A Simple Proof that Ricochet Robots is PSPACE-complete*

Jose Balanza-Martinez Angel A. Cantu Robert Schweller Tim Wylie

Department of Computer Science
University of Texas - Rio Grande Valley
Edinburg, TX 78539-2999, USA
{robert.schweller, timothy.wylie}@utrgv.edu

Abstract

In this paper, we seek to provide a simpler proof that the relocation problem in Ricochet Robots (Lunar Lockout with fixed geometry) is PSPACE-complete via a reduction from Finite Function Generation (FFG). Although this result was originally proven in 2003, we give a simpler reduction by utilizing the FFG problem, and put the result in context with recent publications showing that relocation is also PSPACE-complete in related models.

1 Introduction

In this paper we study the complexity of relocation within the puzzle-game Ricochet Robots [10], which is equivalent to the game Lunar Lockout with fixed geometry [12] (see Figure 1 for pictures). The Ricochet Robots puzzle consists of a 2D grid *board* containing polyomino obstacles and a collection of unit-size robots placed on the board. The player may select any robot and move it maximally in any of the four cardinal directions. With this basic operation, the goal (relocation problem) is to move a target robot to a target goal location on the board.

While deceptively simple, even the single-player puzzle version of this game has proven to be quite complex, with online solvers being written to help develop solving strategies [4, 9, 11]. The relocation problem with Lunar Lockout was originally shown to be NP-hard in 2001 in [7]. With fixed geometry, it was shown to be PSPACE-complete in 2003 in [5]. Recent work has also focused on some parameterized results for Ricochet Robots [6].

The proof of PSPACE-hardness was proven by showing any polynomial-space Turing machine can be transformed into an instance of Lunar Lockout with fixed geometry [5]. In this paper, we show a simpler proof by reducing from Finite Function Generation [8], which was proven to be PSPACE-complete in 1977.

This work fits into a larger landscape of "Tilt" problems that have received recent interest. Given a 2D board with both open locations and blocked locations, as well as a set of unit-size robots placed at open locations on the board, the question of relocating a particular robot to a particular location has been studied under two fundamental variants: *global* signals versus *local* signals, and *unit steps* versus maximal *full steps*. In the case of global signals, each move (in one of the four cardinal directions) moves ALL robots on the board in the specified direction. In terms of step distance, in the *unit step* variant, a moved robot takes just a single step in the specified direction, while in the *full step* variant each robot moves maximally in the specified direction until a wall or another robot is encountered. Together, these variants create four natural versions of the relocation problem for consideration.

In the case of global signals and full steps, the relocation problem was recently shown to be PSPACE-complete in SODA 2020 [1]. In the case of global signals with single-steps, the problem has also recently been shown to be PSPACE-complete [3]. For local signals and single-steps, the problem is easily solved in polynomial time. However, recent work by Brunner, Chung, Demaine, Hendrickson, Hesterberg, Suhl, and Zeff [2] have considered an interesting variant in which each piece has a provided path that it must travel along, making the problem PSPACE-complete. Within this landscape, the remaining of the four natural tilt models is Ricochet Robets with local signals and full steps. A summary of these results is provided in Table 1.

2 Model 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 $B = (O, W)$ of $\{(x, y)|x \in \{1, 2, \ldots, m\}, y \in \{1, 2, \ldots, n\}\}$ where

*This research was supported in part by National Science Foundation Grants CCF-1817602 and CCF-2329918.

(a) Lunar Lockout (b) Ricochet Robots

Figure 1: The board games Lunar Lockout and Ricochet Robots.

Tilt Model	Signal	Polyominoes	Complexity	Reference
Unit	Single	1×1	P	
	Single (paths)	1×1	PSPACE-complete	[2]
	Global	1×1	PSPACE-complete	[3]
Full	Single	1×1	PSPACE-complete	[5], Thm. 3.5
	Global	1×1	PSPACE-complete	[1]

Table 1: An overview of related models and the complexity of the relocation problem. All have concrete or fixed polyominos that can not move in a connected board. The Single (paths) model requires that every tile moves along a specific path given for each tile. This paper gives Theorem 3.5, which proves relocation in full tilt with single signaling.

