PushPush-k is PSPACE-Complete

Erik D. Demaine
Computer Science and Artificial Intelligence Laboratory, MIT
edemaine@mit.edu

Michael Hoffmann
Institute for Theoretical Computer Science, ETH Zürich
hoffmann@inf.ethz.ch

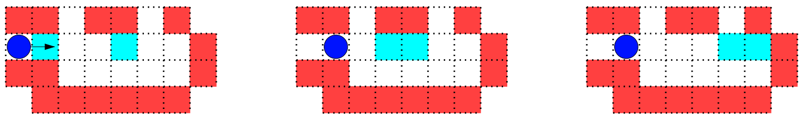
Markus Holzer
Institut für Informatik, Technische Universität München
holzer@in.tum.de

Abstract

We prove that a pushing-block puzzle called PUSHPUSH-k is PSPACE-complete for any fixed $k \geq 1$. In this puzzle, a robot moves on a finite grid. Each grid position is either empty or occupied by a single obstacle block. While moving, the robot may push obstacle blocks in direction of its movement, subject to certain constraints. In particular, once an obstacle block starts moving, it continues to do so until it hits another obstacle or the grid boundary. The problem is to decide whether the robot can navigate from a given start position to a specified goal position on the grid.

1. *Introduction*

Pushing blocks. The generic pushing-block puzzle involves a rectangular grid of squares, each either empty, occupied by one of several *blocks*, or occupied by a unique *robot*. The robot can move horizontally or vertically to an empty square, or the robot can *push* an adjacent block or blocks in the direction of movement, subject to certain constraints. The goal of the puzzle is for the robot to reach a specified goal location, starting from a given configuration of the board. The corresponding decision problem is to decide whether such a puzzle is solvable by a sequence of moves and pushes.



(a) Initially. (b) PUSHPUSH-2. (c) PUSHPUSHPUSH-2.

Figure 1: Example: a single move in different pushing models.

Puzzle Classification. In this paper, we consider the following two pushing constraints; see [4] for a survey of other related constraints and results. The PUSH-k constraint, for a fixed $k \geq 1$, is that the robot can push at most k consecutive blocks at once, and the PUSHPUSH constraint is that, when a block is pushed, it slides to its maximal extent, ending against another block or the boundary of the rectangular grid. Together, these two constraints are called PUSHPUSH-k. If the number of blocks which the robot can push at once is not restricted, we call the resulting constraint PUSHPUSH-$*$.

The PUSHPUSHPUSH-k constraint is similar to PUSHPUSH-k, but blocks do not necessarily stop to move when another block is hit. Instead they continue to slide until the amount of blocks sliding simultaneously exceeds k or they hit the boundary.

Related work. Several papers address specific pushing-block puzzles. It is easy to show that PUSHPUSH (and its variants) are in PSPACE. In particular, PUSHPUSH-1 was introduced by O'Rourke et al. [11], motivated by a computer game[1]. They proved that the natural generalization of the PUSHPUSH-1 puzzle to three dimensions is NP-hard. The proof was extended to the two-dimensional version by Demaine, Demaine, and O'Rourke [3]. An alternate proof, which simultaneously resolved the less-constrained puzzle PUSH-1, was then published by the same authors [2]. At the same time, Hoffmann [8] gave a construction showing that PUSH-$*$ is NP-hard. Subsequently, Demaine and Hoffmann [5] proved NP-hardness of PUSH-k for any $k \geq 1$. In particular, they showed that PUSH-1 and PUSHPUSH-1 remain NP-hard even when the solution path must not cross itself. In this last case, called PUSH-1-X and PUSHPUSH-1-X, the puzzles are also in NP, and hence are NP-complete. Without this noncrossing restriction, the puzzles do not obviously have polynomial-length solutions. Finally, Demaine, Hoffmann, and Hearn [6] considered a model called PUSH-k-F where some blocks may be marked as fixed, that is, immovable for the robot, and proved PUSH-k-F to be PSPACE-complete for $k \geq 2$. Previously, PUSH-$*$-F was shown to be PSPACE-complete by Bremner, O'Rourke, and Shermer [1].

