Agent Motion Planning as Block Asynchronous Cellular Automata: Pushing, Pulling, Suplexing, and More

MIT Hardness Group*, Hayashi Ani[1], Josh Brunner[1], Erik D. Demaine[1], Jenny Diomidova[1], Timothy Gomez[1], Della Hendrickson[1], Yael Kirkpatrick[1], Jeffery Li[1], Jayson Lynch[1], Ritam Nag[1], and Frederick Stock[2]

[1] MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar St., Cambridge, MA 02139, USA,
{joshuaa,brunnerj,edemaine,diomidova,tagomez7,della,yaelkirk,jeli,jaysonl,rnag}@mit.edu
[2] University of Massachusetts, Lowell, MA 01854, USA,
{Frederick_Stock}@student.uml.edu

Abstract. In this paper, we explore how agent reachability problems from motion planning, games, and puzzles can be generalized and analyzed from the perspective of block asynchronous cellular automata, inspired by asynchronous cellular automata, surface chemical reaction networks, and similar rewriting dynamical systems. Specifically, we analyze square grids with three or four cell types (states) that model motion planning with blocks: an agent which occurs uniquely, empty space, blocks, and (optionally) fixed walls. The agent can freely exchange with (walk through) empty space, and can interact with blocks according to a single local asynchronous 3-cell replacement rule; the goal is for the agent to reach a specified destination (reachability). This setting generalizes well-studied motion-planning problems such as Push-1F and Pull-1F, which are known to be PSPACE-complete. We analyze all 40 possible 3-cell replacement rules (22 that conserve blocks so are naturally in PSPACE, and 18 that create or destroy blocks so are naturally in NP), and except for a few open problems, characterize their complexity — ranging from L to NL to as hard as Planar Monotone Circuit Value Problem to P-complete to NP-complete to PSPACE-complete.

Keywords: Asynchronous Cellular Automata · Reductions · Motion Planning

1 Introduction

Cellular automata have been studied extensively in computer science, along with many variations. A traditional ***cellular automaton*** [34, 21, 30, 22] is defined by a set of possible ***states*** for cells of a grid, and a local ***replacement rule*** for how

* Artificial first author to highlight that the other authors (in alphabetical order) worked as an equal group. Please include all authors (including this one) in your bibliography, and refer to the authors as "MIT Hardness Group" (without "et al.").

the state of every cell evolves over time as a function of the states of its local neighborhood (including the cell itself). The most famous examples are Conway's Game of Life (2D) [6] and Rule 110 (1D) [7, 26], which are each Turing-complete in certain senses.

In a traditional cellular automaton, all cells update in lock step, synchronized by a global clock. In ***asynchronous*** cellular automata [15, 13, 19, 18, 27, 17], any cell can update at any time. Updating two neighboring cells in different orders can produce different results, as each cell's update can depend on the other cell's state. Thus the evolution of an asynchronous cellular automaton is nondeterministic. For example, asynchronous cellular automata capture the abstract Tile Assembly Model (aTAM) [35, 1], where a cell can transition from empty to having a particular tile when certain other tiles are neighbors.

Block cellular automata [24, 32, 14] use a different kind of update rule: they define a replacement rule on a constant-size ***block*** of cells (e.g., two consecutive cells in 1D, or a 2×2 square of four cells in 2D) by specifying the new state of the entire block as a function of its old state. In a synchronous cellular automaton, block updates require partitioning the cells into blocks, updating each block in lock step, and then repeating with a shifted version of the partition so that all neighboring cells eventually interact.

Block asynchronous cellular automata. In this paper, we explore *block asynchronous* cellular automata, apparently for the first time. As in block cellular automata, the replacement rule specifies the new state of an entire constant-size block of cells as a function or relation of their old states. But unlike synchronous automata, we do not need a partition of cells into blocks. Instead, as in asynchronous cellular automata, the replacement rule can be applied nondeterministically to any one block at any time.

Block asynchronous cellular automata directly model many existing dynamical systems, with applications to DNA computing, modular robotics, and puzzles and games:

1. Surface Chemical Reaction Networks (sCRNs) [28, 3] are 2D block asynchronous cellular automata with rotatable 1×2 blocks. In other words, an sCRN specifies rules of the form $AB \to A'B'$, where A, B, A', B' are possible states, which can be applied to any two (horizontally or vertically) neighboring cells. There may be multiple rules of the form $AB \to \cdots$, so they form a relation instead of a function, adding to the nondeterminism of the system.
2. Friends-and-strangers graphs [8, 25] are graphical block asynchronous cellular automata with blocks defined by a graph edge. More precisely, for every two friends A, B, we have a ***swap rule*** $AB \to BA$ that can be applied to any two adjacent vertices of the graph.
3. The Fifteen Puzzle and related $n^2 - 1$ puzzles [29, 12] are 2D block asynchronous cellular automata with rotatable 1×2 blocks. Here we have a swap rule $Ae \to eA$ for every state A, where e is one (uniquely occurring) state representing the empty space.

4. Modular pivoting robots [2] are 2D or 3D block asynchronous cellular automata with rotatable constant-size blocks of side length up to 3. Specifically, the rules allow a robot to move from one position to another (similar to $Ae \to eA$ swap rules), but use a larger block to guarantee that certain neighboring cells are empty to avoid collisions during the motion.
5. The abstract Tile Assembly Model (aTAM) [35, 1] is a 2D block asynchronous cellular automaton with non-rotatable constant-size blocks involving a cell and its four cardinal neighbors. The replacement rule specifies how a tile can attach to a growing assembly, modifying only one cell but depending on the neighbors.
6. The puzzle game Lights Out! [16] is a 2D block asynchronous cellular automaton, where a block is a cell and its four cardinal neighbors, and the replacement rule flips the state of all five cells. Similarly, the generalization to graphs [16] is a graphical block asynchronous cellular automaton.

