Pushing Blocks via Checkable Gadgets: PSPACE-completeness of Push-1F and Block/Box Dude

Joshua Ani
Massachusetts Institute of Technology, Cambridge, MA, USA

Lily Chung
Massachusetts Institute of Technology, Cambridge, MA, USA

Erik D. Demaine
Massachusetts Institute of Technology, Cambridge, MA, USA

Yevhenii Diomidov
Massachusetts Institute of Technology, Cambridge, MA, USA

Dylan Hendrickson
Massachusetts Institute of Technology, Cambridge, MA, USA

Jayson Lynch
Cheriton School of Computer Science, University of Waterloo, Waterloo, ON, Canada

Abstract

We prove PSPACE-completeness of the well-studied pushing-block puzzle Push-1F, a theoretical abstraction of many video games (first posed in 1999). We also prove PSPACE-completeness of two versions of the recently studied block-moving puzzle game with gravity, Block Dude — a video game dating back to 1994 — featuring either liftable blocks or pushable blocks. Two of our reductions are built on a new framework for "checkable" gadgets, extending the motion-planning-through-gadgets framework to support gadgets that can be misused, provided those misuses can be detected later.

2012 ACM Subject Classification Theory of computation → Computational complexity and cryptography

Keywords and phrases gadgets, motion planning, hardness of games

Digital Object Identifier 10.4230/LIPIcs.FUN.2022.2

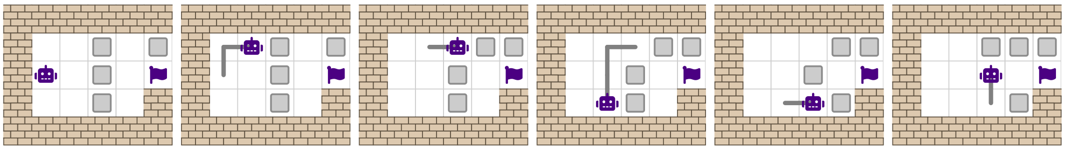
1 Introduction

In the ***Push*** family of pushing-block puzzles, introduced by O'Rourke in 1999 [14], a 1×1 agent must traverse a unit-square grid, some cells of which have a "block", from a given start location to a given target location. Refer to Figure 1. In ***Push-k*** [7, 8]), the agent's move (horizontal or vertical by one square) can ***push*** up to k consecutive blocks by one square, provided that there is an empty square on the other side. In the ***-F*** variation (described in [8, 14] but first given notation in [10]), some of the blocks are ***fixed*** in the grid, meaning they cannot be traversed or pushed by the agent or other blocks. Push-1F has the same allowed moves as the famous ***Sokoban*** puzzle video game, invented in 1982 and analyzed at FUN 1998 [6], but crucially Push-1F's goal is for the agent to reach a target location, which is much simpler than Sokoban's "storage" goal where the blocks must be pushed to certain locations.

In this paper, we prove that Push-1F is PSPACE-complete, settling an open problem from [8, 10], and complementing previous PSPACE-hardness for Push-kF for $k \geq 2$ from 20 years ago [10].

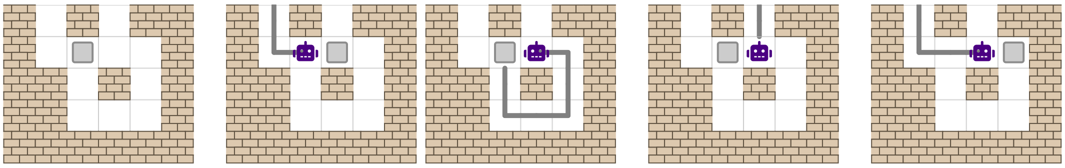
To gain some intuition about why Push-1F is so difficult to prove PSPACE-hard, and how we surmount that difficulty, consider the attempt at a "diode" gadget in Figure 2. The goal of this gadget is to allow repeated traversals from the left entrance to the right (as in Figure 2b), while always preventing "backward" traversal from the right to the left (as in

11th International Conference on Fun with Algorithms (FUN 2022).
Editors: Pierre Fraigniaud and Yushi Uno; Article No. 2; pp. 2:1–2:30
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



(a) (b) (c) (d) (e) (f)

Figure 1 Sample Push-1F puzzle and solution sequence. In steps (c) and (e), for example, the agent cannot push right again. The agent is drawn as a robot head; the traversed path between steps is drawn as a gray line; pushable blocks are drawn as boxes; fixed blocks are drawn as brick walls; and the goal location is drawn as a flag. *Robot and flag icons from Font Awesome under CC BY 4.0 License.*



(a) Gadget **(b)** Intended forward traversal **(c)** Backward traversal impossible **(d)** Breaking the gadget

Figure 2 A broken Push-1F diode gadget.

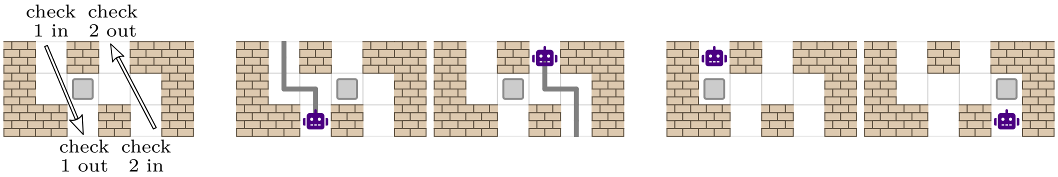
Figure 2c). But given the opportunity for forward traversal, the agent can instead "break" the gadget to allow future forward and backward traversal (as in Figure 2d).

To solve this problem, we introduce the idea of a ***checkable gadget*** where, after the agent completes the "main" gadget traversal puzzle, the agent is forced (in order to solve the overall puzzle) to do a specified sequence of ***checking*** traversals of every gadget, all of which must succeed in order to solve the overall puzzle. If designed well, these checking traversals can detect whether a gadget was previously "broken", and allow traversal only if not. In the case of Figure 2, one can think of the gadget as a four-location gadget (the top three rows) which has its bottom two locations connected. This four-location gadget is "checkable": we will demand that, after completing the main puzzle, the agent follows the two checking traversals shown in Figure 3. In order for these checking traversals to both be possible, the agent cannot push the block into either corner, preventing the agent from breaking the gadget during the main gadget traversal puzzle. We call this process of removing broken states from a gadget by demanding that the checking traversals remain legal ***postselection***.[1]

We develop a general framework of checkable gadgets that enable a reduction to focus on the main gadget traversal puzzle, assuming all gadgets remain unbroken (i.e., the checking traversals remain possible at the end), while the framework ensures that the agent makes these checking traversals at the end (without other unintended traversals). This framework builds upon the motion-planning-through-gadgets framework introduced at FUN 2018 [9] and developed further in [2, 3, 11–13] to handle checkable gadgets.

We also apply our framework to resolve the complexity of ***Block Dude***, a puzzle video

[1] In quantum computing, for example, "postselection is the power of discarding all runs of a computation in which a given event does not occur" [1]. In probability theory, postselection is equivalent to conditioning on a particular event.

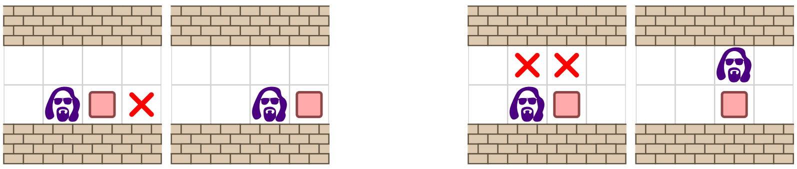


(a) Checkable gadget **(b)** Successful checks **(c)** Failed checks

Figure 3 The top three rows of the Push-1F diode gadget of Figure 2, as a checkable gadget. The checking traversals are "check 1 in → check 1 out" and "check 2 in → check 2 out", denoted by the hollow arrows.

game made over a dozen times on many platforms, originally under the name "Block-Man 1" (Soleau Software, 1994); see [5] for details. Barr, Chung, and Williams [5] recently formalized this game's mechanics, along with several variations, and proved them all NP-hard. In this paper, we prove PSPACE-completeness of three of these variations, including the original video game mechanics:

1. ***BoxDude*** is like Push-1 but where all pushable blocks and the agent experience gravity, falling straight down whenever they have blank spaces below them. In addition to moving horizontally left or right, the agent can "climb" on top of horizontally adjacent blocks (be they pushable or fixed), provided the square above the agent is empty. See Figure 4.



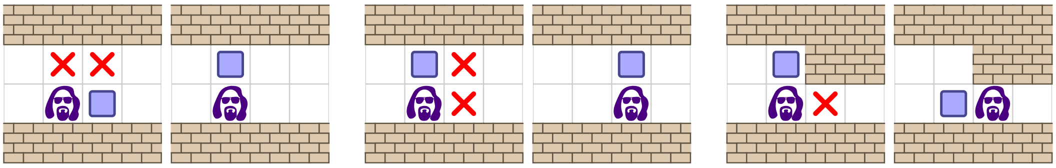
(a) Pushing **(b)** Climbing

Figure 4 Mechanics for BoxDude, with pushable boxes shown in red. Squares marked with a red × must be empty for the move to be possible.

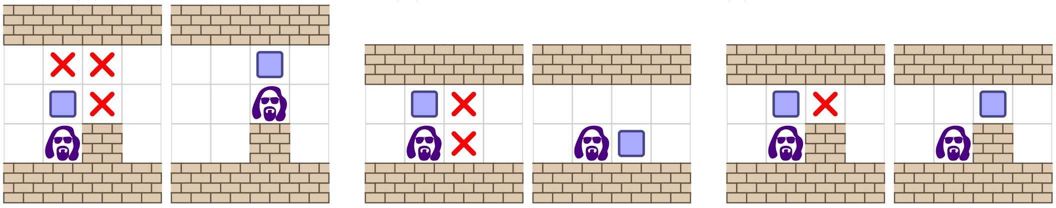
2. In ***BlockDude*** (as in the Block Dude video games), blocks cannot be pushed; instead, nonfixed blocks can be "picked up" by the agent from a horizontally adjacent position to the position immediately above the agent, provided that that position and the intermediate diagonal position are empty. See Figure 5. The agent can then carry one such block to another location (provided the ceiling offers height-2 clearance), and then drop the block in front of them, again provided that that position and the intermediate diagonal position are empty.[2] They can also stack the block on top of another block. If the agent tries to move past a low ceiling while carrying a block, the block will be dropped behind them.
3. In ***BloxDude***, nonfixed blocks can be pushed (as in BoxDude) and/or picked up (as in BlockDude).

The other variations described in [5], called ⋯Duderino instead of ⋯Dude, change the goal of a puzzle to place the k nonfixed blocks into k specified storage locations, as in Sokoban. We leave open the complexity of BoxDuderino, BlockDuderino, and BloxDuderino.

[2] A complication in some implementations of the game is that the agent can only pick up or drop the block in front of them, with the agent's orientation determined by their previous move. (Some implementations allow turning around in place.) This detail will not affect our results.



(a) Lifting a block **(b)** Carrying a lifted block **(c)** Low clearance when carrying



(d) Climbing with a block **(e)** Dropping a lifted block **(f)** Stacking a lifted block

Figure 5 Mechanics for BlockDude, with liftable blocks shown in blue. Squares marked with a red $\times$ must be empty for the following move to be possible.

All of the games we consider can easily be simulated in polynomial space, and thus are in NPSPACE = PSPACE by Savitch's Theorem. Proving PSPACE-hardness is much more complicated, and is the goal of this paper.

The rest of this paper is organized as follows. In Section 2, we review the motion-planning-through-gadgets framework. In Section 3, we prove that BlockDude and BloxDude are PSPACE-complete using standard reductions from motion-planning-through-gadgets. In Section 4, we develop our checkable gadget framework. In Section 5, we prove that BoxDude is PSPACE-complete using our checkable gadget framework. In Section 6, we prove that Push-1F is PSPACE-complete via a much more involved application of our checkable gadget framework.

2 Gadgets Framework

The ***motion-planning-through-gadgets framework*** is an abstract motion planning model used for proving computational hardness results. Here we give the definitions and results we need for this paper; see [11–13] for more details.

A ***gadget*** G consists of a finite set $Q(G)$ of ***states***, a finite set $L(G)$ of ***locations*** (entrances/exits), and a finite set $T(G)$ of ***transitions*** of the form $(q, a) \to (r, b)$ where $q, r \in Q(G)$ are states and $a, b \in L(G)$ are locations. The transition $(q, a) \to (r, b) \in T(G)$ means that an agent can ***traverse*** the gadget when it is in state q by entering at location a and exiting at location b which changes the state of the gadget from q to r. We use the notation $a \to b$ for a traversal by the agent that does not specify the state of the gadget before or after the traversal. A ***traversal sequence*** $[a_1 \to b_1, \ldots, a_k \to b_k]$ on the locations $L(G)$ is ***legal*** from state s_0 if there is a corresponding sequence of transitions $[(a_1, s_0) \to (b_1, s_1), \ldots, (a_k, s_{k-1}) \to (b_k, s_k)]$, where each start state of each transition matches the end state of the previous transition (s_0 for the first transition). We define gadgets in figures using a ***state diagram*** which gives, for each state $q \in Q$, a labeled directed multigraph $G_q = (L(G), E_q)$ on the locations, where a directed edge (a, b) with label r represents the transition $(q, a) \to (r, b) \in T(G)$.

