Completing the Complexity Classification of 2-Solo Chess: Knights and Kings Are Hard

Kolja Kühn
Institute of Theoretical Informatics, Karlsruhe Institute of Technology, Germany

Wendy Yi
Institute of Theoretical Informatics, Karlsruhe Institute of Technology, Germany

Abstract

We extend the study of the 2-Solo Chess problem which was first introduced by Aravind, Misra, and Mittal in 2022. 2-Solo Chess is a single-player variant of chess in which the player must clear the board via captures such that only one piece remains, with each piece capturing at most twice.

It is known that the problem is solvable in polynomial time for instances containing only pawns, while it becomes NP-complete for instances restricted to rooks, bishops, or queens. In this work, we complete the complexity classification by proving that 2-Solo Chess is NP-complete if the instance contains only knights or only kings.

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

Keywords and phrases Solo chess, puzzle games, board games, NP-completeness

Digital Object Identifier 10.4230/LIPIcs.FUN.2026.27

Related Version *Full Version*: https://arxiv.org/abs/2603.01675

1 Introduction

Solo Chess is a single-player variant of chess. Implemented on chess.com [6], given is a chess position consisting only of pieces of the same color. The goal is to find a sequence of captures such that only one piece remains, following the rules of chess except that pieces are allowed to capture other pieces of the same color. Each move has to be a capture, and each piece has a *budget* that limits the number of times it can move. In k-Solo Chess, each piece can capture at most k times. The original game on chess.com is implemented with $k = 2$. (A difference to our variant is that in the chess.com version, the position has at most one king, and if so, it has to be the final remaining piece of the capture sequence.)

The complexity of deciding whether a Solo Chess puzzle can be solved was first studied by Aravind, Misra, and Mittal [1, 2], where the authors generalize the game to be played on a board of arbitrary size. They focus on instances that contain only a single piece type and provide complexity results for pawns, queens, rooks, and bishops. While there is a linear time algorithm for Pawn 2-Solo Chess, they show NP-completeness for Queen 2-Solo Chess. Moreover, they prove hardness for Bishop Solo Chess and Rook Solo Chess variants, where each piece receives an individual capture budget, which may be 0, 1, or 2.

The result for rooks was improved by Bilò, Di Donato, Gualà, and Leucci [4, 3], who showed that Rook Solo Chess is already NP-hard if every rook has budget 2. With a reduction given in [2], this hardness result immediately extends to 2-Solo Chess with only bishops. For knights, it was shown in [4] that 11-Solo Chess is NP-hard. For King Solo Chess, no complexity result was known.

In [5], a variant is considered where each piece has unlimited budget. While all single piece type variants are in P, combining two piece types makes the problem NP-hard again. They also consider a variant where there is a piece (the king in the original game) which must be the final piece remaining. This problem is again in P for a single piece type, while the

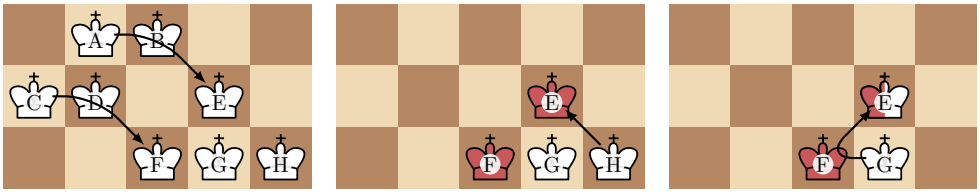


Figure 1 An example instance of King 2-Solo Chess, 2-kings are in white, 1-kings in half-red and 0-kings in red. Middle: Configuration after $(A \to B \to E)$ and $(C \to D \to F)$. Right: Configuration after $(H \to E)$. After $(G \to F \to E)$, the instance is cleared with E as final square.

picture becomes more diverse when considering the problem where the uncapturable piece is of a different piece type than all the remaining pieces. Various piece type combinations are discussed, some of which are decidable in polynomial time while others are NP-hard.

Other variants change the topology of the board such as 1D-Solo Chess (played on a single row) or Graph Capture Game (played on the edges of a given graph) [2].

Our Contribution. In this paper, we complete the complexity classification of 2-Solo Chess with a single piece type. Based on the first author's Master's thesis [7], we prove NP-hardness for both King 2-Solo Chess and Knight 2-Solo Chess by reduction from a well-known variant of 3-SAT.

2 Preliminaries

In this paper, we only consider 2-Solo Chess, and a *piece* is either a king or a knight. Furthermore, each instance contains only pieces of the same type, i.e., only kings or only knights. A *configuration* is a setup of pieces on a board, together with a *budget* for each piece. A piece with budget $b \in \{0, 1, 2\}$ is also called a *b-piece* (or *b-king* or *b-knight*). In each move, a piece with positive budget captures another piece. The move $(X \to Y)$ describes that the piece on square X captures the piece on square Y. If the same piece immediately continues capturing a piece on square Z, we use the notation $(X \to Y \to Z)$. Applying a sequence of captures s to a configuration C, we obtain a new configuration denoted by $C[s]$. A capturing sequence s *clears* C if there is only one piece remaining in $C[s]$. In this case, we call the square of the last piece in $C[s]$ the *final square* of s. Figure 1 shows a clearable instance of King 2-Solo Chess.

We observe that capturing sequences are monotone regarding budgets. Let C and C' be configurations with the identical setup of pieces. We say $C' \geq C$ if the budget of each piece in C' is at least the budget of the corresponding piece in C.

▶ **Observation 1.** *Let C be a configuration and s a capturing sequence. Let C' be a configuration with $C' \geq C$. Then, s is a valid capturing sequence for C', and it is $C'[s] \geq C[s]$.*

For every capturing sequence s of some configuration C, there are pieces in C that never move from their original square. We call these pieces *virtual 0-pieces* with respect to s. All other pieces are called *virtual 2-pieces*. Observe that if we replace each virtual 0-piece in C with a piece with budget 0, then s is still a valid capturing sequence.

Before we fix properties on virtual 0-pieces, it is useful to introduce some graph-theoretical notions. We define a *capture graph* G of C whose vertex set consists of pieces in C. Two vertices are connected by an (undirected) edge if and only if the pieces can capture each other (disregarding the budget). For kings and knights, every capture yields a subgraph of

the original capture graph. The *neighborhood* $N(X)$ of a vertex set X is the set of vertices not in X that have an edge to a vertex in X. A *vertex cut* is a set of vertices whose removal disconnects the graph. The following two lemmas provide properties of virtual 0-pieces.

▶ **Lemma 2.** *Let s be a clearing sequence with final square f. The following properties hold.*

1. *The virtual 0-pieces form a connected subgraph of the capture graph.*
2. *In every vertex cut of the capture graph, there is a virtual 0-piece.*
3. *For every set X of virtual 0-pieces that does* not *contain f, the number of virtual 2-pieces in the neighborhood of X is at least $|X|$.*

Proof. Observe that the final square f holds a virtual 0-piece (provided the instance contains more than a single piece). Any other virtual 0-piece eventually reaches f through a series of captures, so the path it takes consists entirely of virtual 0-pieces. Thus, within the subgraph of virtual 0-pieces, each piece is connected to f by a path, which shows the claim.

For the second property, assume that there is a vertex cut where every piece moves. But then, the capture graph is disconnected afterwards, and thus, s is not a clearing sequence.

Finally, let X be a set of virtual 0-pieces that do not contain f. Then, for each piece of X its original square is cleared at some point. The only way to clear a square without capturing with the piece that is originally present, is to capture into the square with a 2-piece, followed by another capture by that piece. As a result, for each virtual 0-piece of X, one neighboring 2-piece captures it, yielding the claim. ◀

It is clear that the total number of captured pieces in a capturing sequence does not exceed the sum over all virtual budgets. More specifically, the budget of the final piece of a clearing sequence is at most the difference between the sum over all virtual budgets and the total number of captured pieces. We say that a clearing sequence *loses* budget if the budget of the final piece is strictly less than the difference. In particular, budget is lost if a 1-piece is captured.

▶ **Lemma 3.** *Let s be a clearing sequence that does not lose budget, and let X be a set of virtual 2-pieces. The number of virtual 0-pieces in the neighborhood of X is at least $|X|$.*

Proof. Let X be a set of virtual 2-pieces. Since s does not lose budget, each piece in X captures a virtual 0-piece at some point and becomes a 1-piece, which captures another virtual 0-piece. Thus, no square can be captured by two distinct virtual 2-pieces, and every virtual 2-piece has to capture at some point. This means that X has at least $|X|$ virtual 0-pieces in its neighborhood. ◀

Consider now a capture graph that has a leaf u. Observe that if u is not the final square of some clearing sequence s, then the latter contains a capture of u into its only neighbor, and no capture into u. In particular, if u has budget 0, it never leaves its square and we call it *stranded*. If the neighbor v of u has only one other neighbor w, we call $A = (u, v, w)$ a (2-)*antenna* and we can generalize the above observation.

▶ **Observation 4.** *Let $A = (u, v, w)$ be an antenna and let s be a clearing sequence whose final square is not u or v. Then, s contains the moves $(u \to v \to w)$, and it is $C[s] \leq C[s']$ where s' is obtained from s by shifting $(u \to v \to w)$ to the beginning of the sequence.*

The definition and observation generalizes to antennae of arbitrary length.

