Computational Complexity of Generalized Push Fight

Jeffrey Bosboom
MIT CSAIL, 32 Vassar Street, Cambridge, MA 02139, USA
jbosboom@csail.mit.edu

Erik D. Demaine
MIT CSAIL, 32 Vassar Street, Cambridge, MA 02139, USA
edemaine@mit.edu

Mikhail Rudoy[1]
MIT CSAIL, 32 Vassar Street, Cambridge, MA 02139, USA
mrudoy@gmail.com

Abstract

We analyze the computational complexity of optimally playing the two-player board game Push Fight, generalized to an arbitrary board and number of pieces. We prove that the game is PSPACE-hard to decide who will win from a given position, even for simple (almost rectangular) hole-free boards. We also analyze the *mate-in-1* problem: can the player win in a single turn? One turn in Push Fight consists of up to two "moves" followed by a mandatory "push". With these rules, or generalizing the number of allowed moves to any constant, we show mate-in-1 can be solved in polynomial time. If, however, the number of moves per turn is part of the input, the problem becomes NP-complete. On the other hand, without any limit on the number of moves per turn, the problem becomes polynomially solvable again.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness

Keywords and phrases board games, hardness, mate-in-one

Digital Object Identifier 10.4230/LIPIcs.FUN.2018.11

Related Version `https://arXiv.org/abs/1803.03708`

Acknowledgements This work grew out of an open problem session originally started during an MIT class on Algorithmic Lower Bounds: Fun with Hardness Proofs (6.890) in Fall 2014.

1 Introduction

Push Fight [10] is a two-player board game, invented by Brett Picotte around 1990, popularized by Penny Arcade in 2012 [9], and briefly published by Penny Arcade in 2015 [8]. Players take turns moving and pushing pieces on a square grid until a piece gets pushed off the board or a player is unable to push on their turn. Figure 1 shows a Push Fight game in progress, and Section 2 details the rules.

In this paper, we study the computational complexity of optimal play in Push Fight, generalized to an arbitrary board and number of pieces, from two perspectives:

[1] Now at Google Inc.

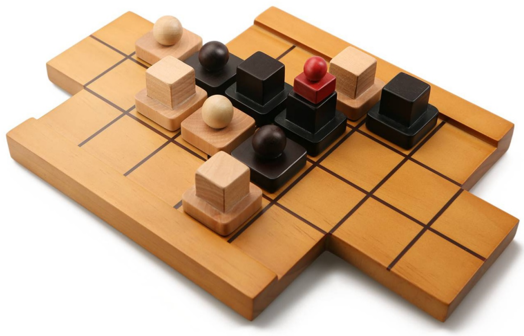


Figure 1 A Push Fight game in progress. Photo by Brettco, Inc., used with permission.

Table 1 Summary of our results.

	Computational complexity of...	
Moves per turn	**Mate-in-1**	**Who wins?**
≤ 2	P	PSPACE-hard, in EXPTIME
$\leq c$ constant	P	open
$\leq k$ input	NP-complete	open
unlimited	P	open

1. **Who wins?** The typical complexity-of-games problem is to determine which player wins from a given game configuration.
2. **Mate-in-1**: Can the current player win *in a single turn*?

Table 1 summarizes our results.

Generalized Push Fight is a two-player game played on a polynomially bounded board for a potentially exponential number of moves, so we conjecture the "who wins?" decision problem to be EXPTIME-complete, as with Checkers [11] and Chess [4]. (Certainly the problem is in EXPTIME, by building the game tree.) In Section 4, we prove that the problem is at least PSPACE-hard, using a proof patterned after the NP-hardness proof of Push-$*$ [7]. Our proof uses a simple, nearly rectangular board, in the spirit of the original game; in particular, the board we use is hole-free and x-monotone (see Figure 8). It remains open whether Push Fight is in PSPACE, EXPTIME-hard, or somewhere in between.

Our mate-in-1 results are perhaps most intriguing, showing a wide variability according to whether and how we generalize the "up to two moves per turn" rule in Push Fight. If we leave the rule as is, or generalize to "up to c moves per turn" where c is a fixed constant (part of the problem definition), then we show that the mate-in-1 problem is in P, i.e., can be solved in polynomial time. However, if we generalize the rule to "up to k moves per turn" where k is part of the input, then we show that the mate-in-1 problem becomes NP-complete. On the other hand, if we remove the limit on the number of moves per turn, then we show that the mate-in-1 problem is in P again. Section 3 proves these results.

The mate-in-1 problem has been studied previously for other board games. The earliest result is that mate-in-1 Checkers is in P, even though a single turn can involve a long sequence of jumps [3]. On the other hand, Phutball is a board game also featuring a sequence of jumps in each turn, yet its mate-in-1 problem is NP-complete [2].

For omitted proofs, see the full version of the paper [1].

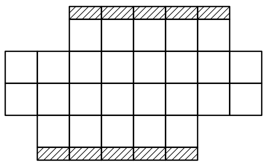


Figure 2 Original Push Fight board. Shaded regions represent side rails.

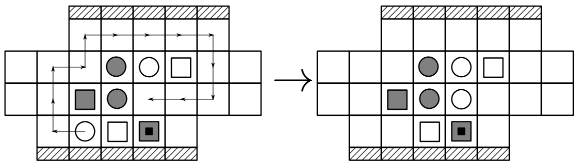


Figure 4 An example move.

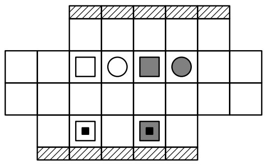


Figure 3 Our notation for pieces, in reading order: a white king, a white pawn, a black king, a black pawn; and white and black anchored kings (in an actual game, there is only one anchor).

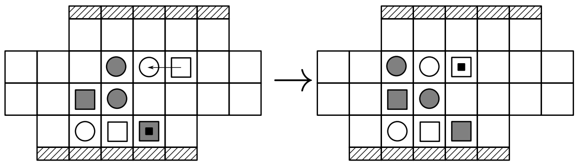


Figure 5 An example push.

2 Rules

The original Push Fight board is an oddly shaped square grid containing 26 squares; see Figure 2. Part of the boundary of this board has *side rails* which prevent pieces from being pushed off across those edges. We generalize Push Fight by considering arbitrary polyomino boards, with each boundary edge possibly having a side rail.

Push Fight is played with two types of pieces, each of which takes up a square of the board: *pawns* (drawn as circles) and *kings* (drawn as squares). Each piece is colored either black or white, denoting which player the piece belongs to. Standard Push Fight is played with three kings and two pawns per player. Additionally, there is a single *anchor* that is placed on top of a king after it pushes (but is never placed directly on the board). Figure 3 shows our notation for the pieces.

Push Fight gameplay consists of the two players alternating *turns*. During a player's turn, the player makes up to two optional *moves* followed by a mandatory *push*.

To make a move, a player moves one of their pieces along a simple path of orthogonally adjacent empty squares; see Figure 4.

To push, a player moves one of their kings into an occupied adjacent square. The piece occupying that square is pushed one square in the same direction, and this continues recursively until a piece is pushed into an unoccupied square or off the board. If this process would push a piece through a side rail, or would push the anchored king, the push cannot be made. Pushes always move at least one other piece. When the push is complete, the pushing king is anchored (the anchor is placed on top of that king). Figure 5 shows a valid push.

A player loses if any of their pieces are pushed off the board (even by their own push) or if they cannot push on their turn.

▶ **Definition 1.** A *Push Fight game state* is a description of the board's shape, including which board edges have side rails, and for each board square, what type of piece or anchor occupies it (if any).

Note that the position of the anchor encodes which player's turn it is: if the anchor is on a white king, it is black's turn, and vice versa. If the anchor has not been placed (no turns have been taken), it is white's turn.

3 Mate-in-1

We consider three variants of mate-in-1 Push Fight, varying in how the number of moves is specified: as a constant in the problem definition, as part of the input, or without a limit.