On the other hand, Holzer and Schwoon [9] proved that the *Atomix* puzzle is PSPACE-complete. Atomix is similar to PUSHPUSH-1 in that only a single block

[1] See http://www.pushpush.net for a history of the game and its implementations.

can be pushed at once, and when pushed, it slides its maximal extent. One difference is that the pushing is not made by a robot, but rather by an external agent (the player), so any block can be pushed at any time (unless it is surrounded by other blocks). Another difference is the goal: in Atomix, the blocks are distinguished and must be assembled into a particular pattern.

Our results. This paper resolves the complexity of both PUSHPUSH-k and PUSH-PUSHPUSH-k by proving them PSPACE-complete for any $k \geq 1$. The reduction reuses certain ingredients from both the construction of Holzer and Schwoon [9] and the NP-hardness proof for PUSHPUSH-1 by Demaine et al. [3].

The paper is organized as follows. In the next section we outline the reduction from intersection of regular languages given by finite automata. Then in Section 3 we describe the implementation of the gadgets and prove their main properties. Finally, in Section 4 we give our main result and summarize the computational complexity of some pushing-block puzzles.

2. *Reduction*

Our reduction is from the PSPACE-complete problem of testing for non-empty intersection of regular languages [7, 10]. More precisely, we are given a number $n \geq 1$, and a sequence $A_1, \ldots, A_n$ of finite automata, with $A_i = (Q_i, \Sigma, \delta_i, q_i, F_i)$, for $1 \leq i \leq n$, all using a common alphabet Σ. The goal is to determine whether there is a word in Σ^* that is accepted by all A_i, $1 \leq i \leq n$.

We construct a PUSHPUSH-k instance that has a solution precisely if there is a word that is accepted by all given automata. The basic idea is to force the player to choose a word that is accepted by all finite automata and then to simulate the transition functions of the automata step-by-step.

The construction is based on the observation that any $(k+1) \times (k+1)$ square group of blocks is immobile by a PUSHPUSH-k robot, so we can view them as effectively pinned to the plane. In fact, most of the blocks in the constructed instance are arranged in such immobile groups; few blocks can be pushed by the robot.

An interesting property of PUSHPUSH-k—as opposed to, for example, PUSH-k—is that an obstacle block is sometimes helpful: it can be used to control the movement of other blocks. One can design a key-lock mechanism that requires an obstacle to be pushed to a certain position before the robot can pass. Hence, a truly movable obstacle block is a scarce and important resource in our instances. The robot has to accept some workload to get such a block into the desired position. In particular, this includes the simulation of a finite automaton.

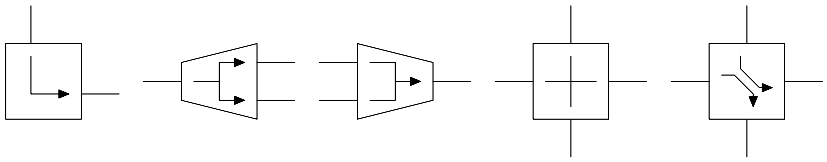


Figure 2: Symbols for main gadgets: (a) one-way gadget, (b) choice gadget, (c) merge gadget, (d) cross gadget, and (e) synchronize gadget.

2.1. *Gadget Symbols and Properties*

We introduce some symbols and basic properties of gadgets; see Fig. 2. Each gadget puts certain restrictions on how a single movable obstacle block can be routed through it. None of the basic gadgets restricts the robot's movement in any way. We delay describing the actual gadgets until Section 3, when their development is motivated.

The main properties of these gadgets are as follows.

One-way gadget: A block can be routed through this gadget in one direction only. One-way gadgets are implicit throughout other gadgets by arrow notation.

Choice gadget: An incoming block can be routed to either outgoing port.

Merge gadget: An incoming block from either port can be routed to outgoing port.

Cross gadget: Blocks can be routed through horizontally and vertically, but a block cannot take a turn between horizontal and vertical.