Figure 6 shows the state diagram of a key gadget called the ***locking 2-toggle*** [11]. This gadget has four locations (drawn as dots) and three states $1, 2, 3$. The central state, 2, allows for two different transitions. Each of those transitions takes the gadget to a different state,

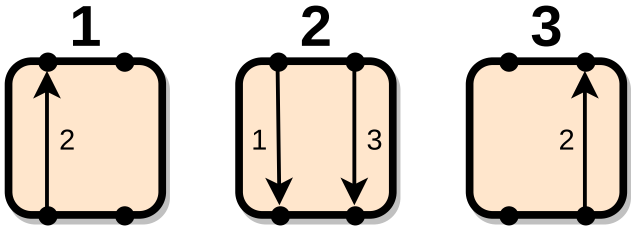


Figure 6 State diagram for the locking 2-toggle gadget. Each box represents the gadget in a different state, in this case labeled with the numbers $1, 2, 3$. Dots represent the four locations of the gadget. Arrows represent transitions in the gadget and are labeled with the states to which those transitions take the gadget. In state 2, the agent can traverse either tunnel going down, which blocks off both downward traversals until the agent reverses that traversal.

from which the only transition returns the agent to the prior location and returns the gadget to state 3.

A ***system of gadgets*** S consists of a set of gadgets, an initial state for each gadget, and a ***connection graph*** on the gadgets' locations. If two locations a, b of two gadgets (possibly the same gadget) are connected by a path in the connection graph, then an agent can traverse freely between a and b (outside the gadgets).[3] We call edges of the connection graph ***hallways***, and for clarity in figures, we add extra vertices to the connection graph called ***branching hallways***, which we can equivalently think of as a one-state gadget that has transitions between all pairs of locations. A ***system traversal*** is a sequence of traversals $a_1 \to b_1, \ldots, a_k \to b_k$, each on a potentially different gadget in S, where the connection graph has a path from b_i to a_{i+1} for each i. We write such a traversal as $a_1 \to^* b_k$, ignoring the intermediate locations. A system traversal is ***legal*** if the restriction to traversals on a single gadget G is a legal traversal sequence from the initial state of G assigned by S, for every G in S. Note that gadgets are "local" in the sense that traversing a gadget does not change the state (and thus traversability) of any other gadgets.

The ***reachability*** or ***1-player motion planning*** problem with a finite set of gadgets $\mathcal{G}$ asks whether there is a legal system traversal $s \to^* t$ from a given start location s to a given goal location t (by a single agent) in a given system of gadgets S, which contains only gadgets from $\mathcal{G}$.

Because we are working with 2D games, we also consider ***planar motion planning***, where every gadget additionally has a specified cyclic ordering of its vertices and the system of gadgets is embedded in the plane without intersections. More precisely, a system of gadgets is ***planar*** if the following construction produces a planar graph: (1) replace each gadget with a wheel graph, which has a cycle of vertices corresponding to the locations on the gadget in the appropriate order, and a central vertex connected to each location; and (2) connect locations on these wheels with edges according to the connection graph. In ***planar reachability***, we restrict to planar systems of gadgets. Note that this definition allows rotations and reflections of gadgets, but no other permutation of their locations.

[3] Equivalently, we can think of identifying locations a and b topologically, thereby contracting the connected components of the connection graph. Alternatively, if we think of the gadgets as individual "levels", then the connection graph is like an "overworld" map connecting the levels together.

2.1 Simulation

To define a notion of gadget simulation, we can think of a system of gadgets as being characterized by its set of possible traversal sequences (as formalized by the related ***gizmo*** framework of [12]).

▶ **Definition 1.** *A* ***(local) simulation*** *of a gadget G in state q consists of a system S of gadgets, together with an injective function m mapping every location of G to a distinct location in S, such that a traversal sequence $[a_1 \to b_1, \ldots, a_k \to b_k]$ on the locations in G is legal from state q if and only if there exists a sequence of system traversals $m(a_1) \to^* m(b_1), \ldots, m(a_k) \to^* m(b_k)$ that is legal in the sense that the concatenation of the restrictions of the system traversals $m(a_i) \to^* m(b_i)$ to traversals on a single gadget G is a legal traversal sequence for G from the initial state of G assigned by S, for every G in S.*

A ***planar simulation*** *of a gadget G in state q is a simulation (S, m) where S is furthermore a planar system of gadgets, and the cyclic order of locations of G must map via m to locations in cyclic order around the outside face of S.*

A [planar] simulation of an entire gadget G consists of a [planar] simulation of G in state q, for all states $q \in Q(G)$, that differ only in their assignments of initial states. A finite set $\mathcal{G}$ of gadgets ***[planarly] simulates*** *a gadget G if there is a [planar] simulation of G using only gadgets in $\mathcal{G}$.*

These definitions of simulation imply that, if we take a larger system of gadgets and replace each instance of gadget G with the system S using the appropriate initial states (matching up locations that correspond via m), then the entire system behaves equivalently. In particular, this substitution preserves reachability of locations from one another. Furthermore, if the larger system and the simulation are both planar, then the full resulting system is planar. More formally:

▶ **Lemma 2.** *Let H be a gadget, and let $\mathcal{G}$ and $\mathcal{G}'$ be finite sets of gadgets. If $\mathcal{G}$ [planarly] simulates H, then there is a polynomial-time reduction[4] from [planar] reachability with $\{H\} \cup \mathcal{G}'$ to [planar] reachability with $\mathcal{G} \cup \mathcal{G}'$.*

2.2 Known Hardness Results

We can now formally state the problems we will reduce from in this paper.

In Section 3, we use the locking 2-toggle to show PSPACE-completeness of BlockDude puzzles.

▶ **Theorem 3.** [11, Theorem 10] *Planar reachability with any interacting-k-tunnel reversible deterministic gadget is PSPACE-complete.*

The locking 2-toggle is an example of an interacting-k-tunnel reversible deterministic gadget [11] and thus we obtain PSPACE-completeness of planar reachability with the locking 2-toggle. We recommend readers interested in this more general dichotomy to refer to [11].

We also use the nondeterministic locking 2-toggle shown in Figure 7. This is used in Section 5 to show PSPACE-completeness of BoxDude puzzles. Its behavior resembles that of the locking 2-toggle, but because it is not deterministic it is not covered by the prior theorem.

[4] Throughout this paper, reductions are ***many-one/Karp***: a reduction from A to B maps an instance of A to an equivalent (in terms of decision outcome) instance of B.

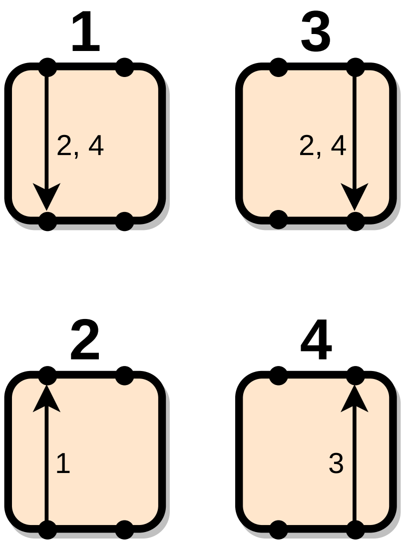


Figure 7 State diagram for a nondeterministic locking 2-toggle. From state 1, the left tunnel can be traversed so as to leave the gadget in either state 2 or state 4. Formally, in the multigraph for state 1 there are two different edges, one labeled 2 and the other labeled 4.

▶ **Theorem 4.** [2, Theorem 3.1] *Planar reachability with the nondeterministic locking 2-toggle is PSPACE-complete.*

The final main gadget we will make use of is a type of self-closing door shown in Figure 8. This gadget will be used in our result on Push-1F in Section 6.

▶ **Theorem 5.** [3, Theorem 4.2] *Planar reachability with any normal or symmetric self-closing door is PSPACE-hard.*

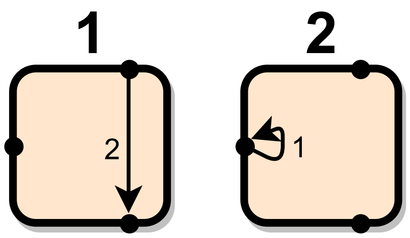


Figure 8 State diagram for the directed open-optional self-closing door. The door must be opened by visiting its opening location before every traversal.

3 BlockDude and BloxDude are PSPACE-complete

In this section, we show that BlockDude and BloxDude are PSPACE-complete using a reduction from planar reachability with locking 2-toggles, shown in Figure 6, which is PSPACE-complete by Theorem 3. Recall from Section 1 in this model blocks can be picked up by BlockDude from an adjacent square. BloxDude allows both picking up and pushing blox, and the reduction will be a small modification to the BlockDude proof.

We will build hallways allowing the player to move between connected locations on gadgets. To connect more than two locations, we need a branching hallway, which is shown in Figure 9. This allows the player to freely move between any of the three entrances.

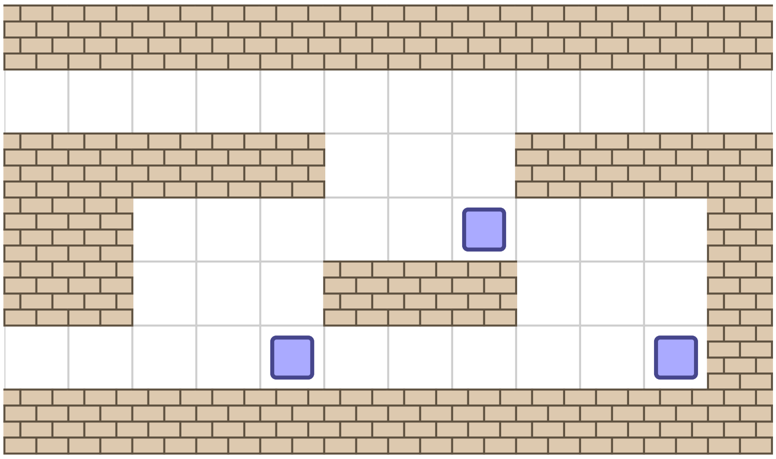


Figure 9 A branching hallway for BlockDude. Blue squares represent blocks (which can be picked up).

We now describe how the player can use the branching hallway in a way that always lets them move between any of its entrances. Whenever the player is outside the branching hallway, both bottom blocks will be in their original positions, and the top block will be somewhere on the middle platform, depending on the most recently taken exit. When the player arrives at the branching hallway, they will first move the top block to the right side of the middle platform (the position in Figure 9). The only case where this is nontrivial is when the player enters at the bottom with the top block on the left. In this case, the player can go under the middle platform and climb up from the right by moving both bottom blocks. Then they can pick up the top block and step back down on the right, causing the carried block to fall onto the right end of the middle platform. Finally, they can reset the bottom blocks and return to the bottom entrance. Once the top block is on the right, the player can take whichever exit they need. If they take the top left exit, they will move the top block to the left first.

To embed an arbitrary planar graph in BlockDude, we also need to be able to turn hallways and in particular to make vertical hallways despite gravity. Fortunately, the branching hallway in Figure 9 can achieve both goals. If we ignore the top-right entrance, the agent can turn around and make some vertical progress. By chaining these switchbacks in alternating orientation, we can build an arbitrarily tall vertical hallway.

To complete the proof of PSPACE-hardness, we only need to build a locking 2-toggle. We will construct the locking 2-toggle out of simpler pieces, as shown in Figure 11. The simpler pieces are two kinds of 1-toggle: one just for the player, and one that the player can carry a block through. The state diagram for a 1-toggle is given in Figure 10. When the player arrives at (say) the bottom left entrance, they can grab the block in the middle and bring it to the left side, and use it to reach the top left entrance. With the block stuck on the left, the right side cannot be traversed until the player returns to the top left, puts the block back, and exits the bottom left. The player cannot move through this gadget in any way not allowed by a locking 2-toggle. They may leave the block on the left side when the exit the bottom left, but this does not achieve anything; it only prevents them from traversing the

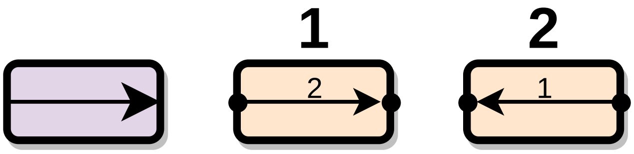


Figure 10 Icon and state diagram for the 1-toggle. Leftwards and rightwards traversals must alternate.

