all. In terms of self-assembly, work has been done on designing boards to efficiently self-assemble target shapes in a pipelined manner [7], in parallel [29], in

*This research was supported in part by National Science Foundation Grant CCF-2329918.

Model	Tilt Seq.	Relocation	Occupancy	Vacancy	Reconfig.	Poly. Size	Poly. > 1 Count	Ref.
FT	Det.	PSPACE-C	PSPACE-C	PSPACE-C	PSPACE-C	2	1	Thm. 2
	General	PSPACE-C	PSPACE-C	PSPACE-C*	PSPACE-C	1	0	[4, 14, 18]
SS	Det.	PSPACE-C	PSPACE-C	PSPACE-C	PSPACE-C	2	1	Cor. 2
	General	PSPACE-C	PSPACE-C	PSPACE-C*	PSPACE-C	1	0	[4, 14, 18]

Table 1: Summary of our results for the problems of relocation, occupancy, vacancy, and reconfiguration for deterministically repeating tilt cycles. The new results are in bold. The dash lines follow from the other results. “Poly. Size” is the maximum size of any polyomino, and “Poly. > 1 Count” is the number of polyominos in the system that are not singletons. *This has not been formally proven, but was conjectured [18] and is likely true based off [14].

Model	Cycle Length	Single Tape TM	Multi-Tape TM	Ref.	Cycle Length	Threshold Circuits	Ref.
Full-Tilt	4	$O(1)$	$O(1)$	Thms. 1, 3	4	$O(d)$	Thm. 4
Single-Step	280	$O(1)$	$O(T)$	Cors. 1, 3	6	$O(dw)$	Cor. 5

Table 2: Summary of deterministic computation results related to other models in terms of the cycles required for simulation. T refers to the length of the tape of the Turing Machine. For circuits, d and w are the depth and width of the circuit, respectively.

3D [20], and in a *universal* manner where a single board can be programmed by a tilt sequence to construct a selected shape or pattern from a general class [4, 5]. Despite its power, the simplicity of global control permits applications at the macro, micro, and nano scale, including examples of large population robot swarms ranging from naturally occurring magnetotactic bacteria [19, 22, 23] to manufactured light-driven nanocars [16, 30].

1.1 Our Focus and Results

Our focus is on a limited version of Tilt models in which the tilt sequences are a fixed, deterministic cycle of commands. Specifically, we utilize what may be the simplest such sequence: a repeated clockwise rotation. Previous complexity results no longer apply in this scenario, and the complexity and power of this limited variant of the model has remained open until now. We present three main results: the efficient simulation of space-bounded Turing machines, the PSPACE-completeness of classic problems in the model, and the efficient simulation of threshold circuits. An overview of these results is presented in Tables 1 and 2.

Turing Machine Simulation. Our first result is an efficient simulation of any given space-bounded Turing machine using a rotational tilt sequence (Section 3). The size of the board is polynomial in the size of tape and the states of the Turing machine, and simulates one computational step every $O(1)$ rotations. We further show that the tape of the Turing machine can be programmed by a preliminary tilt-sequence preceding the rotational sequence used to run the machine. While previous work has implemented key components for computation within deterministic tilt-cycles (e.g., dual-rail logic and binary counters [6]), this is the first result to show that the fullest possible computational power of this model variant is realizable. The direct and efficient simulation of Turing machines further provides a plausible framework for rotational Tilt systems to be implemented as general-purpose computational devices.

Complexity: Relocation, Occupancy, Vacancy, Reconfiguration. Our next set of results resolves the complexity of classic problems previously studied within the Tilt model (Section 3.4). We focus on the problems of *relocation* (deciding if a tile can be moved from one location to another), *occupancy* (deciding

if a given board location can be occupied by a tile), *vacancy* (deciding if a location can be emptied), and *reconfiguration* (can a given board configuration be reconfigured into a second given configuration). In the case of general tilt sequences, these problems have been shown to be PSPACE-complete [4, 5], but their complexity for deterministic sequences was an open problem. Our Turing machine simulation, along with special halting configurations, implies PSPACE-completeness for relocation, reconfiguration, occupancy, and vacancy even when there is only a single domino along with singletons in the system. A summary of our complexity results is given in Table 1.

Threshold Circuits. Along with Turing machines, circuits represent one of the most fundamental models of computation. Previous work has shown how rotational Full-Tilt systems can simulate arbitrary circuits consisting of ANDs and ORs with a rotation count on the order of the circuit depth [6]. We expand on this progress by adding the efficient simulation of the *majority* gate that determines which bit value has the highest representation among a given set of input bits (Section 5). This extension yields the efficient simulation of *Threshold* circuits, a provably faster set of circuits that is of particular interest to applications such as machine learning.

1.2 Additional Related Work.

Full-Tilt reconfiguration, even with almost no geometry, remains capable of non-trivial reconfiguration as studied in [2] with the game of 2048. In terms of shape production, heuristic algorithms for planning the assembly of target shapes have been explored [9]. Closely related to the Full-Tilt model is the *Single-Step* model of global control [14, 15] where each polyomino only moves a single step with each command, instead of maximally. [14] and [18] show that Single-Step can be the dual of Full-Tilt, implying a connection in the complexity between the two models for some classic problems. But for some problems, such as shape production [1, 3], the problem is NP-hard even without any real geometry. Other problems, such as relocation, are NP-hard even with only two possible directions for monotone [13], or even no [15], geometry. The efficient assembly of shapes [11] and patterns [12] has also been explored.

2 Preliminaries

Board. A *board* (or *workspace*) is a rectangular region of the 2D square lattice in which specific locations are marked as *blocked*. Formally, an $m \times n$ board is a partition $B = (F, X)$ of $\{(x, y) | x \in \{1, 2, \dots, m\}, y \in \{1, 2, \dots, n\}\}$ where F denotes a set of *open* locations, and X denotes a set of *blocked* locations. We informally refer to blocked locations as “concrete.”

Tiles, Polyominos, and Configurations. A tile is a labeled unit square centered on a non-blocked point on a given board. Since we will not use any dynamic attachment between existing tiles, we simplify our definition. Formally, a tile is simply a coordinate $c = (x, y)$ on the board. A *polyomino* is a finite set of non-overlapping tiles $p = \{t_1, \dots, t_k\}$ that is connected with respect to the coordinates of the tiles. A polyomino that consists of a single tile is informally referred to as a “tile.” A configuration $C = (B, P)$ is an arrangement of polyominos $P = \{p_1, \dots, p_k\}$ on a board B such that there are no overlaps among polyominos, or with blocked board spaces.

Step. A *step* is a way to turn one configuration into another by way of a global signal that moves all tiles/polyominos in a configuration one unit in a direction $\delta \in \{u, r, d, l\}$ (up, right, down, left) when possible without causing an overlap with a blocked position or another tile. Formally, for a configuration $C = (B, P)$, let P' be the maximal subset of P such that translation of all polyominos in P' by 1 unit in the direction d induces no overlap with blocked squares or other polyominos. A step in direction δ is performed by executing the translation of all polyominos in P' by 1 unit in that direction. If a configuration does not change under a step transition for direction δ , we say the configuration is δ -terminal.

Tilt. A *tilt* in direction $\delta \in \{u, r, d, l\}$ for a configuration is executed by repeatedly applying a step in direction $\delta \in \{u, r, d, l\}$ until a δ -terminal configuration is reached. We say that a configuration C can be *reconfigured in one move* into configuration C' (denoted $C \rightarrow_1 C'$) if applying one tilt in some direction δ

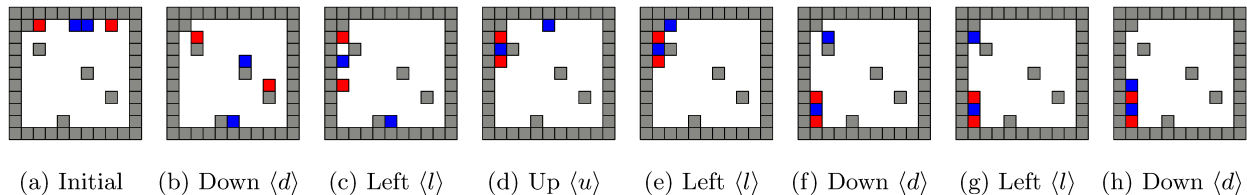


Figure 1: Full-Tilt example of tiles moved through several tilts.

to C results in C' . We define the relation \rightarrow_* to be the transitive closure of \rightarrow_1 . Therefore, $C \rightarrow_* C'$ means that C can be reconfigured into C' through a sequence of tilts.

Tilt Sequence. A *tilt sequence* is a series of tilts inferred from a series of directions $D = \langle d_1, d_2, \dots, d_k \rangle$; each $d_i \in D$ implies a tilt in that direction. For simplicity, when discussing a tilt sequence, we just refer to the series of directions from which that sequence was derived. Given a starting configuration, a tilt sequence corresponds to a sequence of configurations based on the tilt transformation. An example tilt sequence $\langle d, l, u, l, d, l, d \rangle$ and the corresponding sequence of configurations can be seen in Figure 1. A tilt sequence is termed *deterministic* if it consists of repeated applications for a fixed constant-size cycle of tilt instructions. Our focus is a deterministic tilt sequence that is a *rotational* sequence, which is the repeated application of the clockwise rotation $\langle u, r, d, l \rangle$.

Turing Machine. We use a Turing machine (TM) defined as a 7-tuple $(Q, \Sigma, \Gamma, \delta_U, q_0, q_a, q_r)$, where Q is the set of states, and $\Sigma = \Gamma = \{0, 1\}$ for the tape and language alphabets, and q_0, q_a, q_r are the start, accept, and reject states, respectively. For a universal machine, we use a 15-state, 2-symbol tape, as given in [26], which defines the transition function δ_U . We let Q follow zero-based indexing. We assume a bounded tape of length n .

3 Space-Bounded Turing Machine Simulation

In this section, we detail how a rotational tilt system can efficiently simulate any space-bounded Turing machine. The initial simulation is described in Section 3.1, and in Section 3.6 we show how the Turing machine's initial tape can be programmed by a tilt-sequence that precedes the rotational sequence that runs the machine. Finally, we expand the construction to a two-tape Turing machine in Section 4. Our simulations are *efficient* in that they simulate one step of the Turing machine per $O(1)$ rotations. As an implication of our space-bounded Turing machine simulation, we show in Theorem 2 that the previously studied problems [4, 5, 18] of *occupancy* (will a square on a board eventually get filled), *vacancy* (will a square eventually be empty), *relocation* (can a given tile go from one location to another), and *reconfiguration* (can all tiles be located to some set positions) are all PSPACE-complete even in the case of deterministic rotational sequences. While each tilt application is defined as an application of single steps, for the purpose of these problems, we only care if the configuration is reached after a full application of the tilt, i.e., sliding through a target location during a tilt does not count as occupying that location. Here, we establish the following.

Theorem 1. *For any Turing machine \mathcal{M} with $s = |Q|$ states and a bounded tape of length n , there exists a non-bonding rotational Full-Tilt simulation of the machine with board size $O(ns^3)$ that simulates the machine at a rate of one step per $O(1)$ rotations.*

By *simulation*, we mean there exists a mapping from board configurations to bit-configurations of the Turing machine tape, with specific $O(1)$ size sub-regions of the board specifying these bit values. A simulation then requires that the dynamics of the changing board configuration match the dynamics of the corresponding bit configuration for the Turing machine that is being simulated. Our simulation board size is polynomial in the tape size and states of the Turing machine, uses $O(1)$ tilts per computation step, and is *non-bonding*.

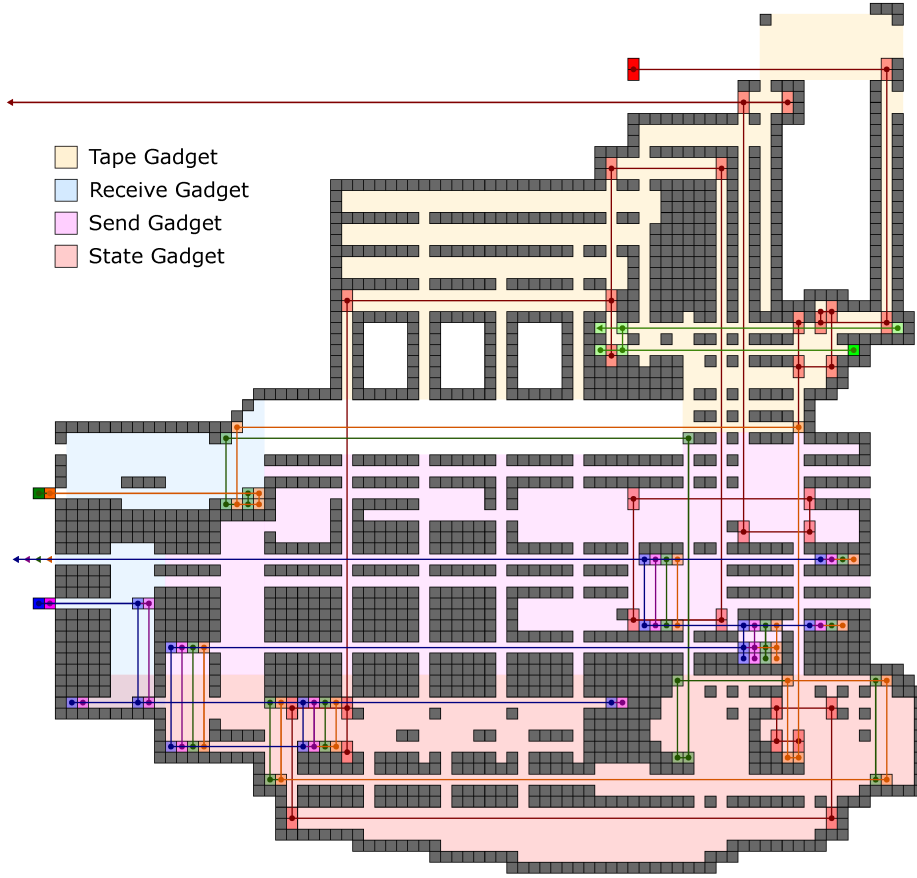


Figure 2: A **State-Cell** construction (a single cell of the tape and the TM state control) with three states (including halt) performing an execution. The process starts with the head-domino entering the **State-Cell** from the left at the dark red domino at the top of the board. In the same cycle, at the leftmost blue and purple tiles, the inert-singletons enter. One cycle later, at the leftmost green and orange tiles, the active-singletons also enter. The head-domino flows through the *Tape gadget* reading 1, *State gadget* setting q_0 , *Tape gadget* writing 1, then sends the 4 inert-singletons left in the *Send gadget* before finally being sent left itself.