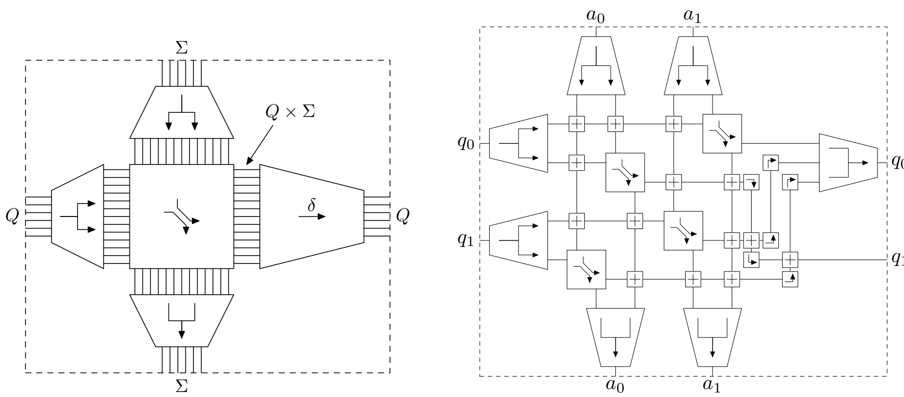
Synchronize gadget: Only when two incoming blocks have arrived they can pass through the central channel and be distributed to the two outgoing ports, with exactly one block exiting along each outgoing port.

Recall that the robot is free to pass through the gadgets in any direction. Also, all gadgets can be generalized to an arbitrary number of input and output channels.

2.2. *Single Step Transition*

Next we describe how to simulate a single step of a finite automaton $M = (Q, \Sigma, \delta, q_0, F)$, where Q is the set of states, Σ is the input alphabet, $\delta : Q \times \Sigma \to Q$ is the (deterministic) transition function, q_0 is the initial state, and F is the set of final (accepting) states. It is then not much harder to simulate several steps and put together several finite automata. Fig. 3 shows in part (a) the symbolic diagram of the transition simulation of the finite automaton, and in part (b) an example for a particular automaton with two states and two symbols, using the transition function $\delta(q_0, a_0) = q_1; \delta(q_0, a_1) = \delta(q_1, a_0) = \delta(q_1, a_1) = q_0$.

On the left of the construction, we suppose that there is a block in exactly one of the state channels $q \in Q$, representing the current state of the finite automaton. On the



(a) Symbolic diagram. (b) Example: 2 states and 2 symbols.

Figure 3: Transition simulation of an automaton.

top of the construction, we suppose that there is a block in exactly one of the symbol channels $a \in \Sigma$, representing the next input symbol.

The idea is that the transition function $\delta : Q \times \Sigma \to Q$ can be viewed as a matrix, where each row corresponds to a state, and each column corresponds to an input symbol. The construction contains a corresponding $|Q| \times |\Sigma|$ matrix of synchronize gadgets. The current state $q \in Q$ of the automaton determines a particular row in this matrix, and the next input symbol $a \in \Sigma$ determines a particular column, and we want the machine to go to the identified cell (q, a) of the matrix. Using choice gadgets, we let the state block choose which column it would like to visit, and we let the symbol block choose which row it would like to visit. For the blocks to make any progress beyond the matrix of synchronize gadgets, they must go to a common cell, which is precisely the desired cell (q, a).

As output, the simulation produces the same symbol a at the bottom of the construction (for later use in another finite automaton) and a new state at the right of the construction, determined by the transition function δ. Specifically, if a block enters the δ gadget on wire $(q, a) \in Q \times \Sigma$, the simulation produces the successor state $\delta(q, a)$. In Fig. 3(a), it is hidden that the implementation of the multiwire synchronize gadget uses a number of one-way and cross gadgets, as shown in Fig. 3(b).

2.3. *Simulation of Several Finite Automata*

Now we piece together several instances of Fig. 3 in order to simulate the intersection non-emptiness problem for several finite automata. Fig. 4 gives a schematic view of the construction; each box labeled "Automaton" corresponds to an instance of Fig. 3.