Our definition of block asynchronous cellular automata is inspired by these various applications, and seems natural to study more broadly. Many applications (e.g., 1–3 above) involve just two-cell blocks.

Our results. In this paper, we study the natural extension of block asynchronous cellular automata with *three-cell blocks*. Specifically, we are motivated by two well-studied agent-based motion-planning games involving reconfiguration of movable blocks, Push-1F and Pull?-1F, which are known to be PSPACE-complete [5, 4]. In both games, an agent (the player) can traverse empty cells of a 2D grid of square cells, where each cell can be empty, contain a movable block, or be a fixed wall, and the goal is for the agent to reach a particular cell. In Push-1F (which has the same dynamics as the famous Sokoban puzzle game), the agent can push a neighboring block, provided that the cell on the other side of the block is empty. In Pull?-1F, the agent can (but does not have to) pull a neighboring block, provided the cell on the other side of the agent is empty. We can model these games as block asynchronous cellular automata with rotatable 1×2 and 1×3 blocks and the following replacement rules:

Push-1F	**Pull?-1F**	
$Ae \to eA$	$Ae \to eA$	(agent A can move through empty space e)
$ABe \to eAB$	$eAB \to ABe$	(agent A can push/pull a block B into empty space e)

The rules do not involve fixed walls F, which captures the desired property that the agent and blocks cannot move into such cells.

The main goal of this paper is to characterize the complexity of ***reachability*** (whether the agent can reach a particular cell) in all block asynchronous cellular automata with a single three-cell block rule involving the same four states: agent A, empty e, movable block B, and fixed wall F. Notably, we require that the agent state A appears uniquely at all times in the configuration, as in Push-1F and Pull?-1F and other agent-based motion-planning problems, and that the fixed walls F are ***frozen*** (never change state). Push-1 was recently shown to be

hard even without fixed walls F [20], so we also consider the complexity with just three states $\{A, e, B\}$, forbidding fixed walls F. Finally, we assume that two-cell movement rule $Ae \to eA$ is always present, and characterize the behavior of an arbitrary single three-cell "game" rule.

We provide an almost complete complexity-theoretic landscape for the (single-agent) reachability problem. There are 40 possible single replacement rules for a three-cell block (subject to the constraints above), and two versions of each (fixed blocks allowed or forbidden). We decompose the rules into 22 "conservative" game rules that conserve the number of blocks, and 18 "bounded" game rules that create or destroy a block. Bounded rules naturally lead to polynomially bounded games, as the game rule can be applied a number of times at most linear in the playing area. Tables 1 and 2 summarize our results for conservative and bounded games respectively, which solve all but five of the 80 variations (two of which are equivalent). We also attempt to give each rule a name to provide intuition behind what the agent is doing, like Push-1 modeling an agent "pushing" a block.

More specifically, we show the following:

1. We start in Section 3 by analyzing what we call ***generalized swaps***. These game rules are very weak and in some sense reduce to a swap, so they can be solved in logarithmic space (L or NL), and some are complete for their respective classes.
2. Next, in Section 4, we analyze ***conservative rules*** where the total number of blocks does not change. This includes famous examples like Push-1 and new games such as Suplex-1 ($eAB \to BAe$), where the agent throws the block over its head into the cell behind it.[3] As shown in Table 1, we give a complete characterization for when fixed walls F are allowed, and leave only four open cases without fixed walls. Many cases are PSPACE-complete, while some are surprisingly in P.
3. Next, in Section 5, we cover ***bounded rules*** where the number of blocks monotonically increases or monotonically decreases. For these rules, we prove that each game is either NP-complete or in P. A few games are P-complete.
4. Finally, in Section 6, we provide initial results for ***bendy rules***, where the rule's block can be any path of three adjacent cells, not necessarily a 1×3 rectangle. This rule type is a natural extension of block asynchronous cellular automata (especially in the graphical view), and is motivated by implementations where rigidity is hard to enforce, such as DNA [28].

The connection between cellular automata and agent-based motion planning can be seen in Langdon's Ant [23] which was later considered from a computational complexity perspective [33, 10]. Motion planning is also becoming of increased interest due to experimental constructions such as robots that walk along DNA origami lattices which perform tasks [31].

[3] This name comes from the wrestling move where an opponent is thrown backward over ones head.