O denotes a set of *open* locations, and W denotes a set of *blocked* locations- referred to as "concrete" or "walls." Based on a geometric hierarchy [1], here we create a *connected* board[1]. A board is said to have *connected* geometry if the set of open spaces O for a board is a connected shape.

Tiles. A tile is a labeled unit square centered on a non-blocked point on a given board. Formally, a tile is an ordered pair (c, a) where c is a coordinate on the board, and a is a tile label. In this work we have no attachments between tiles and simply use it as an identifying label.

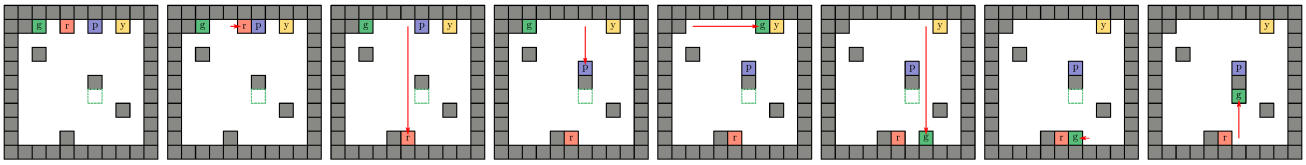
Configurations. A configuration is an arrangement of tiles on a board such that there are no overlaps among tiles, or with blocked board spaces. Formally, a configuration $C = (B, P = \{P_1, \ldots, P_k\})$ consists of a board B, along with a set of non-overlapping tiles P that each do not overlap with the blocked locations of board B.

Particle Step. A *particle step* is a way to turn one configuration into another by way of a signal that moves a tile t in a configuration one unit in a direction $d \in \{N, E, S, W\}$ when possible without causing an overlap with a blocked location or another tile. If a configuration does not change under a step transition for tile t in direction d, we say the configuration is $d(t)$-*terminal*.

Particle Tilt. A *particle tilt* in direction $d \in \{N, E, S, W\}$ for a configuration is executed by repeatedly applying a particle step in direction $d \in \{N, E, S, W\}$ on the same tile t until a $d(t)$-terminal configuration is reached. We denote this as $d(t)$. We say that a configuration C can be *reconfigured in one move* into configuration C' (denoted $C \rightarrow_1 C'$) if applying one particle tilt on a tile t in some direction d 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 particle tilts.

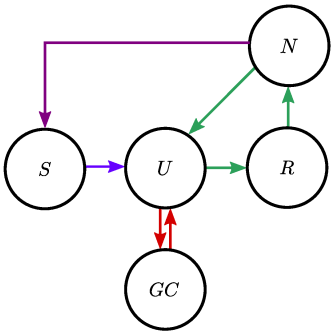
Particle Tilt Sequence. A *particle tilt sequence* is a sequence of particle tilts. For a given tile, the sequence can be inferred from a series of directions $D = \langle d_1, d_2, \ldots, d_k \rangle$; each $d_i \in D$ implies a particle tilt in that direction on a tile. For simplicity, when discussing a particle tilt sequence on a specific tile, we just refer to the

[1]Note that within this model, every board geometry could be considered as rectangular and the blocked spaces are simply robots that are never given a move signal. However, for the reduction, these must be non-movable robots and thus we adhere to the definition of a blocked space instead.



(a) Start (b) E(r) (c) S(r) (d) S(p) (e) E(g) (f) S(g) (g) W(g) (h) N(g)

Figure 2: Ricochet Robots relocation example. Given the starting configuration (Start), a specific tile (green tile g), and a specific location (outlined), this provides a sequence of particle tilts that places g in the location. The tiles are labeled g, r, p, y based on their respective colors of green, red, purple, and yellow. The grey tiles are all blocked locations that can not be moved.



(a)

Com.	Description
1. S	Select function for the first element in the domain.
2. U	Unlock next element.
3. R	Input the element to its lock gadget.
4. N	Input unlocked element to the function determined by previous element.
5. G	Input element to the goal gadget and move to next element.

(b)

Figure 3: (a) Flowchart of the relocation process. Purple paths lead to a state that can only be reached with the last domain element. Blue paths lead to states that can only be reached with the first domain element. Red paths can be traversed only once per element. (b) Descriptions of each state in the flowchart.

series of directions from which that sequence was derived. Given a starting configuration, a particle tilt sequence corresponds to a sequence of configurations based on the tilt transformation. An example particle tilt sequence $\langle E(r), S(r), S(p), E(g), S(g), W(g), N(g)\rangle$ and the corresponding sequence of configurations can be seen in Figure 2.