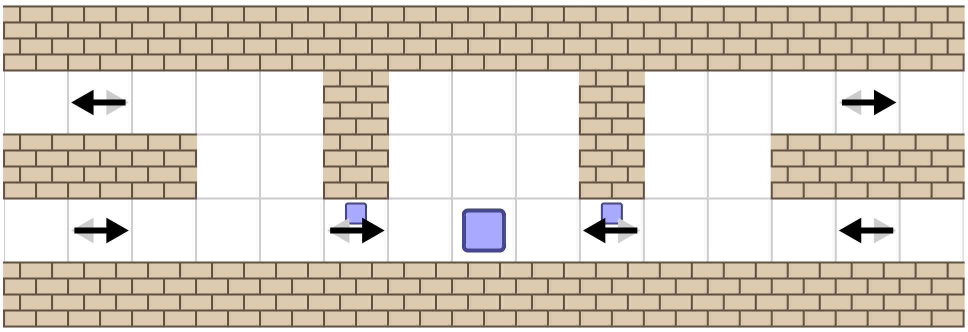


Figure 11 The schematic for our locking 2-toggle for BlockDude. Arrows with a faded backward arrowhead are 1-toggles. Only the player can go through the 1-toggle unless it has a block icon above the arrow, in which case the player can carry a block through.

right side.

Our 1-toggle for just the player is shown in Figure 12. In the state shown, the player can not enter on the right. If they enter on the left, they can move the blocks to exit on the right, but in doing so must block the left entrance. Because of the 1-high hallways, the player can not bring a block through this gadget.

The 1-toggle that lets the player carry a block through is more complicated, and is shown in Figure 13. If the player enters on the left with or without a block, they can get to the right as follows:

- Move the top staircase to the right, so they can climb all the way down.
- Move the top staircase and then the bottom staircase to a single pile in the bottom left corner.
- Move the single pile to the bottom right corner.
- Use three blocks to build a staircase to the middle platform on the right, and move the rest of the blocks up to that platform.
- Use another three blocks to build a staircase to the right exit.

To reach either exit, there must be at least three blocks on the bottom level to form a staircase to the middle platform, and three blocks on the middle platform to form a staircase to the exit. In particular, six blocks must stay inside the gadget, so the player can leave with a block only if they brought one with them. If the player tries to enter the side opposite the one they most recently exited, they will be blocked by both staircases and unable to get across the gadget.

This 1-toggle might break if the player brings several additional blocks to it, but it will never be possible to bring more than one additional block because of the structure of our locking 2-toggles.

With these components, we can fill in our schematic for a locking 2-toggle (Figure 11),

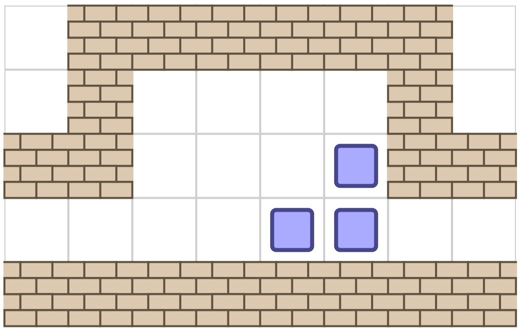


Figure 12 A 1-toggle for BlockDude, currently traversable from left to right.

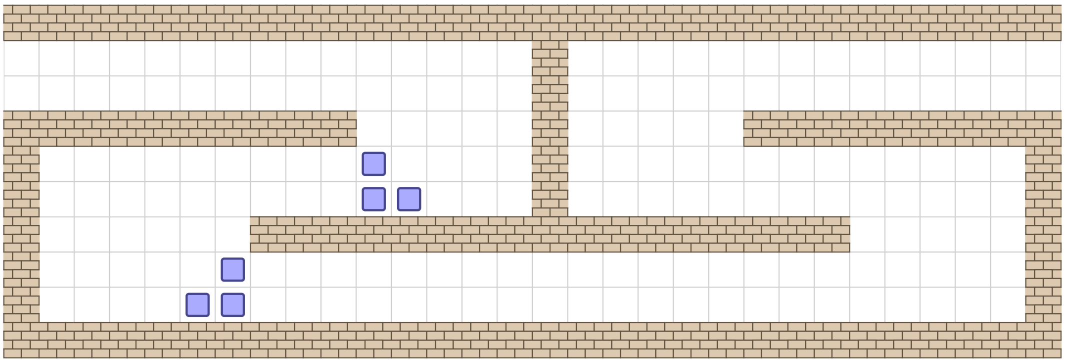


Figure 13 A 1-toggle for BlockDude that lets the player carry a block through it, currently traversable from left to right.

which we show in full in Figure 14. To summarize: the player can enter on either side, at the lower entrance. They can get to the block in the center, but must return to the side they came from. Then they can use this block to reach the top exit on the same side. This makes the center block inaccessible from the other side, so the other side cannot be traversed until the player comes back in the opposite direction and returns the center block.

3.1 BloxDude is PSPACE-complete

In this section we discuss how to adapt the prior proof for BlockDude puzzles to work for blox which can both be picked up and pushed. All the valid traversals from our BlockDude constructions remain and we only need to prevent unwanted movement of the blox due to pushing.

First, whenever there is a hallway in which a blox should not be able to be moved, such as all three hallways from the branching hallway, we add a step in the hallway, as shown in Figure 15. Thus the blox cannot be carried and if it is pushed to the step it will become stuck.

Next we show how to adapt the 1-toggle with block traversal so it works in this setting. This is given in Figure 16. The three-block-tall staircases ensure that bringing a single blox from the wrong direction does not allow deconstructing a staircase from behind. In particular, the middle layer has two blox in a row which cannot be pushed and thus one extra blox will not enable the Dude to deconstruct the staircase from that side.

We also need a regular 1-toggle, and the construction in Figure 12 can be broken in the blox model. Luckily we have a hallway that prevents blox from being carried or pushed

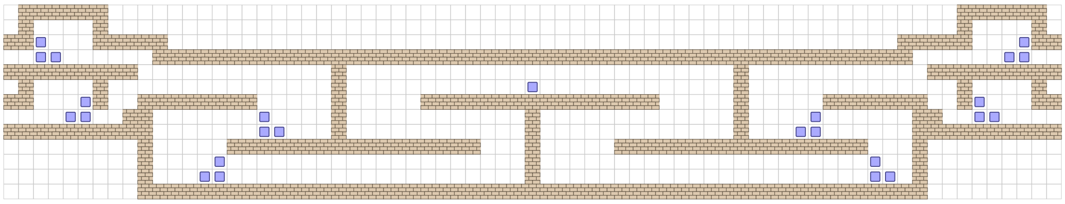


Figure 14 The full locking 2-toggle for BlockDude, combining Figures 11, 12, and 13.

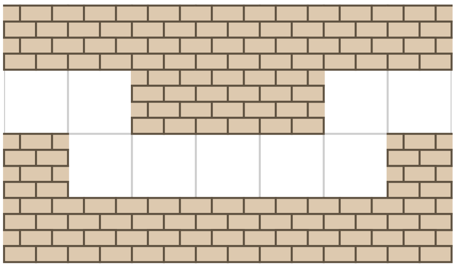


Figure 15 A blox cannot be moved through this hallway.

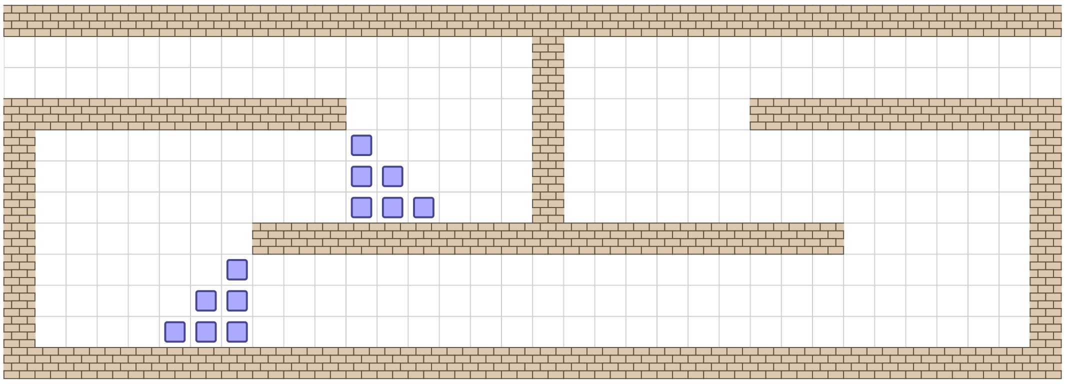


Figure 16 A 1-toggle for BloxDude that lets the player carry a block through it, currently traversable from left to right.

through it, so we can add such a hallway to each end of the gadget in Figure 16 preventing extra blox from entering or leaving. This yields a regular 1-toggle which does not permit blox to pass through.

Once we have the prior two gadgets, it is clear the locking 2-toggle in Figure 11 will still work in the blox model, giving the desired PSPACE-hardness result.

4 Checkable Gadget Framework

In this section, we introduce a new extension to the gadgets framework which will be used in the rest of the paper. This extension allows us to indirectly construct a gadget G by first constructing a "checkable" version of G, and then using "postselection" to obtain G. The checkable G behaves identically to G except that the agent can make undesired traversals into "broken" states which prevent later "checking" traversals. The postselection operation removes these possibilities by guaranteeing that the agent will perform the checking traversals at the end, so to solve reachability, the agent could never perform the undesired traversals.

The price we pay for this ability to constrain the behavior of gadgets is that the resulting simulations are no longer drop-in replacements as in the local simulations of Definition 1; instead we obtain "nonlocal simulations" which require altering the entire surrounding system of gadgets:

▶ **Definition 6.** *A finite set of gadgets $\mathcal{G}$* ***[planarly] nonlocally simulates*** *a gadget H if, for every finite set of gadgets $\mathcal{G}'$, there is a polynomial-time (many-one/Karp) reduction from [planar] reachability with $\{H\} \cup \mathcal{G}'$ to [planar] reachability with $\mathcal{G} \cup \mathcal{G}'$.*

Lemma 2 says that simulations are nonlocal simulations, so this notion is a generalization of Definition 1.

Next we define "checkable" gadgets via "postselection", which transforms a gadget with broken states (where a checking traversal sequence is impossible) into an idealized gadget where those broken states are prevented. At this stage, the prevention is by a magical force, but we will later implement this force with a nonlocal simulation.

▶ **Definition 7.** *Let G be a gadget, C be a traversal sequence on $L(G)$, and $L' \subset L(G)$. Call a state q of G* ***broken*** *if C is not legal from q. Assume that broken states are preserved by transitions on L' in the sense that, if q is broken and there is a transition $(q, a) \to (q', b)$ where $a, b \in L'$, then q' is also broken.*

Define ***Postselect(G, C, L')*** *to be the gadget G' where $L(G') = L'$, $Q(G')$ contains the nonbroken states of G, and $T(G')$ contains the transitions of G restricted to L' and $Q(G')$.*[5] *When there exist C and L' such that Postselect(G, C, L') is equivalent to G', we say that G is a* ***checkable G'****, and we call C the* ***checking traversal sequence****.*

A traversal sequence X is legal for Postselect(G, C, L') from state q if and only if XC is legal for G from q, because both are equivalent to there being a nonbroken state reachable by traversing X. Intuitively, Postselect(G, C, L') is the gadget that results from forcing the agent to traverse C after solving reachability, to ensure that the gadget was left in a nonbroken state, and hiding locations in $L \setminus L'$. Postselect(G, C, L') behaves like G on the locations L' except that transitions into broken states are prohibited.

We now state the main result of the checkable gadget framework, which is in terms of two simple (and often easy-to-implement) gadgets SO (single-use opening) and MSC (merged single-use closing gadgets) defined in Section 4.1.

▶ **Theorem 8.** *For any G, C, and L' satisfying the assumptions of Definition 7, $\{G, SO, MSC\}$ planarly nonlocally simulates Postselect(G, C, L').*

The goal of this section is to prove Theorem 8. Figure 17 provides a schematic overview of the gadget simulations throughout this section that culminate in this result. In Section 4.1, we describe the base gadgets needed for our construction. In Section 4.2, we prove that nonlocal simulations compose in the natural way. In Section 4.3, we introduce a particularly simple kind of checkable gadget, and show that they nonlocally simulate the gadget they are based on. Finally, in Section 4.4 we use all of these tools to prove Theorem 8.

[5] If every state of G is broken, then Postselect(G, C, L') has no states. In this case, it is impossible to use Postselect(G, C, L') in a system of gadgets because that requires specifying an initial state, so all of our theorems hold vacuously.