Rules	Game	Reachability-F	Reachability
$ABe \to eBA$	Leap	L-complete (Thm. 2)	L-complete (Thm. 2)
$eAB \to AeB$	Trivial	L (Cor. 1)	L (Cor. 1)
$AeB \to eAB$	Trivial	L (Cor. 1)	L (Cor. 1)
$AFe \to eFA$	Vault	L-complete (Thm. 2)	L (Cor. 1)
$AFB \to BFA$	Vault Swap	NL-complete (Thm. 2)	NL (Cor. 1)
$ABF \to BAF$	Push Swap at Fixed	NL (Cor. 1)	NL (Cor. 1)
$BAF \to ABF$	Pull Swap at Fixed	NL (Cor. 1)	NL (Cor. 1)
$ABe \to BAe$	Push Swap at Empty	NP-complete (Thm. 3)	NP-complete (Thm. 3)
$ABe \to BeA$	= Push Swap at Empty	NP-complete (Thm. 3)	NP-complete (Thm. 3)
$ABe \to eAB$	Push-1	PSPACE-complete [5]	PSPACE-complete [20]
$ABe \to AeB$	= Push-1	PSPACE-complete [5]	PSPACE-complete [20]
$eAB \to eBA$	Pull Swap at Empty	PSPACE-complete (Thm. 4)	PSPACE-complete (Thm. 4)
$AeB \to eBA$	= Pull Swap at Empty	PSPACE-complete (Thm. 4)	PSPACE-complete (Thm. 4)
$eAB \to ABe$	Pull?-1 [4]	PSPACE-complete [4]	*OPEN*
$AeB \to ABe$	= Pull?-1	PSPACE-complete [4]	*OPEN*
$eAB \to BAe$	Suplex-1	PSPACE-complete (Thm. 4)	P (Thm. 5)
$eAB \to BeA$	= Suplex-1	PSPACE-complete (Thm. 4)	P (Thm. 5)
$AeB \to BAe$	= Suplex-1	PSPACE-complete (Thm. 4)	P (Thm. 5)
$AeB \to BeA$	= Suplex-1	PSPACE-complete (Thm. 4)	P (Thm. 5)
$ABB \to BBA$	Swap-2	PSPACE-complete (Thm. 4)	*OPEN*
$ABB \to BAB$	Push Swap at Block	PSPACE-complete (Thm. 4)	*OPEN*
$BAB \to ABB$	Pull Swap at Block	PSPACE-complete (Thm. 4)	*OPEN*

Table 1: Our results for conservative game rules. "=" indicates that the rule is equivalent to the specified rule when combined with the movement rule.

2 Definitions

2.1 Block Asynchronous Cellular Automata

Definition 1 (States). *In this paper, we will use a size-4* ***state alphabet*** *with the following state symbols and names:*

A — *the agent*
B — *a movable block*
e — *an empty space*
F — *a fixed wall (allowed only in some models)*

Definition 2 (Board). *A* ***board*** *is an $x \times y$ grid graph where each vertex of the grid is assigned a single state.*

Definition 3 (Subconfiguration). *A* ***subconfiguration*** *of size k is a sequence of k distinct vertices on a board. A* ***connected subconfiguration*** *is a sequence of k distinct vertices where each vertex is adjacent to the next vertex in the sequence. A* ***linear subconfiguration*** *is a connected subconfiguration where all vertices lie either in the same row or same column on the board.*

We reconfigure a subconfiguration on a board using a "rule":

Rules	Game	Reach-F	Reach
$ABB \to Aee$	Push Smash	$\longrightarrow$	NP-c (Thm 8)
$ABB \to eBA$	Leap Crush	$\longrightarrow$	NP-c (Thm 8)
$ABB \to BAe$	Swap Zap	$\longrightarrow$	NP-c (Thm 8)
$ABB \to AeB$	Push Merge	$\longrightarrow$	NP-c (Thm 7)
$ABB \to ABe$	Pull Merge	$\longrightarrow$	NP-c (Thm 9)
$BAB \to eAe$	Suplex Smash	$\longrightarrow$	NP-c (Thm 7)
$BAB \to BAe$	Suplex Merge	$\longrightarrow$	NP-c (Thm 7)
$BAB \to eBA$	Clap Merge	$\longrightarrow$	NP-c (Thm 7)
$eAB \to BBA$	Trail Swap	$\longrightarrow$	NP-c (Thm 8)
$ABe \to BAB$	Trail Push	$\longrightarrow$	NP-c (Thm 7)
$eAB \to eAe$	Battering Ram	P-c (Thm. 11)	NL-c (Thm. 13)
$ABe \to Aee$	Knock Over	in NL (Thm. 16)	in NL (Thm. 16)
$ABe \to BBA$	Trail Leap	$\longrightarrow$	L-c (Thm. 2)
$AFB \to AFe$	Zap through Walls	in P (Thm. 11), PMCVP-hard (Cor. 2)	N/A
$AFB \to eFA$	Vault Crush	NL-c (Thm. 2)	N/A
$ABF \to AeF$	Push into Wall	in NL	N/A
$FAB \to FAe$	Wall Suplex	in NL	N/A
$AFe \to BFA$	Trail Vault	L-c (Thm. 2)	N/A

Table 2: Our results for bounded game rules. "$\longrightarrow$" in the F column indicates that the complexity is the same as the right column, as hardness without fixed walls is a stronger result. "N/A" in the right column indicate that the problem is trivial as the rule cannot be applied without fixed walls.

Definition 4 (Rule). *A **rule** of size k is an ordered pair from $\{A, B, e, f\}^k \times \{A, B, e, f\}^k$. A rule is applied to a board by replacing a length k linear subconfiguration that matches the left side of the rule with the states of the right side of the rule. Rules can be applied regardless of reflection and rotation.*

Definition 5 (Rule System). *A **rule system** is a binary relation over $\bigcup_k \{A, B, e, f\}^k$ specifying rules that can be applied.*

Definition 6 (Agent Based Rule System). *A rule system is **agent-based** if every rule in the rule system contains exactly one A on both the left and right hand sides of the rule.*

When $k = 2$, this model is equivalent to a 4 species Surface Chemical Reaction Network [3]. In the real world, surface chemical reaction networks have a difficult time distinguishing between 3 states in a straight line versus 3 states in an L. This then gives rise to a second class of rule, a "bendy" rule.