▶ **Observation 5.** *Let $A = (v_0, \ldots, v_b)$ be an antenna in a configuration C, and let the leaf v_0 of the antenna have a budget of b' with $b' < b$. Then there is no capturing sequence that empties each square of A. Thus, any clearing sequence s of C has its final square in $\{v_0, \ldots, v_{b'}\}$, without loss of generality on $v_{b'}$.*

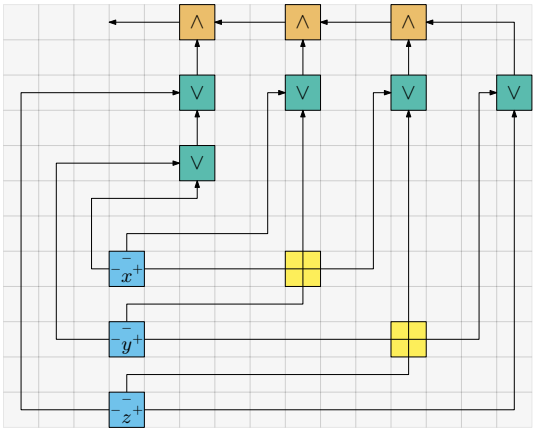


Figure 2 A rectilinear embedding for 3,3-SAT instance $I = (U, C)$ with $U = \{x, y, z\}$ and $C = \{(\neg x, \neg y, \neg z), (\neg x, \neg y), (x, \neg z), (y, z)\}$. The crossings are indicated by yellow tiles.

3,3-SAT

In this section we briefly discuss the 3-SAT-variant which we use for our reductions. A 3,3-SAT instance consists of a set of variables U and a set of clauses C. Each clause contains two or three literals, and each variable occurs exactly three times in the formula. Tovey showed that this variant of SAT remains NP-complete [8, Thm. 2.1]. It is easy to show that we may assume that of these three variable occurrences, two are negative and one is positive.

In both reductions, we assume a rectilinear embedding of the 3,3-SAT instance is given. The rectilinear embedding is constructed as follows. There are variables with three outgoing edges and binary OR- and AND-gates, each placed at integer coordinates. They are connected by rectilinearly drawn directed edges, with turns and crossings at integer coordinates.

Then, the embedding is discretized into tiles for the logic gates, variables, straight wires, wire turns and crossings. See Figure 2 for an example embedding. These tiles fit in a square grid, with variables placed in the first column in the lower rows at sufficient distance to each other. Clauses are placed to the top right of the variables in distinct columns at sufficient distance. In particular, the top row contains an AND-tile for each clause except the top-right clause. Below there are two rows for the up to two OR-tiles of each clause. Each OR-tile is connected to the tile immediately above it, except the top-right OR-tile, which is connected to the AND-tile to its top left. AND-tiles are connected to the AND-tile to their left, except for the leftmost AND-tile which outputs the result. Finally, the first two literals of each clause are connected to the corresponding (lower) OR-tile and, if required, the third literal is connected to the corresponding higher OR-tile. This placement allows for the edges to be drawn such that they do not go downwards and every pair of edges crosses at most once. Then it is easy to see that the resulting directed graph with gates, variables and crossings as vertices is acyclic. The entire embedding is scaled such that each tile has size 100×100.

3 King 2-Solo Chess

We show that King 2-Solo Chess is NP-hard by reduction from 3,3-SAT. Given is a 3,3-SAT instance with a formula Φ together with a drawing as described in the previous section. We now construct an instance I_Φ of King 2-Solo Chess that mimics the drawing of Φ such that Φ is satisfiable if and only if I_Φ is clearable. For this, we have two types of *variable assignment* gadgets VAR and 2VAR, AND, OR, and crossing gadgets (X), as well as

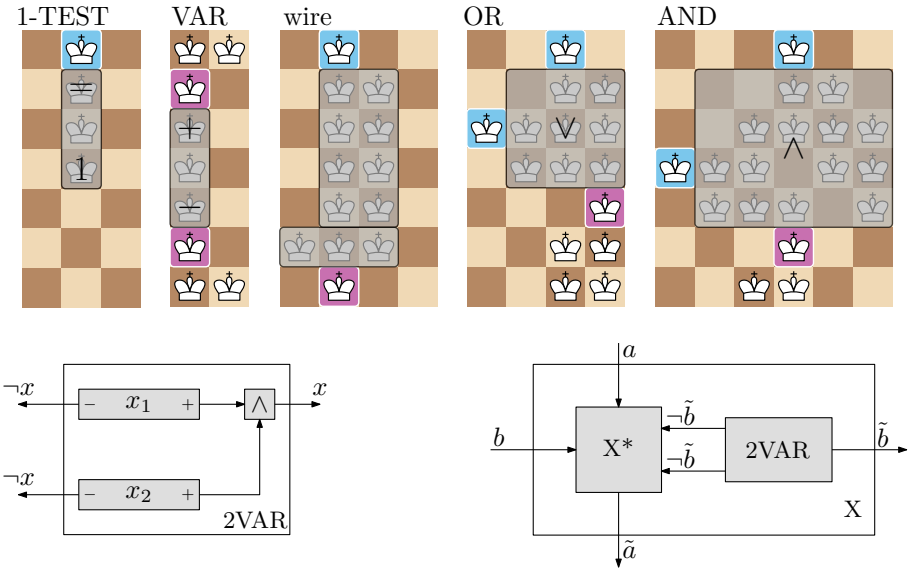


Figure 3 Gadgets from left to right: 1-TEST, VAR, a wire with connector, OR, AND, 2VAR, X. The input squares are marked in blue, the output squares in purple.

a transportation gadget (so-called *wire*) and a checking gadget (1-TEST). Each gadget has up to two input and up to two output squares. All gadgets are depicted in Figure 3. We call the wire, VAR, OR, AND, X* gadget (part of the X gadget), and 1-TEST gadget *atomic gadgets*. Each non-transportation atomic gadget is placed (possibly rotated or mirrored as needed) in the center of the corresponding tile in the given embedding. They are connected via wires in a way that adjacent atomic gadgets share a single square, which is an output square in one gadget and an input square in the other. Other than that, the gadgets do not touch or overlap with other gadgets. Wires are placed along the edges of the embedding. Within a tile, wires are extended to connect the tile input and output with the gadget inputs and outputs, respectively. The size of the tiles for the embedding is chosen such that this is always possible. The 1-TEST gadget is placed such that its input square is the output square of the wire that corresponds to the circuit output. See Figure 4 for a full example instance.

It is useful to assume that in a clearing sequence of I_Φ, the gadgets are cleared in a specific order, starting with the VAR gadgets and following the wires towards the 1-TEST gadget. We later show that every clearable instance indeed admits such a clearing sequence. With such an order, we can think of the capturing of kings as a *signal* that goes from each VAR gadget to the 1-TEST gadget. The budget of the king on an input square of a gadget right before making moves within the gadget indicates an incoming signal. Analogously, the budget of the king on an output square right after clearing the corresponding gadget indicates an outgoing signal. We note that the only two possible budgets for both input and output are 0 and 1. These signal values represent false and true, respectively. In the following, we describe the individual gadgets in more detail.

The 1-TEST gadget checks if its input is 1. It is easy to verify that not all kings in the gadget can be captured and thus, it contains the final square in every clearing sequence. In particular, it is only clearable if its input is a 1-signal.

The VAR gadget is connected to a wire on each end, one side representing the positive literal and the other the negative literal. We note that in every clearing sequence, the middle king of each VAR gadget captures either to the top or to the bottom, producing a 0-signal in