Ricochet Robots. For this paper, we refer to this tilt model as Richochet Robots for convenience.

Finite Function Generation. In our result we make use of *finite function generation* which was shown to be PSPACE-complete in [8]. Let A be a set with N elements, $F = \{f_1, \ldots, f_k\}$ be a finite set of maps $f_i : A \to A$, and $h : A \to A$. Is h generated by some sequence of compositions $s = f_i \circ \cdots \circ f_n$ *s.t.* $\{f_i, \ldots, f_n\} \in F$? More succinctly, $FFGEN = \{F, s, h\}$ $|h$ is generated by some sequence of compositions s whose elements exist in F$\}$.

3 Complexity of Ricochet Robots

Here, we give a reduction from *finite function generation* in which, given m functions $f_i : X \to X$ where $|X|$ is polynomial in size and $1 \leq i \leq m$, and a function $g(x) : X \to X$, is g the composition of some sequence of f_i's? We first discuss the reduction framework in detail.

3.1 Reconfiguration Framework for Individually Controlled Particle Systems

We will employ a framework Figure 3 that will force the elements in the system through the same function, else h cannot be generated. This framework works by selecting a function and inputting the first element through it with a *function selector* gadget. After this, we will need the element to "unlock" the next element in the domain. This next element cannot be input into its function if it is not unlocked first. Attempting to force an element to its function selector without unlocking it first would render the system not reconfigurable. We achieve this with the use of *lock* gadgets which are also used to determine which function the unlocked element will be input through. This process is then repeated with all the elements in the system. Once all the elements have been input through a function, and the first element has been unlocked by the last, it is used to determine which function will follow next. This forces that all element to be input through the same function the same number of times. The elements can only be input

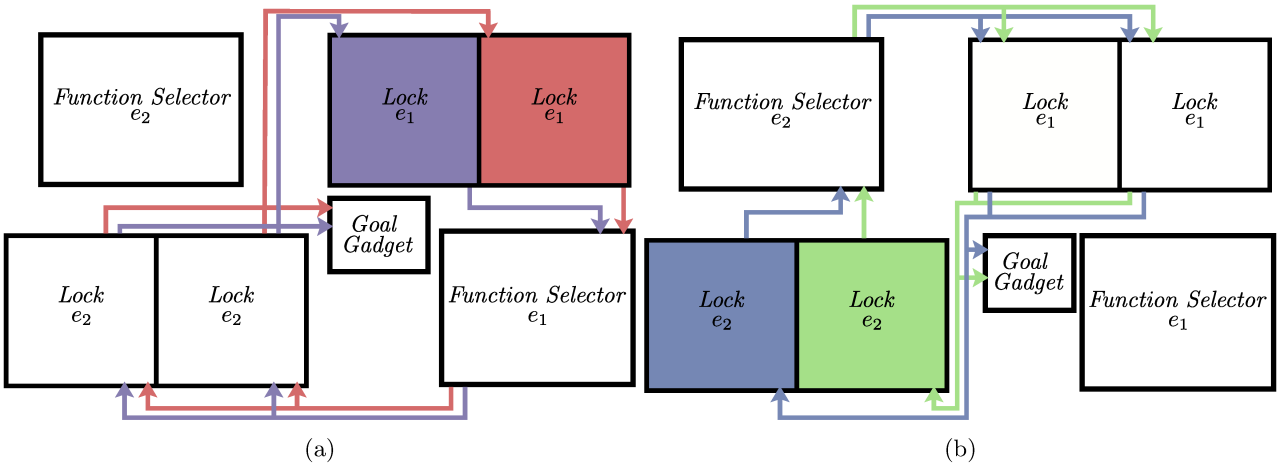
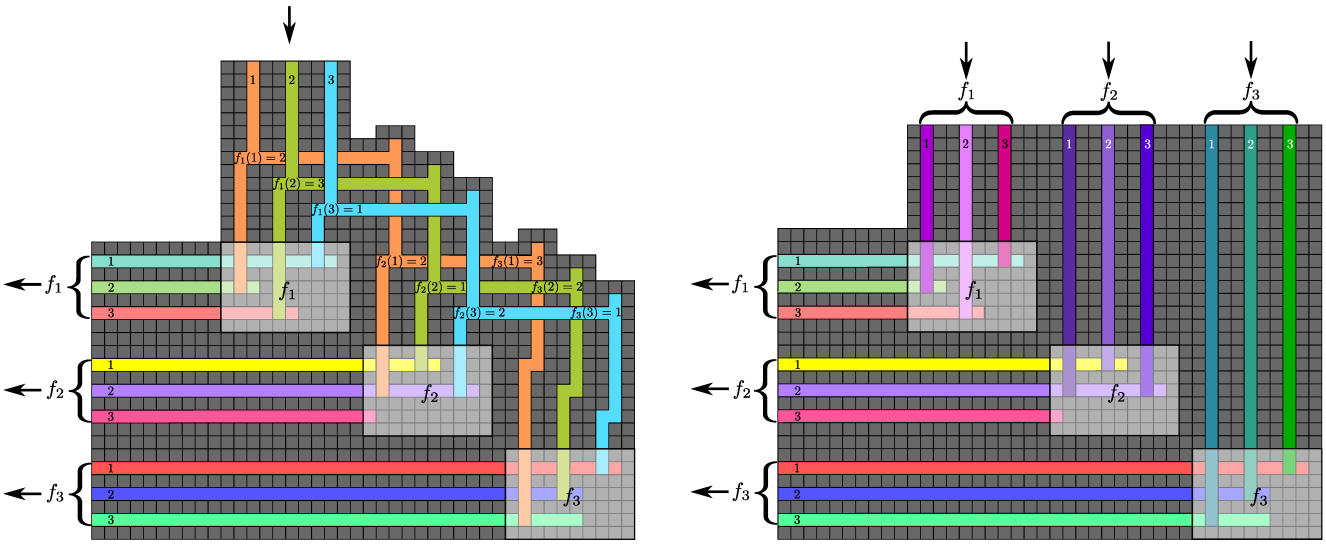