Definition 7 (Bendy Rule). *A **bendy rule** of size k is an ordered pair $\{A, B, e, f\}^k \times \{A, B, e, f\}^k$ where a size-k (connected) subconfiguration is replaced by a new subconfiguration of the same size and shape.*

Note that bendy rules do not require linear subconfigurations; the states can instead "bend" making an L shape. When $k = 2$, bendy rules are identical to normal rules. In this paper, a rule will not be bendy unless otherwise specified.

We study the ***Single Agent Reachability Problem***. This is characterized by tracking a special agent state which the rules and starting configuration ensure only one can exist at a time. If a node becomes the state A we say it is occupied by the agent.

Definition 8 (Single Agent Reachability Problem). *Given a board B with a single agent state A, an agent-based rules system R, and a target location t, does there exist a sequence of rules applied to subconfigurations which causes the node at t to be in state A?*

This problem is commonly studied in the context of block pushing puzzles, and therefore we use a naming convention found in block pushing literature. For example, given the rule $ABe \to eAB$, we call Single Agent Reachability problem Push-1 if we do not allow fixed walls F, Push-1F if we allow fixed walls, and Push-1W if we allow ***thin walls***, which can be thought of as removing edges from the board (changing whether we have a full grid, an induced subgraph, or a general subgraph of the grid). We include the name of each rule in the tables. In the future, when we refer to the Single Agent Reachability problem by a rule r we really mean the Single Agent Reachability Problem using the set or rules consisting of r and the ***movement rule***, $\{r, Ae \to eA\}$. For example, in Table 1, the results for the rule $ABe \to eBA$ are in fact results for the Single Agent Reachability Problem for the set of rules $\{ABe \to eBA, Ae \to eA\}$. This is because in typical pushing block puzzles the agent is allowed to move freely unless it comes in contact with a fixed wall or movable block, where in the latter case the agent can then either push or pull the block. Hence, any ruleset that does not include the movement rule is an incomplete representation of these puzzles.

2.2 Gadgets

Almost all of our hardness results reduce from the motion-planning-through-gadgets model [9, 11]. This framework was introduced by a series of papers [9, 11], and attempts to generalize motion planning problems where an autonomous agent attempts to navigate an environment to get between two distinct points.

A ***gadget*** consists of a finite set of ***states***, a finite set of ***locations***, and a finite set of ***transitions*** of the form $(q, a) \to (r, b)$, meaning that, when the gadget is in state q, an agent can enter at location a and exit at location b while changing the gadget's state to r. Given a system of such gadgets, with locations connected together by a graph, the ***reachability*** problem asks whether the agent can start at one specified location and reach another through a sequence of gadget traversals and connections between gadgets.

As pushing-block puzzles are motion-planning problems at their core, we make extensive use of this framework to show the vast majority of our rules are NP-hard or PSPACE-hard.

3 Generalized Swaps

We start by defining a generalized swap and prove that all generalized swaps are in NL and all symmetric generalized swaps are in L. We also prove that many generalized swaps are L-hard and NL-hard. For hardness, we reduce from non-planar graph reachability and thus we need a crossover gadget.

3.1 Rules in L and NL

Definition 9 (Generalized Swap). *Let P denote a subconfiguration that a rule is being applied to. A set of rules is a generalized swap if for each rule:*

1. *The only vertices that change state are the start s and ending location t of an agent, and*
2. *For all other vertices $p \in P \setminus \{s, t\}$ which do not contain a fixed wall F, adding or removing a block B from p does not impact the ability of the agent to move from s to t, though perhaps using different rules from the set.*

Adding or removing blocks may prevent movement through other positions in P, however. A generalized swap is ***symmetric*** *if the rule with the agent starting at t and moving to s is also a generalized swap.*

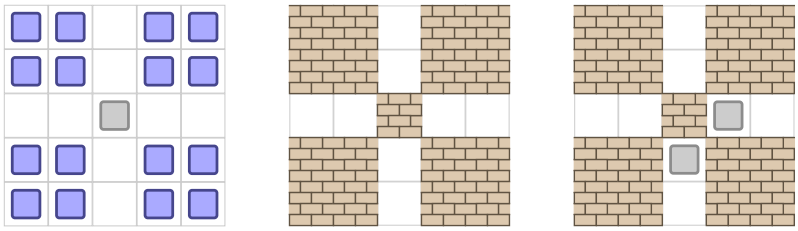
Theorem 1. *Single Agent Reachability with a generalized swap is in NL. If the generalized swap is symmetric, it is in L.*

Corollary 1. *The following games are in L under both the non-bendy setting and the bendy setting:*

- *Trivial ($eAB \to AeB$ and $AeB \to eAB$)*
- *Leap-F ($ABe \to eBA$)*
- *Vault-F ($AFe \to eFA$)*
- *Trail Leap-F ($ABe \to BBA$)*
- *Trail Vault-F ($AFe \to BFA$)*

The following games in in NL under both the non-bendy setting and the bendy setting:

- *Vault Swap-F ($AFB \to BFA$)*
- *Vault Crush-F ($AFB \to eFA$)*
- *Push into Wall-F ($ABF \to eAF$)*
- *Wall Suplex-F ($BAF \to eAF$)*
- *Fixed Wall's Push Swap ($ABF \to BAF$)*
- *Fixed Wall's Pull Swap ($BAF \to ABF$)*



a Leap:
$(ABe \rightarrow eBA)$
Trail Leap:
$(ABe \rightarrow BBA)$

b Vault:
$(AFe \rightarrow eFA)$
Trail Vault:
$(AFe \rightarrow BFA)$

c Vault Swap:
$(AFB \rightarrow BFA)$
Vault Crush:
$(AFB \rightarrow eFA)$

Fig. 1: Undirected crossovers in (a) and (b), and a directed crossover in (c)

3.2 Path Reductions

For L-hardness and NL-hardness we must provide a crossover gadget; whether or not the crossover gadget is directed influences the hardness of the ruleset.