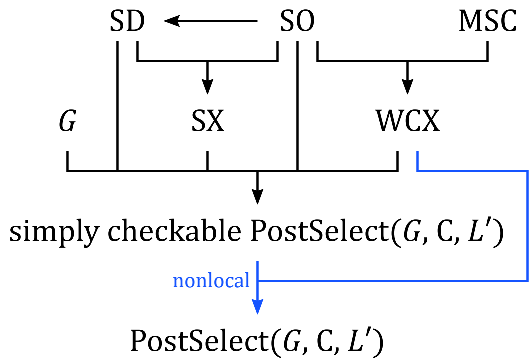
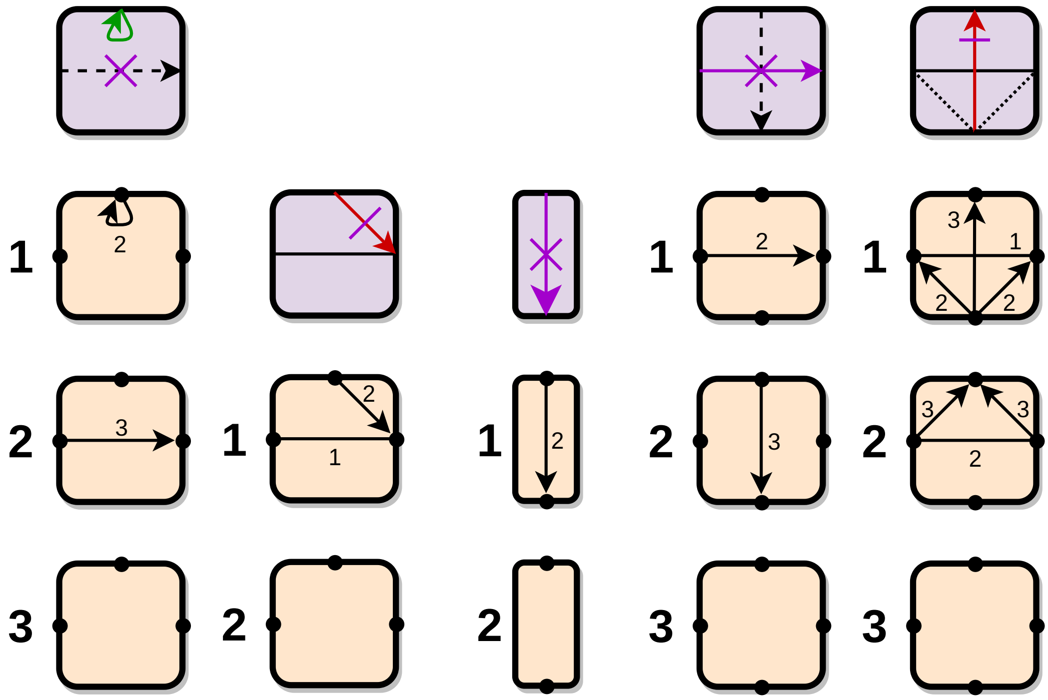


Figure 17 Overview of gadget simulations used for postselection. Black arrows show local simulations and blue arrows show nonlocal simulations.

4.1 Base Gadgets

We now define two base gadgets and three additional derived gadgets, shown in Figure 18, that we use to implement the machinery of checkable gadgets. All five of these gadgets can change state only a bounded number of times; they are "LDAG" in the language of [13].



(a) Single-use opening gadget (SO) **(b)** Merged single-use closing gadget (MSC) **(c)** Dicrumbler/ single-use diode (SD) **(d)** Single-use crossover (SX) **(e)** Weak closing crossover (WCX)

Figure 18 Icons (top) and state diagrams (bottom) for two base gadgets (a–b) and three derived gadgets (c–e). Green arrows show opening traversals, red arrows show closing traversals, and purple crosses indicate traversals that close themselves.

The two base gadgets required for our construction are shown in Figure 18a–18b:

(a) The ***single-use opening (SO)*** gadget, shown in Figure 18a, is a three-state three-location gadget. In state 1, the "opening" location has a self-loop traversal (also called a button, or a port in [3]), which transitions to state 2. State 2 allows a single traversal between the other two locations, after which (in state 3) no traversals are possible.

(b) The ***merged single-use closing (MSC)*** gadget, shown in Figure 18b, is a two-state three-location gadget. In the "open" state 1, horizontal traversals in both directions are freely available. After a traversal from top to right, the gadget transitions to the "closed" state 2, where no traversals are possible.

Next we describe three useful gadgets for our construction which can be built from these base gadgets.

The ***dicrumbler/single-use diode (SD)*** gadget, shown in Figure 18c, is a two-state two-location gadget. In state 1, there is a single directed traversal between the two locations, which permanently closes the gadget in state 2 where no traversals are possible. The SD gadget can be simulated by either of the two base gadgets: it is equivalent to state 2 of SO, and to MSC restricted to the two locations incident to the closing traversal.

The ***single-use crossover (SX)*** gadget, shown in Figure 18d, allows one traversal from left to right and then one from top to bottom. It can be simulated using SO and SD gadgets as shown in Figure 19. The top location in the simulation cannot be entered until the top SO is opened. This opening is possible only after traversing the first two SDs, which prevents any further traversals coming from the left or going to the right. The bottom SO prevents premature traversals going to the bottom.

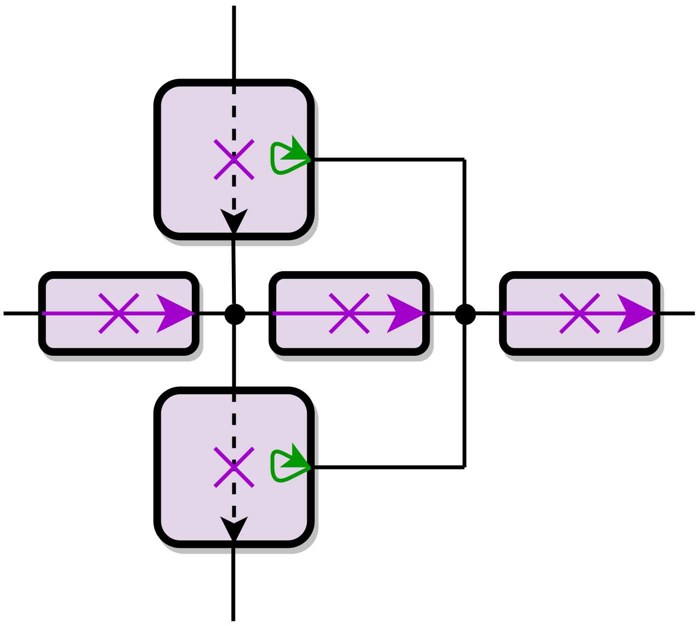


Figure 19 Construction of the single-use crossover from SO and SD gadgets.

The ***weak closing crossover (WCX)***, shown in Figure 18e, initially allows traversals freely between the left and right. If a bottom-to-top traversal is performed, no more traversals are possible. However, a bottom-to-left or bottom-to-right traversal is also possible (which also opens up left-to-top or right-to-top traversals), making the crossover "leaky". The weak closing crossover can be simulated using SO, MSC, and SD gadgets, as shown in Figure 20.

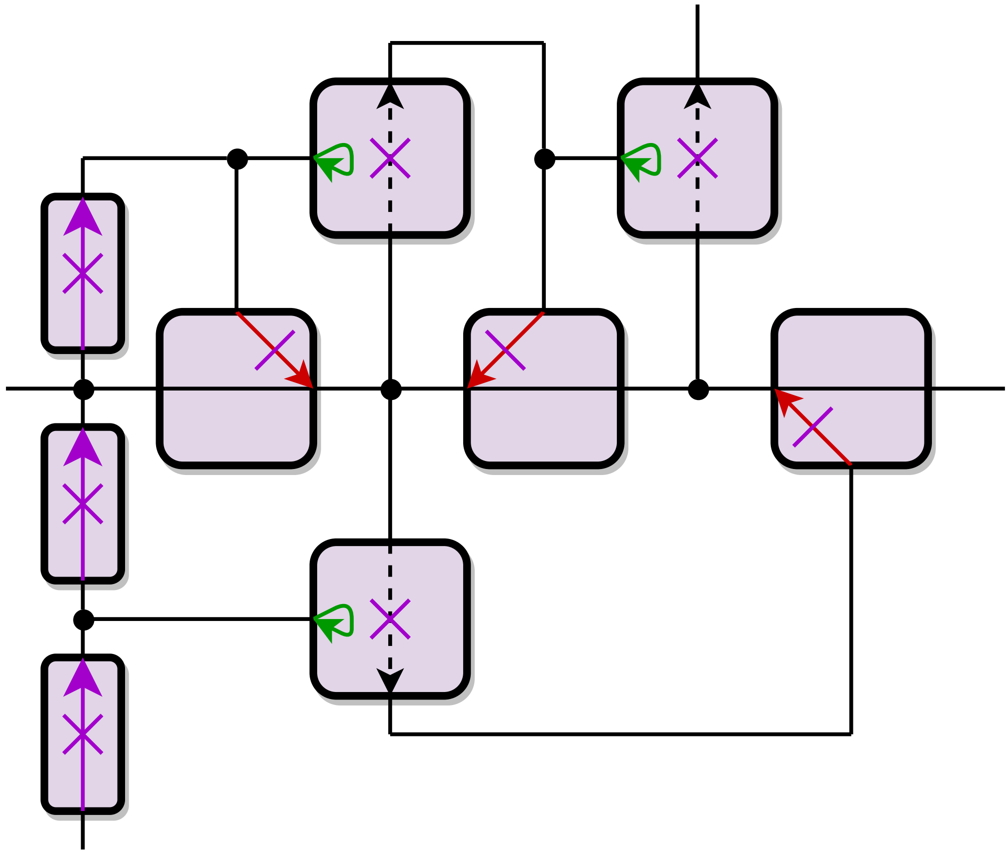


Figure 20 Construction of the weak closing crossover from SD, SO, and MSC gadgets.

To open the upper-right SO, the agent needs to traverse the upper-left SO and then close the middle MSC. To open the upper-left SO, the agent will need to close the leftmost MSC. Having closed both the left and the middle MSCs, the agent is forced to traverse the bottom SO and close the rightmost MSC. The bottom SO can only be opened by the agent traversing entering the bottom and traversing bottom two SDs, preventing any future traversals from the bottom. In summary, in order to exit the top, the agent must have entered the bottom in the past, and have closed all three MSCs. Entering the bottom changes to state 2, and exiting the top changes to state 3.

4.2 Nonlocal Simulation Composition

A crucial fact about nonlocal simulation is that nonlocal simulations can be composed:

▶ **Lemma 9.** *Let $\mathcal{G}$ and $\mathcal{H}$ be finite sets of gadgets. Suppose $\mathcal{G}$ [planarly] nonlocally simulates every gadget in $\mathcal{H}$, and $\mathcal{H}$ [planarly] nonlocally simulates another gadget H. Then $\mathcal{G}$ [planarly] nonlocally simulates H.*

Proof. For a finite set of gadgets $\mathcal{G}'$, we must find a polynomial-time reduction from reachability with $\{H\} \cup \mathcal{G}'$ to reachability with $\mathcal{G} \cup \mathcal{G}'$. Let $\mathcal{H} = \{H_1, \ldots, H_n\}$, where $n = |\mathcal{H}|$, and let $\mathcal{H}_i$ be the prefix $\{H_1, \ldots, H_i\}$, so $\mathcal{H}_n = \mathcal{H}$. Then we construct a chain of reductions between reachability with different sets of gadgets:

$$\{H\} \cup \mathcal{G}' \;\rightarrow\; \mathcal{G} \cup \mathcal{H}_n \cup \mathcal{G}' \;\rightarrow\; \mathcal{G} \cup \mathcal{H}_{n-1} \cup \mathcal{G}' \;\rightarrow\; \cdots \rightarrow \mathcal{G} \cup \mathcal{H}_1 \cup \mathcal{G}' \;\rightarrow\; \mathcal{G} \cup \mathcal{G}'.$$

The first reduction is because $\mathcal{H} = \mathcal{H}_n$ nonlocally simulates H. The remaining reductions come from the assumption that $\mathcal{G}$ nonlocally simulates each $H_i \in \mathcal{H}$, which implies that there is a polynomial-time reduction from reachability with $\{H_i\} \cup \mathcal{G} \cup \mathcal{H}_{i-1} \cup \mathcal{G}' = \mathcal{G} \cup \mathcal{H}_i \cup \mathcal{G}'$ to reachability with $\mathcal{G} \cup \mathcal{G} \cup \mathcal{H}_{i-1} \cup \mathcal{G}' = \mathcal{G} \cup \mathcal{H}_{i-1} \cup \mathcal{G}'$. ◀

4.3 Simply Checkable Gadgets

Next, we define a special kind of checkable gadgets, called "simply checkable" gadgets. A simply checkable G is essentially a checkable G where the checking sequence consists of a single traversal between two locations not in $L(G)$, called c_{in} and c_{out}. Simply checkable gadgets will be a useful as an intermediate step in our proof of Theorem 8.

▶ **Definition 10.** *For a gadget G, a **simply checkable** G is a gadget G' satisfying the following properties:*

1. *$L(G') = L(G) \sqcup \{c_{in}, c_{out}\}$ has two new locations c_{in}, c_{out}. For planar gadgets, the cyclic orderings of the shared locations $L(G)$ are the same. (Locations c_{in} and c_{out} can be added to the cyclic order anywhere.)*
2. *There is a function $f : Q(G) \to Q(G')$ assigning a state of G' to each state of G.*
3. *For any traversal sequence X that is legal for G from state q, the concatenated traversal sequence $X \cdot [c_{in} \to c_{out}]$ is legal for G' from $f(q)$.*
4. *Every traversal sequence that ends at c_{out} and is legal for G' from state $f(q)$ has the form*

$$X \cdot [c_{in} \to \bullet, \bullet \to \bullet, \ldots, \bullet \to c_{out}]$$

where X is legal for G from state q and the omitted $\bullet$ locations (if any) belong to $L(G)$.

Intuitively, a simply checkable G in state $f(q)$ behaves the same as G does in state q, provided that afterward the agent performs a traversal sequence from c_{in} to c_{out} (which may involve the agent exiting and re-entering the gadget, but only via nonchecking locations). The gadget can do essentially anything in a traversal sequence not ending in c_{out}.

