



Complexity of SOLO CHESS Variants

Master's Thesis of

Kolja Kühn

At the Department of Informatics
Institute of Theoretical Informatics (ITI)

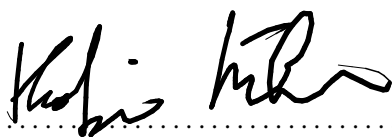
Reviewer: T.T.-Prof. Dr. Thomas Bläsius
Second reviewer: Dr. rer. nat. Torsten Ueckerdt
Advisor: Wendy Yi

2024.05.09 – 2024.11.08

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

I declare that I have developed and written the enclosed thesis completely by myself. I have not used any other than the aids that I have mentioned. I have marked all parts of the thesis that I have included from referenced literature, either in their original wording or paraphrasing their contents. I have followed the by-laws to implement scientific integrity at KIT.

Karlsruhe, 2024.11.08

A handwritten signature in black ink, appearing to read 'Kolja Kühn', written over a horizontal dotted line.

(Kolja Kühn)

Abstract

SOLO CHESS is a single-player variant of the game of chess. The player is given a chess position consisting of only pieces of the same color and is tasked with finding a sequence of captures that clears the board. Every move must be a capture, and the number of captures of each piece must not exceed its budget. In particular, in an instance of B -SOLO CHESS, every piece starts out with a budget of B captures. The game ends when only a single piece remains.

The corresponding decision problem generalizes the game to an unbounded board and asks whether a clearing capture sequence exists. Prior work has shown that 2-SOLO CHESS is NP-complete, even when instances are restricted to contain only rooks, only bishops or only queens, while instances containing only pawns can be decided in linear time. In addition, 11-SOLO CHESS is NP-complete for instances containing only knights.

The main focus of our work is on 2-SOLO CHESS. We show that the problem is NP-complete for instances containing only kings, as well as instances containing only knights (previously only known for budgets $B = 11$). This yields a complete characterization of 2-SOLO CHESS. We also give an alternative proof for the NP-completeness of 2-SOLO CHESS played with only rooks. Finally, we discuss B -SOLO CHESS played on a one-dimensional board. We show that instances containing n pieces can be decided in time $\mathcal{O}(f(B) \cdot n)$, i.e., linear time for any fixed $B \in \mathbb{N}$.

Zusammenfassung

SOLO CHESS ist eine Einzelspieler-Variante des Schachspiels. Spielende erhalten eine Schachposition, die nur aus Figuren derselben Farbe besteht. Ziel des Spiels ist, eine Folge von Zügen zu finden, die das Brett leert. Jeder Zug muss ein Schlagzug sein, und die Anzahl der Schlagzüge jeder Figur darf ihr Budget nicht überschreiten. Insbesondere hat in einer Instanz von B -SOLO CHESS jede Figur anfangs ein Budget von B Schlagzügen. Das Spiel endet, wenn nur noch eine Figur übrig bleibt.

Das entsprechende Entscheidungsproblem verallgemeinert das Spiel auf ein unbeschränktes Brett und fragt, ob eine räumende Zugfolge existiert. Frühere Arbeiten haben gezeigt, dass 2-SOLO CHESS NP-vollständig ist, selbst wenn die Instanzen nur Türme, nur Läufer oder nur Damen enthalten, während Instanzen mit nur Bauern in Linearzeit entschieden werden können. Darüber hinaus ist 11-SOLO CHESS NP-vollständig für Instanzen, die nur Springer enthalten.

Der Schwerpunkt unserer Arbeit liegt auf 2-SOLO CHESS. Wir zeigen, dass das Problem NP-vollständig ist für Instanzen, die nur Könige enthalten, sowie für Instanzen, die nur Springer enthalten (bisher nur für Budgets $B = 11$ bekannt). Dies liefert eine vollständige Charakterisierung von 2-SOLO CHESS. Der Weiteren präsentieren wir einen alternativen Beweis für die NP-Vollständigkeit von 2-SOLO CHESS beschränkt auf Türme. Schließlich betrachten wir B -SOLO CHESS auf einem eindimensionalen Brett. Wir zeigen, dass Instanzen mit n Figuren in Zeit $\mathcal{O}(f(B) \cdot n)$ entschieden werden können, d.h. in Linearzeit für jedes feste $B \in \mathbb{N}$.