Theorem 2. *The following games are complete for their corresponding complexity classes:*

- *Leap ($ABe \rightarrow eBA$) is L-complete.*
- *Vault-F ($AFe \rightarrow eFA$) is L-complete.*
- *Trail Leap ($ABe \rightarrow BBA$) is L-complete.*
- *Trail Vault-F ($AFe \rightarrow BFA$) is L-complete.*
- *Vault Crush-F ($AFB \rightarrow eFA$) is NL-complete.*
- *Vault Swap-F ($AFB \rightarrow BFA$) is NL-complete.*

4 Conservative Games

We present the complete set of "conservative" rules on three nodes. A rule is ***conservative*** if and only if:

1. The number of movable blocks present on the board before the rule is applied is equal to the number of movable blocks after, and
2. The number of fixed walls is the same before and after applying the rule, and fixed walls do not change position.

There are several rules which are "congruent" to other rules. Two rules R_1, R_2 are ***congruent*** if applying the movement rule to either the left or right hand states of R_1 can produce identical states to R_2. If the movement rule is congruent to a rule R, then R is ***trivial***.

4.1 Rules in NP

Theorem 3. *Single-Agent Reachability for Push Swap at Empty ($ABe \rightarrow BAe$) is NP-complete.*

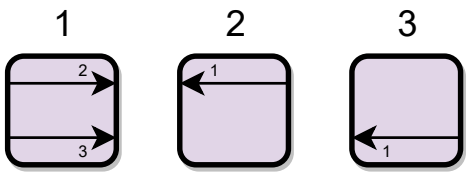
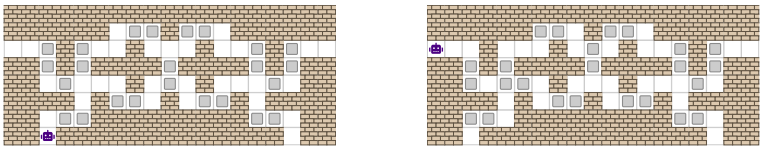


Fig. 2: The Locking 2-Toggle (L2T) gadget and its states from the motion planning framework. The numbers above indicate the state and when a traversal happens across the arrows, the gadget changes to the indicated state.

4.2 PSPACE-complete Rules

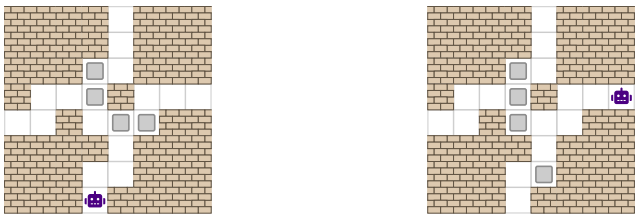
Theorem 4. *Single-agent Reachability for the following rules is PSPACE-complete:*

- *Swap-2 ($ABB \to BBA$)*
- *Pull Swap at Empty ($eAB \to eBA$)*
- *Suplex-1F ($eAB \to BAe$)*
- *Pull Swap at Block ($BAB \to ABB$)*
- *Push Swap at Block ($ABB \to BAB$)*



a Initial state b Locked state

Fig. 3: An L2T gadget for Swap-2 ($ABB \to BBA$) model



a Initial state b Locked state

Fig. 4: An L2T gadget for Suplex-1F ($eAB \to BAe$)

4.3 Polynomial Time Rules

Here we show agent reachability Suplex-1 with no fixed walls is in P. Recall that Suplex-1 is the rule $eAB \to BAe$ (or equivalently, $eAB \to BeA$). We will work with the version with the rule $eAB \to BAe$. We call applications of the this rule (which does not move the agent) ***suplex moves*** to distinguish it from moving the agent with the movement rule. We call empty locations ***holes***; because the agent needs a hole behind it in order to move blocks, keeping track of the location of holes will be key to our algorithm. Call a location ***reachable*** if it is possible for the agent to ever reach that location, and call a move a ***turn*** if it is in a different direction than the previous move (i.e., horizontal after a vertical or vice versa). We will think of the path the agent takes as a sequence of ***segments*** and turns, where a segment is the sequence of consecutive moves in the same direction. Most of the time, when following a path, the agent will alternate between using suplex moves to clear the space in front of them, and then moving into the newly cleared space. Using these ideas we show Suplex-1 without fixed blocks in P.

Theorem 5. *The set of locations S as defined in Lemma* **??** *is exactly the set of reachable locations. Thus Suplex-1 ($eAB \to BAe$) is in P.*

5 Bounded Games

In this section we study what we call "bounded" games. In a ***bounded game***, the total number of movable blocks monotonically increases or decreases with each application of the game rule. We can divide the bounded games into two types: ***Zap***, which deletes blocks in each rule application, and ***Trail***, which adds blocks. The results are outlined in Table 2. We then prove the remaining bounded games are all easy via a greedy algorithm. Finally we investigate which of these games are P-complete. We start with the proof bounded games are all in NP.

Theorem 6. *All bounded games are in NP.*

5.1 NAND Gadgets

In many of these games we get NP-hardness by building a NAND gadget. This gadget has two tunnels which both start open. Traversing either tunnel closes the other and leaves the first open. We use two variants: anti-parallel and crossing. The first is shown in Figure 5. For the last reduction we introduce a new version of the NAND called the ***Delayed NAND***. This NAND adds an extra initial state where the gadget must be "activated" by a button before solving. It is of note that all of these NP-complete reductions do not use fixed walls.