Design. Typically, we envision a TM machine as a single control state with an infinite tape. In this model, our control structures are built from blocked locations (concrete) that are stationary, so the data must be sent from the current tape location to the control states. This is easily done in the Full-Tilt model, but we also need to keep track of the tape cell being used to send the data back. This is simple with two dominos (one to stay at the correct tape-cell location, and the other to handle state change), but is tricky with only a single domino. Further, no single-step implementation could be efficient since any given step might have an unbounded distance to traverse on the tape.

Here, we integrate the tape and control into a single structure where the control is repeated at every tape cell (overview in Figure 2). We can do this efficiently because a 2-symbol universal TM is possible with only 15 states [26]. Thus, the state control is constant size, and the controlling data is simply passed left or right with the head to process the data on the tape. This also allows for the single-step model to be universal for a constant cycle size since it now has a constant bounded distance to travel at any point.

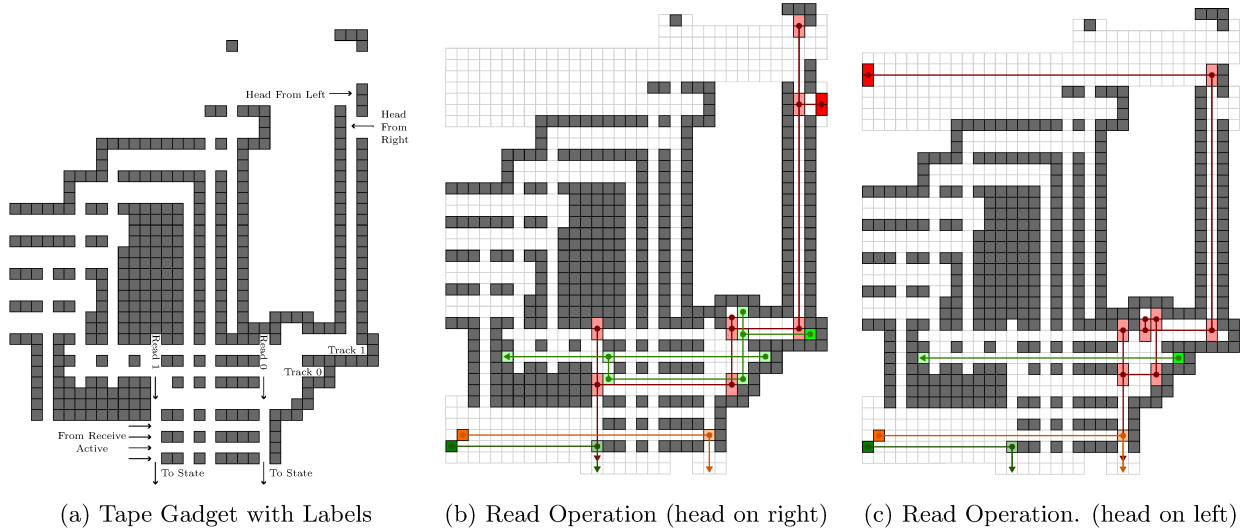


Figure 3: (a) The *Tape gadget* construction, with labeled regions. (b-c) A *read operation* performed in the *Tape gadget*. Observe how the placement of the data-singleton affects where the head-domino exits at the end of the operation.

Programming. From an implementation standpoint, programming a TM by placing particles on the tape could be difficult at a micro or nanoscale. We also discuss how the tape can easily be programmed by deviating from the deterministic cycle $O(n)$ or $O(\log n)$ times, where n is the length of the tape. The linear programming is constant height, while the logarithmic method requires the tilt board to have $O(\log n)$ additional height.

Extensions. After presenting the single-tape machine, we show efficiency extensions. For Full-Tilt, we show how a 2-tape TM can be built with multiple dominos that are needed to communicate between the two tapes. For Single-Step, since the head and processing is local to the tape cell, we can extend the single-tape implementation to linear systolic arrays [10, 21, 24] with a single domino for each tape head. The Full-Tilt model can also do systolic arrays on the single-tape TM, but the Single-Step model can not efficiently implement the 2-tape TM due to the distance required for the two tape heads to interact.

3.1 Simulation Construction

Here, we provide the construction of our simulation of a single-tape space-bounded Turing machine. Given a space-bounded Turing machine $\mathcal{M} = \{Q, \Sigma, \Gamma, \delta_U, q_0, q_a, q_r\}$ and an input tape w of size n , we construct a board B that simulates \mathcal{M} over w . B consists of n copies of a **State-Cell** that represents a specific tape cell on w and a set of particles that model the behavior of \mathcal{M} .

Particle Functions. The set of particles consists of $2|Q| + n$ singletons and one domino. They fall into a few categories based on function.

- Head-domino. Implements the TM functionality and tracks the tape head location.
- State-singletons. The singletons collectively represent \mathcal{M} 's current state based on the number of singletons used. At all points, the state-singletons are divided into *active-singletons* and *inert-singletons*.
 - Active-singletons. Tiles that collectively encode the current state of the Turing machine \mathcal{M} . Specifically, the number of active-singletons is equal to $2i$ where $q_i \in Q$ is the current state of the Turing machine.

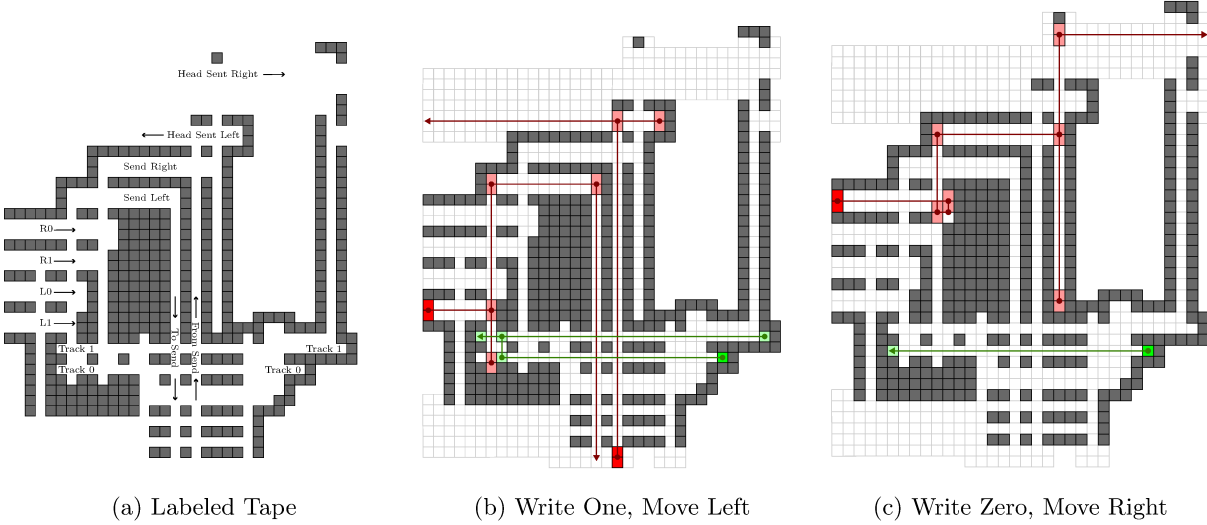


Figure 4: (a) The *Tape gadget* construction with the regions used for write labeled. (b) A write $L1$ operation performed in the *Tape gadget*. The head-domino moves the data-singleton from 0 to 1, then moves into the *Send gadget*. It returns and is sent left. (c) A write $R0$ operation performed in the *Tape gadget*. The head-domino does not interfere with the data-singleton, then is sent right.

- Inert-singletons. Any state-singleton that is not active is inert, and is stored in case the number of active-singletons needs to be increased.
- Data-singletons. Each *Tape gadget* has a singleton that is either in the ‘0’ or ‘1’ holding row for a dual rail implementation of the current symbol.

State-Cell. Each **State-Cell** is composed of four gadgets: the *Tape gadget*, *State gadget*, *Send gadget*, and *Receive gadget*. See Figure 2.

- *Tape Gadget* (Sec. 3.2.1). The *Tape gadget* holds the **State-Cell**’s data-singleton, whose location encodes the **State-Cell**’s symbol. The gadget’s operations are Read and Write. Read sends the head-domino to the same **State-Cell**’s *State gadget* via one of two paths, which is determined by the location of the *Tape gadget*’s data-singleton. Write receives the head-domino at one of four locations. The location determines where the data-singleton is placed, and which direction the State-singletons and head-domino are sent.
- *State Gadget* (Sec. 3.2.2). The *State gadget* serves the same purpose as the transition function δ_U of \mathcal{M} . It takes two pieces of information as its input: the holding row the *Tape gadget*’s data-singleton was in and the current count of the active-singletons. According to what δ_U maps the input to, the *gadget* updates the active-singleton count and selects which path the head-domino will return to the *Tape gadget* on.
- *Send Gadget* (Sec. 3.2.3). The *Send gadget* moves the state-singletons out of the cell construct to the left or right, according to the position of the head-domino.
- *Receive Gadget* (Sec. 3.2.4). The *Receive gadget* catches the state-singletons from another cell construct and directs them to the *Tape gadget* and *State gadget*.

Together, these gadgets receive the head-domino and state-singletons from another **State-Cell**. They read the data-singleton’s location, and based on what was read and the number of active-singletons, the gadgets update the count of the active-singletons, re-position the data-singleton, and move the head-domino and state-singletons to an adjacent **State-Cell**.

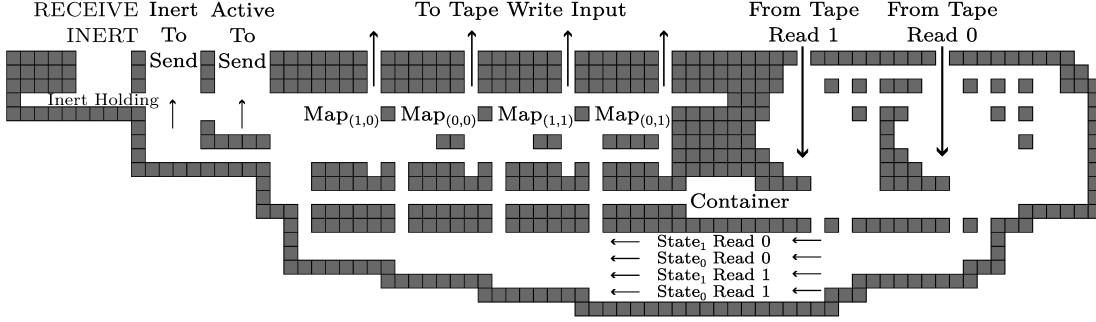


Figure 5: State Labeled. According to the State: x , and the Symbol read: y , the head-domino is sent left at $\text{STATE}_x \text{ READ } y$ (details in Figure 6). Or, if q_x is a halt state, the head-domino is sent into CONTAINER (details in Figure 8). Each $\text{STATE}_x \text{ READ } y$ path leads to $\text{MAP}_{(x,y)}$ where the state is updated (details in Figure 7). The new active-singletons and inert-singletons are sent to the *Send gadget*, and the head-domino is sent to one of the *Tape gadget's* write inputs.

Geometric Alternating Cell Types. The **State-Cell** constructions are split into two types. Due to geometrical constraints, the location from which a **State-Cell** sends the head-domino to their right neighboring **State-Cell's** *Tape gadget* is to the left of the location where they will receive the head-domino from their left neighboring **State-Cell's** *Tape gadget*. Because these locations serve different purposes, they cannot be at the same height. However, the location where a cell receives the head-domino from the left must be vertically aligned with the location in the leftward adjacent cell from which the head-domino is sent right. Therefore, **State-Cell** constructions alternate between sending the head-domino high and low. The send high constructions have the location from which they will send the head-domino right placed above the location where they will receive head-domino from the left. The send low constructions have the opposite design. Thus, by alternating the constructions, the location where the head-domino is sent right in any given **State-Cell** will be at the same height as the location its right neighbor receives the head-domino from the left. This is also applied for the active-singletons and inert-singletons in the *Send gadget*. All figures for this paper are of the **SEND HIGH**-type for simplicity.

3.2 Gadgets

We now discuss the gadgets in detail, and then walk through an execution cycle.

3.2.1 Tape Gadget

Read Operation. The *read operation* begins upon the head-domino entering the *Tape gadget*, shown in Figure 3a, from another **State-Cell's** *Tape gadget*. Regardless of whether the head-domino arrives at the **FROM LEFT** or **FROM RIGHT** region, it traverses to **TRACK 1**. Then, according to the placement of the data-singleton, the head-domino is placed at **READ 1** or **READ 0**. If the data-singleton is in **TRACK 1**, then it will interrupt the head-domino's path and cause it to arrive at **READ 1**. In the process of interrupting, the data-singleton is moved to **TRACK 0**. Otherwise, the singleton is in **TRACK 0**, and the head-domino will be placed at **READ 0** without affecting the singleton. In either case, after the head-domino is at one of the **READ** locations, the active-singletons arrive from the *Receive gadget* with half stopping under **READ 1**, and half under **READ 0**. Finally, the head-domino will traverse down its chosen **READ** with half the active-singletons below it, while the rest of the active-singletons traverse down the other **READ**. Examples of the *read operation* are shown in Figures 3.