3.1 c-Move Mate-in-1

▶ **Problem 2.** c-Move Push Fight Mate-in-1: *Given a Push Fight game state, can the player whose turn it is win this turn by making up to c moves and one push?*

The standard Push Fight game has $c = 2$.

▶ **Theorem 3.** c-Move Push Fight Mate-in-1 *is in P.*

Proof Sketch. The number of possible turns is $\leq A^{2c+4}$ on a board of area A. ◀

3.2 k-Move Mate-in-1 is in NP

▶ **Problem 4.** k-Move Push Fight Mate-in-1: *Given a Push Fight game state and a positive integer k, can the player whose turn it is win this turn by making up to k moves and one push?*

In this section, we prove the following upper bound on the number of useful moves in a turn:

▶ **Theorem 5.** *Given a Push Fight game state on a board having n squares, if the current player can win this turn, they can do so using at most n^6 moves followed by a push.*

Proof Sketch. We divide the reachable game states into $\leq n^4$ equivalence classes, and show that two equivalent configurations can be reached via $\leq n^2$ moves within that class. ◀

Our bound directly implies an NP algorithm for k-Move Push Fight Mate-in-1:

▶ **Corollary 6.** k-Move Push Fight Mate-in-1 *is in* NP.

A turn consists of making some number of moves followed by a single push. For the purpose of analyzing a single turn, kings other than the single king that pushes are indistinguishable from pawns, so we can assume the current player first chooses a king, then replaces all of their other kings with pawns before making their moves and push. The following definitions are based on this assumption.

▶ **Definition 7.** Given a single-king game state, a *board configuration* is a placement of pieces reachable by the current player making a sequence of moves.

▶ **Definition 8.** The *pawnspace* of a board configuration is the (possibly disconnected) region of the board consisting of the empty squares and the squares containing the current player's pawns. Equivalently, the pawnspace is the region consisting of all squares not occupied by the current player's king or the other player's pieces.

▶ **Definition 9.** The *signature* of a board configuration is a list of nonnegative integers, where each integer is a count of the current player's pawns in a connected component of the configuration's pawnspace, ordered according to row-major order on the leftmost topmost square in the corresponding connected component.

▶ **Definition 10.** Given two board configurations C_1 and C_2 derived from the same game state, we say that $C_1 \equiv C_2$ if and only if

1. C_1 and C_2 have the same pawnspace (that is, the current player's only king occupies the same square in C_1 and C_2) and
2. C_1 and C_2 have the same signature (that is, each connected component of the pawnspace contains the same number of the current player's pawns in C_1 and C_2).

Relation $\equiv$ is clearly reflexive, symmetric, and transitive, so it is an equivalence relation inducing a partition of the set of board configurations derived from a given game state into equivalence classes. We need the following two lemmas about $\equiv$ for our proof of Theorem 5:

▶ **Lemma 11.** *For a given game state on a board with n squares, there are at most n^4 equivalence classes of board configurations.*

▶ **Lemma 12.** *If $C_1 \equiv C_2$, then C_2 can be reached from C_1 in at most $n^2 - 1$ moves without leaving the equivalence class of C_1.*

We are now ready to prove Theorem 5:

▶ **Theorem 5.** *Given a Push Fight game state on a board having n squares, if the current player can win this turn, they can do so using at most n^6 moves followed by a push.*

Proof. By our assumption that the current player can win this turn, there exists a sequence of moves for the current player after which they can immediately win with a push, corresponding to a sequence of board configurations $C_1, C_2, \ldots, C_l$. Configuration C_1 is obtained from the initial game state by replacing all of the current player's kings, except the one that ends up pushing, with pawns. Each C_{i+1} can be reached from C_i in one move, and C_l is a configuration from which the current player can win with a push.

We now define *simplifying* a sequence of board configurations over an equivalence class E. If the sequence contains no configurations from E, then simplifying the sequence over E leaves it unchanged. Otherwise, let A_i be the first configuration in the sequence in E and A_j be the last configuration in the sequence in E. By Lemma 12, there exists a sequence of fewer than $n^2 - 1$ moves that transforms A_i into A_j, corresponding to a sequence of board configurations $A_i = D_0, D_1, \ldots, D_u = A_j$ with $u \le n^2 - 1$. Then simplifying over E consists of replacing all configurations between and including A_i and A_j with the replacement sequence $D_0, D_1, \ldots, D_u$.

Notice that simplifying a sequence (over any class) never changes the first or last configuration in the sequence, and each configuration in the resulting sequence remains reachable in one move from the previous configuration in the resulting sequence. After simplifying over a class E, the only configurations in the resulting sequence in E are those in the replacement sequence, so the number of configurations in the sequence in E is at most n^2. Furthermore, all configurations in the replacement sequence are in E, so simplifying over E never increases (but may decrease) the number of configurations falling in other classes.

Let $C'_1, C'_2, \ldots, C'_l$ be the result of simplifying $C_1, C_2, \ldots, C_l$ over every equivalence class. By Lemma 11, there are at most n^4 such classes, and by the above paragraph there are at most n^2 configurations from each class in $C'_1, C'_2, \ldots, C'_l$, so the length of $C'_1, C'_2, \ldots, C'_l$ is at most n^6. Each configuration in $C'_1, C'_2, \ldots, C'_l$ is reachable in one move from the previous configuration, and that sequence of at most n^6 moves leaves the current player in position to win with a push, as desired. ◀

3.3 Unbounded-Move Mate-in-1

▶ **Problem 13.** Unbounded-Move Push Fight Mate-in-1: *Given a Push Fight game state, can the player whose turn it is win this turn by making any number of moves and one push?*

▶ **Theorem 14.** Unbounded-Move Push Fight Mate-in-1 *is in P.*

We can of course solve Unbounded-Move Push Fight Mate-in-1 by trying all possible sequences of moves to find a board configuration from which the current player can win with a push, but there are exponentially many board configurations, so such an algorithm takes exponential time. Instead, we can use the fact that any two configurations in the same equivalence class are reachable from each other in a polynomial number of moves (from Lemma 12) to search over equivalence classes of board configurations instead of searching over board configurations. There are at most n^4 equivalence classes (by Lemma 11), so they can be searched in polynomial time.

We will make use of the following definitions:

▶ **Definition 15.** Two equivalence classes of board configurations C_1 and C_2 are *neighbors* if there exist board configurations $b_1 \in C_1$ and $b_2 \in C_2$ such that b_1 can be reached from b_2 with a king move of exactly one square. The *equivalence class graph* is a graph whose vertices are equivalence classes of board configurations and whose edges connect neighboring equivalence classes.

An equivalence class of board configurations C is a *winning equivalence class* if there exists a board configuration $b \in C$ such that the player whose turn it is can win with a push.

The key idea for our algorithm is the following:

▶ **Lemma 16.** *There exists a path in the equivalence class graph from the equivalence class of the initial board configuration to a winning equivalence class if and only if there exists a winning move sequence.*

The size of the equivalence class graph is polynomial in n (by Lemma 11), so provided the graph can be constructed and the winning equivalence classes identified, this type of path in the equivalence class graph, if it exists, can be found in polynomial time.

Recall from Definition 10 that equivalence classes of board configurations are defined by the pawnspace and signature, and that, for configurations derived from the same game state (i.e., having the other player's pieces in the same positions), the pawnspace is defined by the position of the current player's king. Thus we can uniquely name a class using the king position and signature.

▶ **Definition 17.** The *class descriptor* of an equivalence class of board configurations for a given game state is the ordered pair of the position of the current player's king and the signature defining that class.

To prove Theorem 14, we need to give polynomial-time algorithms to compute the neighbors of an equivalence class and to decide whether a class is a winning equivalence class.