Any simply checkable G is also a checkable G: if G' is a simply checkable G, then $\mathsf{Postselect}(G', [c_{\text{in}} \to c_{\text{out}}], L(G))$ is equivalent to G.

We show that a simply checkable G can nonlocally simulate G while preserving planarity, using an auxiliary gadget. First, define the hallway gadget to be the one-state two-location gadget with transitions in both directions between the locations (i.e., a "branching hallway" with only two locations). A ***checkable hallway crossover*** is a simply checkable hallway where the added locations c_{in} and c_{out} are not adjacent in the cyclic order, i.e., they interleave with the two hallway locations. For example, the weak closing crossover from Figure 18e is a checkable hallway crossover, where the horizontal traversal corresponds to the hallway, the bottom location is c_{in}, and the top location is c_{out}.

▶ **Lemma 11.** *Let G' be a simply checkable G and let CHX be a checkable hallway crossover. Then*

1. *$\{G'\}$ nonlocally simulates G; and*
2. *$\{G', CHX\}$ planarly nonlocally simulates G.*

Proof. For any gadget set $\mathcal{G}'$, we construct a polynomial-time reduction from reachability with $\{G\} \cup \mathcal{G}'$ to reachability with $\{G'\} \cup \mathcal{G}'$, or from planar reachability with $\{G\} \cup \mathcal{G}'$ to planar reachability with $\{G', \text{CHX}\} \cup \mathcal{G}'$. Suppose we have a [planar] system S of gadgets from $\{G\} \cup \mathcal{G}'$, along with a designated starting location s and target location t. Let $G_1, \ldots, G_n$ denote the copies of G in S, and let $q_1, \ldots, q_n$ be their respective initial states in S. We build a new system S' of gadgets from $\{G'\} \cup \mathcal{G}'$ as follows; refer to Figure 21.

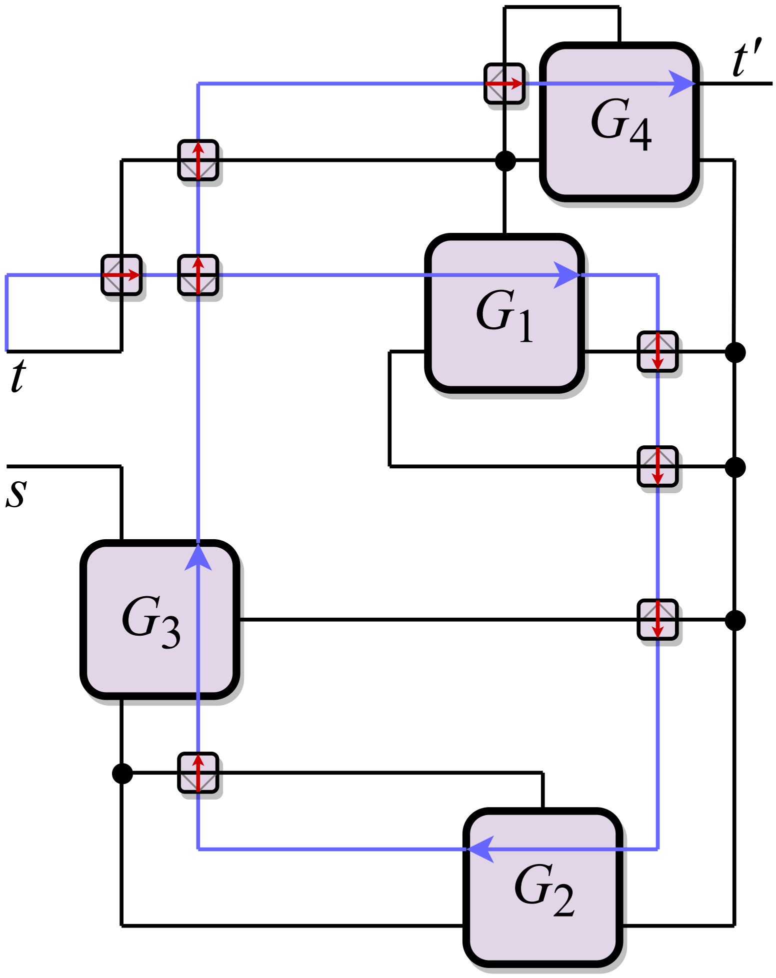


Figure 21 Our nonlocal simulation for the proof of Lemma 11. The system is modified by replacing each copy of G with a copy of G' and adding the blue path from t through $c_{\text{in}} \to c_{\text{out}}$ on each one.

1. Replace each copy G_i of gadget G with initial state q_i in S by a corresponding copy G'_i of G' with initial state $f(q_i)$, whose copies of c_{in} and c_{out} are named $c_{\text{in},i}$ and $c_{\text{out},i}$.
2. Connect t to $c_{\text{in},1}$. In the planar case, we place a copy of CHX on each crossing this creates, with the check line on the way from t to $c_{\text{in},1}$.
3. Connect $c_{\text{out},i}$ to $c_{\text{in},i+1}$ for each i. In the planar case, we place a copy of CHX on each crossing this creates, with the check line on the way from $c_{\text{out},i}$ to $c_{\text{in},i+1}$.

Our reduction outputs this new system S' along with the same start location s and the new target location $t' = c_{\text{out},n}$.

This construction clearly takes polynomial time. To prove that the reduction is valid, we must show that there is a legal system traversal $s \to^* c_{\text{out},n}$ in S' if and only if there is a legal system traversal $s \to^* t$ in S.

First suppose there is a legal system traversal $s \to^* t$ in S. Then this solution can be extended to a legal system traversal $s \to^* c_{\text{out},n}$ in S' by appending the traversal $c_{\text{in},i} \to c_{\text{out},i}$ on G'_i for each i in increasing order, and in the planar case, adding the needed traversals of the inserted copies of CHX (including the check traversals needed to get from t to $c_{\text{in},1}$

and from each $c_{\text{out},i}$ to $c_{\text{in},i+1}$). The appended $c_{\text{in},i} \to c_{\text{out},i}$ traversals are all valid because Property 3 of Definition 10 requires that any legal traversal sequence for G can be extended by $c_{\text{in}} \to c_{\text{out}}$ to yield a legal traversal sequence for G'. For the same reason, the appended $c_{\text{in}} \to c_{\text{out}}$ traversals in copies of CHX are valid. Also, the inserted hallway traversals of the copies of CHX are all valid from the definition of checkable hallway crossover, because they occur before all appended $c_{\text{in}} \to c_{\text{out}}$ traversals.

Now suppose that there is a legal system traversal $s \to^* c_{\text{out},n}$ in S'. Define $c'_{\text{in},i}, c'_{\text{out},i}$ to be the check in and out locations for all checkable gadgets (copies of both G' and CHX), in the order that these check traversals occur in the intended solution described above. By Property 4 of Definition 10, the agent can only exit the ith checkable gadget (G' or CHX) at $c'_{\text{out},i}$ if it previously entered at the corresponding $c'_{\text{in},i}$. In S', the only location connected to $c'_{\text{in},i+1}$ is $c'_{\text{out},i}$ (ignoring hallway traversals of CHX gadgets), so this property implies that $c_{\text{out},i}$ was previously visited as well. By induction, the solution must have reached $c'_{\text{in},1}$ via t, and then traversed all of the $c'_{\text{in},i}$ and $c'_{\text{out},i}$ locations (possibly with some detours). Consider the prefix X' of the solution up to the first time t is visited, and let X be the modification to remove any hallway traversals of the copies of CHX. We claim X is a solution for S. Clearly X is a system traversal $s \to^* t$ and satisfies all unmodified gadgets (from $\mathcal{G}'$). By Property 4 of Definition 10, $c'_{\text{in},i}$ and $c'_{\text{out},i}$ are visited at most once in the full solution, and the prefix of the solution prior to visiting $c'_{\text{in},i}$ is legal for the ith checked gadget. Because each $c'_{\text{in},i}$ is visited after t, it is not visited in X, and thus X is legal for G_i. Similarly, X makes only hallway traversals of CHX, so removing those traversals is valid in S where there were direct connections before the crossings were introduced. Therefore X is a valid system traversal $s \to^* t$ in S. ◀

4.4 Postselected Gadgets

We now finally prove our main result, Theorem 8: postselection can be achieved using only the two base gadgets from Section 4.1, while preserving planarity.

It will be convenient to assume all of our gadgets are ***transitive***: if there are two transitions $(q_1, \ell_1) \to (q_2, \ell_2) \to (q_3, \ell_3)$, then there is also a transition $(q_1, \ell_1) \to (q_3, \ell_3)$. For reachability, this makes no difference: we can replace any gadget with its transitive closure without affecting the answers to any reachability problems, since we can always think of the transition $(q_1, \ell_1) \to (q_3, \ell_3)$ as a sequence of two transitions. That is, every gadget is equivalent for reachability to some transitive gadget, and in particular there are nonlocal simulations in both directions.

Proof of Theorem 8. Assume without loss of generality that G is transitive, by replacing G with its transitive closure.

We will show that $\{G, \text{SO}, \text{MSC}, \text{SD}, \text{SX}, \text{WCX}\}$ planarly *locally* simulates some gadget G' which is a simply checkable $\mathsf{Postselect}(G, C, L')$. As shown in Section 4.1 (Figures 19 and 20 in particular), $\{\text{SO}, \text{MSC}\}$ planarly locally simulates WCX, SX, and SD. By combining these local simulations, we obtain that $\{G, \text{SO}, \text{MSC}\}$ planarly locally simulates the same G'. By Lemma 2, this is also a nonlocal simulation. By Lemma 11, for any checkable hallway crossover gadget CHX, $\{G', \text{CHX}\}$ planarly nonlocally simulates G'. Because $\{\text{SO}, \text{MSC}\}$ planarly simulates the weak closing crossover (Figure 20), which is a checkable hallway crossover, it follows from Lemma 9 that $\{G, \text{SO}, \text{MSC}\}$ planarly nonlocally simulates $\mathsf{Postselect}(G, C, L')$, proving the theorem.

Now we show that $\{G, \text{SO}, \text{MSC}, \text{SD}, \text{SX}, \text{WCX}\}$ planarly locally simulates some gadget G' which is a simply checkable $\mathsf{Postselect}(G, C, L')$. Unpacking the definitions of "simply checkable" and Postselect, we must simulate a gadget G' that satisfies the following properties:

1. $L(G') = L' \sqcup \{c_{\text{in}}, c_{\text{out}}\}$.
2. There is a function f from unbroken states of G to states of G'.
3. For any traversal sequence X on L', if XC is legal for G from state q, then $X \cdot [c_{\text{in}} \to c_{\text{out}}]$ is legal for G' from state $f(q)$.
4. Any traversal sequence that ends with c_{out} and is legal for G' from state $f(q)$ has the form $X \cdot [c_{\text{in}} \to \bullet, \bullet \to \bullet, \dots, \bullet \to c_{\text{out}}]$, where X is a traversal sequence on L', XC is legal for G from state q, and all the omitted $\bullet$ locations are in L'.

We construct our simulation of the gadget G' starting from G as follows; refer to Figure 22.

1. For purposes of description, orient so that G has all of its locations on the top of its bounding box. We will place the locations for the simulated gadget on a horizontal line L above G (so they will lie on the outside face).
2. For each location $l \in L'$, add a long upward edge e_l connecting l in G to a new location l' on L. Because the edges are all vertical, they do not cross each other, and the l' locations appear in the same cyclic (left-to-right) order as $l \in L'$.
3. Place c_{in} on L left of all e_l edges. Starting from c_{in}, draw a non-self-crossing path that crosses each of the e_l in one rightward pass, then turn down, then cross each e_l a second time in one leftward pass in between the first pass and G. We ensure any further crossings with the edges e_l take place between these two delimiter passes, which we call the top and bottom delimiters, by routing paths across the bottom delimiter before crossing any e_l. These delimiters serve to "cut off" the rest of the construction, preventing leakage.
4. For each traversal $a_i \to b_i$ in the sequence $C = [a_1 \to b_1, \dots, a_k \to b_k]$, add a single-use opening gadget O_i and a dicrumbler D_i, near locations b_i and a_i respectively. Connect the opening location of O_i to the entrance of D_i (routing up across the bottom delimiter, then horizontally, then down). Connect the exit of D_i to a_i, and connect b_i to the entrance of O_i.
5. Connect the exit of each O_i to the opening location of O_{i+1}, routing up across the bottom delimiter, then all the way left, then up, then right, then down.
6. Finally, connect c_{in} to the opening location of O_1 after the two delimiter passes; and connect the exit of O_k to c_{out}, routing up across the bottom delimiter, then all the way left, then up.

We call the path we have constructed from c_{in} to c_{out} the ***checking path***. For an unbroken state q of G, the corresponding state $f(q)$ of G' is simulated by placing G in state q and all other gadgets in their usual initial states.

This construction is nonplanar in two ways: our new checking path crosses the edges e_l and also crosses itself. In the former case we replace the crossing with a weak closing crossover, oriented so that the checking path closes e_l. In the latter case we replace the crossing with a single-use crossover, oriented correctly so that the agent can traverse the two directions in the expected order detailed below. We must prove this construction has the properties stated above. By construction, its locations are $L' \sqcup \{c_{\text{in}}, c_{\text{out}}\}$.