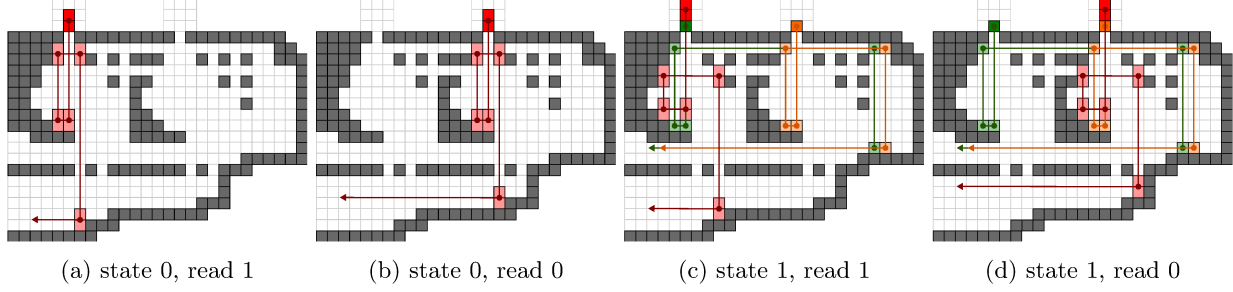


Figure 6: The four possible non-halt comparisons in the 3 state *State gadget*. Depending on the initial location of the head-domino and the count of active-singletons, different heights are traversed by the head-domino. Recall that the number of active-singletons encodes the state: q_x and the initial location of the head-domino indicates the **State-Cell**'s symbol: y . Given that, the height traversed represents (q_x, y) , the input to \mathcal{M} 's transition function.

Write Operation. The *write operation* begins when the head-domino is returned to the *Tape gadget* from the same **State-Cell**'s *State gadget*. The head-domino returns at one of four locations: $R0$, $R1$, $L0$, or $L1$, shown in Figure 4a. Each location maps to a direction-value combination, which dictates how the head-domino moves and edits the data-singleton. For example, $L1$ has the direction-value combination Left-1. If the write operation receives the head-domino from that location, it writes 1 and moves the head-domino left.

Recall that after the read operation, the data-singleton is placed in TRACK 0. To write 1, the head-domino blocks TRACK 0, sending the data-singleton to TRACK 1. To write 0, because the read operation leaves 0 written, the head-domino does not enter either of the TRACKS and instead waits one cycle. After writing, the head-domino traverses to either MOVE LEFT or MOVE RIGHT. MOVE LEFT sends the head-domino into the *Send gadget*, which will return it 2 cycles later. Upon returning, the head-domino is sent to the left neighboring **State-Cell**. MOVE RIGHT delays the head-domino once, then sends it to the right neighboring **State-Cell**.

3.2.2 State Gadget

The *State Gadget* receives half the active-singletons through FROM TAPE READ 1 and the other half through FROM TAPE READ 0 (Figure 5). In the same tilt, the head-domino is placed atop one of the halves. The path the head-domino takes forward is dependent on both the number of active-singletons offsetting it, which we define as x , and that FROM TAPE READ y it entered through. If q_x is one of the Turing machine's halt states, then the head-domino stops at the height of the active-singletons. The next left tilt locks the active-singletons and the head-domino in the container. An example is shown in Figure 8. Otherwise, the head-domino traverses left across the bottom section of the *State gadget* at the height STATE_{q_x} , READ y . The height is set by the process in Figure 6. Meanwhile, the active-singletons are positioned to merge with the inert singletons, which have been cycling at WAIT. Next, the geometry of $\text{MAP}_{x,y}$ sets a new count of active-singletons equal to $2 \times a$ for $(q_x, y) \rightarrow (q_a, b, c)$ (see Figure 7). The new active and inert-singletons then traverse separately to the *Send gadget*, while the head-domino is sent to $R0$, $R1$, $L0$, or $L1$ (from Figure 4a) according to which move-write combination δ_U maps (q_x, y) to.

3.2.3 Send Gadget

Upon receiving the active-singletons and the inert-singletons from the *State gadget*, the *Send gadget* must delay them until the *Tape gadget*'s write operation might send the head-domino down. If the *Tape gadget* received the head-domino at $L1$ or $L0$, then the head-domino enters the *Send gadget* to the right of LEFT INDICATOR₁. After the delay, if the head-domino is present, the *gadget* sends the state-singletons left via the process in Figure 10c. Otherwise, they are sent right, shown in 10b.

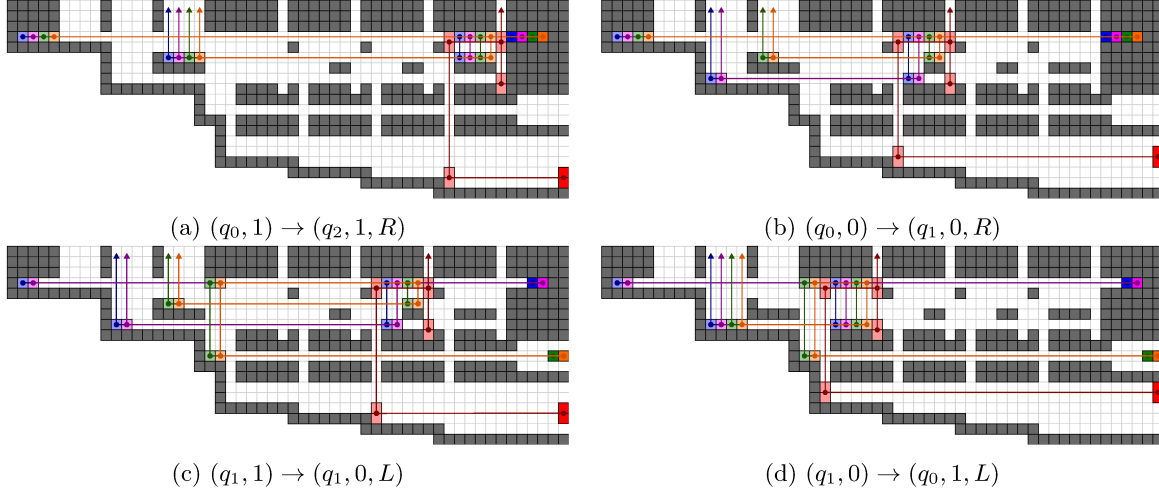


Figure 7: The 4 possible non-halt $\text{MAP}_{(x,y)}$ options in a 3-state *State gadget*. The head-domino enters this section at a height indicating the input to \mathcal{M} 's transition function. Traversing from each height places the head-domino above a unique $\text{MAP}_{(x,y)}$. According to the transition function $(q_x, y) \rightarrow (q_a, b, c)$, the geometry under $\text{MAP}_{(x,y)}$ will set the number of active-singletons to $2 \times a$. (a) sets 4 active-singletons (b) sets 2 active-singletons (c) sets 2 active-singletons (d) sets 0 active-singletons. In addition, the tunnel the head-domino exits $\text{MAP}_{(x,y)}$ from is connected to the cb input for the *Tape gadget* write operation. If curious, in Figure 2, we can see the specific cb tunnel connections. They are: (a) goes to $R1$, (b) goes to $R0$, (c) goes to $L0$ and (d) goes to $L1$.

3.2.4 Receive Gadget

The *Receive gadget* directs the active and inert-singletons at the beginning of each execution. The inert-singletons are received from INERT FROM RIGHT or INERT FROM LEFT as shown in Figure 11a. They are moved to WAIT in the *State gadget* where they will cycle until accessed by the head-domino. If the active-singletons are received from ACTIVE FROM LEFT, they are set to different heights, then sent to the *Tape gadget*. If they are received from ACTIVE FROM RIGHT, they are first delayed twice at DELAY 1 and 2, then likewise have their heights changed and are sent to the *Tape gadget*. The FROM LEFT and FROM RIGHT cases are shown in Figures 11b and 11c, respectively.

3.3 Execution Walkthrough

Here, we walk through a single execution cycle shown in Figure 2, focusing on how the gadgets interact. Two inert-singletons enter the *Receive gadget* at FROM LEFT SEND INERT. The head-domino enters the *Tape gadget* at HEAD FROM LEFT, which begins the *read operation*. One tilt cycle later, during the *read operation*, the two active-singletons arrive at FROM LEFT SEND ACTIVE in the *Receive gadget*, and the inert-singletons are sent to WAIT in the *State gadget*. The active-singletons are delayed one cycle to align with the *read operation*, then are sent from the *Receive gadget* to the *Tape gadget*. At this point, the *read operation* finishes, positioning the head-domino in READ 0 with one active-singleton below it, and the other below READ 1.

The head-domino and active-singletons traverse down to the *State gadget*. The *State gadget* receives as input: 1 active-singletons per TAPE READ, and the head-domino through FROM TAPE READ 0. The Turing Machine's transition function δ_U dictates that $(q_1, 0) \rightarrow (q_0, 1, L)$. Thus, in the simulation, we see the head-domino sent to $\text{MAP}_{(1,0)}$ where the active-singleton count is set to 0. The four inert-singletons are sent to the *Send gadget*, and as δ_U orders, the head-domino is sent to tunnel $L1$ in the *Tape gadget*.

In the *Send gadget*, the inert-singletons are delayed. Because the *Tape gadget* received the head-domino

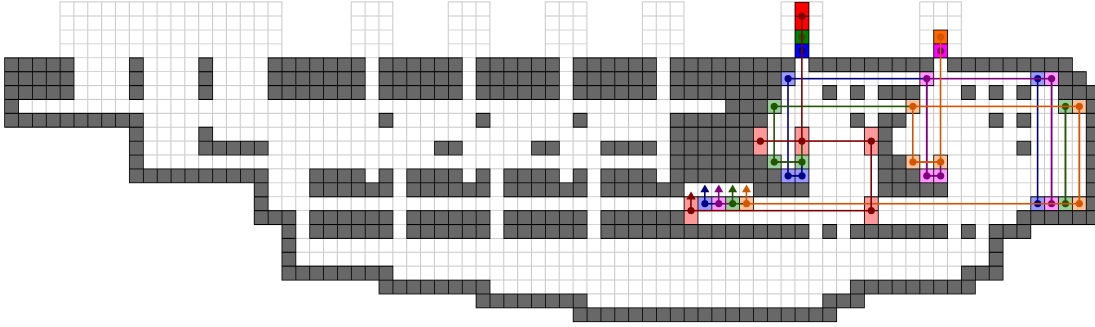


Figure 8: Simulating the Turing Machine Halting. The Halt States are encoded as all state-singletons being active. Instead of the head-domino moving down to a STATE_x READ y combination, it stops one below the active-singleton line. Upon the following leftward tilt, the head-domino catches the active-singletons, locking them, and itself, into the CONTAINER.

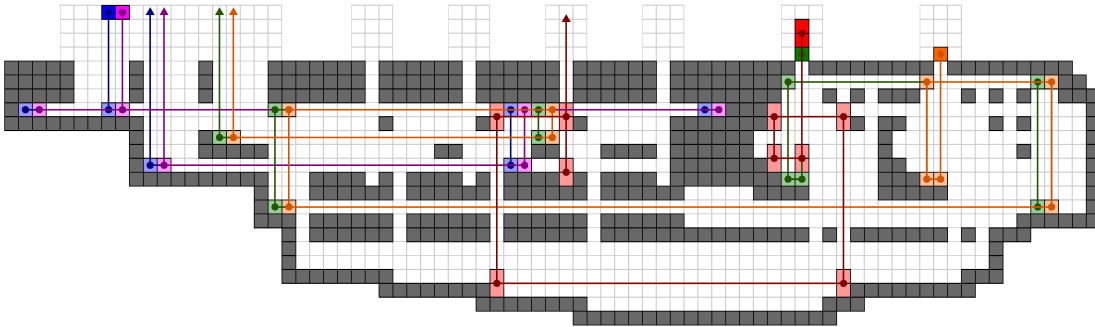


Figure 9: Example of full *State gadget* execution. The head-domino comes FROM TAPE READ 1. There is one active-singleton below it, indicating the simulation is in state q_1 . The head-domino goes to STATE_1 READ 1, and the two active-singletons join the two inert-singletons. The head-domino moves up, blocking the state-singletons at $\text{MAP}_{(1,1)}$. The Turing Machine's transition for $(1,1)$ is $(q_1, 1) \rightarrow (q_1, 0, L)$. In the *State gadget*, we can see the active-singleton count being set to 2 according to that transition. For the same reason, the existing head-domino traverses to $L0$.

from $L1$, it begins the *write operation* by setting the data-singleton to 1. The head-domino then traverses the MOVE LEFT tunnel to the *Send gadget*. There, the inert-singletons have finished delaying, so they check for the head-domino. Since it is present, they are redirected to INERT GOING LEFT, then are sent to the **State-Cell**'s left neighbor's *Receive gadget*. Meanwhile, the head-domino attempts to repeat the process of redirecting with the nonexistent *active-singletons*, then traverses back up into the *Tape gadget*, where it is sent to the **State-Cell**'s left neighbor's *Tape gadget*.

3.4 Complexity Results

The simulation described above does not solve the occupancy, relocation, vacancy, or reconfiguration problems. This is because they deal with specific locations on the board. Our simulation has a clear indicator for the Turing machine's halt states, but the indicator could be at any of the n **State-Cell** constructions. We make a small modification to all **State-Cells**, such that they eject the state-singletons when the halt state is reached (Figure 12a). From this and a few other small changes, we achieve the following results.