▶ **Lemma 18.** *Given an initial game state and a class descriptor for some class C, we can compute in polynomial time the equivalence classes (as class descriptors) neighboring C.*

▶ **Lemma 19.** *Given an initial game state and a class descriptor for some class C, we can decide in polynomial time whether C is a winning equivalence class.*

We are now ready to prove Theorem 14:

▶ **Theorem 14.** Unbounded-Move Push Fight Mate-in-1 *is in P.*

Proof. First, compute the class descriptor for the equivalence class of the initial board configuration. Then perform a breadth- or depth-first search of the equivalence class graph, using the algorithm given in the proof of Lemma 18 to compute the neighboring class descriptors and the algorithm given in the proof of Lemma 19 to decide if the search has found a winning equivalence class. Each of these procedures takes polynomial time. By Lemma 11, there are only polynomially many equivalence classes, so the search terminates in polynomial time. By Lemma 16, there exists a winning move sequence if and only if this search finds a path to a winning equivalence class. ◀

The key idea of the above proof is that, if we do not care how many moves we make inside an equivalence class, then it is sufficient to search the graph of equivalence classes. Thus the above proof does not apply to k-Move Push Fight Mate-in-1, and in the next section, we prove k-Move Push Fight Mate-in-1 is NP-hard.

3.4 *k*-Move Mate-in-1 is NP-hard

To prove k-Move Push Fight Mate-in-1 hard, we reduce from the following problem, proved strongly NP-hard in [5]:

▶ **Problem 20.** Integer Rectilinear Steiner Tree: *Given a set of points in $\mathbb{R}^2$ having integer coordinates and a length ℓ, is there a tree of horizontal and vertical line segments of total length at most ℓ containing all of the points?*

▶ **Theorem 21.** k-Move Push Fight Mate-in-1 *is strongly* NP*-hard.*

Proof Sketch. The basic idea of our reduction is to create a game state mostly full of the current player's pawns, but with a few empty squares (*holes*). The player must "move" the holes (by moving pawns into them, creating a new hole at the pawn's former square) to free a king that can push one of the other player's pieces off the board. Initially each pawn can only travel one square (into an adjacent hole) per move, but once two holes have been brought together, a pawn can travel two squares per move, and so on. Bringing the holes together optimally amounts to finding a Steiner tree covering the holes' initial positions.

Reduction: Suppose we are given an instance of Integer Rectilinear Steiner Tree consisting of points $p_i = (x_i, y_i)$ with $i = 1, \dots, n$ and length ℓ. For convenience, and without affecting the answer, we first translate the points so that $\min x_i = 2$ and $\min y_i = 4$ and reorder the points such that $y_1 = 4$.

We then build a Push Fight game state with a rectangular board with a height of $\max y_i$ and a width of $n + \max x_i$, indexed using 1-based coordinates with the origin in the bottom-left square; refer to Figure 6. The entire boundary of the board has side rails except the edge adjacent to square $(x_1, 1)$. There is a white king in square $(x_1 + n, 2)$ and a black king with the anchor in square $(x_1 - 1, 2)$. There is a black pawn in square (x, y) if any of the following are true:

1. $y = 3$ and $x \neq x_1$,
2. $y = 2$ and either $x < x_1 - 1$ or $x > x_1 + n$, or
3. $y = 1$.

The squares (x_i, y_i) with $1 \leq i \leq n$ (corresponding to the points in the Integer Rectilinear Steiner Tree instance) are empty. All remaining squares are filled with white pawns. The output of the reduction is this Push Fight board together with $k = \ell + 3$. ◀

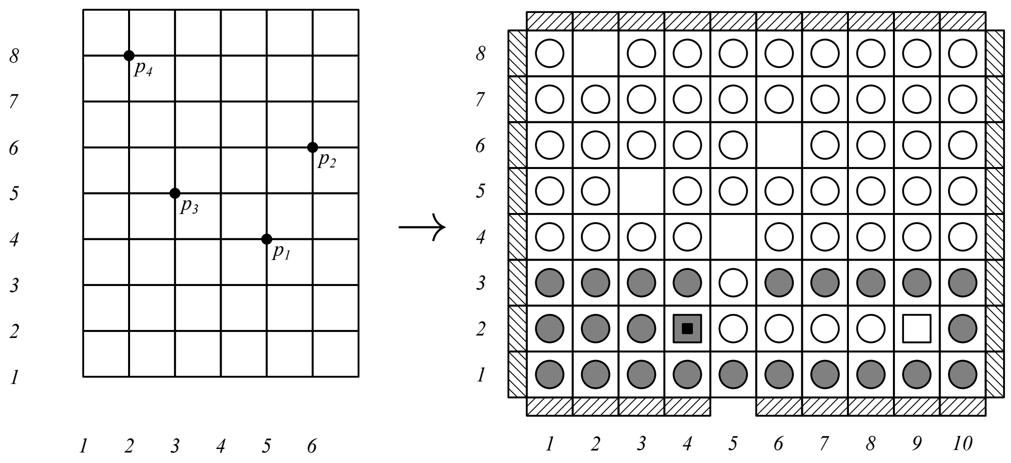


Figure 6 A Push Fight board (right) produced during the reduction from the points in an example rectilinear Steiner tree instance (left).

4 Push Fight is PSPACE-hard

In this section, we analyze the problem of deciding the winner of a Push Fight game in progress.

▶ **Problem 22.** PUSH FIGHT: *Given a Push Fight game state, does the current player have a winning strategy (where players make up to two moves per turn)?*

▶ **Theorem 23.** PUSH FIGHT *is* PSPACE*-hard.*

To prove PSPACE-hardness, we reduce from Q3SAT, proved PSPACE-complete in [12, 6]:

▶ **Problem 24.** Q3SAT: *Given a fully quantified boolean formula in conjunctive normal form with at most three literals per clause, is the formula true?*

Our proof parallels the NP-hardness proof of PUSH-$*$ in [7]. PUSH-$*$ is a motion-planning problem in which a robot (agent) traverses a rectangular grid, some squares of which contain blocks. The robot can push any number of consecutive blocks when moving into a square containing a block, provided no blocks would be pushed over the boundary of the board. The PUSH-$*$ decision problem asks, given a initial placement of blocks and a target location, can the robot reach the target location by some sequence of moves? In our proof, the white king takes the place of the PUSH-$*$ robot[2] and white pawns function as blocks. Our proof has the additional complication that Black sets the universally quantified variables, and that White's moves and Black's push must be forced at all times to keep the other gadgets intact.

Figure 7 shows an overview of the reduction. The sole white king begins at the bottom-left of the *variable gadget I* block, setting existentially quantified variables as it pushes up and right. The *variable gadget II* block contains black pawns and holes that allow Black to set the universally quantified variables. After all the variables have been set, the white king traverses the *bridge* to the *clause gadget* block. The variable and clause gadgets interact via a pattern of holes in the *connection block* encoding the literals in each clause. The white

[2] The PUSH-$*$ robot can move without pushing blocks, so the correspondence is not exact.

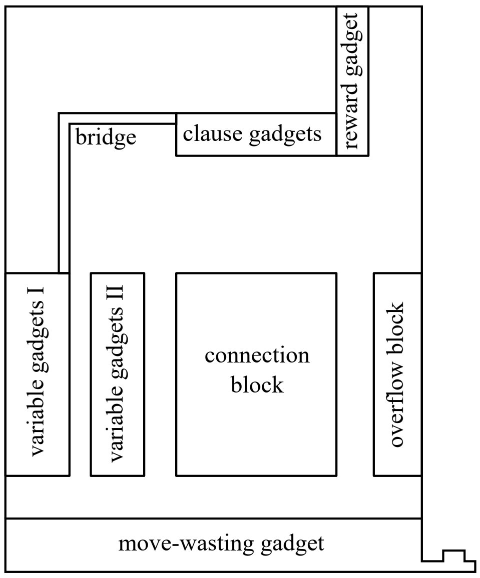


Figure 7 An overview of the Push Fight board produced by our reduction.

king can traverse the clause gadgets only if the variable gadgets were traversed in a way corresponding to a satisfying assignment of the variables. The *reward gadget* contains a boundary square without a side rail, such that the white king can push a black pawn off the board if the white king reaches the reward gadget. The *overflow block* contains empty squares needed by the variable gadgets that were not used in the connection block (for variables appearing in few clauses). The *move-wasting gadget* forces White's moves and Black's push, ensuring the integrity of the other gadgets. Finally, all other squares on the board are filled with white pawns, and the boundary has side rails except at specific locations in the reward and move-wasting gadgets. Figure 8 shows an example output of the reduction.