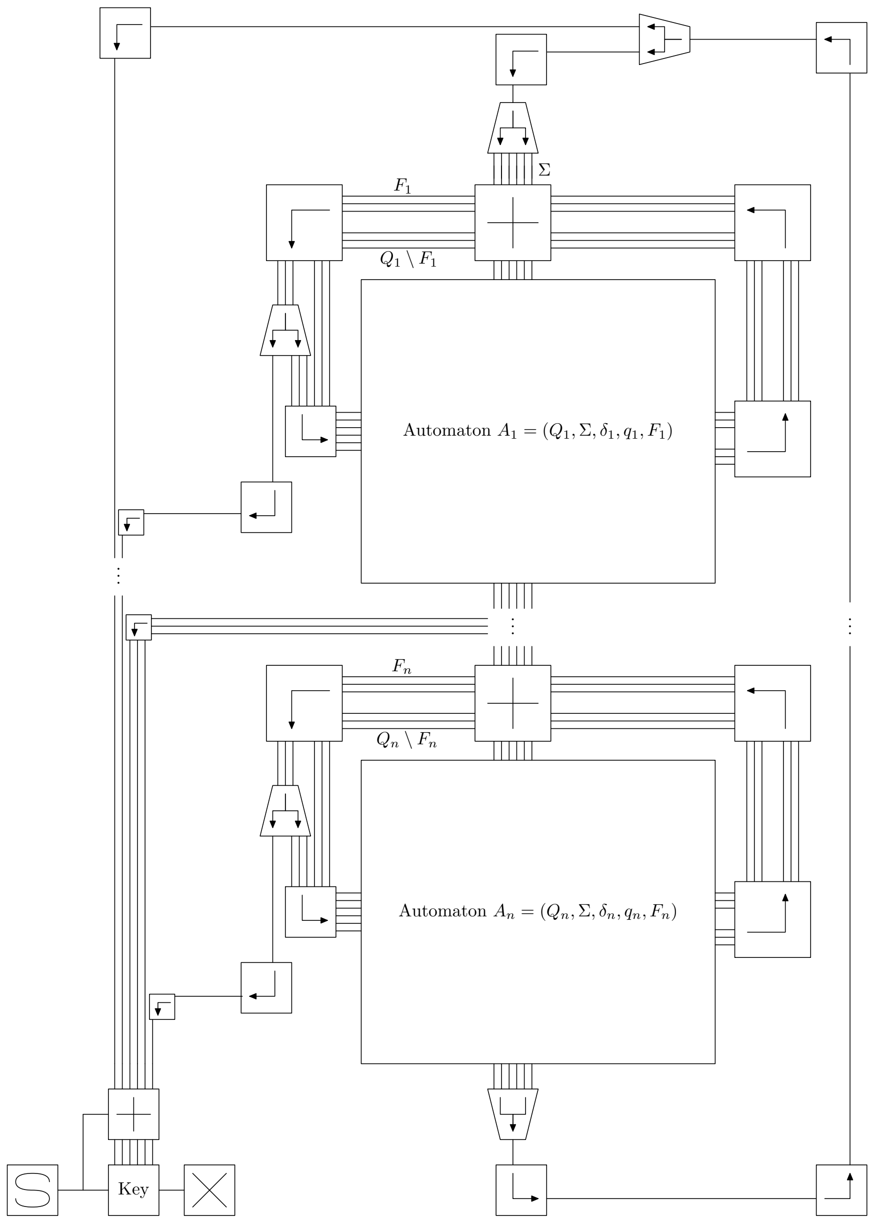


Figure 4: Non-empty intersection simulation for several finite automata.

First, the input state channel of each finite automaton contains an initial free block, thus specifying the initial state of the automaton under consideration. Moreover, each input state channel corresponding to a final (accepting) state has an alternate exit route labeled F_i. These exits enable the robot to route a block toward the lock gadget, which is located in the lower-left corner of the figure and is labeled by the word "Key." This lock corresponds to the automaton termination and accepting the guessed word.

Second, the output state channels loop back to the input state channels of each finite automaton, permeating this information. Third, the symbol channels are strung through all the finite automata, and finally loop back again, with a nondeterministic choice at the very top; this is where the input word is generated, symbol-by-symbol.