Theorem 2. *Relocation, reconfiguration, occupancy, and vacancy are PSPACE-complete in the deterministic*

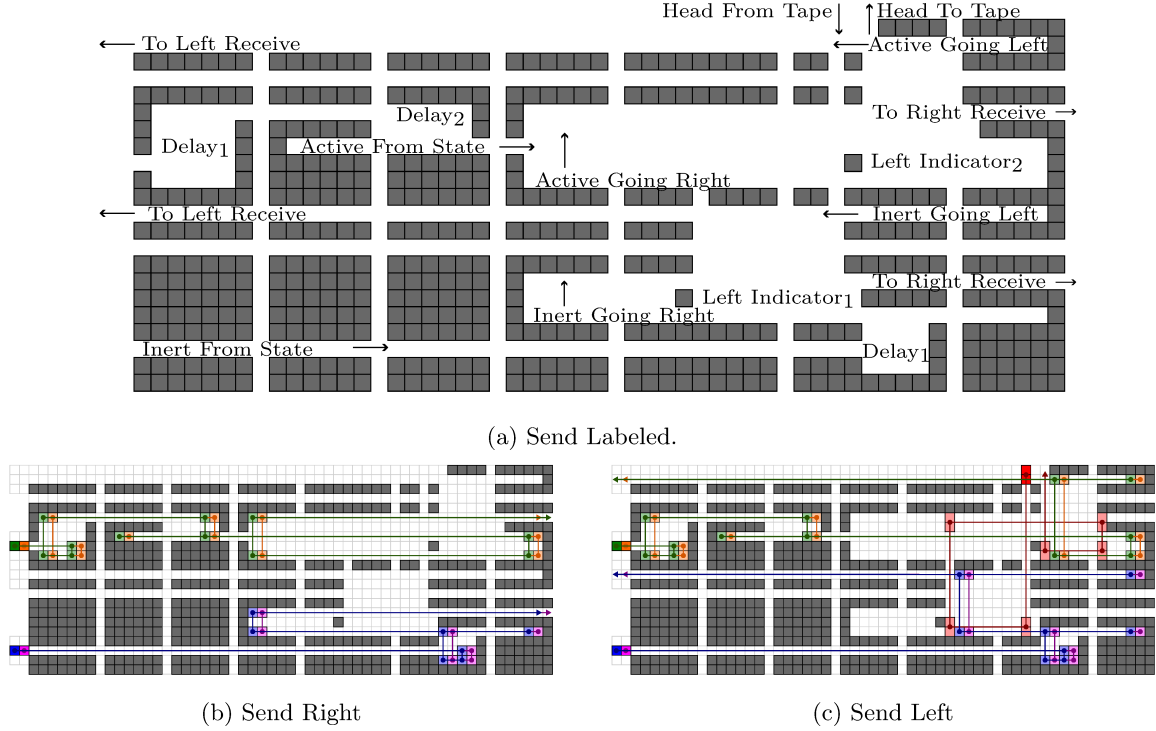


Figure 10: (a) Labeled *Send gadget*. The *Send gadget* receives the state-singletons divided into Active and Inert. The inert-singletons are delayed once, then enter the lower chamber. At this point, there are two cases shown in (b) and (c). (b) shows the case where the head-domino has not been sent by Tape write. This indicates MOVE RIGHT. The inert-singletons pass LEFT INDICATOR₁ and go to INERT GOING RIGHT. They exit TO RIGHT RECEIVE. Active-singletons follow the same process in the upper chamber one cycle later. (c) shows the case where the head-domino is sent down by the Tape write operation, triggering SEND LEFT. Instead of moving past LEFT INDICATOR₁ the head-domino is caught on it. The inert-singletons are sent up to the INERT GOING LEFT tunnel. While this happens, the head-domino is sent up and cycles around to LEFT INDICATOR₂. The active-singletons are caught by it, behaving similarly to the inert-singletons. After Redirecting the active-singletons, the head-domino is sent back into the *Tape gadget* as in the bottom of Figure 4b.

Full-Tilt model with rotational sequences (cycle-length of 4, using 4 directions), even for non-binding systems with singletons and a single polyomino of size 2.

Proof. Given Theorem 1, it is PSPACE-complete to know whether the halting state of any Turing machine encoded as a Full-Tilt system can be reached. In this construction, the halting state causes the state-singletons to be ejected (Figure 12a), which we connect from all the CONTAINERS from every **State-Cell** to one location. This immediately proves relocation and occupancy can simulate a bounded Turing Machine, as after falling out of any container, the head-domino is moved left to a predetermined location. Vacancy can be solved by attaching a *Vacancy gadget* (Figure 12b) to that location such that the head-domino follows the path in Figure 12c, allowing the yellow-singleton to move for the first time.

Reconfiguration requires assignment of a target location for every particle. This includes the data-singletons. We solve this problem after reaching the halt state by setting every data-singleton to ‘1’. This is done by sending the head-domino through mildly modified *Tape gadgets*, as shown in Figure 17c. It is passed to the leftmost cell’s modified *Tape gadget* after exiting the containment area. The final configuration is all data-singletons set to ‘1’, the state-singletons trapped in a cycle from the move after halt, and the head-domino located above and to the right of the rightmost cell. \square

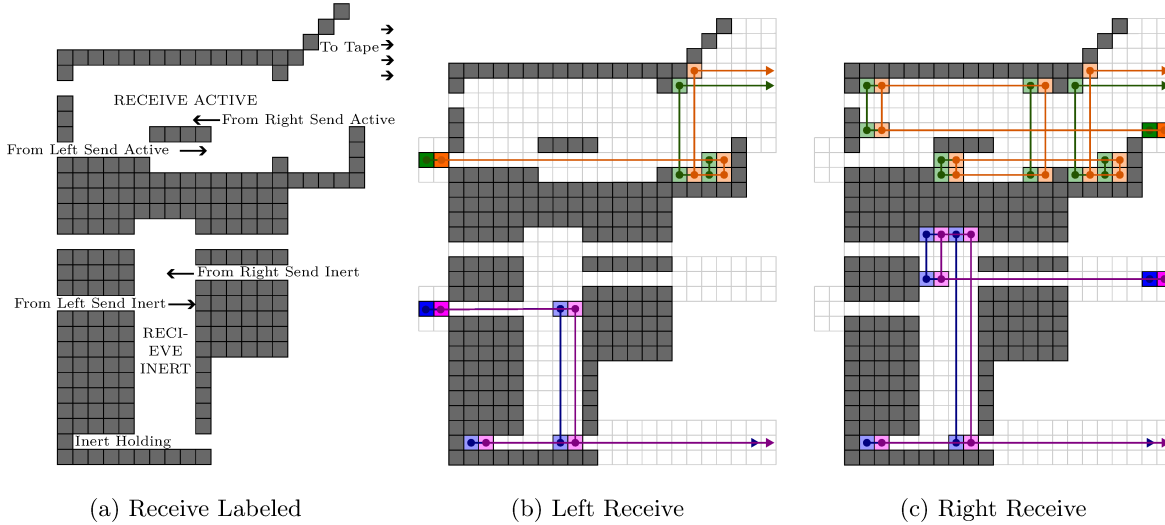


Figure 11: (a) The *Receive gadget* construction with labeled sections. This gadget catches the state-singletons (active and inert) from the adjacent cell to the right or left, then directs the active-singletons to the Tape Read input, and the inert-singletons to the *State gadget* wait area. (b) Shows the gadget receiving two active and two inert-singletons from the Left. (c) Shows the gadget receiving two active-singletons and two inert-singletons from the Right.

3.5 The Single-Step Model

Here, we note that the main complexity problems are still hard in the Single-Step model for a constant cycle length. This is achieved by adapting the Full-Tilt construction. Then with the same modifications to the Full-Tilt TM construction for each specific problem, we can achieve the corresponding results for the complexity questions.

Corollary 1. *For any Turing machine M with $s = |Q|$ states and a bounded tape of length n , there exists a non-bonding rotational Single-Step simulation of the machine with board size $O(ns^3)$ that simulates the machine at a rate of one step per $O(1)$ rotations.*

Proof. Given a TM M , we encode it to be used in a universal 15-state 2-symbol TM, and we create that universal machine using the method outlined in Theorem 1 for Full-Tilt. Since the size of the universal TM is constant, the maximum distance any tilt causes a tile to move within a cell is 70, and thus the same construction works with a cycle of $\langle u^{70}, r^{70}, d^{70}, l^{70} \rangle$. Only a slight modification is needed to move pieces between **State-Cells**. Every 70 steps, a small holding area of simple geometry (see Figure 26a) is needed. \square

Corollary 2. *Relocation, reconfiguration, occupancy, and vacancy are PSPACE-complete in the deterministic Single-Step model with rotational sequences (cycle-length of 280, using 4 directions), even for non-binding systems with singletons and a single polyomino of size-2.*

Proof. Using the **State-Cell** constructions and gadgets from Theorem 2. Since the maximum distance within any gadget is less than 70, they work the same as in Full-Tilt. We connect the **State-Cell** constructions with Single-Step paths with 70-spaced holds (e.g., Fig. 26a). \square

We note that we can actually improve Corollary 2 to show that the problems are still PSPACE-complete for a cycle length of 40 as $\langle u^{10}, r^{10}, d^{10}, l^{10} \rangle$, but the proof uses a different technique. It is proven by implementing gadgets in the motion planning framework [17]. Due to the additional preliminaries and explanations that would be required, and for consistency and elegance of presentation, we have omitted this result in favor of the less optimal one that follows as a corollary.

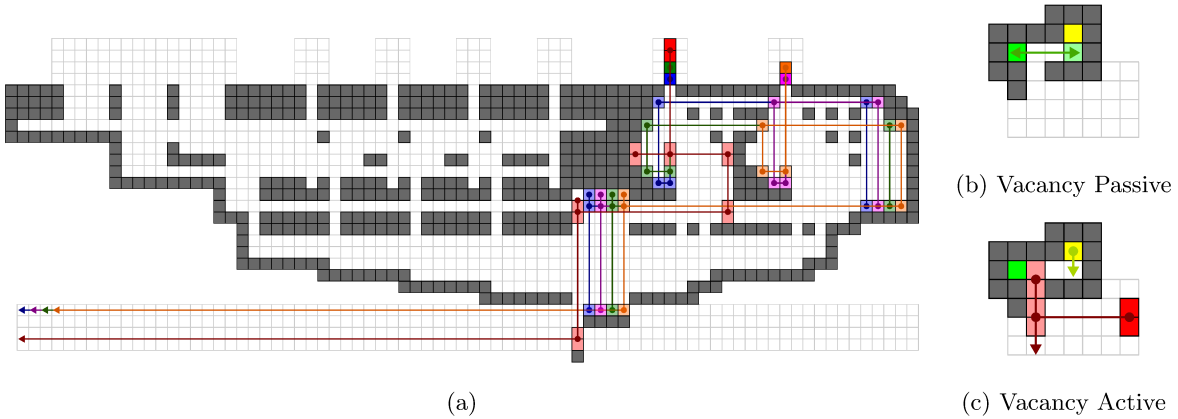


Figure 12: Updated State gadget for complexity results. (a) The modified *State gadget* that abstracts out the cell at which a halt occurred. (b) The behavior of the vacancy gadget when not receiving the head-domino. The yellow tile can never move. (c) How vacancy is solved when the head-domino is received. The head-domino blocks the green tile, and the yellow tile can move down.

3.6 Programming the Tape

Given that a 2-symbol universal Turing Machine can be made with as few as 15 states [25], and the applications at the nanoscale may make placing singletons tedious, there is good motivation to create a tape that is programmable at run time through the global external forces rather than by placing the tiles in the tape directly. However, for a deterministic system, there is no way to achieve this. We provide two methods for initializing the tape: the first being a simple linear encoding that requires $O(n)$ tilts for a size n tape, and a second that uses specialized bit-selection gadgets that can place k 1's on the tape (at any desired positions) with a $O(k \log n)$ -length sequence.

We make a slight modification to our deterministic system to allow a set number of modifications to the cycle at the beginning of the simulation. Once the “programming” is executed, the system goes into the standard $\langle u, r, d, l \rangle$ cycle and may not be modified again. To ensure the tilt sequence modifications of the programming sequence do not interfere with the rest of the construction, we can assume 1) the head-domino is stored in a holding cell and can only be extracted with a $\langle d, l, d, l \rangle$ sequence and 2) that the Turing machine starts in state zero with all state-singletons in the area labeled WAIT of the *State gadget* (Fig. 5).

Linear Edits. In our Full-Tilt TM, a 0 (1) is represented by a single tile in TRACK 0 (TRACK 1) of the *Tape gadget* (Figure 4a). Before programming, we assume neither track contains a tile. An example of a linear programming process is shown in Figure 13a where the $\langle u, r, d, l \rangle$ cycle walks a domino to each tape cell and assigns a tile to either Track 0 or Track 1. To program a ‘1’ in a tape cell, the $\langle u, r, d, l \rangle$ sequence only needs a slight edit by replacing the down and left commands to give $\langle u, r, u, r \rangle$. Thus, programming the tape requires $O(k)$ edits where k is the number of 1's on the tape and a total of $O(n)$ cycles where n is the length of the tape. The tiles being sent can be spaced correctly to be intercepted by extending the TRACK 1 and TRACK 0 tunnels to the right of the *Tape gadget*.

Logarithmic Tilts. We exploit the global control available and use *Bit-selector gadgets* that allow us to select precisely which bits should become 1's without traversing the length of the tape, nor requiring the use of a domino. Figure 13b shows the components of these gadgets, and Figure 14 presents a full example for programming a 4-bit sequence. The *Bit-selector gadgets* are comprised of the left-select, right-select, 0-holding, and 1-holding gadgets. Each gadget has a 0-holding component at its base (initialized with a tile in it), followed by a sequence of left- and right-select components, and is capped with a 1-holding component.