Suppose XC is legal for G from state q. We can perform $X \cdot [c_{\text{in}} \to c_{\text{out}}]$ in the simulation where G starts in q by first performing X in the natural way (using the edges e_l) and then following the checking path: starting at c_{in}, for each i we visit the opening location of O_i, then go through D_i, then traverse $a_i \to b_i$ via G, then traverse O_i. This path brings us to c_{out} at the end, and its restriction to G is exactly XC.

Now suppose that there is a legal traversal sequence for G' from state $f(q)$ ending in c_{out}. Putting ourselves in the position of a forgetful agent, we find ourselves at c_{out} and must determine how we got there. We can induct backwards along the checking path (as in the

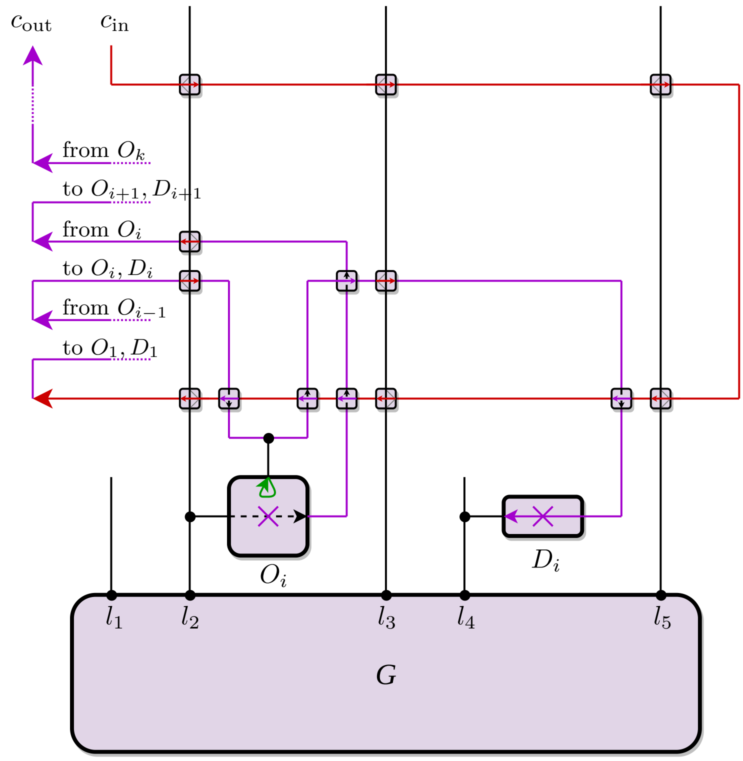


Figure 22 The simulation of a simply checkable, postselected version of the gadget G. The two initial crossings of the edges e_l connecting locations in L' to the outside are shown in red. The rest of the checking path is shown in purple. All further crossings of the checking path with edges e_l occur between the two initial crossings. In this example, $L = \{l_1, l_2, l_3, l_4, l_5\}$ and $L' = \{l_2, l_3, l_5\}$. The ith checking traversal $[l_4 \to l_2]$ is enforced by O_i and D_i.

proof of Lemma 11) to show that we must have visited c_{in}, using the facts that in order to exit the closing side of a weak closing crossover we must have entered it on the opposite side, and that in order to exit from O_i we must have visited its opening location.

Thus at some point in the path we entered G' through c_{in}, crossed all the e_l twice, and then for every $a_i \to b_i$ of C in order we opened O_i, traversed D_i, and later traversed O_i. Crossing each e_l twice closes the weak closing crossovers, making e_l no longer traversable. Between traversing D_i and O_i, we somehow must have gotten from a_i to b_i. We cannot have used the edges e_l because they were already closed during the initial crossings. So we must have made transitions only in G, of the form $(q_1, \ell_1 = a_i) \to (q_2, \ell_2) \to \cdots \to (q_k, \ell_m = b_i)$. Since G is transitive, we could equivalently have made the single transition $(q_1, a_i) \to (q_k, b_i)$, and in particular have traversed $a_i \to b_i$.

Similarly, after the initial two crossings of the e_l, we can't have left this simulated gadget or entered G except for the traversals of C. Finally, we take advantage of the fact that before entering c_{in}, the simulation behaves exactly like G except that only locations in L' are accessible. So the full path through the simulation G' ending at c_{out} must have the following form:

1. We use G' as if it were G (restricted to the locations of L') with initial state q, performing some traversal sequence X.
2. We enter G' through c_{in}.
3. We possibly leak out of G' or into G via locations in L', through the weak closing

crossovers at the initial two crossings with each e_l. Call the sequence of traversals made during this phase Y.

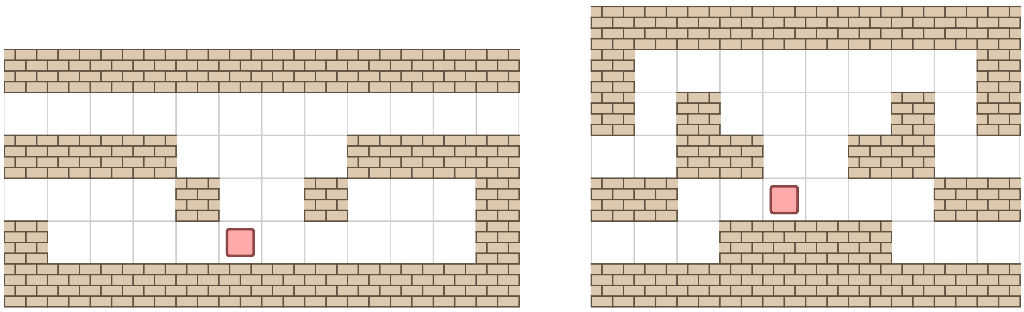
4. Eventually, we finish all of initial crossings with e_l, and moved to the O_is and D_is.
5. We perform the traversal sequence C in G without any additional traversals in G in between and without leaving G'.
6. Finally, we leave G' through c_{out}.

Therefore the sequence of traversals on G' has the form $X \cdot [c_{\text{in}} \to \bullet, \bullet \to \bullet, \dots, \bullet \to c_{\text{out}}]$ and the sequence of traversals just on G is XYC, where X and Y are traversal sequences on L' and the omitted $\bullet$ locations are in L'. In particular, XYC is legal for G from state q, so by the assumption that broken states are preserved by transitions on L', XC is legal for G from q. This is the final condition we needed, so G' is a simply checkable Postselect(G, C, L'). ◀

5 BoxDude is PSPACE-complete

We now show that BoxDude is PSPACE-complete via a reduction from reachability with nondeterministic locking 2-toggles. In this model, boxes can be pushed horizontally by the Dude but cannot be picked up. We will make use of the postselection construction from Section 4 in order to nonlocally simulate nondeterministic locking 2-toggles.

Similarly to BlockDude we must build a branching hallway in order to connect the locations of our gadgets. This time, we also build a directed crossover gadget. These gadgets are shown in Figure 23. Directed crossovers can be used to construct undirected crossovers as in Figure 24. This allows us to connect locations in nonplanar ways, and reduce from reachability instead of planar reachability. We note a diode gadget is easy to build by simply having a height 2 drop.

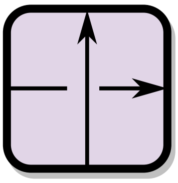


(a) Branching Hallway **(b)** Directed Crossover

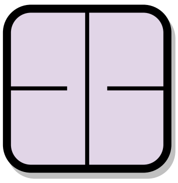
Figure 23 Hallway connection gadgets for BoxDude. Pushable boxes are in red. The branching hallway gadget is fully traversable from any of its three locations to the others. The directed crossover can be traversed only from bottom-left to top-right or from bottom-right to top-left.

Postselection requires us to additionally simulate the gadgets SO and MSC. These gadgets are shown in Figure 25.

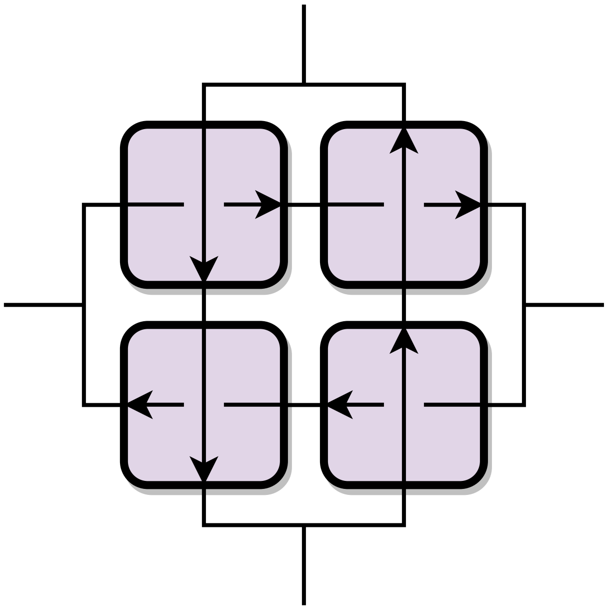
Next we build a checkable ***leaky door*** gadget. A leaky door has two states (“open” and “closed”), and three locations, called “opening”, “entrance”, and “exit”. Similar to a self-closing door [3], the gadget can be traversed in the open state from entrance to exit, but doing so transitions the door to the closed state. In the closed state, it is not possible to enter the gadget through the entrance at all, but visiting the opening location allows the gadget to transition back to the open state. Unlike a self-closing door, it is possible to go from the entrance to the opening location when the gadget is in the open state. It is also



(a) Directed Crossover Icon

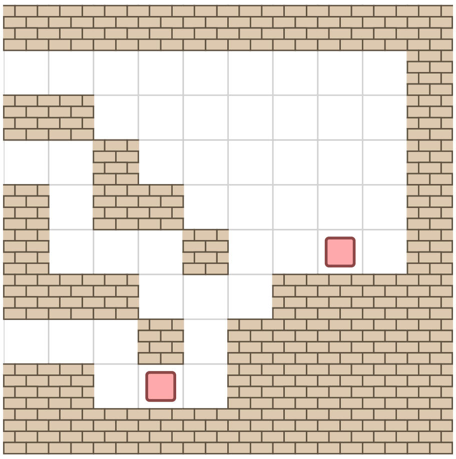


(b) Crossover Icon

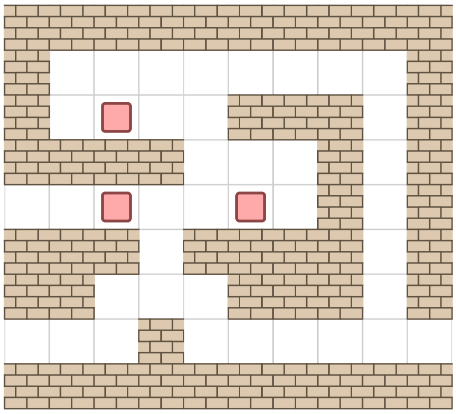


(c) Construction of an undirected crossover from a directed crossover.

Figure 24 Icons for directed and undirected crossovers. The undirected crossover can be constructed from four directed crossovers as shown in [10].



(a) SO



(b) MSC

Figure 25 SO and MSC gadgets for BoxDude.

always possible to go from the opening location to the exit, but doing so transitions the door to the closed state. The full state diagram for the leaky door is shown in Figure 26.

The checkable leaky door is shown in Figure 27. We apply postselection to this gadget with the checking traversal sequence [opening $\to$ opening, entrance $\to$ opening].[6] We now

[6] The first check from opening to opening does not enforce anything but merely allows access to the location in case the gadget was last left in the closed state. The check from entrance to opening cannot be done if the gadget is in the closed state.

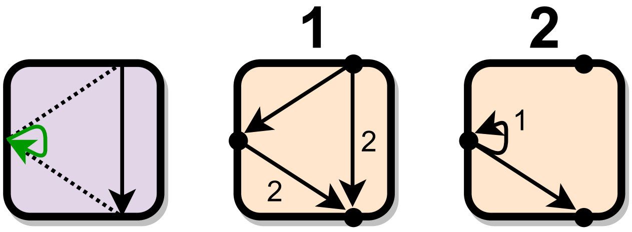


Figure 26 Icon and state diagram for the leaky door gadget.

analyze which states are ***broken*** in the sense that this traversal sequence is impossible from those states.

- If the left box is further to the left than its current location, the gadget state is broken since the entrance is unusable.
- If the left box is more than one square to the right of its current location, the gadget state is broken because the opening location is unreachable from the entrance.
- If the two boxes are adjacent, the gadget state is broken for the same reason.

Moving the right box more than one square to the right is never advantageous for the player, so we assume it does not occur.