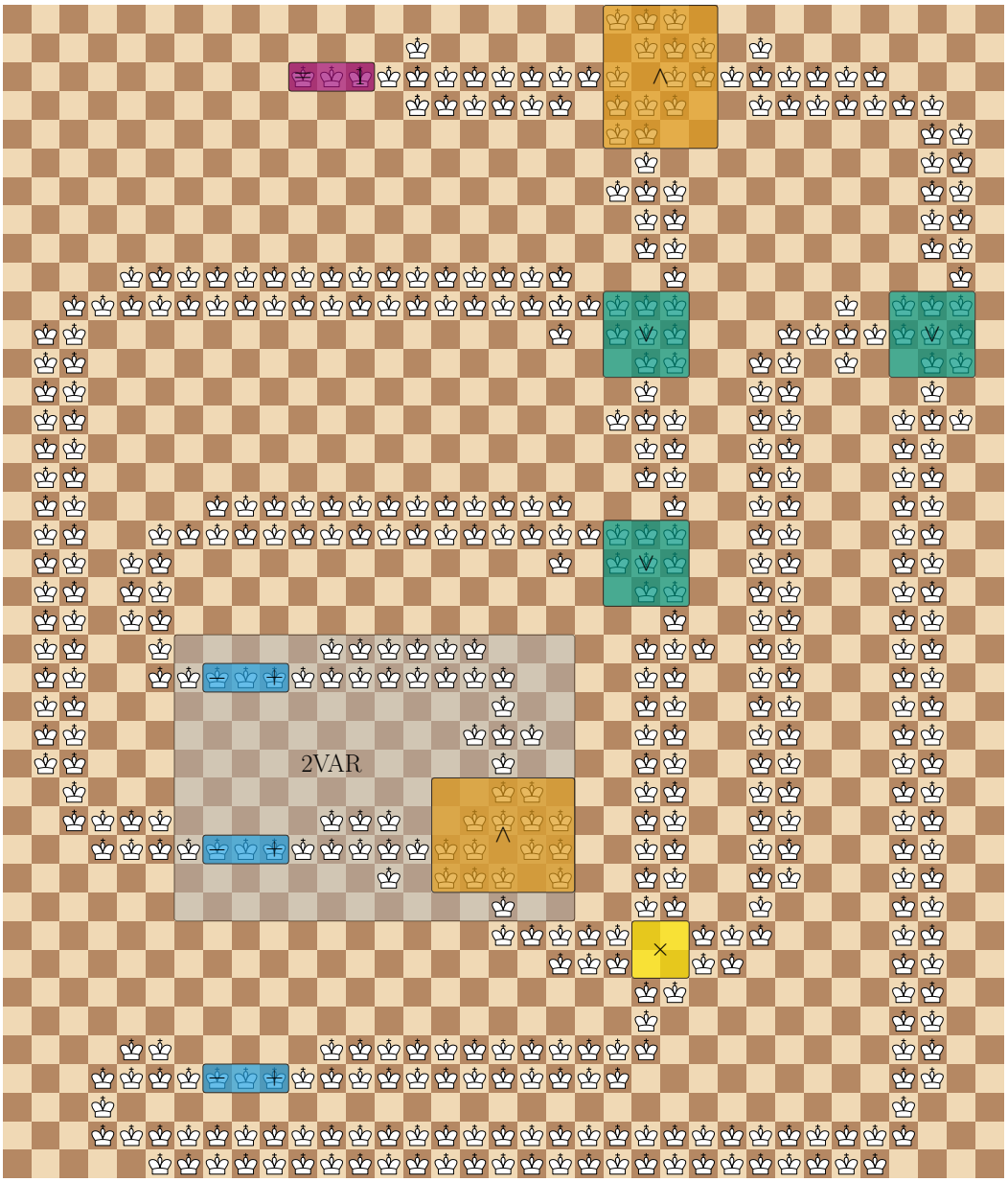


Figure 4 An example instance I_Φ for 3,3-SAT-instance Φ with variables $\{x, y\}$ and clauses $\{\{\neg x, \neg x, y\}, \{x, \neg y\}\}$. The crossing gadget (in yellow) is omitted due to its size. Moreover, I_Φ is constructed without considering tiles to save space.

the direction of the capture and a 1-signal on the other side. The 2VAR gadget produces two copies of the value on the side representing the positive literal. As one would expect, variable gadgets never produce 1-signals on both the positive and the negative side, which we show in the proof of correctness.

The wire gadget propagates signals. A wire starts with a single input square (the output square of the previous gadget) and ends with a *connector*, which consists of a row with three kings and one row with one king, the output square. The output square is an input square of the subsequent gadget. A wire can be placed both vertically and horizontally, and corner gadgets (see Figure 5) allow a wire to turn.

The OR and AND gadget both have two input squares and one output square. The X gadget, which consists of a 2VAR gadget and an X* gadget (omitted due to its size), has two pairs consisting of an input and an output square, and each output square should output the incoming signal at its corresponding input square.

Correctness of Reduction. As the correctness of gadgets requires significant case work, it is convenient to assume for now that the wire, OR-, AND-, and X gadget work as expected. This means that they can output the correct value for all possible inputs and cannot do better. To define this formally, we introduce the notion of *normalized* capturing sequences. A capturing sequence of a gadget is normalized if it clears the full gadget except for the output squares. Note that in a gadget with only one output square, every normalized capturing sequence is also a clearing sequence of the gadget (but not vice versa). Let f be a function that maps i input signals to one output signal. We say that a gadget with one output *computes* function f if the following holds for every possible combination X of input signals:

1. There is a normalized capturing sequence that outputs $f(X)$.
2. Every normalized capturing sequence outputs $f(X)$ or a signal with lower value.

This notion can be easily extended to the X gadget, which has two outputs. There should be a normalized capturing sequence such that both outputs are as expected and for every other capturing sequence, no individual output is better than expected.

Before proving that King 2-Solo Chess is NP-hard, we show that if I_Φ is clearable, then it can be cleared in a specific order, which we call a *canonical* clearing sequence. For this, we define the *gadget graph*, whose vertices consist of the atomic gadgets. There is a directed edge from a gadget A to gadget B if an output square of A is an input square of B. Note that the gadget graph is acyclic and thus admits a topological order of the gadgets where the 1-TEST gadget is last. In a canonical clearing sequence, the moves are ordered as follows.

1. VAR gadgets are cleared first.
2. If a move is made within an atomic gadget, then all previous atomic gadgets in the gadget graph are already cleared.
3. The captures within an atomic gadget form a normalized capturing sequence that is consecutive in the clearing sequence.

▶ **Lemma 6.** *If I_Φ is clearable, then it admits a canonical clearing sequence.*

Assuming the correctness of the variable, wire, OR, AND, and X gadgets, we are now ready to prove that 2-King Solo Chess is NP-hard. The idea is that in a canonical clearing sequence, we may determine an input value of a gadget A as the budget of the king on the respective input square right before the first move within A is made. Thus, following a canonical clearing sequence is equivalent to evaluating the corresponding 3,3-SAT-instance. See the full paper for the proof of the following theorem.

▶ **Theorem 7.** *King 2-Solo Chess is* NP*-hard.*

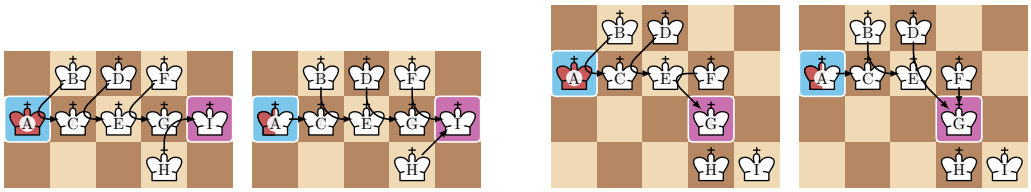


Figure 5 Left: Capturing sequences for a straight wire with input 0: $(B \to A \to C)$, $(D \to C \to E)$, $(F \to E \to G)$, $(H \to G \to I)$, with input 1: $(A \to C)$, $(B \to C \to E)$, $(D \to E \to G)$, $(F \to G \to I)$, $(H \to I)$. Right: wire with corner (same capturing sequences as for straight wire).

Correctness of Gadgets

In the following, we prove that the variable, wire and the OR, AND, and X gadget each compute the desired function. For every gadget apart from the variable gadgets, we describe a normalized capturing sequence that yields the expected output(s) for every combination of inputs. Moreover, we show that we cannot do better. Note that the difference between the total virtual budget and the total number of kings is an upper bound for the output signal. For every normalized capturing sequence, we show that it has too many virtual 0-kings or that it loses too much budget to have a better output than expected.

Variable Gadgets. We first argue briefly that the variable gadgets are work as expected. Note that a VAR gadget never produces a 1-signal on both the positive and the negative side. This also holds for 2VAR gadgets: if it produces a 1-signal on the positive side, then both input values of the AND gadget are 1. Thus, both VAR gadgets produce 0-signals on the negative side. Moreover, if both a 0- and a 1-signal are produced on the side with two outputs, it is never worse to produce two 1-signals instead by monotony (Observation 1). Thus, we may assume that the signals on this side are equal.

Wire Gadget. The wire gadget can be seen in Figure 5. We show that the wire can propagate the input signal correctly. On the other hand, we show that there are no normalized capturing sequences with better output, which yields the following lemma.

▶ **Lemma 8.** *A wire computes the identity function $f(x) = x$ for $x \in \{0, 1\}$.*

Proof. Figure 5 shows a normalized capturing sequence that produces the correct output for each possible input. In both cases, the signal eats up the wire, and every two moves the number of columns with kings is reduced by 1 while preserving the input signal.

On the other hand, we show that there is no normalized capturing sequence that outputs 1 if the input is 0. Let k be the number of columns with two 2-kings. Then, without the input square, the wire has $k+2$ columns and $2k+4$ kings in total. Each column of the wire except for the input and the output square forms a vertex cut in the capture graph. Thus, each of the $k+1$ columns contains one virtual 0-king by Lemma 2. Moreover, the output king never captures in a normalized clearing sequence and is thus a virtual 0-king as well. Thus, with input i, every normalized capturing sequence has at most $i+2\cdot(2k+4-(k+1)-1) = i+2k+4$ total virtual budget, and at most $i+2k+4$ kings can be cleared. Since the input is a 0-king, which does not contribute additional budget, any clearing sequence results in a 0-king. ◀

Normalized capturing sequences for the corner gadget with expected outputs are also depicted in Figure 5. As the capture graph of a wire with corner is a subgraph of the capture graph of a straight wire, there is no normalized capturing sequence for input 0 that outputs 1, i.e., a wire with corners also computes the identity function.

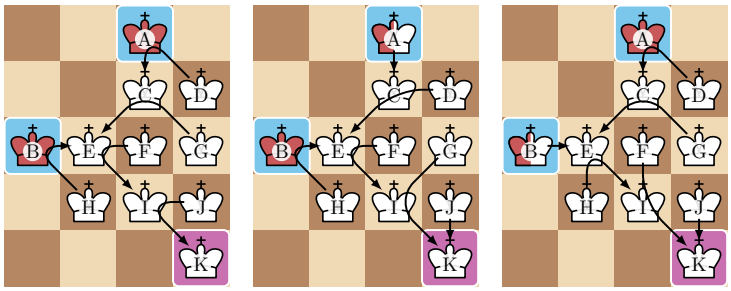


Figure 6 An OR gadget. If both inputs are 0: $(D \to A \to C), (H \to B \to E), (G \to C \to E), (F \to E \to I), (J \to I \to K)$ outputs a 0-king. If input A is 1, then $(H \to B \to E), (A \to C), (D \to C \to E), (F \to E \to I), (G \to I \to K), (J \to K)$ outputs a 1-king. If input B is 1, then $(D \to C \to A), (B \to E), (G \to C \to E), (H \to E \to I), (F \to I \to K), (J \to K)$ outputs a 1-king.