We first prove the behavior of each of the gadgets, then describe how the gadgets are assembled.

4.1 Move-wasting gadget

The move-wasting gadget requires White to use both moves to prevent Black from winning on the next turn (unless White can win in the current turn). The move-wasting gadget contains the only black king, thus consuming (and allowing) Black's push each turn. When analyzing the other gadgets, we can thus assume White can only push and Black can only move. The move-wasting gadget comprises the entire bottom three rows of the board, but pieces only move in the far-right portion. Figure 9a shows the initial state of the gadget. Throughout this analysis, we assume White cannot win in one turn; Section 4.5, which analyzes the reward gadget, describes the position in which White can immediately win in one turn, and can therefore disregard the threat from Black in the move-wasting gadget.

In the initial state, the anchor is on the black king, so it is White's turn. White must move the pawn above the black king to avoid losing next turn. There are only two reachable empty squares, both in the column left of the black king. If the other square in that column remains empty, Black can move the black king into it and push the white pawn in that column off the board. Thus White must fill the other square in that column, and the only way to do so is to move the pawn two columns left of the white king one square right. Figure 9b shows the resulting position (after White pushes elsewhere in the board).

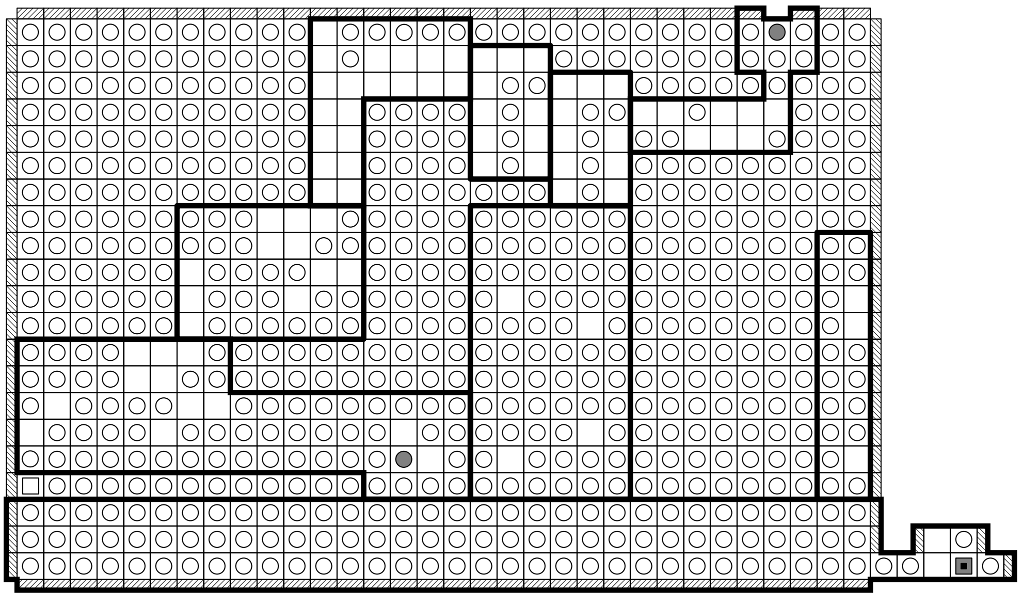
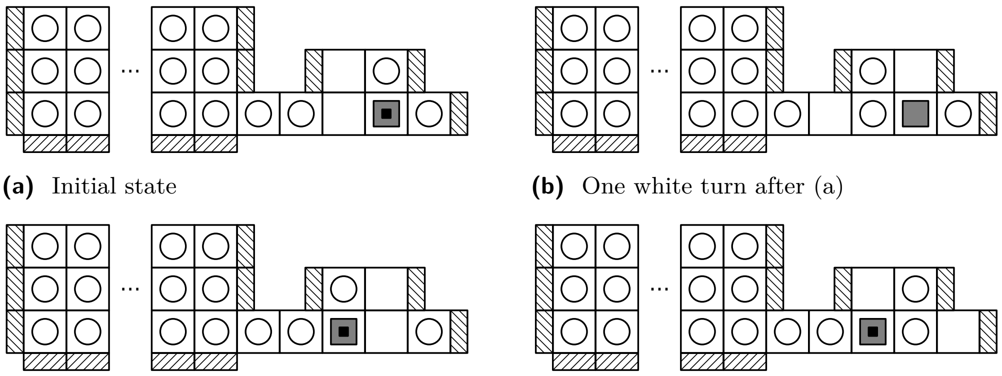


Figure 8 The result of performing the reduction on the formula $\forall x \exists y\ (x \vee \neg y) \wedge (\neg x \vee y)$. Gadgets and blocks are outlined.



(a) Initial state

(b) One white turn after (a)

(c) One black turn after (b)

(d) One white turn after (c)

Figure 9 The move-wasting gadget.

Black's only legal push is to the left, resulting in the position shown in Figure 9c.

The rightmost four columns in Figure 9c are simply the reflection of those columns in Figure 9a, so by the same argument White must fill the column to the right of the black king, resulting in Figure 9d.

Again, the rightmost four columns of Figures 9d and 9b are reflections of each other. Black's only legal push is to the right, restoring the gadget to the initial state shown in Figure 9a. Thus until White can win in one turn, White must use both moves in the move-wasting gadget, and at all times Black must (and can) push in the move-wasting gadget. In the analysis of the remaining gadgets, if the white king reaches a position from which it cannot push, we conclude that White immediately loses, because if White moves a pawn or the king into position to push, Black can win on the next turn as explained above.

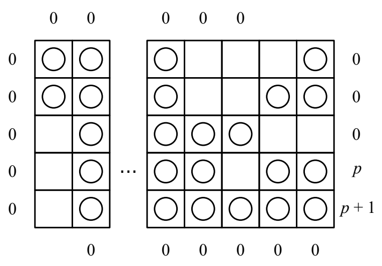


Figure 10 Existential variable gadget.

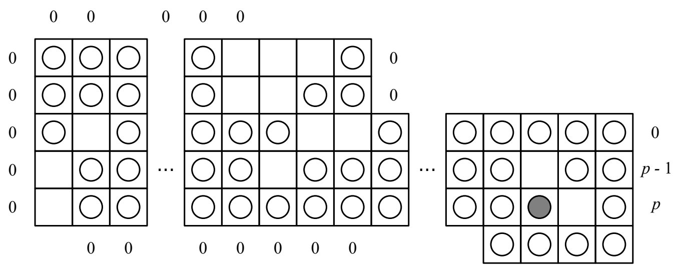


Figure 11 Universal variable gadget.

4.2 Variable gadgets

The existential variable gadget forces White to fill all empty squares in one row of the connection block, corresponding to setting the value of that variable. The universal variable gadget allows Black to choose the value of the corresponding variable, then forces White to similarly fill a row of empty squares. We first analyze a core gadget; the existential variable gadget is a minor variant of the core gadget and its correctness follows directly, while the universal variable gadget has an additional component to allow Black to choose the variable's value. Throughout our analysis, we take advantage of the board being filled with white pawns to limit the number of pieces that can leave the gadget.