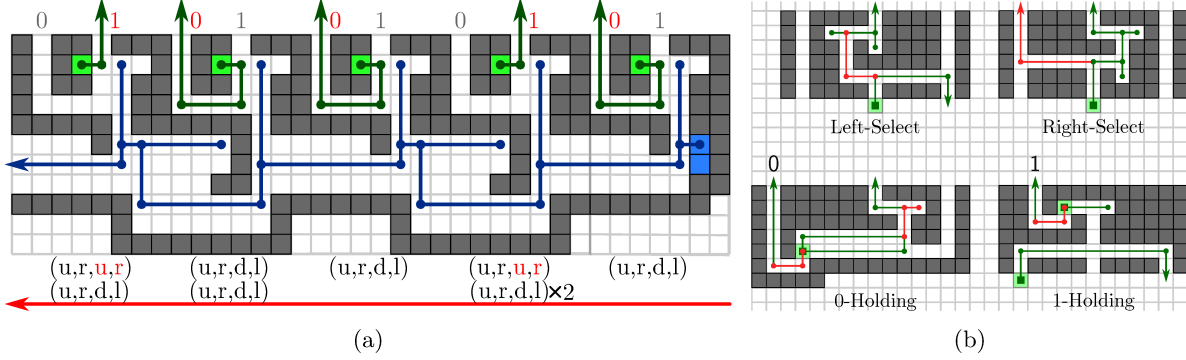


Figure 13: (a) In Full-Tilt, selecting k 1's in the tape in $O(n)$ time where n is the length of the tape and with $O(k)$ edits to the cycle. The cycle ejects a tile in the '0' location, unless an additional $\langle u, r, u, r \rangle$ is added, which causes the tile to exit in the '1' location. After the tilts, the domino exits to the starting cell of the tape. (b) Components of a bit-selection gadget. The left and right-select gadgets dictate the selection sequences for the bit selectors. The 0-holding gadget holds tiles that were not selected as 1's. The 1-holding gadget keeps selected tiles in place until all tiles are ready to be extracted. The head domino is locked in place until a $\langle d, l, d, l \rangle$ sequence, which is the sequence to extract all selected tiles.

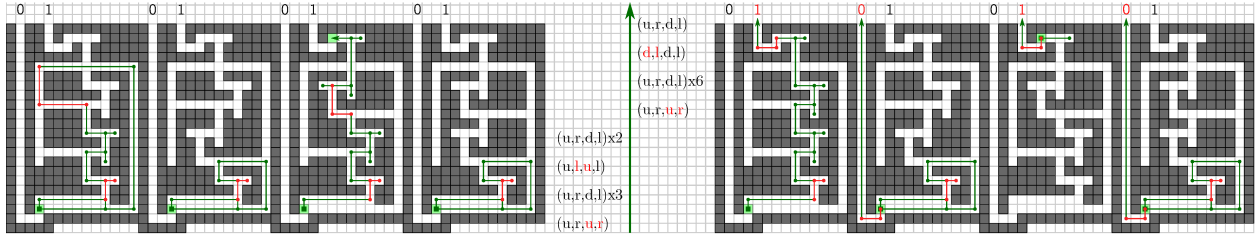


Figure 14: Full *Bit-selector gadgets* for a 4-bit tape. (Left) A sequence of tilt rotations to flip the second bit to a 1. (Right) A subsequent sequence of tilt rotations to flip the fourth bit to a 1 and extract all four bits from the gadgets. In general, k many 1's can be selected for a length- n tape in $O(k \log n)$ tilts and with $O(k \log n)$ edits to the cycle.

By using a unique sequence of left- and right-select components per bit, the gadget repeatedly eliminates half the bits per tilt cycle, until one bit is chosen to become a 1.

To begin this process, we modify the first $\langle u, r, d, l \rangle$ rotation to become $\langle u, r, u, r \rangle$ in order to move all tiles through the top of the 0-holding component. To advance through a left-select gadget, we use another modified rotation step of $\langle u, l, u, l \rangle$. To advance through a right-select component, the standard $\langle u, r, d, l \rangle$ rotation is used. After $O(\log n)$ of these choices, exactly one tile is in the 1-holding component. All others have been redirected back to the 0-holding component through the side tunnels. This process is repeated until all desired 1's have been selected. After this, a final modified tilt rotation of $\langle d, l, d, l \rangle$ prepares all bit tiles (0's and 1's) to be sent to the *Tape gadget* at the same time. Thus, we can program any n -bit tape with k many 1's using $O(k \log n)$ tilt rotations with $O(k \log n)$ edits to the tilt cycle. As before, the tiles exiting the selection gadgets can be positioned to intersect with the correct track of the *Tape gadget*.

4 Space-bounded Two-tape Turing Machine

We now show how the same **State-Cell** construction from Section 3.1 can be extended to simulate a Turing machine with two tapes: α and β . By modifying the *State gadget*, we are able to make new gadgets that have the head-dominos of 2 tapes interact and share information. This information is then sent back into

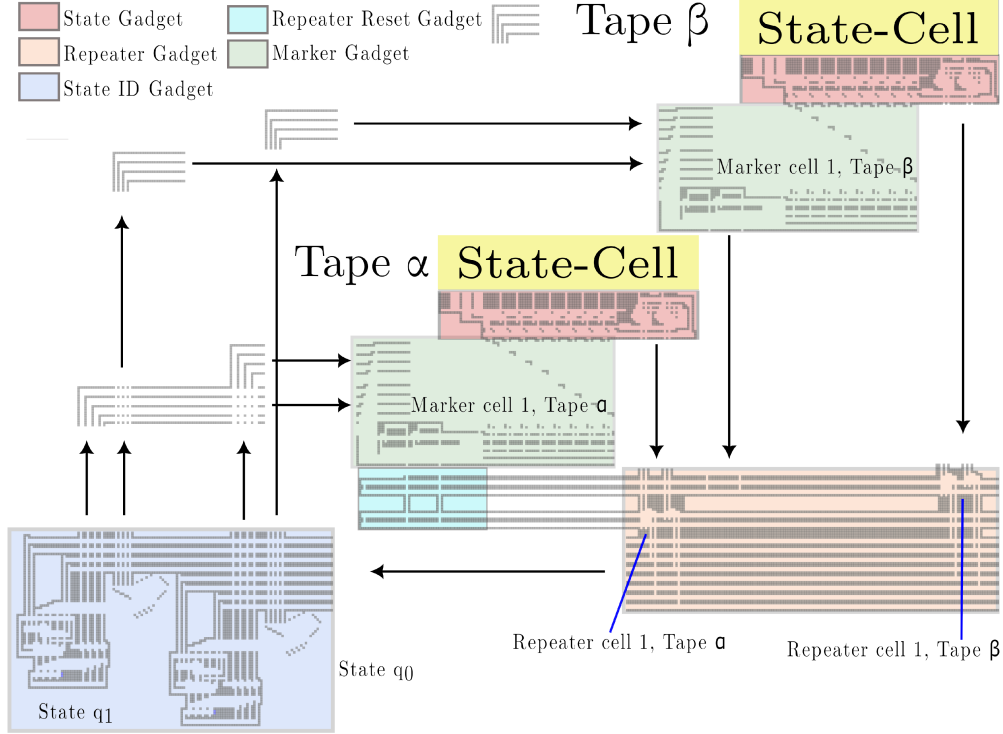


Figure 15: A full diagram of the first cells in a two-tape Turing machine. The yellow boxes indicate the upper half of the **State-Cell**, which works the same as the one in Figure 2. The arrows show how the Turing machine processes information. The unshaded tunnels redirect any helper-dominos moving into the different gadgets.

each tape, which will accordingly execute a transition on one of their **State-Cell** constructs, following the process described throughout Section 3.1.

Two-tape Helper Dominos. In addition to the particles already defined in Section 3.1, we introduce helper-dominos used in the two-tape gadgets.

- **Repeater-dominos.** The orange dominos in Figure 17. These dominos are used to forward information from the head-domino to the new gadgets and interact with the information from another tape.
- **ID-domino.** The blue domino in the *State ID gadget* that assists with sharing the information of the two repeater-dominos, shown in Figure 19.
- **Marker-dominos.** A set of pink dominos in one of the *Marker gadgets* on each given tape. These dominos cycle below the current **State-Cell** as shown in Figure 21a and redirect the repeater-domino into the modified *State gadget*. Each Tape needs a marker domino per State-move-write combination. Thus, there are $4 \times |Q|$ marker dominos on each tape.

Two-tape TM Gadgets. Additional gadgets allow for the information of two different **State-Cells** to interact without modifying the single-tape **State-Cell** too drastically.

- *State gadget* (Sec. 4.1). The Modified *State gadget* allows the head-domino to share the symbol read with the head-domino from the other tape without leaving the **State-Cell**.
- *Repeater gadget* (Sec. 4.2). The *Repeater gadget* reads the information from the head-domino and copies it into a repeater-domino. It then sends that repeater-domino to the correct *State ID gadget*. Every **State-Cell** has a *Repeater gadget*.

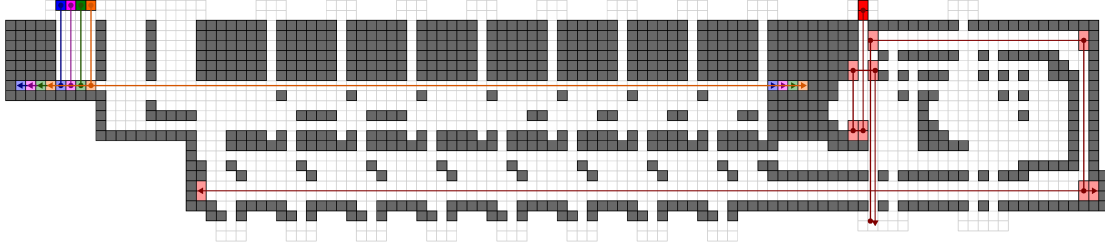


Figure 16: Modified *State gadget*. The head-domino reads the state and goes down to the *Repeater gadget*, and then returns and waits in the bottom tunnel. The State-singletons cycle at WAIT.

- *State ID gadget* (Sec. 4.3). The *State ID gadget* receives the information from both tapes and identifies which transition each tape needs to execute. There is a *State ID gadget* for every state.
- *Marker gadget* (Sec. 4.4). The *Marker gadget* marks which **State-Cell** the head-domino is at and redirects the transition from the *State ID gadget* into that **State-Cell**'s *Modified State Update*. Every **State-Cell** has a *Marker gadget*.
- *Repeater Reset gadget* (Sec. 4.5). The *Repeater Reset gadget* catches the repeater-domino and resets it back to its initial tunnel. There is one *Repeater Reset gadget* per tape.
- *Modified State Update* (Sec. 4.6). The *Modified State Update* is the section of the *Modified State Gadget* responsible for updating the state according to the information read. The process is similar to the one in Figure 7.

A diagram of a two-tape single-cell Turing machine with the attached two-tape gadgets is shown in Figure 15. Following the arrows, we can see how the machine works. First, each **State-Cell** reads its symbol and its state. Then, the *Repeater gadget* forwards the symbol from each **State-Cell** to the *State ID gadget* that matches the state. In the *State ID gadget*, the two repeated signals merge their information to know what specific transition the Turing machine should execute. This transition is then sent to the *Marker gadget* through a tunnel unique to each transition. Then, the Turing machine executes the transition in the cell, the same as in Section 3.1.

4.1 Modified State Gadget

We reuse the *State gadget* from 3.2.2, with a modification that sends the head-domino down into the *Repeater gadget*. Then, the modified *State gadget* catches the head-domino on the next up tilt and sends it into the left half of the modified *State gadget*, where it waits for the transition to update the state as described in Section 4.6. This is shown in Figure 16.

4.2 Repeater Gadget

After the modified *State gadget* sends the head-domino out of the **State-Cell**, the information carried by the head-domino needs to interact with the information from the other tape. We use the *Repeater gadget* shown in Figure 17a to copy this information into a repeater-domino and send each head-domino back into its own **State-Cell**. This gadget has two orange vertical dominos traversing freely left through CHECK 0 and CHECK 1 as shown in Figure 18a and 18b. The gadget receives the head-domino from the modified *State gadget* in the cell on a down tilt. On the following left tilt, the head-domino blocks one of the two CHECK tunnels. The tunnel blocked indicates the value read, and the position represents the machine's current state. The different interactions of the head-domino and the orange repeater-dominos are shown in Figure 18a and 18b. After the interaction, the head-domino goes back up into the **State-Cell** and moves into the *State gadget*. One repeater-domino exits the *Repeater gadget* through one of the four tunnels in its bottom. There is a unique tunnel for each of the four possible interactions. These tunnels send the repeater-domino

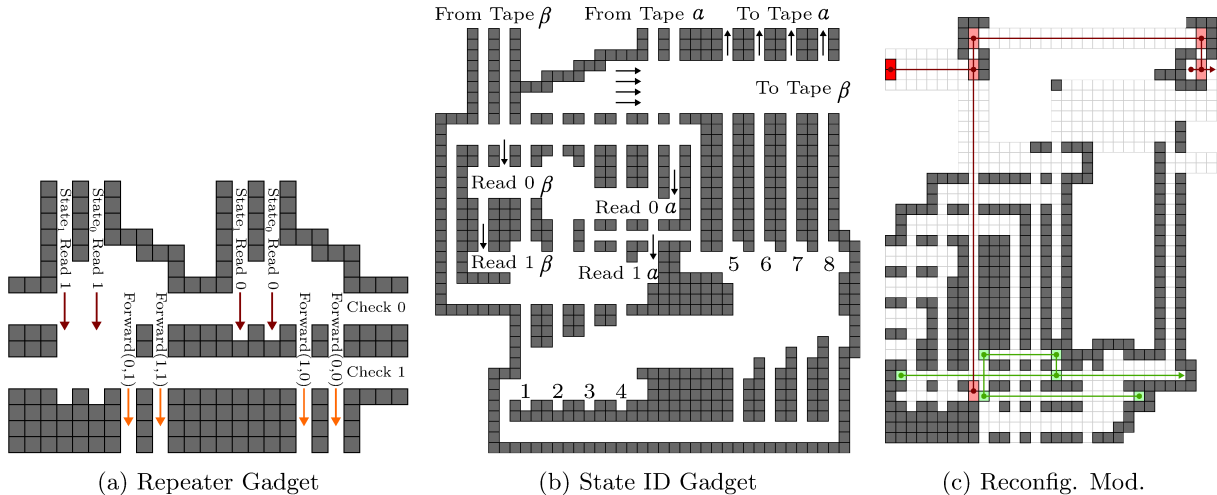


Figure 17: (a) The *Repeater gadget*. Every input tunnel forwards $\text{STATE}_x \text{ READ } y$ into the output tunnel $\text{FORWARD}(x, y)$. The $\text{CHECK } 0$ and $\text{CHECK } 1$ tunnels extend horizontally throughout the board. Every cell in the tape uses the same tunnels. (b) The *State ID gadget*. If the Turing machine is in its state, it receives the repeater-dominoes from each Tape's *Repeater gadgets* at FROM TAPE . Which output tunnel the head-dominoes take at their TO TAPE encodes what the current cell on both tapes read. There is one gadget per different state. (c) A modification to the *Tape gadget* in order to prove the complexity of reconfiguration. To set all data-singletons on the tape to 1, this gadget receives the head-domino from the top left. Then, if the data-singleton is in $\text{TRACK } 0$, the head-domino moves it to track 1. Whether the data-singleton was moved or not, the head-domino then moves to the **State-Cell**'s right neighbor to repeat the process.

to the corresponding *State ID gadget*. Each **State-Cell** has its own *Repeater gadget*. A *Repeater gadget* from Tape α is in line with the other *Repeater gadgets* from all **State-Cell** in Tape α .