Theorem 7. *The following games are NP-complete:*

- *Push Merge ($ABB \to AeB$)*
- *Bendy Push Merge-F ($ABB \to AeB$)*

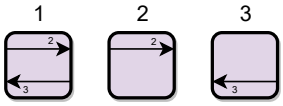
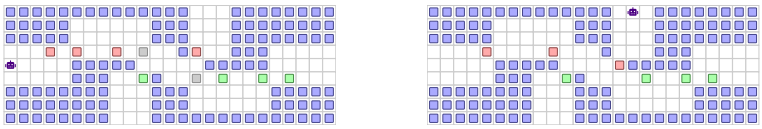


Fig. 5: Antiparallel NAND gadget shown. This gadget is drawn with all transitions directed however some of our gadgets are undirected or mixed, i.e., having both types. In all of these settings the motion planning problem is NP-complete.



a Initial state b Traversed state

Fig. 6: Anti-Parallel NAND for the Clap Merge ($BAB \to eBA$) model

- *Suplex Smash* ($BAB \to eAe$)
- *Bendy Suplex Smash-F* ($BAB \to eAe$)
- *Suplex Merge* ($BAB \to BAe$)
- *Bendy Suplex Merge-F* ($BAB \to BAe$)
- *Clap Merge* ($BAB \to eBA$)
- *Bendy Clap Merge-F* ($BAB \to eBA$)
- *Trail Push* ($ABe \to BAB$)

Theorem 8. *The following rules are NP-complete:*

- *Trail Swap* ($eAB \to BBA$)
- *Swap Zap* ($ABB \to BeA$)
- *Leap Crush* ($ABB \to eBA$)
- *Push Smash* ($ABB \to Aee$)

Theorem 9. *Pull Merge* ($ABB \to ABe$) *is NP-complete.*

5.2 More Easy Cases

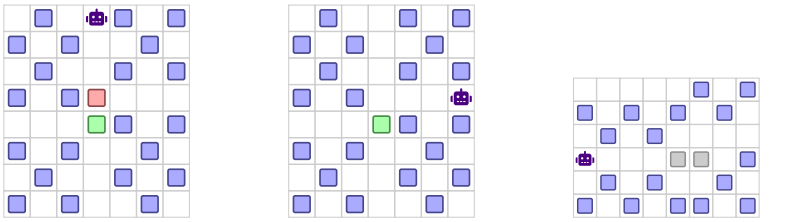
First we will describe a greedy algorithm which gives membership in P for a few games. We only include rules here that aren't covered by log-space algorithms described in Section 3.

Theorem 10. *The following games are in P:*

- *Battering Ram-F* ($eAB \to eAe$)
- *Knock over-F* ($ABe \to Aee$)
- *Zap Through Walls-F* ($AFB \to AFe$)

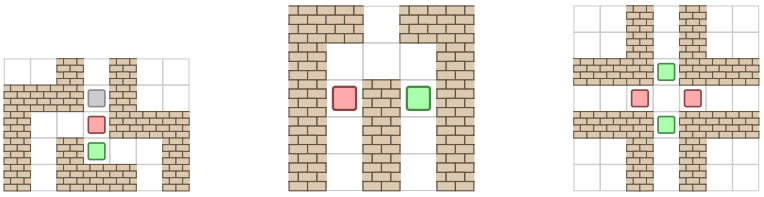
Theorem 11. *The following games are P-complete:*

- *Battering Ram-F* ($eAB \to eAe$)
- *Zap Through Walls-F* ($AFB \to AFe$) *in 3D*
- *Zap Through Walls-F* ($AFB \to AFe$) *with Multiple Agents*



a Push Merge ($ABB \to AeB$)
b Push Merge ($ABB \to AeB$)
c Diode for Push Merge ($ABB \to AeB$)

Fig. 7: Initial states of Antiparallel NANDs



a AND
b OR
c Crossover

Fig. 8: Battering Ram ($eAB \to eAe$) P-complete Reduction

Zap-Through-Walls

Corollary 2. *Zap-Through-Walls-F ($AFB \to AFe$) is PMCVP-hard.*

Theorem 12. *Bendy Zap Through Wall ($AFB \to AFe$) is PMCVP-hard in 2D and P-complete in 3D.*

Battering Ram Here, we give a result for Battering Ram ($eAB \to eAe$) with no fixed blocks:

Theorem 13. *Battering Ram is NL-complete.*

Knock Over Knock Over ($ABe \to Aee$) is of interest because the problem is P-complete if we allow thin walls or if the problem is in 3D, but is in NL if we only allow fixed walls. This is the only section we have where -W vs -F makes a distinction so we specify this in our Theorems. Note that Theorem 10 holds for -W.

Theorem 14. *Knock Over-W ($ABe \to Aee$) is P-complete.*

Theorem 15. *Knock Over-F ($ABe \to Aee$) in 3D is P-complete.*

Theorem 16. *Knock Over-F ($ABe \to Aee$) is in NL.*

	-W	Ref.	-F	Ref.
3D	P-complete	Thms. 14, 15	P-complete	Thm. 15
2D	P-complete	Thm. 14	NL	Thm. 16

Table 3: Knock Over ($ABe \to Aee$) Table

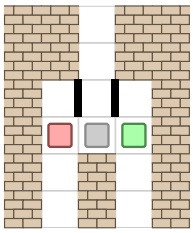


Fig. 9: Knockover-W ($ABe \to Aee$)

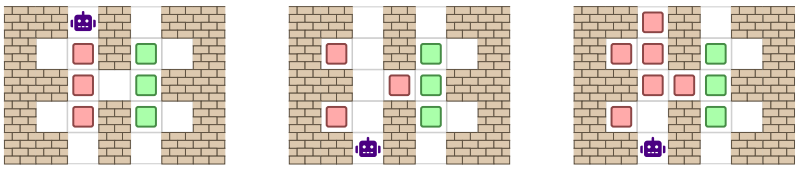
6 Bendy Rules

Here, we consider length-3 bendy rules, that have not been considered in earlier sections. Recall that length-3 bendy rules can be applied in an L-shape. For example, for Push-1 ($ABe \to eAB$), allowing the rule to be bendy allows for an agent to push a block around a corner.