Finally, there are gadgets responsible for the robot's start and finish. Initially, the robot is located at the place marked with the large S. The sequence of locks prevent the robot from immediately accessing its goal position that is labeled with a large X. Because the symbol channels are all strung together, in fact every automaton must terminate and accept at the same time. Thus, one block per automaton can reach the "finish," unlocking the corresponding lock, precisely if there is a common word accepted by all of the finite automata.

3. *Gadgets*

This section details the PUSHPUSH gadgets used in the reduction of the previous section. We begin with some basic observations and conventions, following [3].

First, observe that any $(k + 1) \times (k + 1)$ group of blocks is immobile by a PUSHPUSH-k robot, so we can view them as effectively pinned to the plane. Such groups of blocks are shaded dark in our figures. We use such immobile groups to build *channels*: a channel is a $1 \times x$ or $x \times 1$ group of empty cells surrounded on both sides by two $(k+1) \times x$ or $x \times (k+1)$ groups of pinned blocks.

In all but the synchronize gadget, our constructions use these width-1 channels connected at orthogonal junctions of degree between two and four. We can view such a construction as an orthogonal graph, where the edges represent channels and the vertices represent orthogonal junctions. We use circles to denote movable blocks in the channels or at the channel junctions.

For the remainder of this section, we restrict our attention to PUSHPUSH-1. It is easy to extend the gadgets to the general PUSHPUSH-k constraint, in all cases except the lock gadget, by thickening the channel insulation. Note that even for $k > 1$ there is only one symbol block plus one state block for each automaton to be pushed around. In fact, the construction is done in such a way that the robot can never use its ability to push several blocks simultaneously.

3.1. *One-way, Choice, Merge, and Cross Gadgets*

We start with the gadgets depicted in Fig. 2 save the synchronize gadget. For each gadget, Fig. 5 shows the orthogonal-graph view on the top and an actual PUSHPUSH-1 puzzle on the bottom. These gadgets have the following obvious properties, which we state without proof. Except for the catalyst/synchronize gadget and the lock gadget, we assume that only one movable block resides in a particular gadget at any time. This property is fulfilled in our construction.

LEMMA 1 *In the one-way, choice, merge, corner, and cross gadget, the robot is free to travel in any direction. For blocks, the following properties hold, assuming that at most one block occupies the gadget at once: (1) In the one-way gadget, the robot may move a block from x to y, but not from y to x. (2) In the choice gadget, the robot may move a block only from x to y, or from x to z. (3) In the merge gadget, the robot may move a block only from x to z, or from y to z. (4) In the cross gadget, the robot may move a block only from x to z or y to z', or vice versa.*

3.2. *Synchronize Gadget*

We next detail the synchronize gadget, which is built out of two *catalyst chambers* wired in sequence as illustrated in Fig. 6(a). The difference between catalyst chamber and synchronize gadget is that a catalyst chamber has no "central channel", so the two blocks are not required to leave the gadget simultaneously. The second catalyst gadget acts as an "after-burner", ensuring that both blocks leave the first gadget. The catalyst gadget is illustrated in Fig. 6(b,d–f).

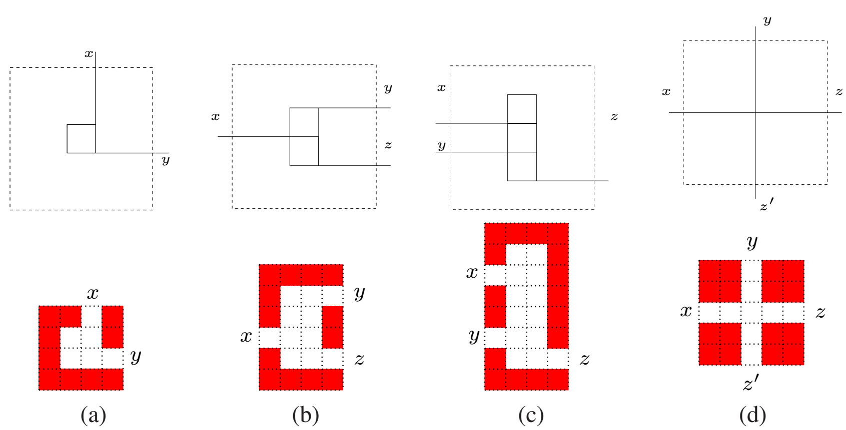


Figure 5: Implementations of (a) one-way, (b) choice, (c) merge, and (d) cross gadgets.