We will show that the postselection of this gadget is exactly the leaky door gadget. When the right box is in its current location, we say that the gadget is in the closed state; when it is one square to the right the gadget is in the open state. Because the left box cannot move more than one square to the right, it follows that any traversal to the exit location must leave the gadget in the closed state. In the closed state, no traversals are possible from the entrance without breaking the gadget by putting two boxes adjacent. Visiting the opening allows transitioning to the open state. In the open state, additional traversals are available from the entrance. The agent may go from entrance to exit by using the connected opening locations to reset the gadget to the closed state and then using the right block to reach the exit. It is also possible to leak from the entrance to the opening location, and from the opening location to the exit (transitioning to the closed state). Thus the traversals within unbroken states are exactly those allowed by the leaky door gadget. By Theorem 8 the checkable leaky door, along with the SO and MSC gadgets built earlier, nonlocally simulate the leaky door.

We now build a 1-toggle gadget, shown in Figure 10, using a pair of leaky doors. This construction is shown in Figure 28. It can be seen that none of the leaks are useful to an agent traversing the gadget, since the most they accomplish is bringing the agent back to its starting location without changing any state.

We are now in a position to build a nondeterministic locking 2-toggle. By Theorem 4, reachability with this gadget is PSPACE-complete. The final construction, shown in Figure 29, is quite simple in appearance; the complexity is hidden in the 1-toggles used to protect the locking 2-toggle's locations. Traversing from A to B is only possible when the box is on the left side of the gadget, and conversely for C to D. Since the box's position can only be changed when exiting the gadget through A or C (corresponding to which side the gadget is locked to), the gadget simulates a locking 2-toggle. Note that this gadget cannot be broken by moving the box further to the left than its current position, since doing so renders the gadget fully untraversable. This is because in this state location A is permanently unusable and B and D cannot be reached from inside the gadget. The agent can only exit out of C,

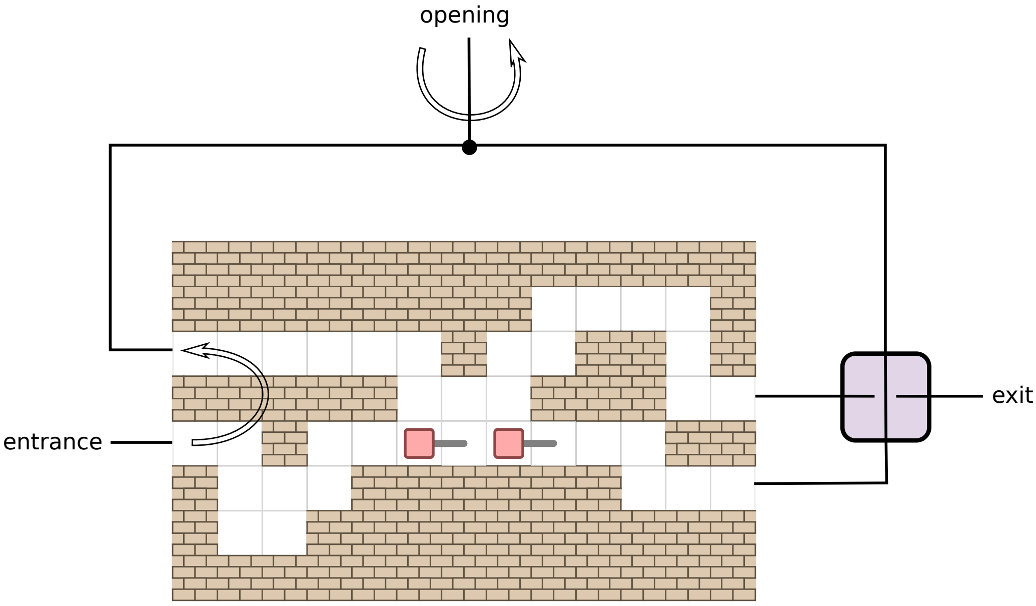


Figure 27 A checkable leaky door, shown in the closed state. The crossover and branching hallway needed to connect the top left and bottom right hallways have been abstracted. Horizontal "tracks" display the range of locations for each box in unbroken states. (The right box can move farther right but it is never advantageous to do this.) The two boxes may not be adjacent in unbroken states.

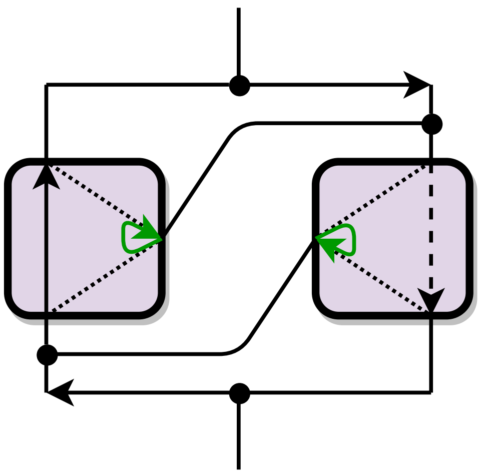


Figure 28 A 1-toggle built from leaky doors. Solid or dashed arrows inside gadgets show the traversal from entrance to exit in an open or closed leaky door, respectively. Green self-loops are opening locations of leaky doors. Arrows outside gadgets are diodes.

so that C's 1-toggle points inwards. Since C's and D's 1-toggles always point in different directions, D is also permanently unusable. The only remaining traversal is $B \to C$, but this is impossible also because C's 1-toggle points inwards.

Using Theorem 8 and Lemma 9, our simulations imply that the BoxDude gadgets we have explicitly built nonlocally simulate a nondeterministic locking 2-toggle. In particular,

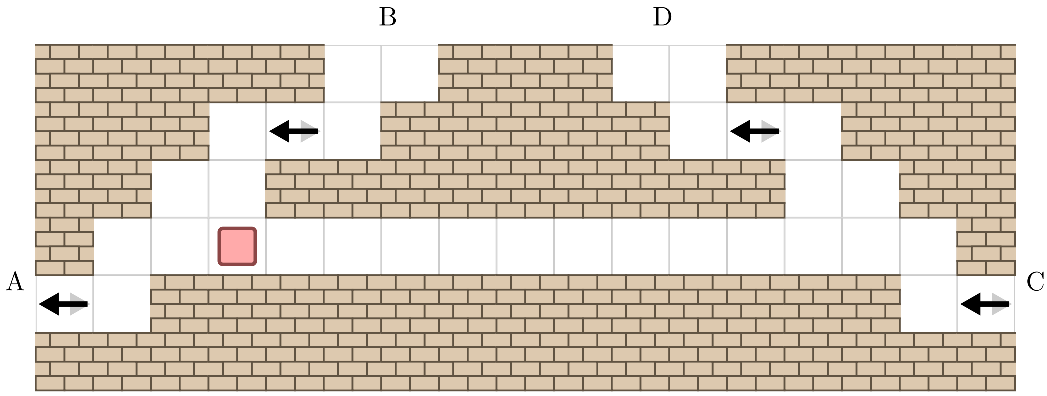


Figure 29 A nondeterministic locking 2-toggle, currently locked to the left side. Locations B and C are protected with inwards-directed 1-toggles; locations A and D with outwards-directed 1-toggles. (Note: the middle portion of the gadget would actually need to be wider than shown in this diagram in order to make enough space to route locations B and D away from each other.)

there is a polynomial-time reduction from planar reachability with nondeterministic locking 2-toggles, which is PSPACE-complete by Theorem 4, to BoxDude. Hence BoxDude is PSPACE-complete.

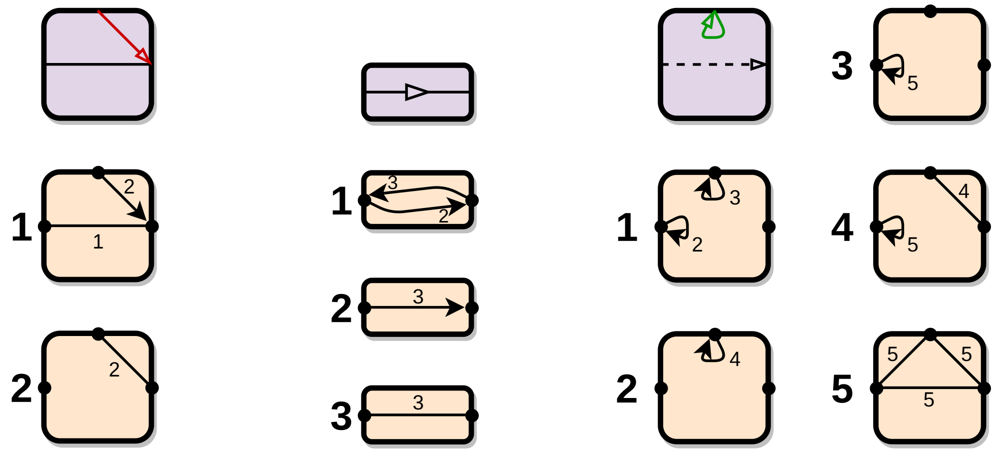
6 Push-1F is PSPACE-complete

In this section, we show that Push-1F is PSPACE-complete using a reduction from planar reachability with self-closing doors, shown in Figure 8, which is PSPACE-complete by Theorem 5. Recall that in this model there is no gravity, and the agent can push one block at a time in any direction. We will make several uses of postselection from Section 4 in order to nonlocally simulate various gadgets along the way.

In order to use postselection, we must build single-use opening (SO) and merged single-use closing (MSC) gadgets. We start by building a ***weak merged closing*** gadget, based on the Lock gadget from [8]. The weak merged closing gadget acts like the MSC except that the closing traversal can be performed multiple times. We also use a gadget introduced in [8] called a ***no-return*** gadget. After a no-return gadget is traversed from left to right, it cannot immediately be traversed from right to left. However, initially traversing it from the right or traversing left to right twice breaks the gadget, making it fully traversable. Finally, we build a ***weak opening*** gadget. A weak opening gadget's exit cannot be used in traversals until both of its input locations are visited separately. Figure 30 shows the state diagrams for these gadgets, and Figure 31 shows how to implement them in Push-1F.

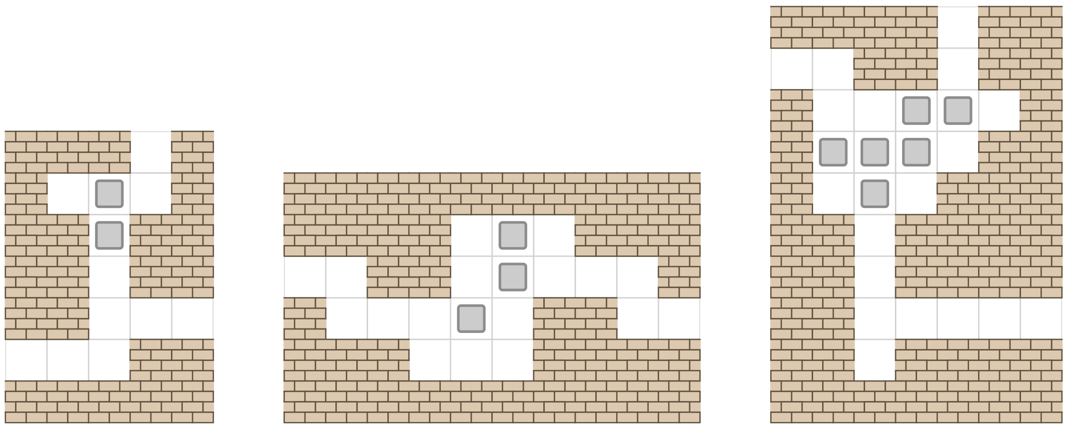
We combine the weak merged closing, no-return, and weak opening gadgets to make a dicrumbler; this allows us to simulate ordinary SO and MSC gadgets using the gadgets we have built so far. These simulations are shown in Figure 32. Having built these gadgets, we can now take advantage of the machinery of checkable gadgets. The structure of the remaining gadget simulations used in this section is outlined in Figure 33.

We first nonlocally simulate a diode, which allows traversal in only one direction. We accomplish this by building a checkable ***protodiode***, where the protodiode is a certain four-location gadget which easily simulates a diode. Refer to Figure 34. We apply postselection to the checkable protodiode with the checking traversals $[A \to C, D \to B]$ to nonlocally simulate the protodiode. The nonbroken states are exactly those in which the block is confined to the middle two squares. Connecting the bottom two locations of the protodiode yields a diode.



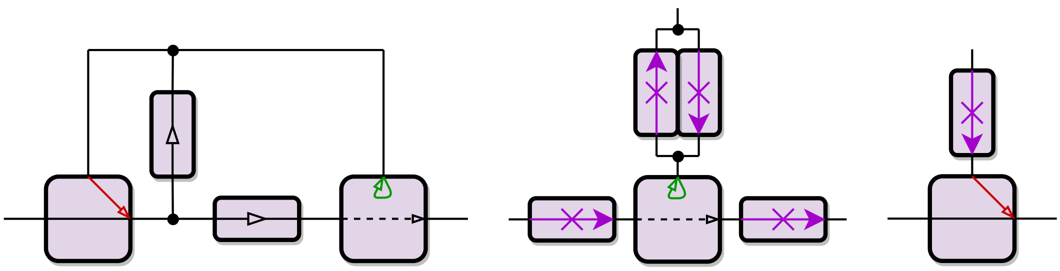
(a) Weak merged closing (b) No-return (c) Weak opening

Figure 30 Icons and state diagrams for Push-1F base gadgets.



(a) Weak merged closing (b) No-return (c) Weak opening

Figure 31 Constructions of base gadgets for Push-1F.