The core gadget occupies a rectangle of width $p + 5$ and height 5. When instantiated in the reduction, the gadget lies entirely within the *variable gadget I* block. Integer p is one more than the maximum number of occurrences of a literal in the input formula. The initial state of the core gadget is shown in Figure 12. Each number along the boundary of the figure gives the number of empty squares outside the gadget in that direction, and thus an upper bound on the number of pieces that can leave the gadget via that edge.

The following lemma summarizes the constraints we prove about the core gadget.

▶ **Lemma 25.** *Starting from the position in Figure 12, and assuming the white king does not push down or left from this position,*

(i) *the white king leaves in the second-rightmost column, and*

(ii) *when the white king leaves either*

  (a) *the gadget is as shown in Figure 13 and $p + 1$ white pawns have been pushed out along the bottom row of the gadget, or*

  (b) *the gadget is as shown in Figure 14 and p white pawns have been pushed out along the second-to-bottom row of the gadget,*

(iii) *and no other pieces have left the gadget.*

We will construct the existential and universal variable gadgets from the core gadget such that the assumption holds. Lemma i ensures we can chain variable gadgets together

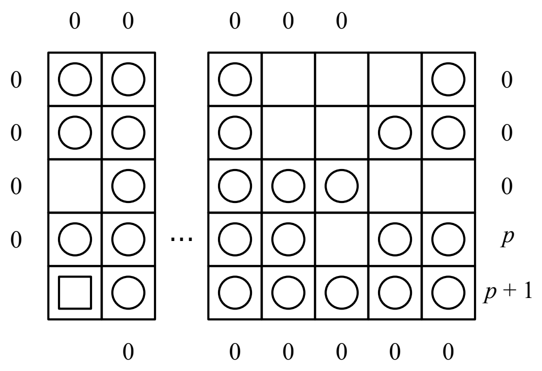


Figure 12 The initial configuration of the core gadget together with upper bounds on the number of pushes out of the gadget at each boundary edge. Omitted columns do not have a given upper bound.

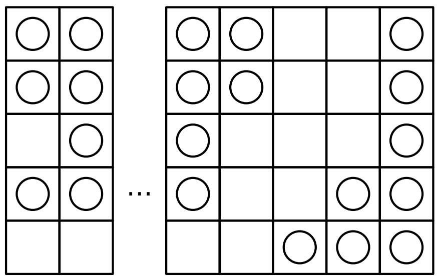


Figure 13 The final configuration of the core gadget after setting the variable to true.

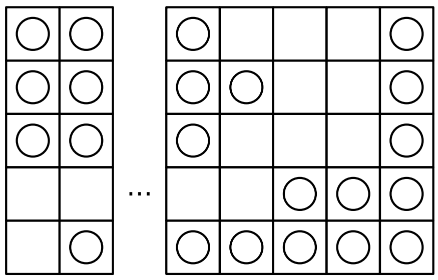


Figure 14 The final configuration of the core gadget after setting the variable to false.

in sequence without the white king escaping. The outcomes implied by Lemma iia and iib correspond to setting the variable to true or false (respectively) by filling in the empty squares in the connection block that could be used to satisfy a clause gadget for a clause containing the opposite literal; that is, pushing pawns out along the bottom row of a gadget prevents all negative literals from being used to satisfy a clause, and similarly for the second-to-bottom row and positive literals.

Proof. We proceed by case analysis starting from Figure 12. The move-wasting gadget consumes White's moves, and there are no black pieces in the core gadget, so we need only analyze the sequence of White's pushes.

Suppose the white king first pushes right. Because of the upper bounds along the top and bottom edges of the gadget, the only legal push in the resulting configuration is to the right, and this remains the case until the white king reaches the fourth column from the right of the gadget. At this point $p+1$ pawns have been pushed off the right edge along the bottom row of the gadget, so there are no empty squares remaining in that row, so pushing right is no longer possible and the only legal push is up. Then the only legal push is again up because of the constraints on the left edge of the gadget. Figure 15 shows the result of this sequence of pushes.

If the white king pushes left from this position, the only possible next push is down, after which there are no legal pushes, resulting in a loss for White. Figure 16 shows this sequence of pushes.

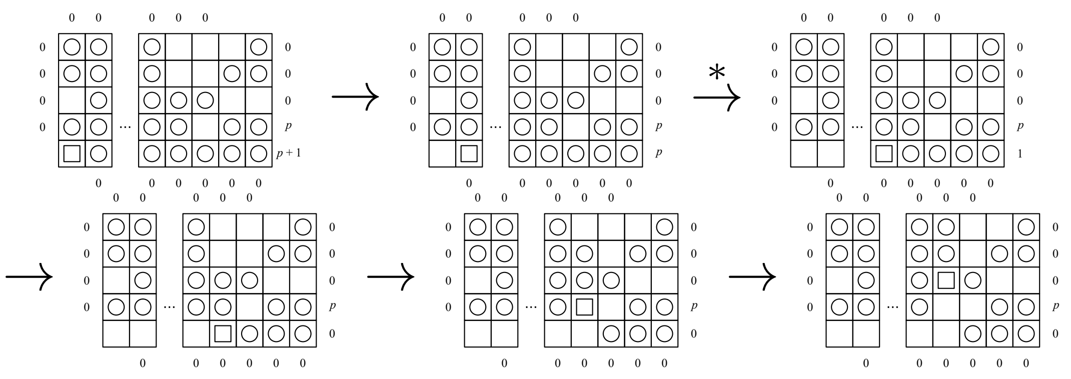


Figure 15 One possible push sequence starting from the initial state of the core gadget. The starred arrow elides a series of pushes to the right.

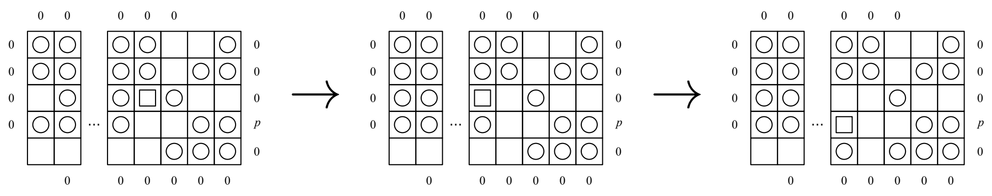


Figure 16 The result of pushing left and down from the last position in Figure 16. White has no legal pushes in the final position.

The only other legal push from the last position in Figure 15 is to the right, after which pushes right, up, up and up again are the only legal pushes. This sequence results in the white king, preceded by a white pawn, exiting the top of the gadget in the second-rightmost column, as desired by Lemma i. Figure 17 shows the positions resulting from this sequence. The final position reached is the position in Figure 13, $p + 1$ pawns were pushed out of the gadget to the right along the bottom row, as desired by Lemma iia, and and no other pieces were pushed out of the gadget, as desired by Lemma iii.

Now suppose that the white king pushes up from the initial configuration. Because of the constraints on the gadget boundary, the only legal push is to the right until the white king reaches the fourth column from the right of the gadget. At this point p pawns have been pushed off the right edge along the second-to-bottom row of the gadget, so there are no empty squares remaining in that row, so pushing right is no longer possible and the only legal push is up. Then the only legal push is again up because of the constraints on the left edge of the gadget. Figure 18 shows the result of this sequence of pushes.

If the white king pushes up from this position, there are no legal pushes in the resulting position, resulting in a loss for White. Figure 19 shows this push and the resulting losing position.

The only other legal push from the last position in Figure 18 is to the right, after which pushes right, up, up and up again are the only legal pushes. This sequence results in the white king, preceded by a white pawn, exiting the top of the gadget in the second-rightmost column, as desired by Lemma i. Figure 20 shows the positions resulting from this sequence. The final position reached is the position in Figure 14, and p pawns were pushed out of the gadget to the right along the second-to-bottom row, as desired by Lemma iib. No other pieces were pushed out of the gadget, as desired by Lemma iii.