6.1 Other Conservative Bendy Results

Here, we obtain partial results for some conservative bendy rules not mentioned in earlier sections:

Theorem 17. *Bendy Push-1F ($ABe \to eAB$) is NP-hard.*



a Initial state b Bendy Push-1 traversal c Bendy Trail Push traversal

Fig. 10: An undirected NAND gadget for Bendy Push-1F ($ABe \to eAB$) and an undirected matched crumblers gadget for Bendy Trail Push ($ABe \to BAB$).

Theorem 18. *Bendy Push Swap at Block-F ($ABB \to BAB$) is NP-hard.*

Theorem 19. *Bendy Pull Swap at Block-F ($BAB \to ABB$) is NP-hard.*

Here, we note a particularly surprising easiness result:

Theorem 20. *Bendy Suplex-1F ($AeB \to BAe$ and congruent) is in L.*

6.2 Other Bounded Bendy Results

Here, we obtain some results (some partial, some full) for bounded bendy rules that are not mentioned in earlier sections:

Theorem 21. *Bendy Trail Push-F ($ABe \to BAB$) is NP-complete.*

Theorem 22. *Bendy Push Smash-F ($ABB \to Aee$) and Bendy Swaplex-F ($ABB \to BAe$) are NP-complete.*

Theorem 23. *Bendy Pull Merge-F ($ABB \to ABe$) is NP-complete.*

Theorem 24. *Bendy Trail Swap-F ($eAB \to BBA$) is NP-complete.*

Theorem 25. *Bendy Knock-Over-F ($ABe \to Aee$) is PMCVP-hard in 2D and P-complete in 3D.*

Theorem 26. *Bendy Battering Ram-F ($AeB \to eAe$) is in L.*

7 Future Directions

Many other variants of Push-1F and Pull?-1F have been introduced and studied. Some of them require further extenions to our block asynchronous cellular automata framework. For example, Pull!-1 [4] forces the agent to pull a block when it moves away from a block. We can model this by replacing the $Ae \to eA$ swap with the more restrictive $eAe \to Aee$ as the movement rule, in addition to the rule $eAB \to ABe$ representing the pull. It would be interesting to consider the other games with this movement rule. Pull-1W [4] allows thin 0×1 walls. Thin walls can be modeled as missing edges in the graph of cells. Games like Push-$*$, where the agent can push any number of blocks in front of it, can be modeled as an infinite set of game rules $AB^ie \to eAB^i$ for all $i > 0$. PushPull-1 is a game that would have two game rules, $\{ABe \to eAB, eAB \to ABe\}$, or as a single reversible rule $ABe \leftrightarrow eAB$. The puzzle game Sokoban can be modeled with the same rules as Push-1, expect that the goal is to reach a target configuration of blocks. It would be interesting to study the other puzzles we have introduced in these variants as well.

References

1. L. Adleman, Q. Cheng, A. Goel, M.-D. Huang, D. Kempe, P. M. De Espanes, and P. W. K. Rothemund. Combinatorial optimization problems in self-assembly. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, pages 23–32, 2002.
2. H. A. Akitaya, E. D. Demaine, A. Gonczi, D. H. Hendrickson, A. Hesterberg, M. Korman, O. Korten, J. Lynch, I. Parada, and V. Sacristán. Characterizing universal reconfigurability of modular pivoting robots. In *Proceedings of the 37th International Symposium on Computational Geometry*, 2021.
3. R. M. Alaniz, J. Brunner, M. Coulombe, E. D. Demaine, J. Diomidova, T. Gomez, E. Grizzell, R. Knobel, J. Lynch, A. Rodriguez, R. Schweller, and T. Wylie. Complexity of Reconfiguration in Surface Chemical Reaction Networks. In H.-L. Chen and C. G. Evans, editors, *Proceedings of the 29th International Conference on DNA Computing and Molecular Programming (DNA 29)*, volume 276 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 10:1–10:18, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.
4. H. Ani, S. Asif, E. D. Demaine, J. Diomidova, D. Hendrickson, J. Lynch, S. Scheffler, and A. Suhl. PSPACE-completeness of pulling blocks to reach a goal. *Journal of Information Processing*, 28:929–941, 2020.
5. H. Ani, L. Chung, E. D. Demaine, J. Diomidova, D. Hendrickson, and J. Lynch. Pushing blocks via checkable gadgets: PSPACE-completeness of Push-1F and Block/Box Dude. In *Proceedings of the 11th International Conference on Fun with Algorithms (FUN 2022)*, 2022.
6. E. R. Berlekamp, J. H. Conway, and R. K. Guy. What is Life? In *Winning Ways for Your Mathematical Plays*, volume 4. A K Peters, 2nd edition, 2004.
7. M. Cook. Universality in elementary cellular automata. *Complex Systems*, 15(1), 2004.
8. C. Defant and N. Kravitz. Friends and strangers walking on graphs. *Combinatorial Theory*, 1, 2021.
9. E. D. Demaine, I. Grosof, J. Lynch, and M. Rudoy. Computational complexity of motion planning of a robot through simple gadgets. In *Proceedings of the 9th International Conference on Fun with Algorithms (FUN 2018)*, 2018.
10. E. D. Demaine, R. A. Hearn, D. Hendrickson, and J. Lynch. PSPACE-completeness of reversible deterministic systems. In *Proceedings of the 9th Conference on Machines, Computations and Universality (MCU 2022)*, pages 91–108, Debrecen, Hungary, August–September 2022.
11. E. D. Demaine, D. H. Hendrickson, and J. Lynch. Toward a general complexity theory of motion planning: Characterizing which gadgets make games hard. In *Proceedings of the 11th Innovations in Theoretical Computer Science Conference (ITCS 2020)*, pages 62:1–62:42, 2020.
12. E. D. Demaine and M. Rudoy. A simple proof that the $(n^2 - 1)$-puzzle is hard. *Theoretical Computer Science*, 732:80–84, 2018.
13. A. Dennunzio, E. Formenti, L. Manzoni, G. Mauri, and A. E. Porreca. Computational complexity of finite asynchronous cellular automata. *Theoretical Computer Science*, 664:131–143, 2017.
14. J. Durand-Lose. Representing reversible cellular automata with reversible block cellular automata. *Discrete Mathematics & Theoretical Computer Science Proceedings*, AA: Discrete Models: Combinatorics, Computation, and Geometry (DM-CCG 2001), Jan. 2001.