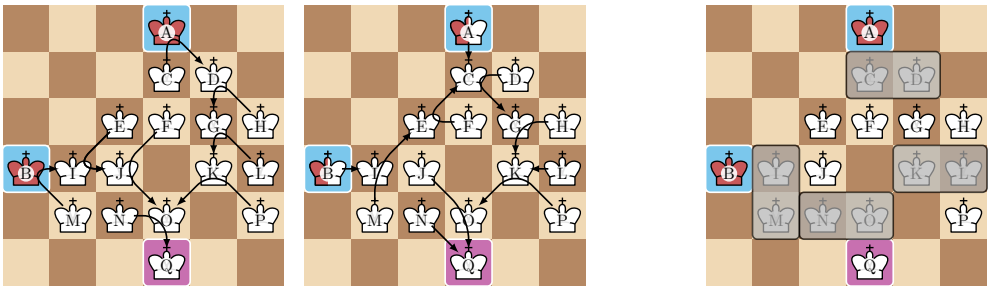


Figure 7 An AND gadget. Left: If both inputs are 0, then $(M \to B \to I), (E \to I \to J), (F \to J \to O), (C \to A \to D), (H \to D \to G), (L \to G \to K), (P \to K \to O), (N \to O \to Q)$ outputs a 0-king. If both inputs are 1, then $(B \to I), (A \to C), (M \to I \to E), (F \to E \to C), (D \to C \to G), (H \to G \to K), (L \to K), (P \to K \to O), (J \to O \to Q), (N \to Q)$ outputs a 1-king. Right: Cut vertex sets which contain one virtual 0-king each.

OR Gadget. The OR gadget can be seen in Figure 6 with normalized capturing sequences that produce the correct output for each possible input. It remains to show that there is no normalized capturing sequence with better output.

▶ **Lemma 9.** *The OR gadget computes the OR-function.*

Proof. We show that if both inputs are 0, it is impossible to output 1. The capture graph of the OR gadget has three disjoint vertex cuts $\{C, D\}, \{E, H\}, \{I, J\}$. By Lemma 2, each vertex cut contains a virtual 0-king. Moreover, the output king never captures in a normalized clearing sequence. As the gadget (including output but excluding input) has nine kings in total, every normalized capturing sequence has a total virtual budget of at most $2 \cdot (9-4) = 10$ (without the inputs). With this budget, at most ten kings can be cleared, which is exactly the number that needs to be captured. Since both inputs are 0-kings, which do not contribute additional budget, every normalized clearing sequence results in a 0-king. ◀

AND Gadget. We show that the AND gadget can output the expected signal, but there is no normalized capturing sequence with better output.

▶ **Lemma 10.** *The AND gadget computes the AND-function.*

Proof. Figure 7 shows normalized capturing sequences with the desired output if both inputs are 0 or if both inputs are 1. In the other two cases, where one input is 0 and the other is 1, there are normalized capturing sequences that output 0 by Observation 1.

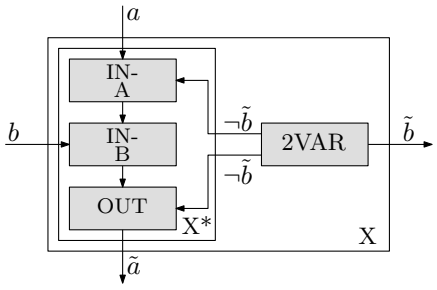


Figure 8 The wire crossing gadget with its four components.

It remains to show that if at least one input is 0, then it is impossible to output 1. We first argue in the following that the number of virtual 0-kings is at least 7 (without the inputs) for every normalized capturing sequence. In a normalized capturing sequence, the output square always is a virtual 0-piece as it does not move. The gadget contains the vertex cuts $\{C, D\}, \{I, M\}, \{K, L\}, \{N, O\}$ (see Figure 7), which contain at least one virtual 0-king each by Lemma 2.2. Moreover, virtual 0-kings are connected by Lemma 2.1. We observe that to connect the virtual 0-kings in the four vertex cuts and the output king, at least seven virtual 0-kings in total are needed. Without the inputs, this gives us a virtual budget of at most 16, which is also the number of kings that need to be cleared. Thus, if at least one input is 0, then no capturing sequence with more than seven virtual 0-kings clears the gadget.

Consider a normalized capturing sequence s with seven virtual 0-kings. If both inputs are 0, the output of s cannot be 1 since the virtual budget is not sufficient.

If one input is 1, we argue that s loses budget by showing that there is no placement of the virtual 0-kings such that all needed properties from Lemma 3 are satisfied. We distinguish between cases based on whether king L is a virtual 0-king.

If king L is a virtual 0-king, then the only way to connect the four vertex cuts using seven virtual 0-kings is to choose $\{L, G, C, E, I, N\}$ and the output king. Any other way uses more than seven 0-virtual kings. But then, $\{H, K, P\}$ are three virtual 2-kings that only have two virtual 0-kings in their neighborhood, which is a contradiction to Lemma 3.

If king L is not a virtual 0-king, then king K is a virtual 0-king. We further distinguish the cases which kings in $\{F, G, H\}$ are virtual 0-kings. If none of them or only king F is a virtual 0-king, then $\{H, L, P\}$ are virtual 2-kings that only have two 0-kings in their neighborhood. If at least two of $\{F, G, H\}$ are virtual 0-kings, then there are at least eight virtual 0-kings and the budget is not sufficient to clear the gadget.

Otherwise, either king G or king H is a virtual 0-king. Then, king D is, too: This holds for king G since otherwise, $\{H, L, P\}$ does not have enough virtual 0-kings in its neighborhood. For king H, this holds since otherwise, the regions cannot be connected using seven virtual 0-kings. Moreover, the kings C, E and P are not virtual 0-kings since otherwise, the regions cannot be connected using seven virtual 0-kings. With the same argument, king O is a virtual 0-king. Now, we know that D, K, O, Q and either G or H are virtual 0-kings, while C, E, F, L, P and either G or H are not. If input a is 1, then $\{C, G, H, L, P\}$ contains four virtual 2-kings, but is only connected to at most three virtual 0-kings (D, K, and either G or H). If input b is 1, then $\{P, L, H, G, F, E, I, M\}$ contains six virtual 2-kings (since I and M cannot both be chosen). However, they are all not connected to input a, to king O and the output king. Thus, they have at most five virtual 0-kings in their neighborhood.

Thus, by Lemma 3, s loses budget, and the output cannot be 1. ◀

Table 1 Input/output table for the function X*, as well as the components of X*. The $\perp$ symbol indicates that the gadget is not clearable.

a	b	$\neg\tilde{b}$	$X^*(a,b,\neg\tilde{b})$
0	0	0	$\perp$
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	$\perp$
1	0	1	1
1	1	0	1
1	1	1	1

a	$\neg\tilde{b}$	IN-A
0	0	0
0	1	0
1	0	1
1	1	2

IN-A()	b	IN-B
0	0	0
0	1	1
1	0	0
1	1	2
2	0	2
2	1	2

IN-B()	$\neg\tilde{b}$	OUT
0	0	$\perp$
0	1	0
1	0	0
1	1	0
2	0	1
2	1	1

Crossing Gadget. The crossing gadget takes two inputs a and b and outputs two signals $\tilde{a}$ and $\tilde{b}$. It contains a 2VAR gadget that feeds $\neg\tilde{b}$ into the X* gadget and outputs $\tilde{b}$. The idea of the crossing gadget is that there is a normalized capturing sequence if and only if $\tilde{b} \leq b$. Moreover, in this case, the gadget should output $\tilde{a} \leq a$. These conditions ensure that an incoming 0-signal never becomes a 1-signal. The first table in Table 1 shows the desired outputs for the X* gadget for the three inputs a, b and $\neg\tilde{b}$. There are three additional gadgets IN-A, IN-B and OUT that check the inputs (see Figure 8), which are concatenated by a modified wire. Other than the usual gadgets, the output of IN-A and IN-B as well as the input of IN-B and OUT is not a single square, but a combination of two squares. Moreover, the wire between IN-A and IN-B as well between IN-B and OUT does not end with a wire connector. As a consequence, we can input and output more types of signals. The 0-signal and the 1-signal is the same as before: one square has a 0- or a 1-king, respectively, and the other square is empty. Additionally, there is a 2-signal where one square has a 2-king and the other square has a 1-king. Note that a 2-signal can be trivially transformed into a 1-signal. Moreover, the modified wire can output a 2-signal if the input is a 2-signal, see Figure 9. Conversely, we note that 0- and 1-inputs can never become 2-signals. This means that monotony from Observation 1 also applies to 0-, 1-, and 2-signals.

Table 1 shows the functions for IN-A, IN-B, and OUT. We note that concatenating the three functions as shown in Figure 8 yields the desired function X*. We have a gadget for the IN-B-function and the OUT-function, for the IN-A-function, we can use the same gadget as for IN-B by swapping the inputs.

Assuming the correctness of each gadget, the correctness of the X gadget directly follows. Figure 9 shows the IN-B gadget and the OUT gadget. In the IN-B gadget, the upper input of the IN-B gadget corresponds to the output of the IN-A gadget while the left input is signal b. In the OUT gadget, the left input corresponds to the output of the IN-B gadget while the upper input is signal $\neg\tilde{b}$. The following lemmas prove that the gadgets work as expected; see the full paper for the proofs.