LEMMA 2 *In the catalyst gadget, (1) if the robot moves one block into the gadget alone from either x or y, it cannot move the block to either z or w; (2) if the robot moves two blocks into the gadget, there is a sequence of moves such that both blocks can leave the gadget through different output channels; and (3) if the robot moves two blocks into the gadget, the blocks cannot both be pushed through the same output channel z or w. The robot is free to travel in any direction.*

Proof: Property 1 follows because, once a block is moved into the gadget alone, it can only be pushed around cyclically inside the gadget, but never enter one of the output channels. Property 2 is shown in Fig. 6(d–g). Finally, for Property 3, the blocks can basically only be pushed around in the shaded cycle C in Fig. 6(b). As soon as the first block is pushed out from C, it either leaves the gadget via an output channel, leaves the gadget via an input channel, or it gets stuck in a corner forever. In the latter two cases, we are done. In the former case, which can only occur if both blocks are placed immediately adjacent on C, we are left with one block on C. Either this block is immediately pushed out, or it will be stuck in C as in Property 1. This concludes the proof, because there are no two neighboring positions on the cycle from which blocks can be pushed out to the same channel. □

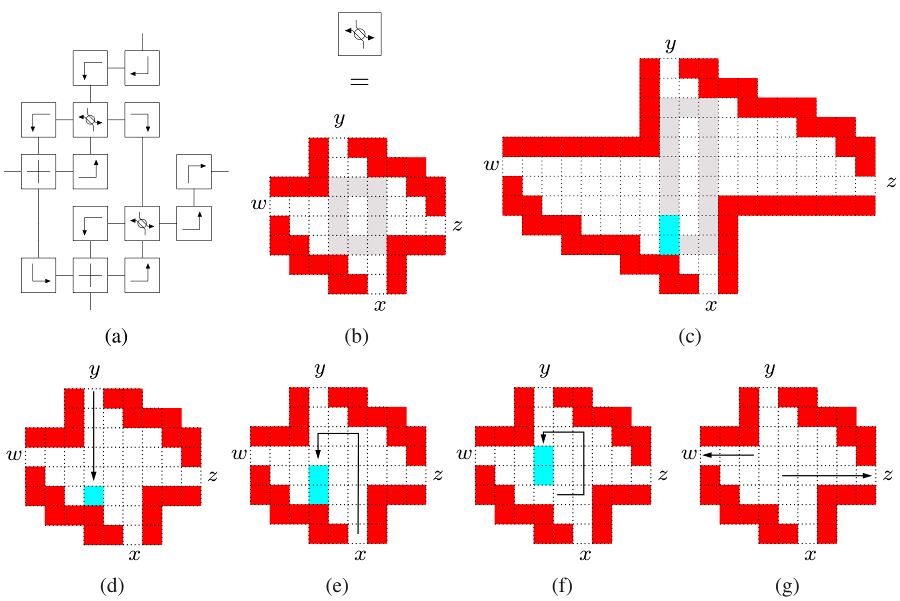


Figure 6: (a) Synchronize gadget, constructed in terms of (b) catalyst gadget, (c) a catalyst gadget for PUSHPUSHPUSH-3, and (d)–(g) the traversal sequence of (b).

No changes are required for the catalyst gadget to work with the PUSHPUSH-k constraint for $k > 1$. However, for PUSHPUSHPUSH-k with $k > 1$, the operations shown in Fig. 6(e–f) do not lead to (f) but back to (e) instead, since the $2 \leq k$ blocks continue to move down. Therefore the gadget has to be "blown up" so that a sequence analogous to Fig. 6 ends up with a sequence of $k+1$ blocks stacked up in the middle. For an arbitrary $k \geq 1$ the catalyst gadget for PUSHPUSHPUSH-k has an interior cycle C of height $2(k+1)$ and width 3, and the sizes of the staircase steps on its boundary decrease gradually from k to 1 when going from x toward w or z toward y, respectively. Moreover, the gadget initially contains $k-1$ movable blocks, stacked up in a corner of C. Fig. 6(c) shows the instance for PUSHPUSHPUSH-3.