Figure 4: Example of a two element system.



(a) Function Selector

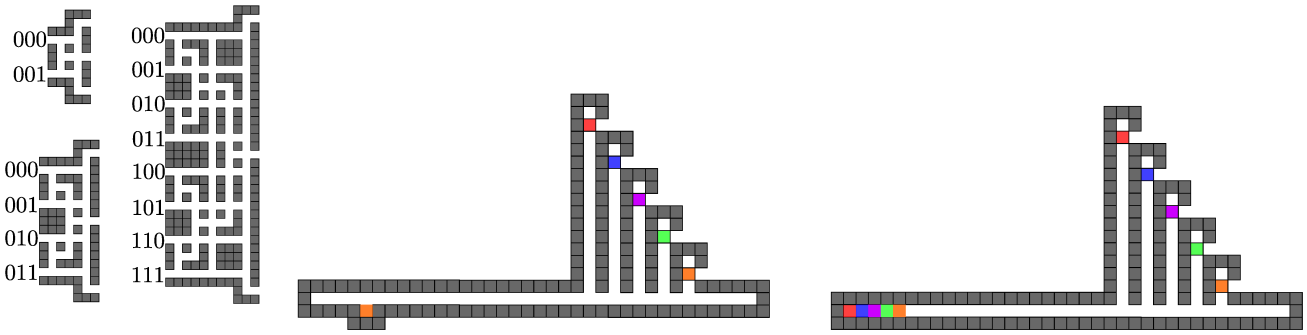
(b) Function Enforcer

Figure 5: An example of function selector and function enforcer gadgets with domain $D = \{1, 2, 3\}$ and functions f_1, f_2, f_3 *s.t.* $f_1(1) = 2, f_1(2) = 3, f_1(3) = 1, f_2(1) = 2, f_2(2) = 1, f_2(3) = 2, f_3(1) = 3, f_3(2) = 2, f_3(3) = 1$. The arrows represent the possible input and output of the gadgets.

into a *goal gadget* after unlocking the next element, but before being input into their lock gadget. This ensures that all the tiles will be input through the same function before being input to the goal gadget.