▶ **Lemma 11.** *The IN-B gadget computes the IN-B-function, where the upper input signal is in* $\{0,1,2\}$*, and the left input signal is in* $\{0,1\}$*.*

▶ **Lemma 12.** *The OUT gadget computes the OUT-function, where the upper input signal is in* $\{0,1\}$ *and the left input signal is in* $\{0,1,2\}$*. There is no normalized capturing sequence if both inputs are* 0*.*

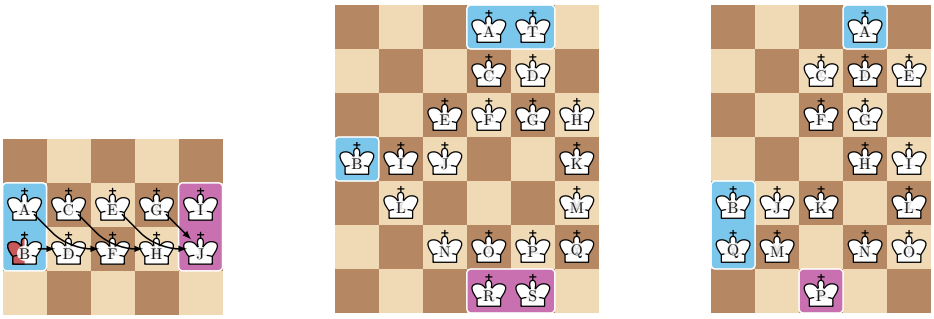


Figure 9 Left: Propagation of a 2-signal in a wire without wire connector. Middle: The IN-B gadget. Right: The OUT gadget.

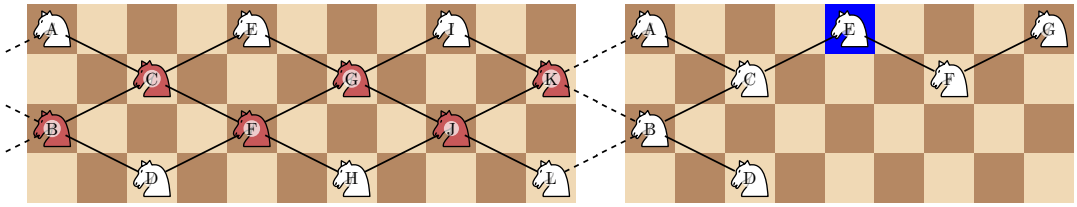


Figure 10 Left: The WIRE in Knight 2-Solo Chess. Originally, all pieces have a budget of 2. We show that the knights drawn in red are virtual 0-pieces. Right: A WIRE ending in a 1-TEST.

4 Knight 2-Solo Chess

In this section, we establish the NP-hardness of Knight 2-Solo Chess, again by reduction from 3,3-SAT. The core idea of the reduction is the same as for King 2-Solo Chess, however, the specific gadgets differ. The main challenge that the knight poses in comparison to the king is the parity constraint in its movement: a knight on a dark square always moves to a light square and vice versa. As a result, the capture graph of any knight configuration is bipartite and contains none of the odd cycles that enabled many of the kings gadgets. We see an example of this in the following subsection on the wire.

Nevertheless, on a high level, the approach is the same: Given a 3,3-SAT formula Φ and a tiled embedding, we construct an instance I_Φ of Knight 2-Solo Chess that is clearable if and only if Φ is satisfiable. To this end, we once more present gadgets for wires, wire crossings, variable assignments, OR-gates, AND-gates, and a 1-TEST. As before, these gadgets allow for a polynomial-time reduction. Assuming the correctness of each of the gadgets, we directly conclude:

▶ **Theorem 13.** *Knight 2-Solo Chess is* NP*-complete.*

It remains to discuss the gadgets themselves. We begin with the wire and the signals that it carries.

4.1 The Wire

The wire is shown in Figure 10. Like the king's wire, it consists of columns of two pieces each. However, for the above mentioned parity reasons, the wire's columns alternate between being placed on light and dark squares, to allow knights to move from one column to the next. Another difference between the knight and the king is that the squares attacked by the former are more spread out. This sparsity not only "stretches" the knight's wire, it also leads to only one of the two diagonal edges between adjacent columns being present. As a result, one knight of each column has degree 4, while the other knight only has degree 2.

This break of symmetry is reflected in the possible signal values. Where the king only had one 1-signal, being interpreted as true, the knight's wire can propagate two different 1-signals that are dual to each other. To account for this, we define our signals as two-dimensional binary vectors. We encode false as $(0,0)$ and true as $(1,0)$. These are the signals that travel between variable assignments, OR- and AND-gates, and the final 1-TEST.

Additionally, we give two auxiliary signal values that are (only) used within the AND gadget and allow the latter to be decomposed into simpler parts. These values are the "dual 1-signal" $(0,1)$, as well as the $(1,1)$-signal which can be thought of as a 2-signal. Once more, these signal values adhere to a form of monotony, where $(0,0) \leq (1,0) \leq (1,1)$ and $(0,0) \leq (0,1) \leq (1,1)$. The signal values $(1,0)$ and $(0,1)$ are incomparable. We give a description of how the signals are realized in the wire later in this subsection.

We adapt the notion of a gadget *computing* a function to Knight 2-Solo Chess, using the above defined partial order on signal values. This would be ill-defined for a gadget which for some input could output either $(1,0)$ or $(0,1)$ (and not $(1,1)$), however, this is never the case in our constructions. Using this notion we state the main lemma, which we prove later.

▶ **Lemma 14** (Wire Lemma). *The wire computes the identity function.*

In the following, we always assume that the wire is cleared from left to right. To show the main lemma, we first show that any capturing sequence s clearing the wire indeed induces the highlighted set of virtual 0-knights. Note that these are precisely the knights with degree 4 in the full wire, we call them *inner knights*, and degree 2 vertices are called *outer knights*.

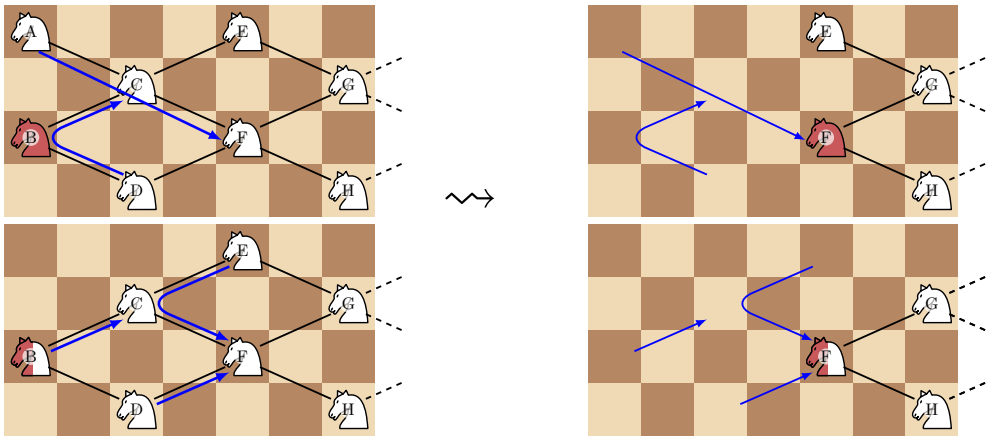
▶ **Lemma 15.** *The set of virtual* 0*-knights of a wire is exactly its set of inner knights.*

Proof. Each column of the wire forms a cut, containing one inner knight and one outer knight. By Lemma 2.2, at least one of of these is a virtual 0-knight. Thus, it suffices to show that no outer knight is a virtual 0-knight. Assume for contradiction that there is an outer knight that is a virtual 0-knight, say E. By Lemma 2.3, at least one neighbor of E, without loss of generality G, is a virtual 2-knight. Then, F, H and J are the virtual 0-knights of the cuts $\{F,G\}$, $\{G,H\}$ and $\{G,J\}$ respectively. As a result, both neighbors of virtual 0-knight H are virtual 0-knights themselves, a contradiction to Lemma 2.3. The claim follows. ◀

▶ **Corollary 16.** *In every capturing sequence that clears the wire from left to right, the inner knight of each column captures the inner knight of the column to its right.*

Proof. By the previous lemma, the inner knight of the column does not capture the outer knight to its right. If it captures to the left, the outer knight in its column becomes a cut vertex and virtual 0-knight, again violating the previous lemma. ◀

Using these properties, we define an ordered way to clear the wire: For each column, all captures from or to its left happen strictly before all captures from or to its right. Any capturing sequence clearing the wire can be brought into this form by repeatedly swapping consecutive moves that are ordered incorrectly. If the two moves do not share a square, they can safely be swapped. Otherwise, for a move from the right and a move from the left to interfere, they must share their destination square, namely the inner knight of the middle column. By Corollary 16, the move from the right is by an outer knight. Then, performing the left move *followed by* the right move results in a knight with budget 1, which is no less than without the swap. Thus, the resulting configuration is greater or equal than without the swap, and by monotony, the capturing sequence remains valid.



(a) Propagating a $(1,0)$- and a $(0,1)$-signal ... **(b)** ... retains a $(1,0)$- and a $(0,1)$-signal.