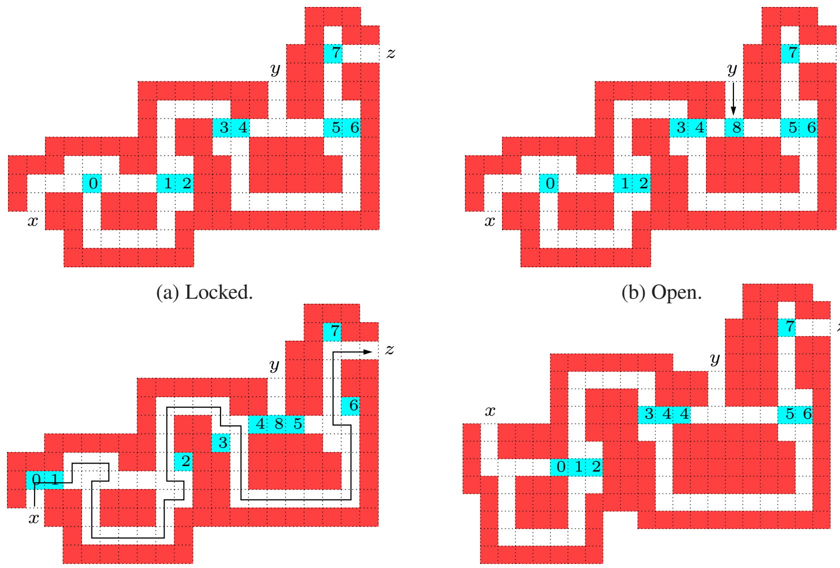
3.3. *Lock Gadget*

Finally, we describe the lock gadget. It prevents the robot from reaching its final destination unless it has enough blocks to unlock the gadget. A single lock gadget is shown in Fig. 7. By cascading several of these gadgets in sequence, we obtain a generalized lock gadget with several key input channels, each requiring a key.

The lock is quite different compared to the gadgets we have seen so far in that it restricts the robot's movement. There are a number of movable blocks which may change position whenever the robot traverses a lock. That is, a lock has an associated state. We distinguish between three different states: a lock as shown in Fig. 7(a) is *closed*, if a key block has been pushed in from y (see Fig. 7(b)), the lock is *open*, and all other possible states are referred to as *broken*. The possible traversals of a lock gadget and the resulting changes in state are summarized by the following lemma.

LEMMA 3 *The robot may pass through a lock gadget from x to z if and only if the lock is open. The robot may pass through a lock gadget from x to y if and only if the lock is closed. After any of these traversals the lock is broken. The robot may change the state of a lock from locked to open by pushing in a block from y. Except for the traversals described above, there is no way for the robot to pass through the gadget between any two distinct ports ($\{x, y, z\}$) in any state.*

Proof: First consider a locked gadget: obviously, it cannot be traversed from either y or z. In fact, Block 0 is the only block that can be pushed at all. If it is pushed down, no further block movement is possible in the gadget. If Block 0 is pushed right, Block 1 can only be pushed up afterward such that it gets stuck in the corner and blocks the connection to y or z forever. Hence, there is only one way to proceed: pushing Block 0 left and thereby blocking port x forever (including the way back). Moreover, the only way to get out of the gadget is to proceed as in Fig. 7(c), except



(a) Locked. (b) Open.

(c) Traversal. (d) Lock for PUSHPUSH-2.

Figure 7: Lock gadget.

that there is no Block 8 such that Block 4 moves all the way until it hits Block 5, thereby forever blocking port z as well.