As an example, we can refer to Figure 4. Here, e_1 is located in either of its two lock gadgets and the element is not locked. We perform step S by inputting e_1 to its function selector gadget and choosing a function. Once the function has been chosen and e_1 has been mapped to its new value, we perform U and unlock e_2. For that we need to choose which lock gadget e_2 is located at with the use of a *lock selector* gadget which can be seen in the appendix. Once we have unlocked e_2 we perform R and return e_1 to its lock gadget. Now we can perform N and input e_2 to its function selector, and perform U and R respectively to unlock e_1 and return e_2 to its lock gadget. Since e_2 is the last gadget, we can now perform S with e_1 again. However, if we wanted to input the elements to the goal gadget, we would do so before returning the element to its lock gadget.

For each element in the domain A, there is one function selector gadgets and $|A|$ lock gadgets. Each lock gadget represents an element's possible value. Since an element might match to any of the domain inputs, there must be a lock gadget for each domain value. Therefore, an element's lock gadget represents the element's value and is mapped to the correct input in its function selector. However, once an element has been output through a function selector, it has to be relocated to the lock gadget where the next element resides. Since the next element could reside in any of its lock gadgets we utilize a *lock selector* gadget to choose between the next element's lock gadgets. There are



(a) (b) (c) Relocation Goal (d) Reconfiguration Goal

Figure 6: (a-b) An example of lock selector gadgets of a system with domain of size 2, 4, and 8. (c) Goal gadget for relocation. (d) Goal gadget for reconfiguration.

$|F| \times |A|$ lock selectors, one for every element in every function. This gadget connects the element's function selector or enforcer, lock gadgets and the next element's lock gadget.

3.2 Reconfiguration Under Ricochet Robots is PSPACE-Complete

Ricochet Robots systems use particle tilts, and motion is not universal. Formally, given two configurations $C = (B, P)$ and $C_{success} = (B, P')$, does there exist a tilt sequence such that $C \rightarrow_* C_{success}$? More formally, to represent an $FFGEN$ system we will use a set of configurations A_C and let $F_C = \{f_{C1}, \ldots, f_{Ck}\}$ be a set of maps $f_{Ci} : A_C \rightarrow A_C$, $h_C : A_C \rightarrow A_C$, and $s_C = f_{Ci} \circ \cdots \circ f_{Cn}$ $s.t.$ $\{f_{Ci}, \ldots, f_{Cn}\} \in F_C$. An initial configuration C can be reconfigured into $C_{success}$, iff $h_C(C) = C_{success}$ and $s_C(C) = C_{success}$. If the above is not true, the system would result in a terminal configuration C_{fail} where $Cfail \not\rightarrow_* C_{success}$. The tilts used to move the tile through the function selector and function enforcer gadgets can be seen as an element being input through different functions. One would be able to reconfigure the system to the goal configuration if and only if h_C was generated by some sequence $f_{Ci} \circ \cdots \circ f_{Cn}$. Finally, we can be more specific and talk about tiles being input and output into gadgets, since moving tiles and changing configurations is equivalent. However, it is easier to explain some gadgets when talking about tiles as input, instead of configurations.