4.3 State ID Gadget

This gadget takes as input the repeater-domino. Since there is a *Repeater gadget* for each cell, there will always be two repeater-dominoes coming into the *State ID gadget*, one for Tape α and one for Tape β , as shown in Figure 19. The purpose of the *State ID gadget* is to read the symbols from both tapes and send the same repeater-domino to carry that information back to **State-Cell**. Each of the four different combinations of symbols between the two tapes has a tunnel to carry this information, as shown in Figure 17b. We use the yellow domino as the repeater-domino from Tape α and the orange domino as the repeater-domino from Tape β . The two dominoes enter the gadget in a position denoting the tape they come from and the symbol they read. According to their position, they can interact in four different ways. According to the interaction, the orange domino will fall into one of the positions labeled 1-4 in Figure 17b. This will stop the blue ID-domino so it can stop the yellow domino in the correct position of those labeled 5-8. This way, the orange and yellow dominoes are each positioned according to the two symbols, and can separately take the correct transition to their tape. The ID-domino then makes its way back to its starting position. The four different interactions are shown in Figure 19.

4.4 Marker Gadget

The *Marker gadget* keeps track of which cell the head-domino is currently reading and allows the repeater-dominoes leaving the *State ID gadget* to identify which **State-Cell** construct they should stop at. The *Marker gadget* uses pink marker-dominoes that cycle as shown in Figure 21a. There are $4 \times |Q|$ marker-dominoes per tape on the board at all times. When the repeater-domino enters (after a right tilt), one of the marker-

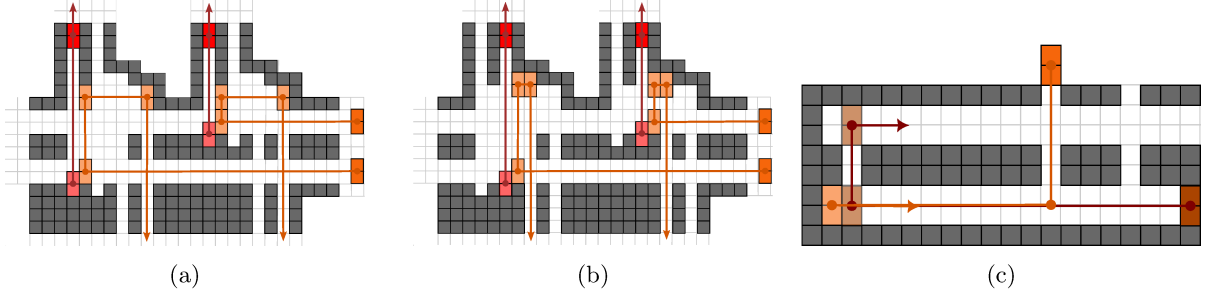


Figure 18: (a) The *Repeater gadget* forwarding a symbol in state q_1 . (b) The *Repeater gadget* forwarding a symbol in state q_0 . (c) The *Repeater Reset gadget*. This gadget is below the leftmost *Marker gadget*.

dominos will catch the repeater-domino and send it up into the modified *State gadget*, as shown in Figure 21b. Then, the same repeater-domino is redirected back down into one of the two tunnels labeled ‘Send Left Indicator’ and ‘Send Right Indicator’ in the bottom part of the *Marker gadget*. Depending on which tunnel the repeater-domino stops at, it will send the marker-domino right or left to the *Marker gadget* of the neighboring **State-Cell**, as shown in Figures 21c and 21d. An alternating height difference is needed for this gadget due to the sending and receiving mechanic, similar to the one in the **State-Cell** described in Section 3.1.

4.5 Repeater Reset Gadget

Once out of the *Marker gadget*, the repeater-domino needs to return to its original tunnel in the *Repeater gadget*. To do this, it gets dropped into the tunnel labeled ‘check for 1’ in Figure 17a. Then, in the case that there already is a domino in the tunnel, one of the two dominos will be pushed up into the ‘check for 0’ tunnel. This interaction is shown in Figure 18c. Note that the CHECK 1 and CHECK 0 tunnels extend horizontally from the *Repeater Reset gadget* all the way to the rightmost *Repeater gadget*.

4.6 Modified State Update

The way the state is updated is similar to the one in the *State gadget* in Section 3.2.2. The head-domino comes in from the *State gadget* and is trapped in the bottom part tunnel. When the repeater-domino from the *Marker gadget* comes up, it will stop the head-domino and send it up to execute a transition to change state, as shown in Figure 22. The head-domino stops the state-singletons such that upon tilt down, they are separated into inert-singletons and active-singletons. Then, the singletons and the head-domino continue to their respective gadgets described in Section 3.1. Using these gadgets, along with the **State-Cell** from Section 3.1, we can establish the following:

Theorem 3. *For any two-tape Turing machine \mathcal{M}_2 with $s = |Q|$ states and a space bound n such that each tape of \mathcal{M}_2 uses at most n tape cells, there exists a non-bonding rotational Full-Tilt simulation of the machine with board size $O(ns^3)$ that simulates the machine at a rate of one step per $O(1)$ rotations.*

Given a two-tape Turing machine \mathcal{M}_2 , we design the board as shown in Figure 15. There are $2n$ **State-Cells** with height offset to separate Tape α from Tape β , $2n$ *Marker gadgets* below the **State-Cells**, and s *State ID gadgets* in the bottom left corner with a height and width offset to separate the input and output tunnels. A single *Repeater Reset gadget* is placed below the leftmost *Marker gadget* and is connected to the checker-tunnels from the *Repeater gadgets*. For every additional state, the *Repeater*, *State ID*, and *Marker gadgets* need an extra pair of input/output tunnels. This construction will require $O(s)$ domino particles. This yields a board of height $O(s)$ and width $O(ns^2)$.

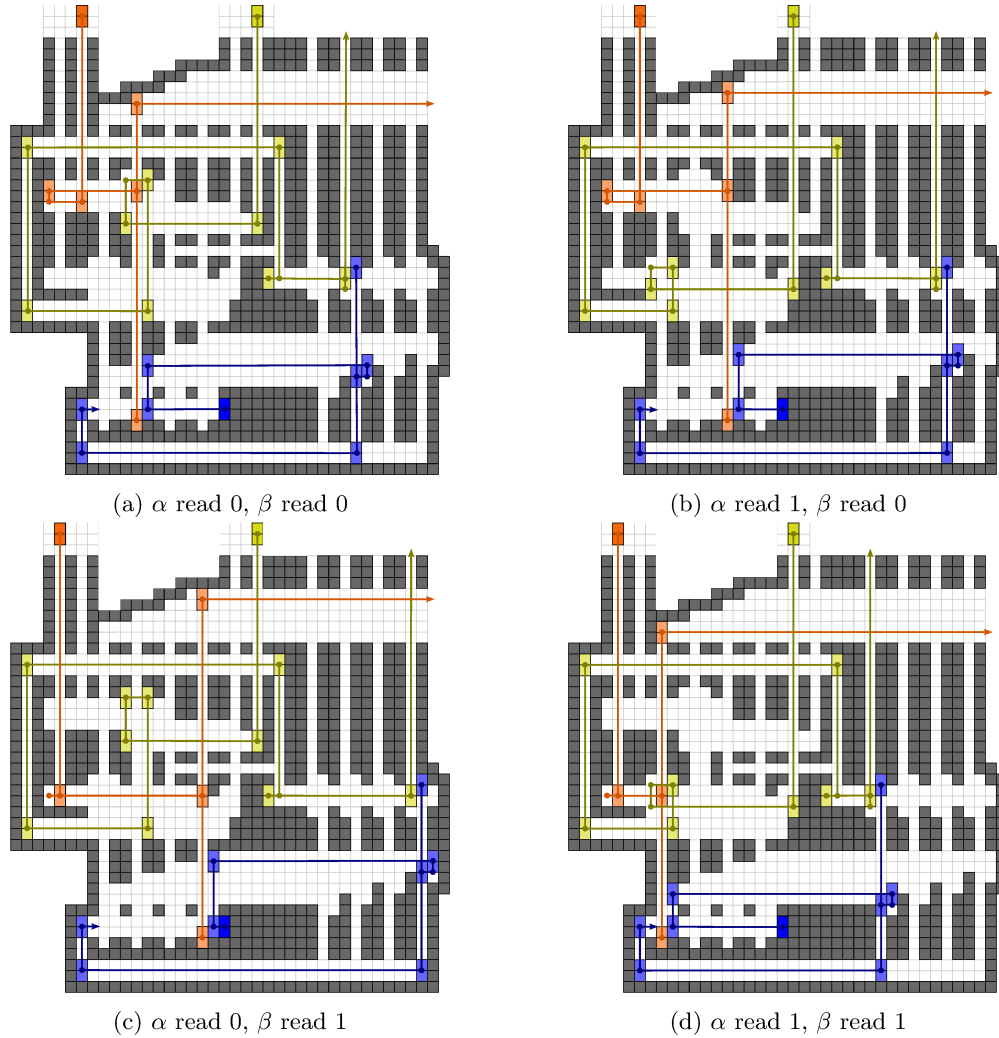


Figure 19: The four possible interactions in a *State ID gadget*. Depending on the symbols read by both tapes, different output tunnels will be used by the yellow (Tape α) and orange (Tape β) repeater-dominos.

4.7 The Single-Step Model

Similar to the single-tape modification in Section 3.5, we modify the two-tape machine for the Single-Step model. As before, the **State-Cell** spacing is bounded, but we can not achieve a constant number of cycles for each step of the TM. Since the two tape heads need to communicate, the distance between them may be as far as the length of the tape (T). Thus, the number of cycles per step is $O(T)$ for any constant cycle.

Corollary 3. *For any two-tape Turing machine \mathcal{M}_2 with $s = |Q|$ states and a space bound n such that each tape of \mathcal{M}_2 uses at most n tape cells, there exists a non-bonding rotational Single-Step simulation of the machine with board size $O(ns^3)$ that simulates the machine at a rate of one step per $O(T)$ rotations.*

Systolic Arrays. In some sense, the Single-Step model's limited travel range prevents many possible efficiency improvements over the single-tape TM. However, due to the use of the **State-Cell** constructions, multiple computations could be occurring simultaneously in different parts of the tape if multiple dominos and state-singletons are included. This would require coordination to prevent collisions and limit each computation to a section of the tape.

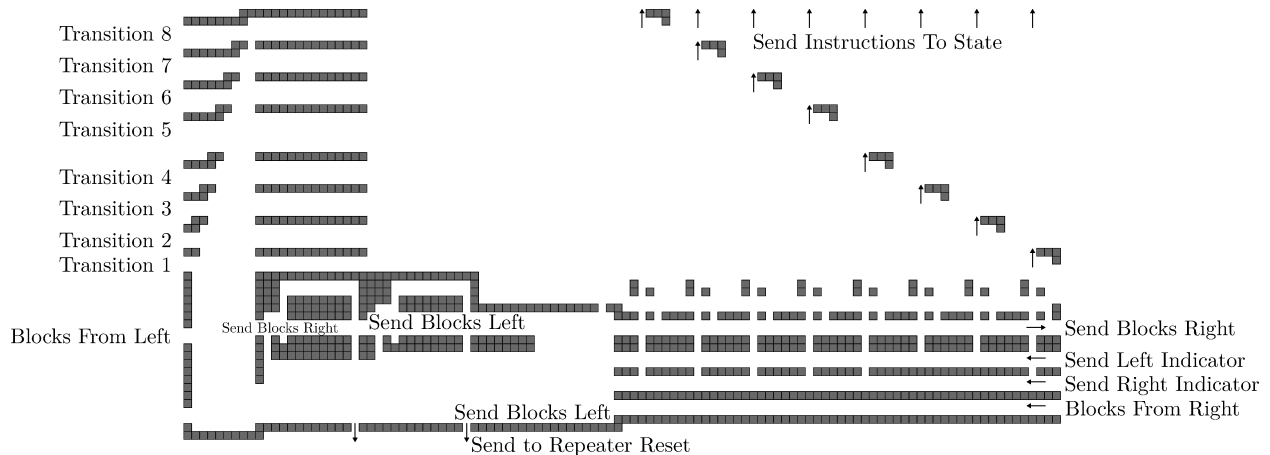


Figure 20: *Marker gadget*. The top half of the gadget is where the repeater-domino gets stopped and redirected to the *state gadget* to execute the transition output its location indicates. In the bottom half, the repeater-domino sends the marker-dominoes left or right, depending on the transition’s direction.