This completes the case analysis. ◀

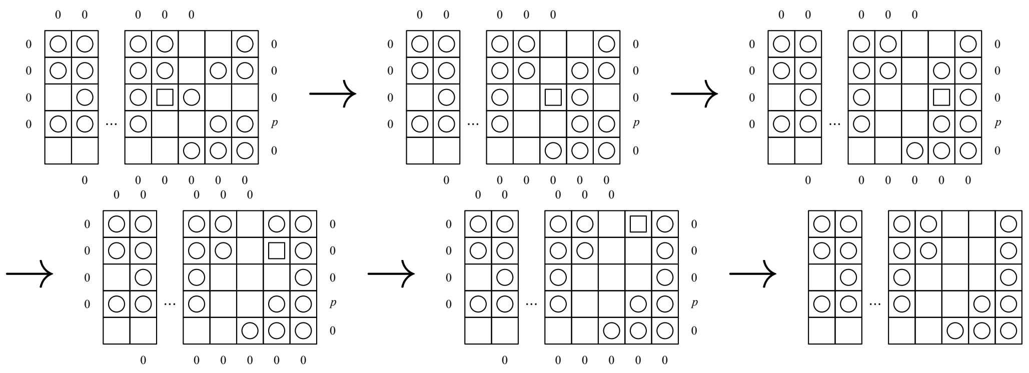


Figure 17 The result of pushing right from the last position in Figure 15, reaching the position in Figure 13.

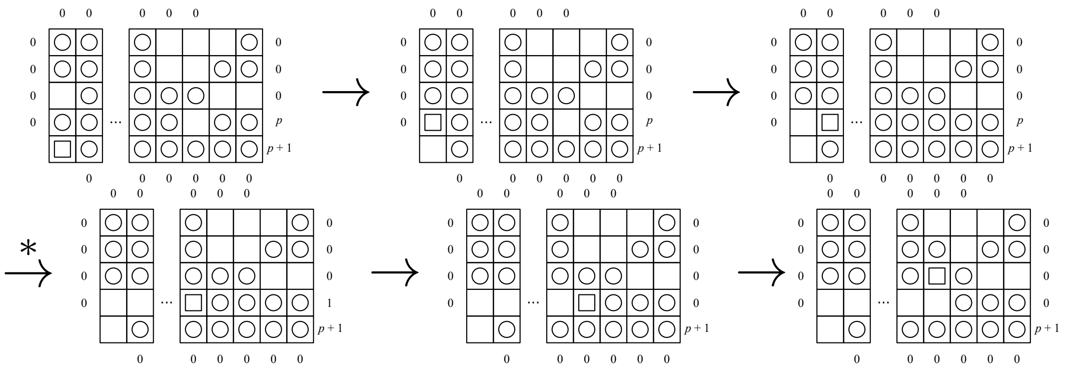


Figure 18 The other possible push sequence starting from the initial state of the core gadget. The starred arrow elides a series of pushes to the right.

Existential variable gadget:

The existential variable gadget, shown in Figure 10, is nearly the same as the core gadget, differing only in the bottom of the leftmost column. When instantiated in the reduction, the white king enters the gadget by pushing a white pawn up into the leftmost column, becoming exactly the core gadget. From the position immediately after the white king enters the gadget, the white king cannot push left (because there are no empty spaces in the row to the left) nor down (because it just pushed up, leaving an empty space in its former position), satisfying the assumption in Lemma 25. Thus by Lemma i, the white king leaves the existential variable gadget in the second-rightmost column with a white pawn above it, and by either Lemma iia or iib, all empty squares in one of two rows of the connection block are now filled by pawns pushed out of the existential variable gadget.

Universal variable gadget:

The universal variable gadget consists of two disconnected regions. The left subregion of the gadget occupies a $(p+6) \times 5$ rectangle in the *variable gadget I* block. As the white king proceeds through the left region of the gadget, a subregion of the gadget reaches the initial state of the core gadget. The right region of the gadget occupies a 4×4 rectangle in the *variable gadget II* block and contains a black pawn to allow Black to control the value of the variable. The bottom of the right region is one row lower than the bottom of the left

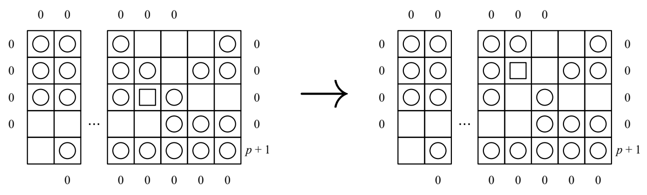


Figure 19 The result of pushing up from the last position in Figure 18. White has no legal pushes in the final position.

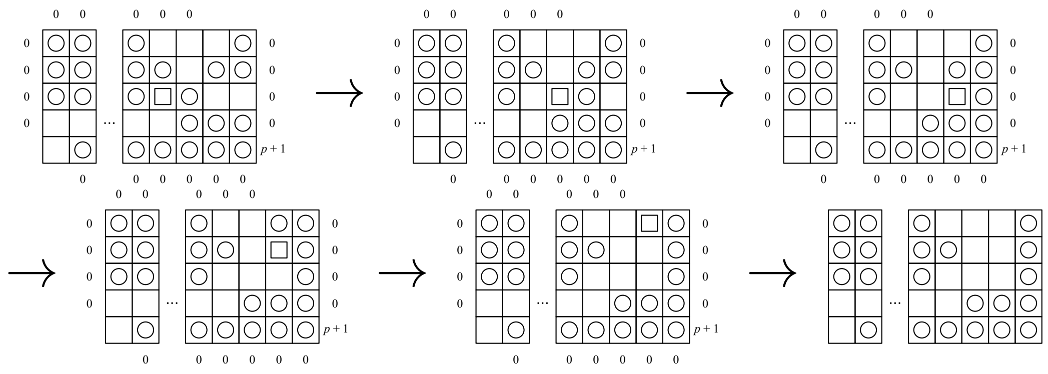


Figure 20 The result of pushing right from the last position in Figure 18, reaching the position in Figure 14.

region. The area between the two regions of the gadget (in the three rows shared by both) is entirely filled by white pawns. Figure 11 shows the universal variable gadget, including the pawn-filled area between the regions.

As with the existential variable gadget, when instantiated in the reduction, the white king enters the universal variable gadget by pushing a white pawn up into the leftmost column. Figure 21 shows the resulting position. Regardless of Black's move, White's only legal push is to the right. By moving the black pawn, Black can choose between the two positions in Figure 22, depending on which of the two rows the black pawn is in when White pushes.

In both of the resulting positions, the black pawn is surrounded, so Black can no longer influence events in this gadget. The left region of the gadget, without the leftmost column, is identical to the initial position of the core gadget. In both positions, the white king cannot push left (empty space) or down (no empty spaces down in the column), satisfying the assumption in Lemma 25. Thus either Lemma iia or Lemma iib holds. Because of the edge constraints, in Figure 22a, only Lemma iia is possible, resulting in Figure 23a. Similarly, in Figure 22b, only Lemma iib is possible, resulting in Figure 23b. By moving the black pawn to select one of these two cases, Black sets the value of the corresponding variable. Then by Lemma i, the white king leaves in the second-rightmost column of the left region (in the *variable gadget I* block) of the gadget. In both cases, the black pawn remains surrounded by white pawns in the right region of the gadget.