3.2.1 Function Selector

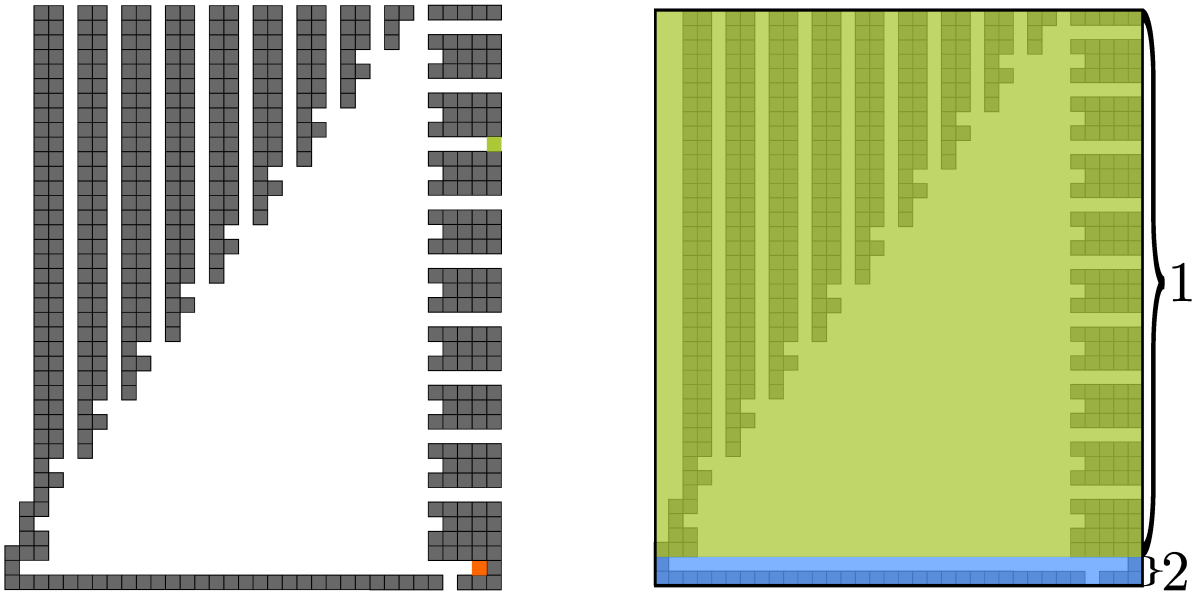
Our function selection gadget takes a tile as input through a *mapping tunnel* that dictates the input value to that function, and through a series of particle tilts selects the function all the tiles will be input through. It will output the tile through a mapping tunnel that indicates the function and the resulting value. This gadget has $|A|$ input mapping tunnels to represent all the possible values an element can have and $|F| \times |A|$ output mapping tunnels to represent the possible values and to indicate the function for the rest of the elements. Once a tile has been output, it cannot enter the function selector again through the output mapping tunnel. An example is seen in Figure 5, we can see how elements are mapped to others and a function is selected when being input through the function selector. The Particle Tilt Sequence for traversal of this gadget is SF found in Figure 9.

3.2.2 Function Enforcer

A *function enforcer* gadget is responsible for enforcing the rest of the domain elements are input through the same function. For this gadget there are $|F| \times |A|$ mapping tunnels, since it is necessary to preserve the function chosen by the function selector. It similarly as the function selector, except a function cannot be chosen. The Particle Tilt Sequences for traversal of this gadget is EF found in Figure 9.

3.2.3 Lock Selector

Once an element exits through a function selector it will be input to one of $|A|$ *lock selectors*. The purpose of these selectors is to input the element to the next element's lock gadget. Since there are $|A|$ lock gadgets, which preserve the value of its domain element, the element can be located in any of these lock gadgets. Once you have unlocked the next element, the current element has to be input back to the lock selector to reach its own lock gadget. The



(a) (b)

Figure 7: (a) Lock Gadget with labeled functions and sample imputs. The paths of the locked tile(orange) and the unlocking tile (light green). The unlocked tile cannot complete its traversal without the unlocking tile. (b) Lock Gadget with labeled sections. Section one is the area where the unlocking tile can exist. Section two is the intersection between section one and two, and is the area where the unlocking tile and the locked tile interact. It is also the area where the locked tile can exist.

lock selectors also preserve the value of the element by inputting it to the lock gadget that corresponds to the correct value. An example of lock selectors for different domain sizes can be seen in Figure 6.