(a) Dicrumbler (b) SO (c) MSC

Figure 32 Constructions of gadgets required for postselection in Push-1F.

We now nonlocally simulate a ***precursor*** gadget, which will be used to build a 1-toggle and a checkable self-closing door. The precursor's state diagram is shown in Figure 35d. We

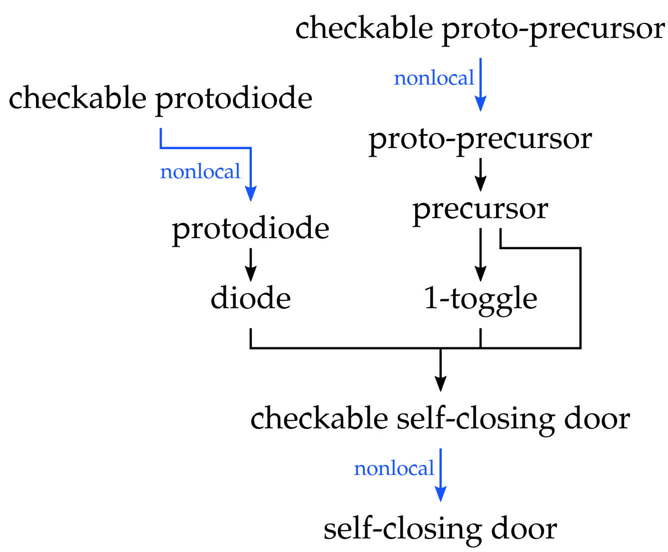
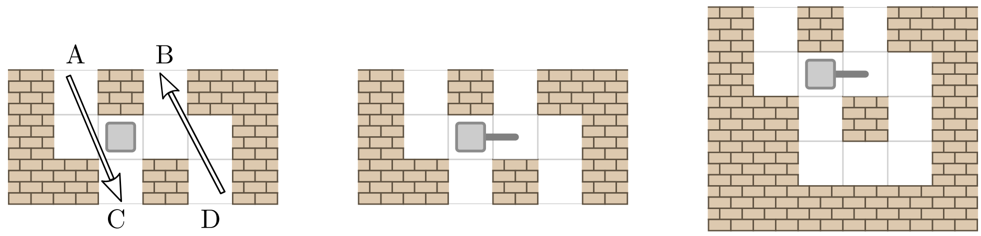


Figure 33 Overview of gadget simulations used for Push-1F. Black arrows show local simulations and blue arrows show nonlocal simulations.



(a) Checkable protodiode and checking traversals

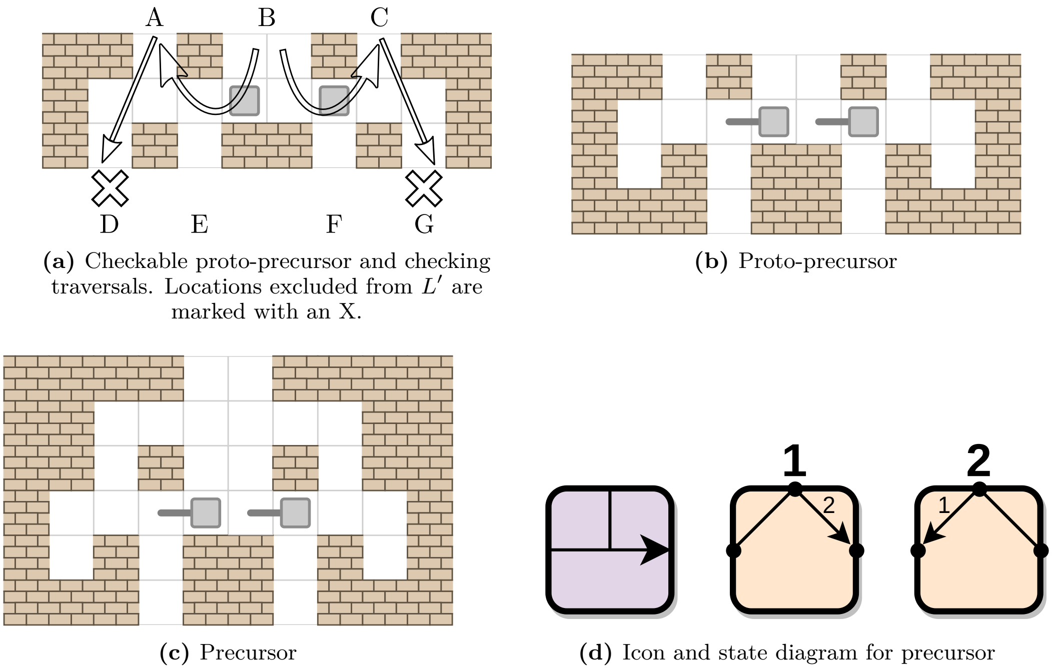
(b) Protodiode

(c) Diode

Figure 34 Nonlocal diode simulation for Push-1F. Horizontal tracks show where the block is allowed to move in the protodiode and diode, as if it is confined by a magical force.

begin by building a checkable ***proto-precursor***, where again the proto-precursor is a certain gadget which easily simulates the precursor. Refer to Fig 35. We apply postselection to the checkable proto-precursor with the checking traversals $[A \to D, C \to G, B \to A, B \to C]$. We close off locations D and G during postselection by not including them in the set $L' = \{A, B, C, E, F\}$ of locations on the proto-precursor. The nonbroken states are exactly those in which the blocks are confined to the four center-most spaces, and the two blocks are not adjacent. Entering a broken state is irreversible with respect to transitions on the locations in L' because D and G were excluded in L'. (If D or G were included then it would be possible to un-break the gadget from some broken states by pushing a block back into the center.) Thus we can use postselection to nondeterministically simulate the proto-precursor; joining its upper three locations together yields the precursor gadget. Additionally, closing the top location of the precursor gadget produces a 1-toggle.

Finally, we nonlocally simulate a self-closing door. Our construction of a checkable self-closing door is shown in Figure 36. This gadget is almost identical to a self-closing door, except that it permits a traversal from the opening location to the exit location exactly once, after which the gadget is fully untraversable. We eliminate this problem by applying postselection with the checking traversal sequence $[\text{opening} \to \text{opening}, \text{entrance} \to \text{exit}]$.



(a) Checkable proto-precursor and checking traversals. Locations excluded from L' are marked with an X.

(b) Proto-precursor

(c) Precursor

(d) Icon and state diagram for precursor

Figure 35 Nonlocal precursor simulation for Push-1F. As before, horizontal tracks in the proto-precursor and precursor show spaces to which blocks are magically confined. The magical force also prevents the pair of blocks in the proto-precursor and precursor from being adjacent.

The sole broken state is the fully untraversable one arising from the aforementioned undesired traversal. If we imagine that a magical force prevents the gadget from being left in such a state, then we obtain exactly a self-closing door.

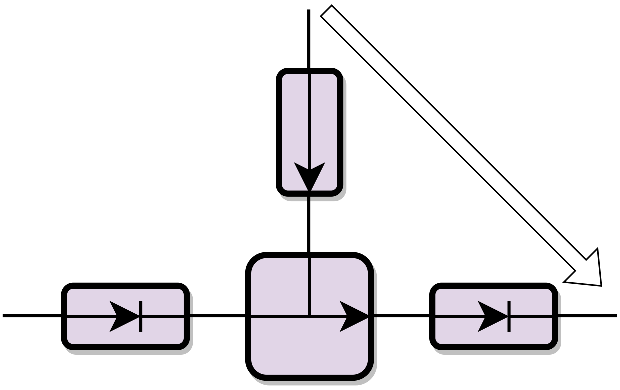


Figure 36 Checkable self-closing door for Push-1F using the precursor gadget, two diodes, and a 1-toggle.

We have demonstrated a series of planar, nonlocal gadget simulations culminating in the planar nonlocal simulation of a self-closing door. Because planar reachability through systems of self-closing doors is PSPACE-complete by Theorem 5, so is Push-1F.

7 Open Problems

The primary remaining question is the complexity of Push-1 block puzzles where there are no fixed blocks allowed in the puzzle. Push-1 can easily simulate fixed blocks using 2×2 arrangements of movable blocks, so we only need to make all fixed areas two blocks thick. Our constructions of the gadgets SO and MSC needed to apply postselection all use two-block thick spacing, so we have shown that postselection is available for Push-1 gadgets. Unfortunately, our postselected constructions for Push-1F critically use one-block-thick spacing.

Another question we do not address is the related block storage question for $\cdots$Dude puzzles, named $\cdots$Duderino in [5], in which the blocks have target locations to occupy. This is comparable to the difference between Push-1F and Sokoban. It is generally expected that the storage version of block-pushing puzzles is at least as hard as reaching a single goal location; however, this result does not directly follow. We believe using the reconfiguration version of the gadgets framework from [4] may help build a gadget-based proof.

We have another open question related to the technique of postselected gadgets. When defining a postselected gadget, we only specified a single traversal sequence to be checked. It seems likely that one could enforce the choice of one of several possible sequences using more complex constructions like those found in the SAT reduction for DAG gadgets in [11]. Are there cases where this sort of flexibility is useful?

Acknowledgments

This work was initiated during extended problem solving sessions with the participants of the MIT class on Algorithmic Lower Bounds: Fun with Hardness Proofs (6.892) taught by Erik Demaine in Spring 2019. We thank the other participants for their insights and contributions.

We would like to thank our reviewers for their detailed and useful feedback.

We would like to thank Aaron Williams for useful discussion including how to restructure the paper and how to better present the results and checkable gadget framework.

Figures produced using SVG Tiler (https://github.com/edemaine/svgtiler), diagrams.net, and Inkscape.

References

1 Scott Aaronson. Quantum computing, postselection, and probabilistic polynomial-time. *Proceedings of the Royal Society A*, 461:3473–3482, 2005. doi:http://doi.org/10.1098/rspa.2005.1546.

2 Joshua Ani, Sualeh Asif, Erik D Demaine, Yevhenii Diomidov, Dylan Hendrickson, Jayson Lynch, Sarah Scheffler, and Adam Suhl. PSPACE-completeness of pulling blocks to reach a goal. *Journal of Information Processing*, 28:929–941, 2020.

3 Joshua Ani, Jeffrey Bosboom, Erik D. Demaine, Yevhenii Diomidov, Dylan Hendrickson, and Jayson Lynch. Walking through doors is hard, even without staircases: Proving PSPACE-hardness via planar assemblies of door gadgets. In *Proceedings of the 10th International Conference on Fun with Algorithms (FUN 2020)*, Favignana, Italy, September 2020.

4 Joshua Ani, Erik D. Demaine, Yevhenii Diomidov, Dylan H. Hendrickson, and Jayson Lynch. Traversability, reconfiguration, and reachability in the gadget framework. In Petra Mutzel, Md. Saidur Rahman, and Slamin, editors, *Proceedings of the 16th International Conference and Workshops on Algorithms and Computation (WALCOM 2022)*, volume 13174 of *Lecture*

Notes in Computer Science, pages 47–58, Jember, Indonesia, March 2022. doi:10.1007/978-3-030-96731-4_5.

5 Austin Barr, Calvin Chang, and Aaron Williams. Block dude puzzles are NP-hard (and the rugs really tie the reductions together). In *Proceedings of the 33rd Canadian Conference on Computational Geometry*, Halifax, Canada, 2021.

6 Joseph Culberson. Sokoban is PSPACE-complete. In *Proceedings of the International Conference on Fun with Algorithms*, pages 65–76, Elba, Italy, June 1998.

7 Erik D. Demaine, Martin L. Demaine, Michael Hoffmann, and Joseph O'Rourke. Pushing blocks is hard. *Computational Geometry: Theory and Applications*, 26(1):21–36, August 2003.

8 Erik D. Demaine, Martin L. Demaine, and Joseph O'Rourke. PushPush and Push-1 are NP-hard in 2D. In *Proceedings of the 12th Annual Canadian Conference on Computational Geometry (CCCG 2000)*, pages 211–219, Fredericton, Canada, August 2000.

9 Erik D. Demaine, Isaac Grosof, Jayson Lynch, and Mikhail 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)*, pages 18:1–18:21, La Maddalena, Italy, June 2018.

10 Erik D. Demaine, Robert A. Hearn, and Michael Hoffmann. Push-2-F is PSPACE-complete. In *Proceedings of the 14th Canadian Conference on Computational Geometry*, pages 31–35, Lethbridge, Canada, August 2002.

11 Erik D. Demaine, Dylan Hendrickson, and Jayson Lynch. Toward a general theory of motion planning complexity: Characterizing which gadgets make games hard. In *Proceedings of the 11th Conference on Innovations in Theoretical Computer Science*, Seattle, Washington, January 2020. arXiv:1812.03592.

12 Dylan Hendrickson. Gadgets and gizmos: A formal model of simulation in the gadget framework for motion planning. Master's thesis, Massachusetts Institute of Technology, 2021.

13 Jayson Lynch. *A framework for proving the computational intractability of motion planning problems*. PhD thesis, Massachusetts Institute of Technology, 2020.

14 Joseph O'Rourke and The Smith Problem Solving Group. PushPush is NP-hard in 3D. arXiv:cs/9911013, November 1999. https://arXiv.org/abs/cs/9911013.