4.3 Bridge gadget

The bridge gadget, shown in Figure 24, brings the white king from the exit of the last variable gadget to the entrance of the first clause gadget. When instantiated in the reduction, the white king enters the bridge gadget from the bottom of the leftmost column, preceded by a

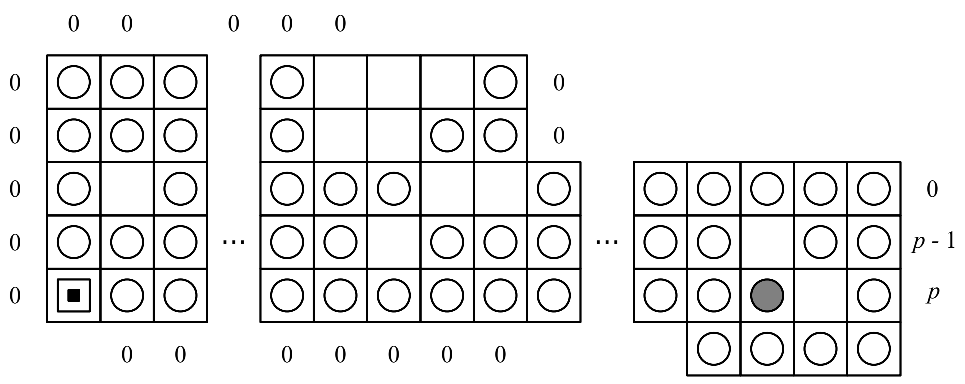


Figure 21 The universal variable gadget after the white king enters.

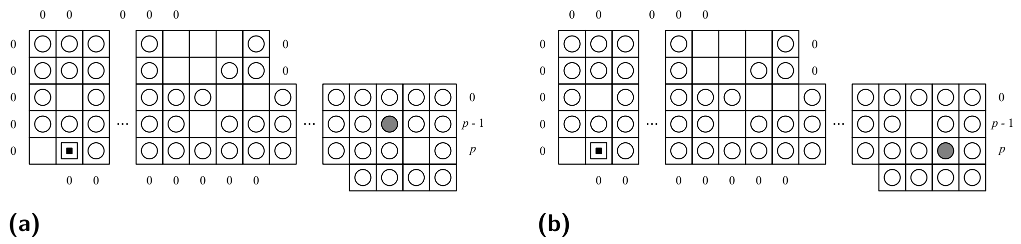


Figure 22 The two possible configurations of the universal variable gadget one white turn after the configuration from Figure 21.

white pawn. The white king's traversal of the bridge gadget is entirely forced. The white king leaves the gadget by pushing a white pawn out to the right in the second-to-top row.

4.4 Clause gadget

The clause gadget, shown in Figure 25, verifies that a column below the gadget contains at least one empty square. When instantiated in the reduction, the white king enters the gadget from the left in the top row, preceded by a white pawn. The resulting sequence of forced pushes includes a push down in the central column of the gadget; if there are no empty squares below the gadget in that column, the white king has no legal pushes and White loses. If there are more empty squares, White can continue to push down, but (when instantiated in the reduction) there are at most three total empty squares in that column, and once those squares are filled, White cannot push. Thus the white king must push right instead and leave the gadget by pushing a white pawn out to the right in the second-to-top row.

4.5 Reward gadget

The reward gadget, shown in Figure 26, allows White to win if the white king reaches the gadget. The black pawn in this gadget cannot move because it is surrounded. When instantiated in the reduction, the white king enters the gadget from the left in the top row, preceded by a white pawn. After pushing right until the white king is in the third column of Figure 26, White can win by moving a white pawn and the white king, then pushing upwards to push the black pawn off the board, as shown in Figure 27. (Recall that the move-wasting gadget no longer binds White once White can win in one turn; Black loses before Black can win using the move-wasting gadget.)

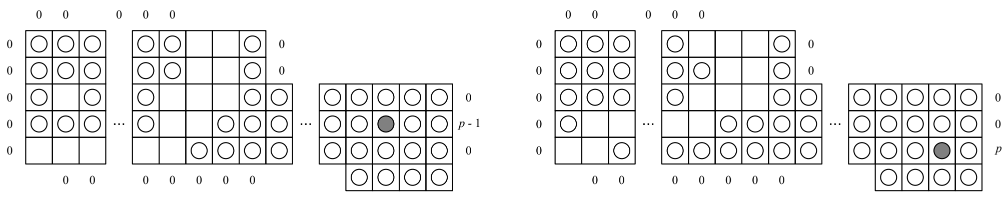

(a) (b)

Figure 23 The two possible final positions of the universal variable gadget after the white king exits.

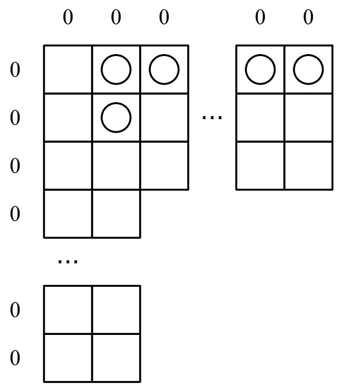


Figure 24 The bridge gadget.

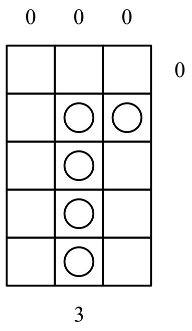


Figure 25 The clause gadget.

4.6 Layout

Having described the gadgets, it remains to show how to instantiate them in a Push Fight game state for a given quantified 3-CNF formula. We first place gadgets with respect to each other, remembering which squares should be left empty, then define the board as the bounding box of the gadgets and fill any squares not recorded as empty with white pawns. The resulting board is mostly rectangular with side rails on all boundary edges, with two exceptions: one edge along the top of the rectangle lacks a side rail as part of the reward gadget, and the board is extended in the bottom-right to accomodate the move-wasting gadget along the bottom of the board.

We begin by building the *variable gadget I* block containing the existential variable gadgets and the left portion of the universal variable gadgets. Gadgets are stacked from bottom to top in the order of the quantifiers in the input formula (using the gadget corresponding to the quantifier), with the leftmost column of each gadget aligned with the second-to-right column of the previous gadget. (Recall that the width of the variable gadgets is defined based on p, one more than the maximum number of occurrences of a literal in the input formula.) This alignment allows (and requires) the white king to traverse the gadgets in sequence as specified by Lemma 25. Figure 29 shows the relative layout of these variable gadgets.

We place the white king one square below the first variable gadget aligned with its leftmost column, and place a white pawn one square above the white king. The white king will push upwards into the first gadget on White's first turn. (If the king was instead placed directly in the variable gadget, if the first variable is universally quantified, Black would not have a move with which to choose the value of the variable before White commits it.)

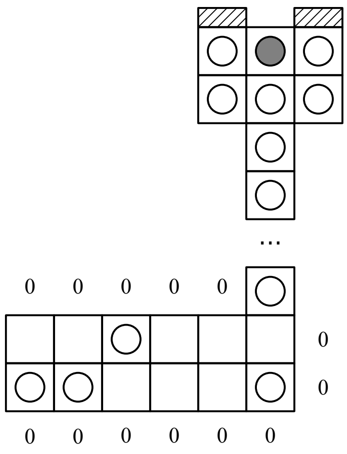


Figure 26 The reward gadget.

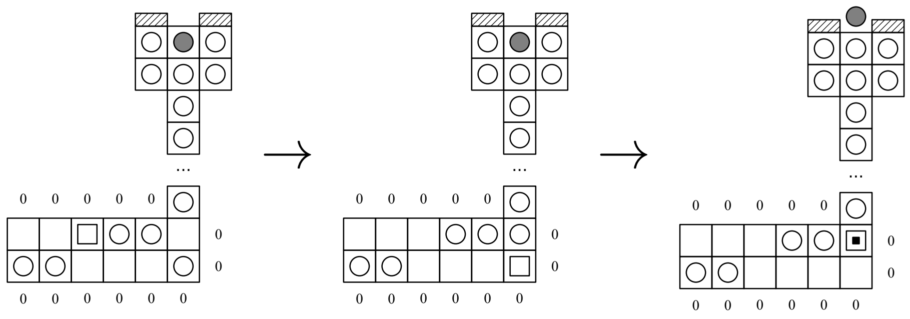


Figure 27 Once the White king reaches the third column of the reward gadget, White can win in a single turn.

We then build the *variable gadget II* block by placing the right regions of the universal variable gadgets to the right of the corresponding left regions in a single column (further right than any part of the variable gadget I section).