3.2.4 Lock Gadget

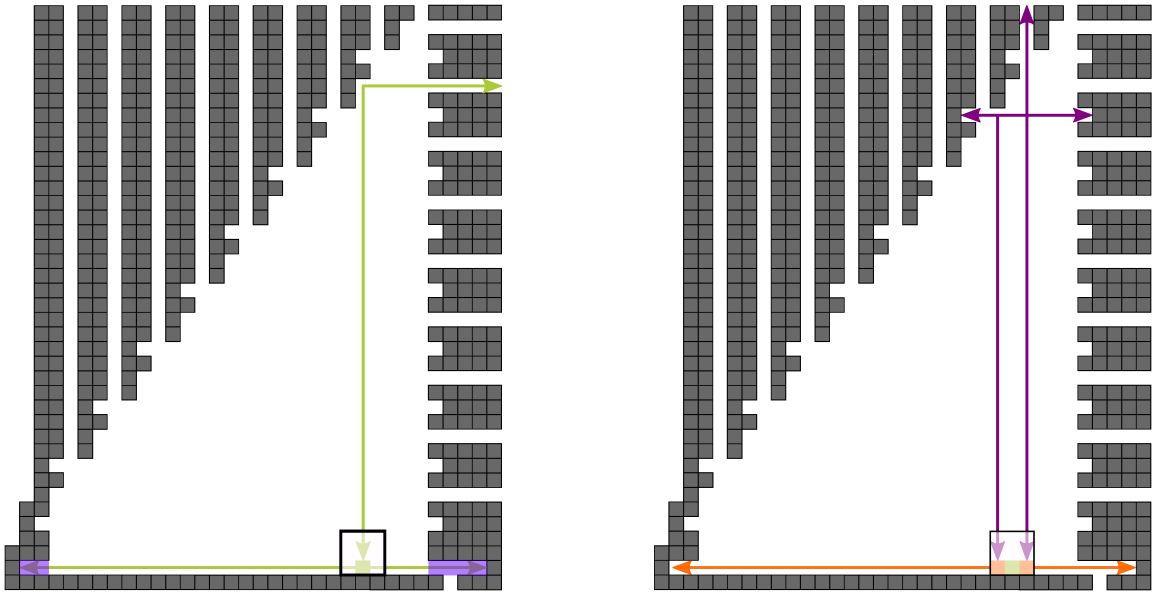
The *lock* gadget is essential in the construction of the proof because it forces the input of all elements through the same function. Its input is two tiles, one for the element to be unlocked and one for the proceeding element performing the unlock. The first tile enters the gadget from an *enforcer tunnel* and must be located in the gadget before the second tile enters the gadget through one of $|F| \times |A|$ enforcer tunnels. The enforcer tunnels ensure that all tiles input and output from the gadget maintain their value and dictates what function the unlocked tile will be input to. The locked tile can only move through the gadget if and only if the unlocking tile is in the corresponding position. If not intended particle tilts are performed the tiles will become trapped and will never be able to leave the lock gadget.

Lemma 3.1. An unlocking tile can only exit a lock gadget through its input mapping tunnel.

We prove this by showing all the possible paths an unlocking tile can take inside a lock gadget, and showing that any particle tilt sequence for the unlocking tile that is not defined in Figure 9 either makes the unlocking tile stuck or does not affect the gadget's output. Figure 8a shows all the possible paths an unlocking tile could take. Since the unlocking cannot use the locked tile to reach any positions it is not supposed to reach, any unexpected particle tilt sequence would lead to a position where the unlocking tile gets stuck. These positions are indicated in section two of Figure 8a.

Lemma 3.2. A locked tile can only exit a gadget through its output mapping tunnel.

We prove this similarly to 3.1 by showing all the possible paths a locked tile can take inside a lock gadget, and showing that any particle tilt sequence for the locked tile that is not defined in Figure 9 either makes the locked tile stuck or does not affect the gadget's output. Figure 8b shows all the possible paths for a locked tile. The locked tile can only interact with the unlocking tile in one section, and there is only one path that would let the locked tile leave the gadget.



(a) (b)

Figure 8: (a) Possible paths for an unlocking tile. The green paths indicate the paths the locked tile can reach by itself. The blue areas indicate when a tile becomes stuck. The highlighted section is where tiles are meant to interact. (b) Possible paths for an unlocking tile. The highlighted section is where tiles can interact. The orange paths indicate the paths the locked tile can reach by itself. Purple paths indicate the paths the locked tile can reach with the help of the unlocking tile. The blue areas indicate when a tile becomes stuck.

3.2.5 Relocation Goal Gadget

The relocation goal gadget is rather simple (see Figure 6c). It consists of the goal for the last robot in the domain, and ensures that the robot must be moved into its goal area last. The theorem follows from Lemmas 3.3 and 3.4, and Lemma 3.3 follows from the correctness of the presented construction.

Lemma 3.3. Relocation within Ricochet Robots with fixed polyominoes and 1×1 movable polyominoes is PSPACE-hard [5].

Lemma 3.4. Relocation within Ricochet Robots with fixed polyominoes and 1×1 movable polyominoes is in PSPACE [5].