Now consider an open lock: again, the gadget cannot be traversed from either y or z. As above, the only way to traverse the lock from x is shown in Fig. 7(c). Afterward, both x and y are blocked forever. □

For PUSHPUSH-k with $k > 2$ a slight modification of the lock gadget is required on top of the usual thickening: Block 4 has to be replaced by a sequence of k blocks. Basically, we copy the column to the right of Block 4 and insert another block to the right of Block 4 (k times). In the proof of Lemma 3 there is a minor difference as well, since for $k > 2$ the robot might first push Block 0 to the right without immediately blocking the whole gadget. To avoid introducing additional states, we just place Block 0 and Block 1 next to each other from the very beginning. Fig. 7(d) shows the resulting gadget for PUSHPUSH-2. Finally observe that the lock gadget immediately extends to PUSHPUSHPUSH-k.

4. *Conclusions*

By the hardness construction and the obvious membership in PSPACE, we have

THEOREM 1 PUSHPUSH-k *and* PUSHPUSHPUSH-k *are PSPACE-complete for any* $k \geq 1$.

Finally, we summarize some of the computational complexity results on pushing-block puzzles in Table 1. The main open problem that remains is to resolve the complexity of PUSH-k and PUSH-$*$ between NP and PSPACE.

Problem	Parameter -1	Parameter -k	Parameter -$*$
PUSH	NP-hard	NP-hard	NP-hard
PUSHPUSH	PSPACE-compl.	PSPACE-compl.	NP-hard
PUSH-X	NP-compl.	NP-compl.	NP-compl.
PUSH-F	NP-hard	PSPACE-compl.	PSPACE-compl.

Table 1: Computational complexity of some block sliding puzzles.

Acknowledgments. This work was initiated during Seminar 02081, "Algorithmic Combinatorial Game Theory," Schloß Dagstuhl, Wadern, Germany, February 17–22, 2002.

References

[1] D. Bremner, J. O'Rourke, and T. C. Shermer. Motion planning amidst movable square blocks is PSPACE complete. Unpublished Draft, 1994.

[2] E. D. Demaine, M. L. Demaine, and J. O'Rourke. PushPush and Push-1 are NP-hard in 2D. In *Proc. 12th Canad. Conf. Comput. Geom.*, pp. 211–219, Aug. 2000. http://www.cs.unb.ca/conf/cccg/eProceedings/26.ps.gz.

[3] E. D. Demaine, M. L. Demaine, and J. O'Rourke. PushPush is NP-hard in 2D. Tech. Rep. 065, Dept. Comp. Sci., Smith College, Jan. 2000. http://arXiv.org/abs/cs.CG/0001019.

[4] E. D. Demaine. Playing games with algorithms: Algorithmic combinatorial game theory. In *Proc. 26th Symp. Math. Found. in Comp. Sci.*, LNCS 2136, pp. 18–32, Aug. 2001.

[5] E. D. Demaine and M. Hoffmann. Pushing blocks is NP-complete for noncrossing solution paths. In *Proc. 13th Canad. Conf. Comput. Geom.*, pp. 65–68, Aug. 2001. http://compgeo.math.uwaterloo.ca/~cccg01/proceedings/long/eddemaine-24711.ps.

[6] E. D. Demaine and M. Hoffmann. Push-2-F is PSPACE-Complete. In *Proc. 14th Canad. Conf. Comput. Geom.*, pp. 31–35, Aug. 2002. http://www.cs.uleth.ca/~wismath/cccg/papers/31.ps.

[7] Z. Galil. Hierarchies of complete problems. *Acta Inf.*, 6(1):77–88, 1976.

[8] M. Hoffmann. Push-$*$ is NP-hard. In *Proc. 12th Canad. Conf. Comput. Geom.*, pp. 205–210, Aug. 2000. http://www.cs.unb.ca/conf/cccg/eProceedings/13.ps.gz.

[9] M. Holzer and S. Schwoon. Assembling molecules in ATOMIX is hard. *Theoret. Comput. Sci.*, 313(3):447–462, Feb. 2004.

[10] D. Kozen. Lower bounds for natural proof systems. In *Proc. 18th Symp. Found. Comp. Sci.*, pp. 254–266, 1977.

[11] J. O'Rourke and the Smith Problem Solving Group. PushPush is NP-hard in 3D. Tech. Rep. 064, Dept. Comp. Sci., Smith College, Nov. 1999. http://arXiv.org/abs/cs/9911013.