We can easily extend the single-tape TM construction to work as a linear systolic array [10, 21, 24] with a single head-domino at each head location and a set of state-singletons. The tape is split into k independent memory sections, each with its own tape head (head-domino and state-singletons). The left/right boundary **State-Cells** of each memory section has a dual-rail *input* area on the left cell and an *output* area on the right cell for the head-domino to pass on information to the neighboring memory areas.

Corollary 4. *For any single-tape Turing machine \mathcal{M} with $s = |Q|$ states, k heads (each with n/k tape cells), and a tape of length n , there exists a non-bonding rotational Single-Step simulation of the machine with board size $O(ns^3)$ that simulates the machine at a rate of one step of each head per $O(1)$ rotations.*

The Full-Tilt model can also achieve this, but since it can also implement a two-tape TM without slowdown, the result is trivial.

5 Efficient Circuit Simulation

In this section, we show how to efficiently simulate Threshold circuits of any level. The results in [6] show that we can implement computationally universal circuits with polynomial board constructions, even for unbounded FAN-IN, with logic gates that compute the results in a constant number of $\langle d, l/r, d, l/r \rangle$ cycles. We provide similar constructions that solve decision problems using gates designed for a fixed $\langle u, r, d, l \rangle$ cycle, and then extend the work with a MAJORITY gate to allow for efficient simulation of *Threshold Circuits*. We improve upon the limitations of a digital sorting network, and show that we can produce monotonic permutations of Boolean values in a constant number of $\langle u, r, d, l \rangle$ rotations. By extension, we give an N-MAJORITY gate gadget in $O(1)$ rotations and $O(n^2)$ size for unbounded, unweighted FAN-IN with n inputs, and show that a Threshold Circuit of depth d can be simulated in a polynomial board in $O(d)$ cycles.

Being topologically atomic in their design (in terms of computation time), similar to the implementations in [6], the gates simulate Boolean circuits. However, a limitation of conventional digital logic gates is their inability to efficiently compute *threshold* gates. Consider a MAJORITY gate, where the expression is *True* when $\lceil \text{FAN-IN}/2 \rceil$ operands are *True*, and *False* otherwise, for odd-parity length inputs.¹

When the *quantity* of *True* operands is known for a given gate, determining if the quantity is greater than k is trivial, i.e., the k^{th} value of a monotonic sequence of operands T . Then $T[k]$ is the MAJORITY. A sorting network permutes T as $\langle 1, 0, 1, 1, 0, \dots \rangle \mapsto \langle 0, 0, \dots, 1, 1, 1 \rangle$.

¹Even-parity lengths are often implementation-defined and thus not addressed.

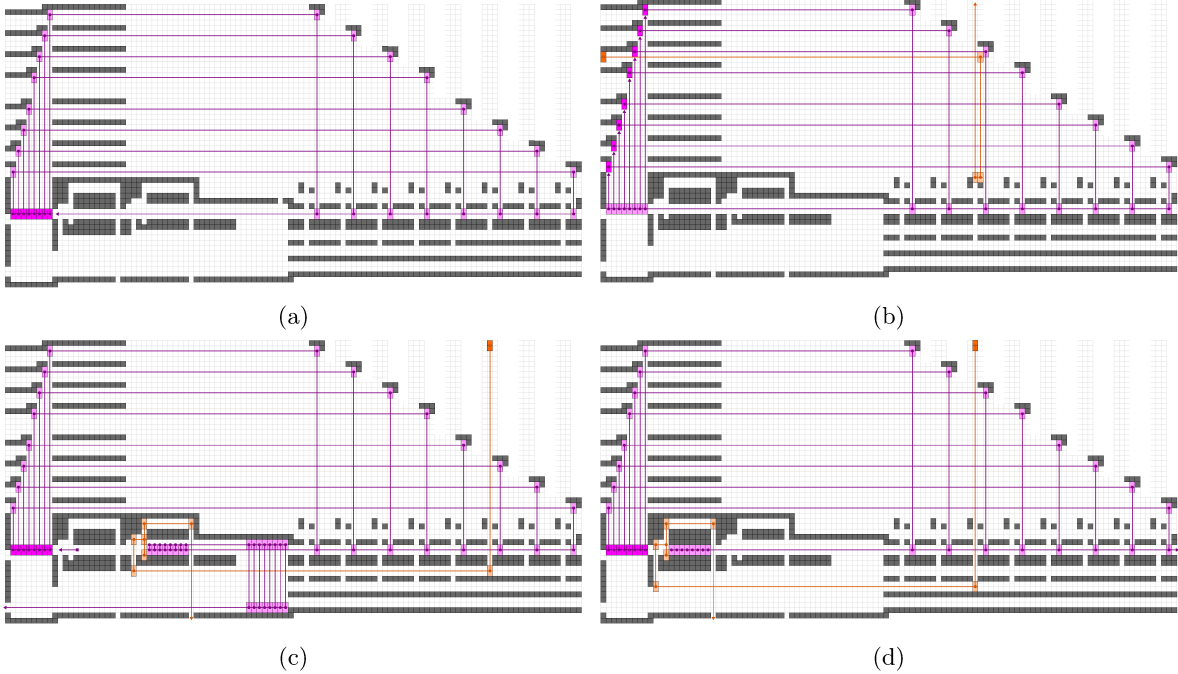


Figure 21: (a) The marker-dominoes cycling in the *Marker gadget*. (b) The repeater-domino being redirected up into the modified *State gadget*. (c) The repeater-domino sending the marker-dominoes to the right and leaving to the *Repeater Reset gadget*. (d) The repeater-domino sending the marker-dominoes to the left and leaving to the *Repeater Reset gadget*.

An intrinsic property of sorting networks is that comparisons between channels are performed in parallel, thus greatly minimizing the depth requirements of comparing and swapping operands for an optimal sorting network (Figure 24). Numerous algorithms and implementations exist for producing optimal or near-optimal sorting networks with depth $O(\log^2 n)$ and size $O(n \log^2 n)$ where n is the number of inputs (assume comparators are $O(1)$), such as MergeSort and the PAIRWISE SORTING NETWORK [28]. A limitation of sorting networks is that the optimality of a network of arbitrary input size is unknown, and determining whether a configuration of a network is a valid sorting network is coNP-complete [27]. Both of these factors make a SORTING-NETWORK MAJORITY gate sub-optimal for sufficient input sizes in Full-Tilt. Thus, we provide an alternative implementation that improves upon digital Boolean sorting within Full-Tilt.

FAN-OUT. We note that [6] proved that in this model, a FAN-OUT requires a domino within the gadget. In order to duplicate the boolean value of some output, additional tiles must be stored, and a domino is required to keep the tiles trapped until the FAN-OUT is activated. Thus, in order to maintain efficiency, a domino exists at every FAN-OUT gate. We give a cleaner arbitrary FAN-OUT in Figure 25b that uses two dominos. It has the possible output tiles trapped internally, which requires a height of $O(n)$ where n is the size of the FAN-OUT output. This affects the height of the board, so we note that the FAN-OUT in [6] has constant height, even if it requires sending the output tiles in as input. Thus, we use the constant height of all gadgets in our construction to achieve the following.

Theorem 4. *For any Threshold Circuit T of depth d and width w , there exists a Full-Tilt system with a board of size $O(w^2 d)$ that simulates T with $O(d)$ $\langle u, r, d, l \rangle$ cycles.*

Proof. An n -MAJORITY gate computed via boolean sorting can be simulated with a Full-Tilt system with a board of size $O(n)$ that determines the MAJORITY in $O(1)$ $\langle u, r, d, l \rangle$ cycles. This is done via sorting the *True*

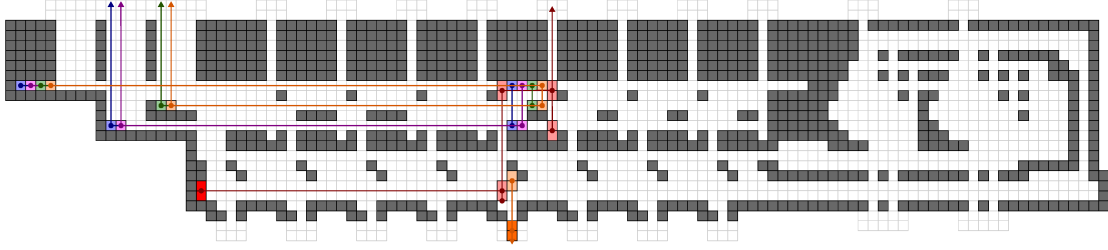


Figure 22: The head-domino executing a state update transition. The repeater-domino blocks the head-domino, sending it up. On the following right tilt the head-domino stops the singletons. Then, the process of separating active-singletons and inert-singletons, and the rest of the information processing in the cell, follows the same process as in Section 3.2.2.

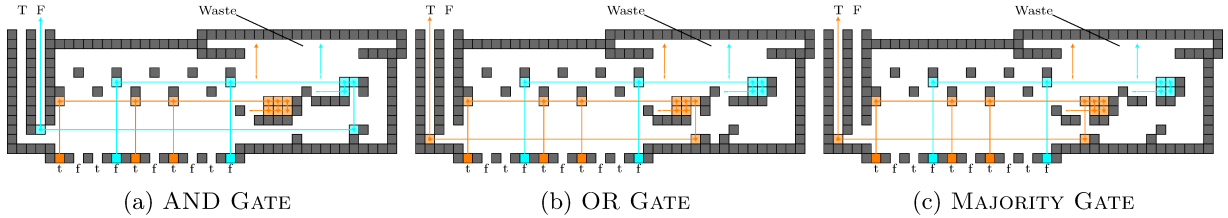


Figure 23: (a) 5-AND gate with 3-True, 2-False inputs. (b) 5-OR gate with 3-True, 2-False inputs. Each gate actually computes *two* expressions that can only be True when the other is False. Both gates compute their True and False expressions in parallel in such a way that the output is always either True or False in dual-rail. Notice that you can take the Boolean inversion of an operand by swapping its respective filters. (c) A dual-rail 5-MAJORITY gate with inputs $\langle 1, 0, 1, 1, 0 \rangle$. Note that, like the Turing Machine, these gates operate with repeated executions of $\langle u, r, d, l \rangle$ cycles.

and False dual-rail input tiles into two columns (Figure 25a), and allowing a single tile from the column with more tiles (more than $\lceil n/2 \rceil$) to exit the correct dual-rail pathway. All other tiles are trapped. We provide an example construction of a 5-MAJORITY gate in Figure 23c. Any gate's input value n must be less than the width of the circuit w . The MAJORITY, AND, OR, and FAN-OUT gates all require $O(n)$ space in width and a constant height (Figure 23). Thus, the total size of the board with each gate gadget is $O(nwd)$, which is $O(w^2d)$. \square

The Single-Step Model. Although the basic idea is the same as with the Full-Tilt model, we incur the distance divided into a number of cycles based on the distance moved per cycle. This is demonstrated in Figure 26a. The gates must be similarly modified, but can be done so with a relatively small cycle length.

Corollary 5. *For any Threshold Circuit T of depth d and width w , there exists a Single-Step system with a board of size $O(w^2d)$ that simulates T with $O(dw)$ $\langle u, r, d, l \rangle$ cycles.*

Proof. Similar to Theorem 4, the board is built using the basic gadgets modified to work for the Single-Step model. This includes changing any tilt path longer than the number of steps in a direction as shown in Figure 26a. Similarly, the comparisons made in the AND, OR, and MAJORITY need to be handled accordingly as well. An example of the Boolean sorting is shown in Figure 26b. Each gate can check the output similar to the MAJORITY check, which is shown in Figure 26c. At each gate, it might take $O(w)$ cycles to walk the tiles to the sides of the gate to compute the output. Thus, the entire simulation takes $O(dw)$ total cycles. \square

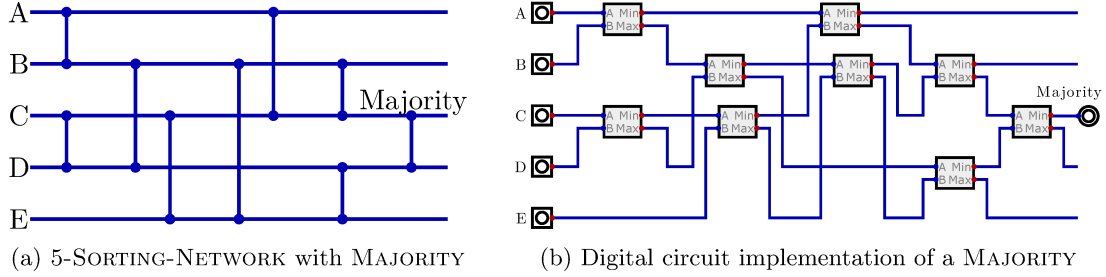


Figure 24: A 5-MAJORITY gate using a sorting network, where the 3rd operand indicates the MAJORITY.