Theorem 3.5. Relocation within Ricochet Robots with fixed polyominoes and 1×1 movable polyominoes is PSPACE-complete [5].

3.2.6 Reconfiguration Goal Gadget

The reconfiguration goal gadget is almost the same as the reconfiguration gadget (see Figure 6d). It consists of the goal areas for all the robots in the domain, and ensures that the robots must be moved into their goal areas in the order that they appear in the domain.

Corollary 3.6. Reconfiguration within Ricochet Robots with fixed polyominoes and 1×1 movable polyominoes is PSPACE-complete.

4 Conclusion

In this paper we gave a simpler reduction to show that relocation and reconfiguration in the tilt model with addressable robots (Ricochet Robots or Lunar Lockout with fixed blocks) is PSPACE-complete even when all particles are 1×1 tiles. This question is still open if there is no fixed geometry (or fixed robots). Is there a way to simulate some fixed geometry with other pieces?

Function Selector and Function Enforcers Sequences

Name	Tilt Sequence	Description
SF	$\langle S\rangle + \langle E, S\rangle^i + \langle W, S, W\rangle$	Selects the function all the elements the domain will be input to. i specifies the function to be selected.
EF	$\langle S, E\rangle$	Enforces all elements are input through the same function as the first element.

Lock Selector Sequences

T_0	$\langle W, N\rangle$	Selects 0 for that bit position.
T_1	$\langle W, S\rangle$	Selects 1 for that bit position.
RF	$\langle N, E\rangle$	Inputs tile to function selector or function enforcer gadget after returning from lock gadget.
G	$\langle S, E\rangle$	Inputs tile to goal gadget after returning from lock gadget.

Lock Sequences

P	$\langle W, S\rangle$	Positions unlocking tile for unlock.
U	$\langle W, N\rangle$	Unlocks locked tile.
R	$\langle N, E\rangle$	Returns tile to lock selector.

Figure 9: Particle Tilt Sequences.

References

[1] Jose Balanza-Martinez, Timothy Gomez, David Caballero, Austin Luchsinger, Angel A. Cantu, Rene Reyes, Mauricio Flores, Robert T. Schweller, and Tim Wylie, *Hierarchical shape construction and complexity for slidable polyominoes under uniform external forces*, Proc. of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA'20, 2020, pp. 2625–2641.

[2] Josh Brunner, Lily Chung, Erik D. Demaine, Dylan Hendrickson, Adam Hesterberg, Adam Suhl, and Avi Zeff, *1 x 1 Rush Hour with Fixed Blocks is PSPACE-complete*, ArXiv e-prints (2020), arxiv:2003.09914.

[3] David Caballero, Angel A. Cantu, Timothy Gomez, Austin Luchsinger, Robert Schweller, and Tim Wylie, *Relocating units in robot swarms with uniform control signals is pspace-complete*, The 32nd Canadian Conference on Computational Geometry, CCCG'20, 2020.

[4] Kevin Cox, *Ricochet robots solver*, `ricochetrobots.kevincox.ca`, 2023, Accessed: 2024-02-20.

[5] Jeffrey R. Hartline and Ran Libeskind-Hadas, *The computational complexity of motion planning*, SIAM Review **45** (2003), no. 3, 543–557.

[6] Adam Hesterberg and Justin Kopinsky, *The parameterized complexity of ricochet robots*, Journal of Information Processing **25** (2017), 716–723.

[7] Martin Hock, *Exploring the complexity of the ufo puzzle*, Master's thesis, 2001, Carnegie Mellon University.

[8] Dexter Kozen, *Lower bounds for natural proof systems*, 18th Annual Symposium on Foundations of Computer Science (sfcs 1977), IEEE, 1977, pp. 254–266.

[9] John Noel, *Ricochet robots solver*, `github.com/johnnoel/ricochet-robots-solver`, 2021, Accessed: 2024-02-20.

[10] Alex Randolph, *Ricochet robots*, 1999, Rio Grande Games.

[11] Carl Scherer and Oli Scherer, *Ricochet robots solver*, `github.com/Lireer/ricochet-robot-solver`, 2016, Accessed: 2024-02-20.

[12] Hiroshi Yamamoto, *Lunar lockout*, 2000, Binary Arts.