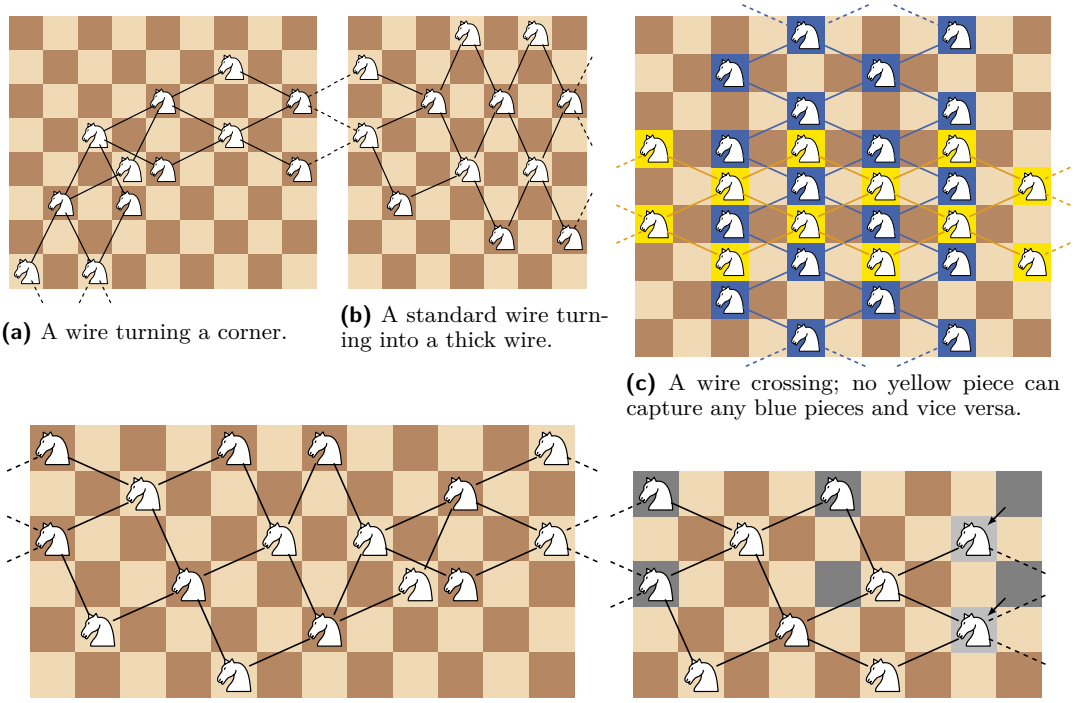
Figure 11 Two Knight Wires propagating a signal each.

Using ordered clearing sequences, we define possible signal values. Observe that due to the knight's parity constraints, wire columns alternate between *dark-squared* and *light-squared* columns. We define how to read off the value of a signal on a dark-squared wire column. This is done right after the final move from the column to its left. At this point, each column to the left is cleared while each column to the right is untouched. Our signal then consists of two binary components, so possible signal values are $(0,0), (0,1), (1,0)$ and $(1,1)$. The first component depends on the outer knight of the signal column. Since no outer knight is captured, the two options are for it to be present as a 2-knight, or to not be present at all. In the first case the component has value 1, in the second case it has value 0. The second component depends on the inner knight of the column. Being a virtual 0-knight, it is not present as a 2-knight, only with budget 0 or 1 (the latter occurs if it was captured by the outer knight to its left). We define the second signal component as the budget of this inner knight. Figure 11 shows two possible signal values and how to propagate them through the wire. The $(1,0)$- and $(0,1)$-signals are unusual in that they are dual to each other. A $(1,0)$-signal reads off a dark-squared column the same as a $(0,1)$-signal would on a light-squared column. Nevertheless, for parity reasons a $(1,0)$-signal is never converted into a $(0,1)$-signal and vice versa. We see this in the proof of Lemma 14.

Proof of the Wire Lemma. Figure 11 sketches how to obtain capturing sequences for each of the four signal values. In particular, for a second component of 0 or 1, the sequence starts with captures $(D \to B \to C)$ or $(B \to C)$ respectively. For a first component of 0 or 1 it continues with $(E \to C \to F)$ or $(A \to C \to F)$. Finally, for a second component of 1, a final move $(D \to F)$ is appended.

To see that signal values never increase, we again consider the difference between the total virtual budget and the number of pieces of the wire. For the extreme case of just one column, this difference is precisely one lower than the number of 1's in the signal, i.e., for signal $(0,0)$ it is -1, for signal $(1,1)$ it is 1 and for signals $(0,1)$ and $(1,0)$ it is 0. Each additional wire column contributes two units of virtual budget and two pieces. Thus, the difference never increases. It follows that signal values propagated through the wire do not increase.

Finally, we show that a $(1,0)$-signal indeed is never converted into a $(0,1)$-signal and vice versa. By the non-increasing difference above, any capturing sequence converting one into the other does not lose budget. Consider the configuration of a $(1,0)$-signal, as seen in



(a) A wire turning a corner.

(b) A standard wire turning into a thick wire.

(c) A wire crossing; no yellow piece can capture any blue pieces and vice versa.

(d) A rotation placement yields a wire with light-squared columns on top and dark-squared columns below.

(e) A shifting placement allows wire lengths that are not a multiple of 4.

Figure 12 Different WIRE placements. Each of them induce the same capturing graph and, therefore, function the same.

Figure 11. The signal-column 0-knight leaves its square eventually by a double capture from an adjacent 2-knight, $(D \to B \to C)$. As this move does not interfere with any other moves, we may assume that it is the first pair of moves in the capturing sequence. In the resulting configuration, knight C has two neighbors with a budget of 2, A and E. If it is captured by E, the eventual capture of A to its only neighbor loses budget. Otherwise, A captures C and F, restoring a $(1, 0)$-signal. The reverse case is analogous. ◀

To align wires and gadgets on the board, we give a number of placements of the wire on the chess board. We note that each of these correspond to the same capture graph, thus, the wire functions identically in each of the placements. Figure 12a shows a wire turning a corner, while Figure 12b has a wire turning into a *thick wire*. Combining these enables the *wire crossing*, seen in Figure 12c, without the need for a dedicated wire crossing gadget.

While the king wire can have any length, the knight wire only repeats every four columns. In Figure 12e, we give an additional placement to shift the wire (diagonally) by a single square to be able to align wires correctly for their gadgets. Note that due to parity, dark-squared columns are always mapped to dark-squared columns, so it is impossible to shift the wire by a single non-diagonal square. However, as inputs to gadgets are of a fixed color, they can still always be aligned correctly. Finally, after passing a series of corners and gadgets, a wire may be oriented so as to have the dark-squared column be aligned towards the top or towards the bottom. To switch between those two modes, we give a rotation placement, seen in Figure 12d.

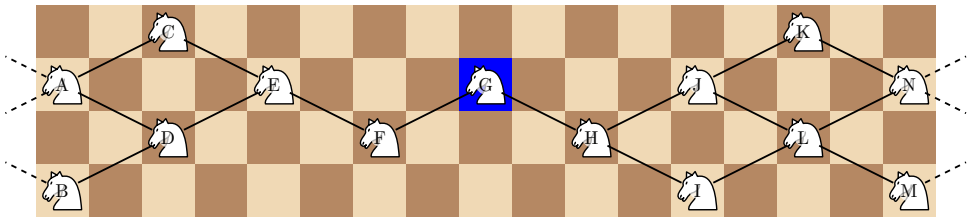


Figure 13 The three valued production gadget produces left and right outputs $(1,1) \leftrightarrow (0,0)$ or $(1,0) \leftrightarrow (0,1)$ or $(0,0) \leftrightarrow (1,1)$.

4.2 The 1-TEST Gadget

The 1-TEST gadget for the knight not only functions the same but is also implemented analogously to its king counterpart. Shown on the right of Figure 10, it consists of an input wire connected to a path of length 3. This antenna of length 3 guarantees, by Observation 5, that any clearing sequence of an instance has its final square in the 1-TEST gadget, without loss of generality on E.

▶ **Lemma 17.** *The 1-TEST gadget reduces to a single piece if it receives a* $(1,0)$*-signal as input. If it receives a* $(0,0)$*-signal as input, it does not reduce to a single piece.*

Proof. For input $(1,0)$, knight A is present with a budget of 2. Then the sequence $(D \to B \to C), (A \to C \to E), (G \to F \to E)$ reduces the gadget to a single piece, as claimed.

For input $(0,0)$, knight A is not present. By Lemma 4, any clearing sequence s (with final square E) can be assumed to start with captures $(G \to F \to E)$. Since knight B is the only neighbor of D, it is captured by D at some point. Its only path to final square E is via C, so s contains the capture $(B \to C)$. As knight B has a budget of at most 1 before this capture, it results in a 0-leaf on C, only adjacent to 0-leaf E. We conclude that no such clearing sequence s exists. ◀

4.3 The Variable Assignment

We now review the VAR gadget. It assigns the value of the literals of a binary variable x, having outputs x and $\neg x$, by producing a true signal to the left and a false signal to the right, or a false signal to the left and a true signal to the right. Translated to our two-dimensional signals, where true is encoded by $(1,0)$ and false is encoded by $(0,0)$ we state:

▶ **Lemma 18.** *A variable assignment gadget produces one* $(1,0)$*- and one* $(0,0)$*-signal.*

While an analogue to the king VAR gadget would be valid, we provide a slightly more general construction that proves useful for the remaining gadgets. To this end, we decompose the gadget into two parts: the *3-valued production* gadget (3-VAL), and the Decrement gadget.

The 3-VAL gadget, shown in Figure 13, has no inputs and two outputs. It can produce three different output pairs.

▶ **Lemma 19** (3-VAL Lemma)**.** *The 3-VAL gadget produces either a* $(1,1)$*-signal to the left and a* $(0,0)$*-signal to the right, or a* $(1,0)$*-signal to the left and a* $(0,1)$*-signal to the right, or a* $(0,0)$*-signal to the left and a* $(1,1)$*-signal to the right.*

We combine it with the Decrement gadget which as the name suggests, decrements a component of a signal, to obtain the VAR gadget (see Figure 14).