Next we place one clause gadget for each clause in the input formula. Each clause gadget is directly to the right of and one square lower than the previous clause gadget. The entire clause gadget block is further right of and above the *variable gadget II* block. Figure 30 shows the relative layout of the clause gadgets. Then we place a bridge gadget such that the entrance of the bridge gadget aligns with the exit of the last variable gadget and the exit of the bridge gadget aligns with the entrance of the first clause.

We place the reward gadget so that its entrance aligns with the exit of the last clause gadget.

We leave empty squares in the connection block to encode the literals in each clause in the input formula. When traversing each variable gadget, the white king pushes pawns to the right in one of two rows. The lower (upper) row corresponds to setting the variable to true (false), or equivalently, preventing negative (positive) literals from satisfying clauses. Associate each row with the literal it prevents from satisfying clauses. Each clause gadget enforces that at least one empty square remains below its middle column, corresponding to at least one of its literals not having been ruled out by the truth assignment. To realize this relation, for each literal in a clause, we leave an empty square at the intersection of the

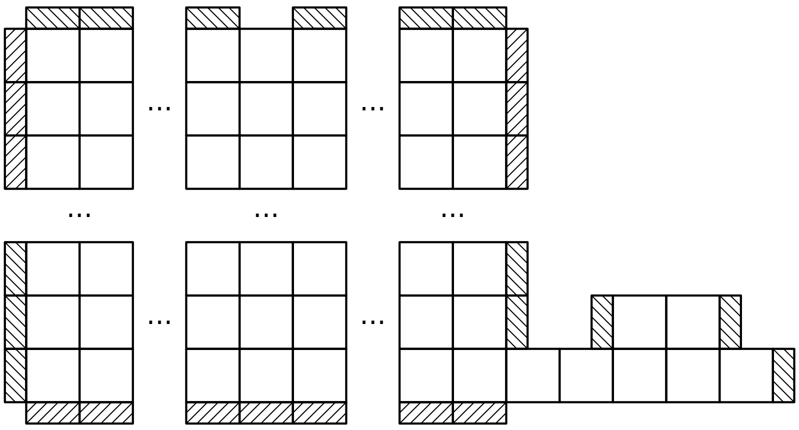


Figure 28 The shape of the Push Fight board produced by the reduction.

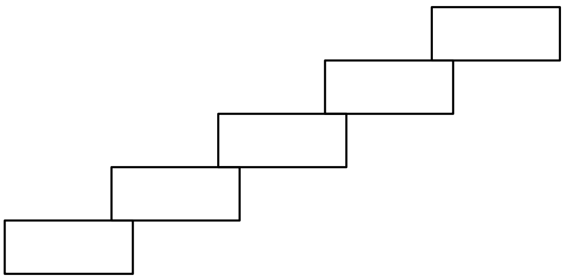


Figure 29 The layout of variable gadgets in the *variable gadget I* block.

column checked by the clause gadget and the row associated with that literal. All other squares in the connection block are filled with white pawns (as are all squares in the board whose contents are not otherwise specified).

The variable gadgets require each row associated with a literal to contain exactly $p-1$, p or $p+1$ empty squares (depending on the type of gadget and whether the row is the upper or lower row). This is at least the number of occurrences of that literal (by the definition of p), but it may be greater. We place any remaining empty squares in each row in columns further right than the reward gadget, forming the overflow block.

The boundary of the board is the bounding box of all the gadgets placed thus far with a move-wasting gadget appended to the bottom of the board. The left column of the move-wasting gadget is aligned with the leftmost column of the first (leftmost) variable gadget and the sixth-from-right column (the rightmost column having height 3) is aligned with the rightmost column of the overflow block. We then fill all squares not part of a gadget nor recorded as empty with white pawns and place side rails on all boundary edges except as described in the move-wasting and reward gadgets. The anchor is on the black king as part of the initial state of the move-wasting gadget.

4.7 Analysis

Our analysis of gadget behavior in the preceding sections constrains the white king's pushes under the assumption that there are a specific number of empty spaces (often 0) in a particular row or column on a side of the gadget. We have already discharged the assumptions regarding the rows associated with literals by our layout of the connection and overflow blocks. For

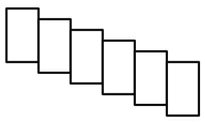


Figure 30 The layout of clause gadgets in the clause gadget block.

every other gadget except the variable gadgets, none of the constrained rows or columns intersects with another gadget, so the constraints on the edges are implied by the dense sea of white pawns outside the gadgets. For the variable gadgets, we assumed that pushing down in the second-to-left column of a variable gadget is not possible, but that column contains the previous variable gadget's rightmost column. We discharge this assumption by noting that in the final state of each variable gadget (after the white king has left the gadget), the rightmost column of that gadget is filled with white pawns, so pushing down in that column is indeed not possible.

Thus the white king must traverse the variable gadgets, setting the value of each variable, then traverse through the bridge gadget to the clause gadgets, where at least one empty space must remain in each checked column for the king to reach the reward gadget. If the choices made while traversing the variable gadgets results in filling all of the empty spaces in a checked column (i.e., the clause is false under the corresponding truth assignment), then White can only push by using a move outside the move-wasting gadget and Black wins on the next turn. If the white king successfully traverses every clause gadget (i.e., every clause is true under the truth assignment), then White wins when the white king pushes the black pawn off the board in the reward gadget. Thus White has a winning strategy for this Push Fight game state if and only if the input quantified 3-CNF formula is true.

References

1 Jeffrey Bosboom, Erik D. Demaine, and Mikhail Rudoy. Computational Complexity of Generalized Push Fight. arxiv:1803.03708, 2018. `https://arxiv.org/abs/1803.03708`.

2 Erik D. Demaine, Martin L. Demaine, and David Eppstein. Phutball endgames are NP-hard. In R. J. Nowakowski, editor, *More Games of No Chance*, pages 351–360. Cambridge University Press, 2002.

3 Aviezri S. Fraenkel, M. R. Garey, David S. Johnson, T. Schaefer, and Yaacov Yesha. The complexity of checkers on an N * N board - preliminary report. In *19th Annual Symposium on Foundations of Computer Science, Ann Arbor, Michigan, USA, 16-18 October 1978*, pages 55–64. IEEE Computer Society, 1978. `doi:10.1109/SFCS.1978.36`.

4 Aviezri S. Fraenkel and David Lichtenstein. Computing a perfect strategy for n x n chess requires time exponential in n. *J. Comb. Theory, Ser. A*, 31(2):199–214, 1981. `doi:10.1016/0097-3165(81)90016-9`.

5 M. R. Garey and D. S. Johnson. The rectilinear Steiner tree problem is *NP*-complete. *SIAM Journal on Applied Mathematics*, 32(4):826–834, 1977. `doi:DOI:10.1137/0132071`.

6 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA, 1979.

7 Michael Hoffmann. Motion planning amidst movable square blocks: Push-* is NP-hard. In *Proceedings of the 12th Canadian Conference on Computational Geometry*, pages 205–210, 2000.

8 Jerry Holkins. Exposition. `https://www.penny-arcade.com/news/post/2015/12/14/exposition`, 2015.

9 Ben Kuchera. Push Fight is the best board game you've never heard of. https://web.archive.org/web/20131211190946/http://penny-arcade.com/report/article/push-fight-is-the-best-board-game-youve-never-heard-of, 2012.

10 Brett Picotte. Push Fight game. http://pushfightgame.com/, 2016. Accessed: 2017-06-22.

11 J. M. Robson. N by N checkers is exptime complete. *SIAM J. Comput.*, 13(2):252–267, 1984. doi:10.1137/0213018.

12 L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time (preliminary report). In *Proceedings of the 5th Annual ACM Symposium on Theory of Computing*, pages 1–9, 1973. URL: https://dl.acm.org/citation.cfm?id=804029.