15. N. Fates. A guided tour of asynchronous cellular automata. In *Proceedings of the International Workshop on Cellular Automata and Discrete Complex Systems*, pages 15–30. Springer, 2013.
16. R. Fleischer and J. Yu. A survey of the game "Lights Out!". In *Space-Efficient Data Structures, Streams, and Algorithms: Papers in Honor of J. Ian Munro on the Occasion of His 66th Birthday*, pages 176–198. Springer, 2013.
17. N. Gershenfeld, D. Dalrymple, K. Chen, A. Knaian, F. Green, E. D. Demaine, S. Greenwald, and P. Schmidt-Nielsen. Reconfigurable asynchronous logic automata. *ACM SIGPLAN Notices*, 45(1):1–6, 2010.
18. E. Goles, D. Maldonado, P. Montealegre, and M. Ríos-Wilson. On the complexity of asynchronous freezing cellular automata. *Information and Computation*, 281:104764, 2021.
19. E. Goles, N. Ollinger, and G. Theyssier. Introducing freezing cellular automata. In *Proceedings of the 21st International Workshop on Cellular Automata and Discrete Complex Systems (AUTOMATA 2015)*, volume 24, pages 65–73, 2015.
20. M. H. Group, J. Brunner, L. Chung, E. D. Demaine, J. Diomidova, D. Hendrickson, and J. Lynch. Pushing blocks without fixed blocks via checkable gizmos: Push-1 is PSPACE-complete. Manuscript under submission, 2024.
21. H. Gutowitz, editor. *Cellular Automata: Theory and Experiment*. MIT Press, 1991.
22. J. Kari. Theory of cellular automata: A survey. *Theoretical Computer Science*, 334(1):3–33, 2005.
23. C. G. Langton. Studying artificial life with cellular automata. *Physica D: Nonlinear Phenomena*, 22(1-3):120–149, 1986.
24. N. Margolus. Physics-like models of computation. *Physica D: Nonlinear Phenomena*, 10(1):81–95, 1984.
25. A. Milojević. Connectivity of old and new models of friends-and-strangers graphs. *Advances in Applied Mathematics*, 155:102668, 2024.
26. T. Neary and D. Woods. P-completeness of cellular automaton Rule 110. In *Proceedings of the 33rd International Colloquium on Automata, Languages and Programming (ICALP 2006)*, pages 132–143, Venice, Italy, July 2006. Springer.
27. N. Ollinger and G. Theyssier. Freezing, bounded-change and convergent cellular automata. *Discrete Mathematics & Theoretical Computer Science*, 24(Automata, Logic and Semantics), 2022.
28. L. Qian and E. Winfree. Parallel and scalable computation and spatial dynamics with DNA-based chemical reaction networks on a surface. In *Proceedings of the DNA Computing and Molecular Programming: 20th International Conference (DNA 20)*, volume 8727, page 114, Kyoto, Japan, Sept. 2014. Springer.
29. D. Ratner and M. Warmuth. The (n^2-1)-puzzle and related relocation problems. *Journal of Symbolic Computation*, 10:111–137, 1990.
30. K. Sutner. On the computational complexity of finite cellular automata. *Journal of Computer and System Sciences*, 50(1):87–97, 1995.
31. A. J. Thubagere, W. Li, R. F. Johnson, Z. Chen, S. Doroudi, Y. L. Lee, G. Izatt, S. Wittman, N. Srinivas, D. Woods, et al. A cargo-sorting DNA robot. *Science*, 357(6356):eaan6558, 2017.
32. T. Toffoli and N. Margolus. Ii.12: The Margolus neighborhood. In *Cellular Automata Machines: A New Environment for Modeling*, pages 119–138. MIT Press, 1987.
33. T. Tsukiji and T. Hagiwara. Recognizing the repeatable configurations of time-reversible generalized langton's ant is PSPACE-hard. *Algorithms*, 4(1):1–15, 2011.
34. J. von Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press, 1966.

35. E. Winfree. *Algorithmic self-assembly of DNA*. PhD thesis, California Institute of Technology, 1998.