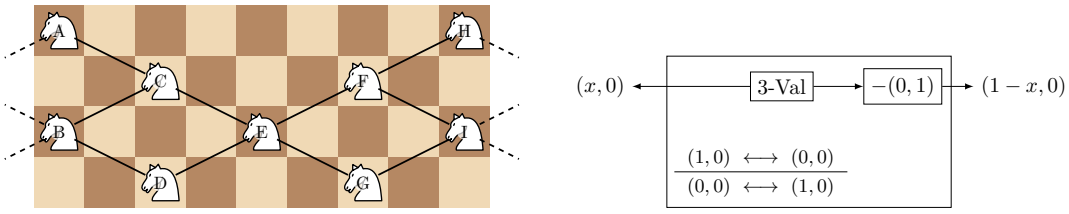


Figure 14 Left: The first-component Decrement gadget is a wire with an outer knight missing. Right: Using a 3-VAL and a Decrement gadget to build a VAR gadget.

▶ **Lemma 20** (Decrement Lemma). *The $-(1,0)$ gadget computes the function that maps $(1,y)$ to $(0,y)$ and $(0,y)$ to $\bot$. An analogous $-(0,1)$ gadget computes the function that maps $(x,1)$ to $(x,0)$ and $(x,0)$ to $\bot$.*

For a 3-VAL output of $(1,1) \leftrightarrow (0,0)$ the VAR gadget produces an error, the other two possible outputs yield the expected outputs of the VAR gadget. This shows Lemma 18.

We now prove the correctness of the two building blocks.

Proof of the 3-VAL Lemma. Consider knight G, highlighted in blue. Observe that, by Observation 5, no clearing sequence contains the capture $(H \to G)$. Instead, knight G can be captured via $(F \to G)$ which after resolving captures $(G \to H), (E \to D)$ yields a $(1,1)$-signal on the left and a $(0,0)$-signal on the right wire, the first of the three possible outcomes.

The remaining options are those where the blue knight is not captured and instead makes a capture itself: If it captures to the left through $(G \to F \to E), (C \to E \to D)$, this creates a $(0,0)$-signal on the left and (after captures $(H \to J \to L), (I \to L)$) a $(1,1)$-signal on the right wire. If instead it captures to the right through $(G \to H)$, this creates a $(0,1)$-signal on the right. Then, after a captures $(F \to E \to D)$, the left wire receives a $(1,0)$-signal. This covers all possible cases for the blue knight and the capture sequence as a whole, and yields the claimed three possible outcomes. ◀

Proof of the Decrement Lemma. We first consider the case where the first input component is 0, i.e., knight A is not present. Then both neighbors B and E of virtual 0-knight C are themselves virtual 0-knights, so by Lemma 2.3, the gadget cannot be fully cleared.

Next consider the case where the first input component is 1, i.e., knight A is present. Propagating the signal as usual in a wire yields the claimed output. We show that no better output is possible: The first output component is 0 by the fact that the would-be outer knight of the column of E already is not present. Furthermore, a virtual budget argument shows that any signal is decreased by at least 1. As a result, $(1,0)$ is mapped to (at most) $(0,0)$ and $(1,1)$ is mapped to (at most) $(0,1)$. ◀

In the following subsections we utilize the counterpart of the Decrement gadget, namely the Increment gadget. The first-component Increment (or $+(1,0)$) gadget increments the first component of a signal, as shown in Figure 15.

▶ **Lemma 21.** *The $+(1,0)$ gadget computes the function $(x,y) \mapsto (1,y)$ for $x,y \in \{0,1\}$. An analogous $+(0,1)$ gadget computes the function $(x,y) \mapsto (x,1)$ for $x,y \in \{0,1\}$.*

Proof. Figure 15 shows that the $(1,0)$-gadget allows a capturing sequence that sets the first component of a signal to 1. To see that no larger (or incomparable) output is possible, it suffices to show that if the second component of the output is 1, then so was the second component of the input. For this, consider an output of $(x,1)$, read off in the column

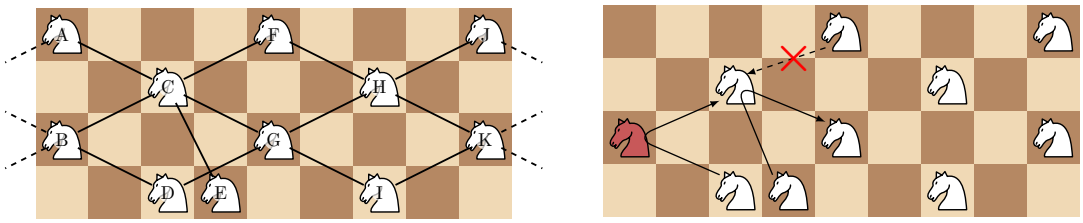


Figure 15 Left: The first-component Increment gadget is a wire with an extra leaf. Right: Example sequence that increments the first component of a signal.

containing knights F and G, i.e., knight G having a budget of 1 and all knights to its right being untouched. We now trace back how the knight got there. It was captured by a 2-knight, which was knight D, as C is a virtual 0-knight and I has not yet moved. Then D did not capture B and, thus, to leave its square towards C, knight B was a 1-knight, meaning the input had a second component of 1.

The $+(0,1)$ gadget has a leaf adjacent to a dark-squared inner knight and functions identically. ◀

This concludes our discussion of the VAR gadget. We remark that as in the king's case, we can combine two VAR gadgets and an AND gadget to create the 2VAR gadget (see Figure 3).

4.4 The OR Gadget

Observe that the OR function can be defined as the maximum function on binary inputs. This extends to our two-dimensional signal values: For $x, y \in \{(0,0),(1,0)\}$ it holds that $x \vee y = \max\{x, y\}$, where max denotes a component-wise maximum. This motivates the MAX gadget, shown in Figure 16, which precisely implements a component-wise maximum. The idea of the gadget is that the yellow signal is propagated as usual while the blue signal captures into the yellow wire. When a component of the blue signal is 1, this allows a blue or white 2-knight to remain, acting as an Increment gadget for the corresponding yellow signal component. This way, a component of the output is 1 if it was set to 1 for at least one of the inputs, thereby computing the component-wise maximum.

▶ **Lemma 22.** *There is a capturing sequence such that the MAX gadget with inputs* (u, v) *and* (w, x) *outputs their component-wise maximum* $(\max\{u, w\}, \max\{v, x\})$.

Proof. Start by reducing the 2-antenna (U, S, R) to a 0-knight R via captures $(U \to S \to R)$.

Consider now the case of a blue $(0,0)$-signal, i.e., knight M is absent, while N has a budget of 0. Then the capturing sequence $(P \to N \to O), (Q \to O \to C), (T \to R \to C)$ leaves behind just the yellow wire (with virtual 0-knight C having an actual budget of 0). Thus, under this capturing sequence the gadget yields the identity function on the yellow signal, as expected.

For a blue input of $(1,0)$, knight M is present and the capturing sequence $(P \to N \to O), (M \to O \to C), (Q \to R \to C)$ leaves behind a first-component Increment for the yellow signal in the form of 2-knight T. Conversely, for a blue input of $(0,1)$, knight N has a budget of 1 and the capturing sequence $(N \to O), (Q \to O \to C), (T \to R \to C)$ leaves behind a second-component Increment in the form of 2-knight P. Finally, for a blue input of $(1,1)$, combining the two approaches through captures $(N \to O), (M \to O \to C), (Q \to R \to C)$ leaves behind both a first- and a second-component Increment in 2-knights T and P. ◀

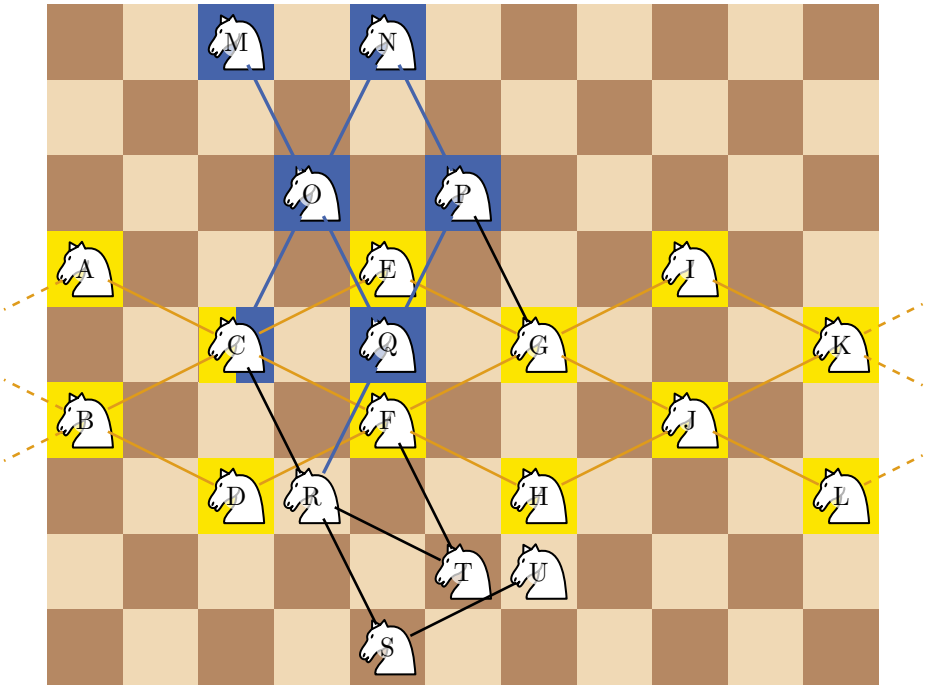


Figure 16 The MAX gadget computes the component-wise maximum of the blue and the yellow signal.