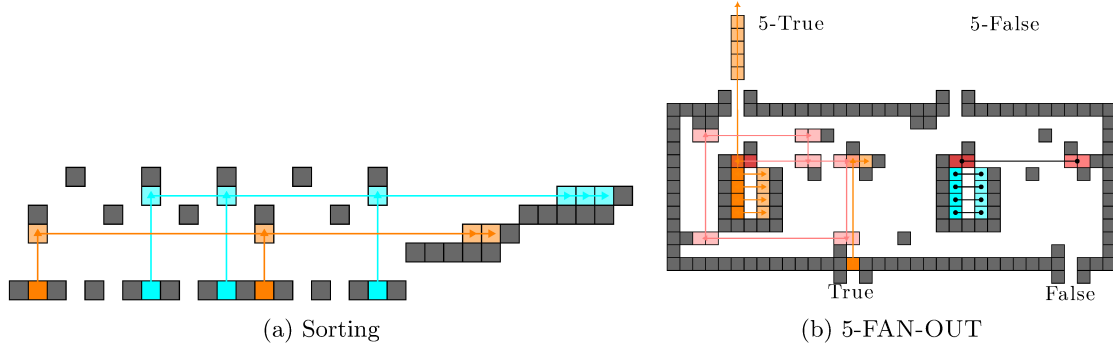


Figure 25: (a) A simple filter gadget that groups *True* and *False* dual-rail operands (represented as orange tiles (left) and cyan tiles (right), respectively) into monotonic groups of *True* and *False* tiles, each in their own row. This gadget demonstrates sorting of an arbitrary number of Boolean operands within a single $\langle u, r, d, l \rangle$ cycle (specifically achieving the monotonic grouping in $\langle u, r \rangle$) with $O(n)$ space for n input operands. This grouping is how AND, OR, and MAJORITY gates are simulated with $\langle u, r, d, l \rangle$ cycles. (b) A 5-FAN-OUT gate gadget that outputs 5 tiles from the *True* side of a dual-rail Boolean gadget when the input is *True*, and from the *False* side when the input is *False*. Dominos (red) block the FAN-OUT tiles until they are unlatched by the input. When a domino is unlatched, 5 tiles exit from the 5-FAN-OUT gate at the beginning of the second $\langle u, r, d, l \rangle$ cycle, and the unlatched domino will re-latch at the end of the cycle. The output tiles then get separated into the corresponding 5 output dual-rail paths (the reverse of sorting).

6 Conclusion and Future Work

We have shown that rotational tilt systems can efficiently simulate space-bounded, programmable Turing machines, which provides a framework for utilizing deterministic tilt systems as computational devices. Through this, we have resolved the complexity of Relocation, Occupancy, Vacancy, and Reconfiguration as PSPACE-complete for deterministic tilt cycles for Full-Tilt with a 4-cycle and Single-Step with a cycle of length 280. Further, we showed that the tape can be programmed directly or via tilts before computation begins.

For efficiency, we show that a two-tape Turing machine can also be simulated in the Full-Tilt model with a constant number of cycles per operation. Single-Step has a harsh movement overhead based on the size of the tape. Thus, we show we can implement linear Systolic Arrays with to achieve some parallel efficiency improvement. Finally, we show how Threshold Circuits (TC) can be efficiently implemented for any level of TC with both Full-Tilt and Single-Step. Full-Tilt only requires a number of cycles based on the depth, and Single-Step requires a number of cycles based on both the depth and width. This work has introduced a number of interesting directions for future work.

- For general tilt sequences, the reachability problems are PSPACE-complete even when restricted to singleton tiles [4]. What is their complexity with deterministic tilt sequences and singleton tiles (no

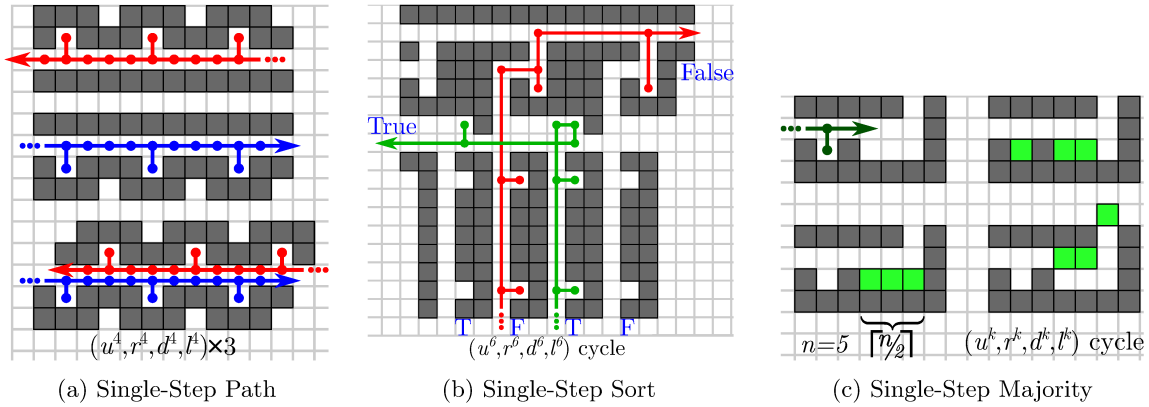


Figure 26: (a) A left, right, and dual tunnel in Single-Step that is traversable for the cycle $\langle u^4, r^4, d^4, l^4 \rangle$. (b) Binary sorting dual-rail input in Single-Step for two variables. All *True* tiles exit on the same path to the left, and all *False* variables exit the same tunnel to the right using a $\langle u^6, r^6, d^6, l^6 \rangle$ cycle. (c) A Single-Step MAJORITY gate for some constant $4k$ -length cycle with five inputs.

single domino)?

- What is the smallest Single-Step cycle where the four reachability-based problems are still PSPACE-complete? What cycles are easy?
- What is the smallest Single-Step cycle that can directly simulate general computation? For both models, what are the fewest cycles needed to compute a given function?
- We focused on rotational cycles, but [6] also considered 4-cycles with only 3 directions $\langle d, l, d, r \rangle$. Our MAJORITY gate can be adapted to work in this model to give Threshold circuits with three directions. What other classes of circuits can be efficiently simulated?
- It may be interesting to study how robust rotation-only systems are to perturbations in the rotation sequence. A future work direction might be to design rotation-based systems with built-in error correction that can withstand reasonable deviations from the intended rotation cycle. Related questions might be to ask what the minimum alteration to a given tilt-sequence is to ensure some property, such as occupancy or vacancy.

References

- [1] Akitaya, H., Aloupis, G., Löffler, M., Rounds, A.: Trash compaction. In: Proc. 32nd European Workshop on Computational Geometry. pp. 107–110 (2016)
- [2] Akitaya, H., Demaine, E.D., Ku, J.S., Lynch, J., Paterson, M., Toth, C.D.: 2048 without merging. Proceedings of the 32nd Canadian Conference on Computational Geometry (2021)
- [3] Akitaya, H.A., Löffler, M., Viglietta, G.: Pushing blocks by sweeping lines. In: Proc. of the 11th Inter. Conf. on Fun with Algorithms. FUN’22 (2022)
- [4] Balanza-Martinez, J., Gomez, T., Caballero, D., Luchsinger, A., Cantu, A.A., Reyes, R., Flores, M., Schweller, R.T., Wylie, T.: Hierarchical shape construction and complexity for slidable polyominoes under uniform external forces. In: Proc. of the 2020 ACM-SIAM Symposium on Discrete Algorithms. pp. 2625–2641. SODA’20 (2020). <https://doi.org/10.1137/1.9781611975994.160>
- [5] Balanza-Martinez, J., Luchsinger, A., Caballero, D., Reyes, R., Cantu, A.A., Schweller, R., Garcia, L.A., Wylie, T.: Full tilt: Universal constructors for general shapes with uniform external forces. In:

- Proc. of the 2019 ACM-SIAM Symposium on Discrete Algorithms. pp. 2689–2708. SODA’19 (2019). <https://doi.org/10.1137/1.9781611975482.167>
- [6] Becker, A.T., Demaine, E.D., Fekete, S.P., Lonsford, J., Morris-Wright, R.: Particle computation: complexity, algorithms, and logic. *Natural Computing* **18**(1), 181–201 (2019). <https://doi.org/10.1007/s11047-017-9666-6>, <https://doi.org/10.1007/s11047-017-9666-6>
- [7] Becker, A.T., Fekete, S.P., Keldenich, P., Krupke, D., Rieck, C., Scheffer, C., Schmidt, A.: Tilt assembly: Algorithms for micro-factories that build objects with uniform external forces. *Algorithmica* **82**(2), 165–187 (2020). <https://doi.org/10.1007/s00453-018-0483-9>, <https://doi.org/10.1007/s00453-018-0483-9>
- [8] Becker, A.T., Habibi, G., Werfel, J., Rubenstein, M., McLurkin, J.: Massive uniform manipulation: Controlling large populations of simple robots with a common input signal. In: 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems. pp. 520–527 (Nov 2013). <https://doi.org/10.1109/IROS.2013.6696401>
- [9] Blumenberg, P., Schmidt, A., Becker, A.T.: Computing motion plans for assembling particles with global control. In: 2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 7296–7302 (2023). <https://doi.org/10.1109/IROS55552.2023.10341556>
- [10] Brent, R.P., Kung, H.T.: Systolic vlsi arrays for polynomial gcd computation. *IEEE Transactions on Computers* **C-33**(8), 731–736 (1984). <https://doi.org/10.1109/TC.1984.5009358>
- [11] Caballero, D., Cantu, A., Gomez, T., Luchsinger, A., Schweller, R., Wylie, T.: Fast reconfiguration of robot swarms with uniform control signals. *Natural Computing* **20**, 1–11 (12 2021). <https://doi.org/10.1007/s11047-021-09864-0>
- [12] Caballero, D., Cantu, A.A., Gomez, T., Luchsinger, A., Schweller, R., Wylie, T.: Building patterned shapes in robot swarms with uniform control signals. In: Proceedings of the 32nd Canadian Conference on Computational Geometry. pp. 59–62. CCCG’20 (2020)
- [13] Caballero, D., Cantu, A.A., Gomez, T., Luchsinger, A., Schweller, R., Wylie, T.: Hardness of reconfiguring robot swarms with uniform external control in limited directions. *Journal of Information Processing* **28**, 782–790 (2020)
- [14] Caballero, D., Cantu, A.A., Gomez, T., Luchsinger, A., Schweller, R., Wylie, T.: Relocating units in robot swarms with uniform control signals is pspace-complete. In: Proceedings of the 32nd Canadian Conference on Computational Geometry. pp. 49–55. CCCG’20 (2020)
- [15] Caballero, D., Cantu, A.A., Gomez, T., Luchsinger, A., Schweller, R., Wylie, T.: Uniform robot relocation is hard in only two directions even without obstacles. *Natural Computing* **24**(1), 3–16 (Mar 2025). <https://doi.org/10.1007/s11047-024-10007-4>, <https://doi.org/10.1007/s11047-024-10007-4>
- [16] Chiang, P.T., Mielke, J., Godoy, J., Guerrero, J.M., Alemany, L.B., Villagómez, C.J., Saywell, A., Grill, L., Tour, J.M.: Toward a light-driven motorized nanocar: Synthesis and initial imaging of single molecules. *ACS Nano* **6**(1), 592–597 (2012). <https://doi.org/10.1021/nn203969b>, PMID: 22129498
- [17] Demaine, E.D., Hearn, R.A., Hendrickson, D., Lynch, J.: PSPACE-completeness of reversible deterministic systems. In: Proceedings of the 9th Conference on Machines, Computations and Universality (MCU 2022). pp. 91–108. Debrecen, Hungary (August 31–September 2 2022)
- [18] Fekete, S., Kramer, P., Reinhardt, J.M., Rieck, C., Scheffer, C.: Drainability and fillability of polyominoes in diverse models of global control. In: The 52nd EATCS International Colloquium on Automata, Languages, and Programming (ICALP) (2025)

- [19] Felfoul, O., Mohammadi, M., Gaboury, L., Martel, S.: Tumor targeting by computer controlled guidance of magnetotactic bacteria acting like autonomous microrobots. In: 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems. pp. 1304–1308 (Sep 2011). <https://doi.org/10.1109/IROS.2011.6094991>
- [20] Keller, J., Rieck, C., Scheffer, C., Schmidt, A.: Particle-based assembly using precise global control. *Algorithmica* **84**(10), 2871–2897 (Oct 2022). <https://doi.org/10.1007/s00453-022-00992-2>, <https://doi.org/10.1007/s00453-022-00992-2>
- [21] Kung, H.T.: Why systolic architectures? *Computer* **15**(1), 37–46 (1982). <https://doi.org/10.1109/MC.1982.1653825>
- [22] Martel, S.: Bacterial microsystems and microrobots. In: *Biomedical Microdevices*. vol. 14, pp. 1033–1045 (2012). <https://doi.org/10.1007/s10544-012-9696-x>
- [23] Martel, S., Taherkhani, S., Tabrizian, M., Mohammadi, M., de Lanauze, D., Felfoul, O.: Computer 3d controlled bacterial transports and aggregations of microbial adhered nano-components. *Journal of Micro-Bio Robotics* **9**(1), 23–28 (Jun 2014). <https://doi.org/10.1007/s12213-014-0076-x>
- [24] Mead, C., Conway, L.: *Introduction to VLSI systems*, vol. -1. Addison-Wesley Publishing Co. (01 1980)
- [25] Neary, T., Woods, D.: Four small universal turing machines. In: Durand-Lose, J., Margenstern, M. (eds.) *Machines, Computations, and Universality*. pp. 242–254. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)
- [26] Neary, T., Woods, D.: Four small universal turing machines. *Fundamenta Informaticae* **91**(1), 123–144 (2009). <https://doi.org/10.3233/FI-2009-0036>
- [27] Parberry, I.: Single-exception sorting networks and the computational complexity of optimal sorting network verification. *Mathematical Systems Theory* **23**, 81–93 (12 1990). <https://doi.org/10.1007/BF02090767>
- [28] Parberry, I.: The pairwise sorting network. *Parallel Processing Letters* **2**, 205–211 (09 1992). <https://doi.org/10.1142/S0129626492000337>
- [29] Schmidt, A., Manzoor, S., Huang, L., Becker, A.T., Fekete, S.P.: Efficient parallel self-assembly under uniform control inputs. *IEEE Robotics and Automation Letters* **3**(4), 3521–3528 (2018). <https://doi.org/10.1109/LRA.2018.2853758>
- [30] Shirai, Y., Osgood, A.J., Zhao, Y., Kelly, K.F., Tour, J.M.: Directional control in thermally driven single-molecule nanocars. *Nano Letters* **5**(11), 2330–2334 (2005). <https://doi.org/10.1021/nl051915k>, pMID: 16277478