It remains to show that no larger outputs are possible. We give this proof in the full version. Here we state without proof:

▶ **Lemma 23.** *Given inputs* $(v, w) \in \{0,1\}^2$ *and* $(x, y) \in \{0,1\}^2$ *the MAX gadget computes their component-wise maximum* $(\max\{v, x\}, \max\{w, y\}) \in \{0,1\}^2$.

▶ **Corollary 24.** *Restricted to inputs* $(x, 0), (y, 0) \in \{(0,0), (1,0)\}$ *the MAX gadget computes the OR function.*

4.5 The AND Gadget

The AND gadget has two inputs, $(x, 0)$ and $(y, 0)$ and one output, $(x \wedge y, 0)$. Shown in Figure 17, it is composed of multiple smaller gadgets connected by wire. We have already seen the Decrement and the MAX gadget. We now introduce the remaining two building blocks.

Asymmetric Value Production Gadget (AVP). Placed at the top right of the AND gadget lies the AVP gadget, another value production gadget. Recall that the VAR gadget combined a 3-VAL and a Decrement gadget to produce outputs $(1,0) \leftrightarrow (0,0)$ or $(0,0) \leftrightarrow (1,0)$. Similarly, the AVP gadget produces outputs $(1,1) \leftrightarrow (0,0)$ or $(0,0) \leftrightarrow (1,0)$, by combining a 3-VAL and a Zeroing gadget (see Figure 18). This Zeroing gadget sets a component of a signal to 0 regardless of its previous value. Consisting of an Increment followed by a Decrement gadget, its correctness follows directly from the correctness of its parts.

To see the correctness of the AVP gadget, we consider the different possible outputs of its 3-VAL gadget. For 3-VAL outputs $(0,0) \leftrightarrow (1,1)$ the gadget produces an overall output of $(0,0) \leftrightarrow (1,0)$. For 3-VAL outputs $(1,1) \leftrightarrow (0,0)$ or $(1,0) \leftrightarrow (0,1)$ the overall outputs are $(1,1) \leftrightarrow (0,0)$ and $(1,0) \leftrightarrow (0,0)$ respectively. By monotony the second of these is not chosen. Thus, this gadget correctly implements the asymmetric value production.

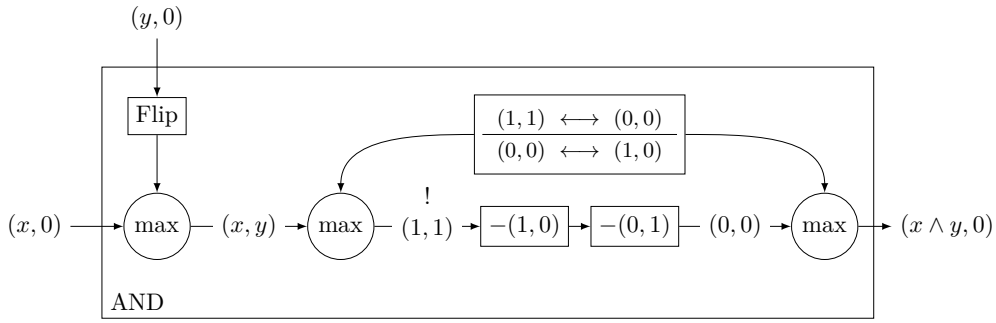


Figure 17 The AND gadget computes the logical conjunction of its two inputs.

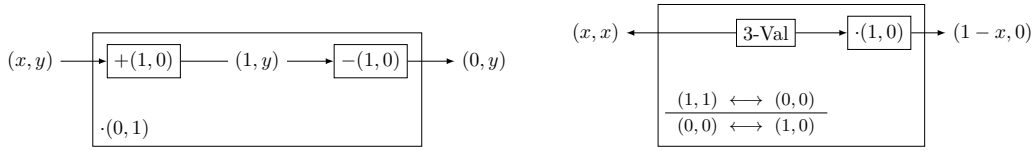


Figure 18 Left: The Zeroing gadget sets a component of a signal to 0. Right: Combining the Zeroing gadget with a 3-VAL gadget yields an AVP gadget.

An analogous version of the gadget swaps the components of the outputs: Placing a $\cdot(0,1)$ gadget to the left of the 3-VAL gadget yields outputs $(0,0) \leftrightarrow (1,1)$ or $(0,1) \leftrightarrow (0,0)$ instead.

Flip Gadget. The top left of the AND gadget houses the Flip gadget, which is unpacked in Figure 19. The Flip gadget computes the function that maps $(x,0) \in \{(0,0),(1,0)\}$ to $(0,x) \in \{(0,0),(0,1)\}$. To show its correctness, we consider the different possible inputs. Observe that the Flip gadget contains a Decrement gadget, enforcing that the output of the MAX gadget to its left has its first component set to 1. Thus, for an input of $(0,0)$ to the Flip gadget, any clearing sequence evaluates the AVP gadget to $(1,1) \leftrightarrow (0,0)$. In this case the signal undergoes the following transformations: $(0,0) \xrightarrow{\max} (1,1) \xrightarrow{-(1,0)} (0,1) \xrightarrow{\cdot(1,0)} (0,0) \xrightarrow{\max} (0,0)$. If the input to the Flip gadget is $(1,0)$, the AVP gadget can evaluate to $(0,0) \leftrightarrow (0,1)$ leading to the following sequence: $(1,0) \xrightarrow{\max} (1,0) \xrightarrow{-(1,0)} (0,0) \xrightarrow{\cdot(1,0)} (0,0) \xrightarrow{\max} (0,1)$. Overall this leads to the desired outputs.

Correctness of the AND Gadget. Having understood its parts, we now turn our attention to the AND gadget itself. Its workings can directly be read off Figure 17. Its inputs $(x,0)$ and $(y,0)$ are combined to (x,y) using a Flip and a MAX gadget. The two Decrements lead to an error, *unless* the signal going into them is a $(1,1)$-signal, in which case they produce

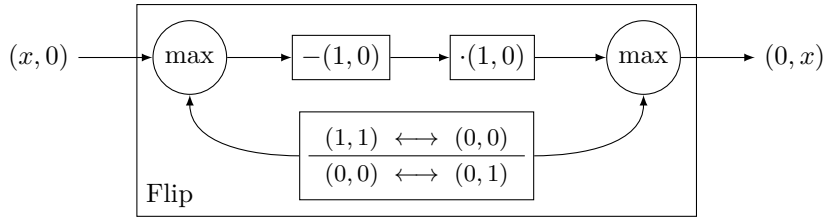


Figure 19 The Flip gadget flips the two components of a signal.

a $(0,0)$-signal. Thus, if $(x,y) \neq (1,1)$, the AVP gadget produces a $(1,1)$-signal to the left, yielding an overall AND gadget output of $(0,0)$. If $(x,y) = (1,1)$, the AVP instead produces a $(0,0)$-signal to the left, yielding an overall output of $(1,0)$. We have seen:

▶ **Lemma 25.** *The AND gadget computes the function that maps inputs* $(x,0),(y,0) \in \{(0,0),(1,0)\}$ *to* $(1,0)$ *if both* x *and* y *are 1, and to* $(0,0)$*, otherwise.*

References

1 N. R. Aravind, Neeldhara Misra, and Harshil Mittal. Chess is hard even for a single player. *Theor. Comput. Sci.*, 1015:114726, 2024. doi:10.1016/j.tcs.2024.114726.

2 N.R. Aravind, Neeldhara Misra, and Harshil Mittal. Chess Is Hard Even for a Single Player. In Pierre Fraigniaud and Yushi Uno, editors, *11th International Conference on Fun with Algorithms*, volume 226 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–5:20, 2022. doi:10.4230/LIPIcs.FUN.2022.5.

3 Davide Bilò, Luca Di Donato, Luciano Gualà, and Stefano Leucci. Uniform-Budget Solo Chess with Only Rooks or Only Knights Is Hard. In Andrei Z. Broder and Tami Tamir, editors, *12th International Conference on Fun with Algorithms*, volume 291 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 4:1–4:19, 2024. doi:10.4230/LIPIcs.FUN.2024.4.

4 Davide Bilò, Luca Di Donato, Luciano Gualà, and Stefano Leucci. Uniform-budget solo chess with only rooks or only knights is hard. *Theor. Comput. Sci.*, 1056:115544, 2025. doi:10.1016/J.TCS.2025.115544.

5 Josh Brunner, Lily Chung, Michael Coulombe, Erik D. Demaine, Timothy Gomez, and Jayson Lynch. Complexity of solo chess with unlimited moves. In Jin Akiyama, Hiro Ito, and Toshinori Sakai, editors, *Discrete and Computational Geometry, Graphs, and Games*, pages 149–174. Springer Nature Switzerland, 2026. doi:10.1007/978-3-032-00281-5_10.

6 Chess.com. Solo chess. URL: https://www.chess.com/solo-chess.

7 Kolja Kühn. Complexity of Solo Chess Variants. Master's thesis, Karlsruhe Institute of Technology, 2024. URL: https://scale.iti.kit.edu/_media/resources/theses/ma_kolja_kuehn.pdf.

8 Craig A. Tovey. A simplified NP-complete satisfiability problem. *Discrete Applied Mathematics*, 8(1):85–89, 1984. doi:10.1016/0166-218X(84)90081-7.