Contents

1. Introduction	1
1.1. Contributions & Outline	2
1.2. Related Work	2
2. Preliminaries	5
2.1. Graphs	5
2.2. SAT	6
2.2.1. And-Or-SAT	6
2.2.2. And-Or Embeddings	8
3. Solo Chess	11
3.1. Chess Board and Pieces	11
3.2. Problem Definition	12
3.3. General Notions of SOLO CHESS	13
3.3.1. The Capture Graph	14
4. Special Cases	17
4.1. 1-SOLO CHESS	17
4.2. One-Dimensional SOLO CHESS	18
5. King 2-SOLO CHESS	23
5.1. The Setup	23
5.2. The Wire	25
5.3. Variable Assignment	28
5.4. The Or Gadget	30
5.5. The And Gadget	31
5.6. Wire Crossings	37
5.6.1. Wire Crossing Functions f and g	38
5.6.2. Wire Crossing Function h	42
5.6.3. The Full Wire Crossing Gadget	43
5.7. The 1-Test Gadget	45
5.8. The Final Reduction	45
6. Knight 2-SOLO CHESS	47
6.1. The Wire	47
6.2. Auxiliary Gadgets	52
6.2.1. Signal manipulation gadgets	52
6.2.2. Value Production Gadgets	55
6.2.3. The Maximum Gadget	56
6.2.4. The Flip Gadget	60
6.3. SAT Gadgets	60
6.3.1. The Variable Assignment Gadget	61

6.3.2.	The Or Gadget	61
6.3.3.	The And Gadget	61
6.3.4.	The 1-Test Gadget	62
6.4.	The Final Reduction	63
7.	Rook 2-SOLO CHESS	65
7.1.	The Setup	65
7.1.1.	The Cleanup Phase	67
7.1.2.	Logic Squares	68
7.1.3.	Gadget inputs	69
7.2.	The Variable Assignment Gadget	72
7.3.	The Left Or Gadget	74
7.4.	The And Gadget	76
7.5.	The Right Or Gadget	82
7.6.	The Final Reduction	83
8.	Conclusion	85
	Bibliography	87
A.	Counter Example for the Original Queen 2-SOLO CHESS Proof	89
B.	King SOLO CHESS Wire Crossing	91

1. Introduction

Chess is a two-player turn-based perfect-information board game. It is played on a quadratic board that holds an 8×8 checkered grid of squares. Each player begins with sixteen chess pieces of either white or black color. The players then take turns to move their pieces and capture their opponent's pieces. The goal of the game is to checkmate the opponent's king, i.e., create a threat to capture the opponent's king that cannot be parried.

Since standard chess is played on a constant sized board and has a bounded maximal game length, finding the optimal strategy in a given position is possible in constant time.

Various generalizations of the game have been studied: Storer showed that chess on an $N \times N$ board with the maximum game length bounded by a polynomial is PSPACE-complete [Sto83]. This matches results for some similar two-player board games, including Othello [IK94], Amazons [Hea05], Hex [Rei81] and Gobang [Rei80]. The same holds true for cooperative versions of chess: Brunner, Demaine, Hendrickson, and Wellman show that solving retrograde or helpmate chess problems is PSPACE-complete [BDHW20].

When the requirement for a polynomial bound on the game length is dropped, chess on an $N \times N$ board becomes EXPTIME-complete [FL81]. This in turn is matched by other two-player board games, such as Checkers [Rob84b], Shogi [AKI87], Xiangqi [Zha19], Jiangqi [Zha19] and Go under Japanese rules [Rob83]. Furthermore, within the framework of chess, the selfmate and reflexmate problems are EXPTIME-complete as well [Zha22].

Some board games, such as Arimaa or Chinese rules Go, forbid a player to repeat a position that previously occurred in the game. Under such an added rule, chess and checkers turn out to even be EXPSPACE-complete [Rob84a]. However, for the mentioned games of Arimaa [RS24] and Chinese rules Go [LS78] only PSPACE-hardness results are known, which do not match the EXPSPACE upper bound.

Many single-player games or puzzle games turn out to be NP-complete. This includes classic Windows games such as Freecell [Hel03], Klondike [LM09], Scorpion Solitaire [AD21] and Spider Solitaire [Ste11]. It also extends to generalizations of physical single-player games, such as finding the shortest solution of a 15-puzzle [RW86] and peg solitaire played on an $N \times N$ board [UI90]. The latter result is particularly interesting in the context of this work, since peg solitaire closely resembles the game of SOLO CHESS.

This single-player variant of the game of chess was introduced on the chess website chess.com according to the following rules: The player is given a chess position placed on a standard 8×8 chess board consisting of pieces of the same color. The goal of the game is to perform a sequence of captures so that only a single piece remains. A restriction is added that every move must be a capture, and each piece may only capture twice.

We study the decision problem based on this game. Here, an instance consists of n pieces placed on an $N \times M$ board. Furthermore, each of the n pieces is given a budget denoting how often it is allowed to capture. In particular, we call the original game 2-SOLO CHESS since each piece has a budget of two captures.

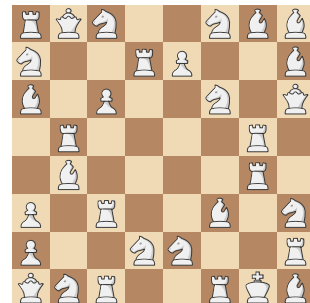


Figure 1.1.: An example position of SOLO CHESS.

It is known that deciding 2-SOLO CHESS is NP-complete, as is deciding 11-SOLO CHESS. More specifically, it is known that 2-SOLO CHESS is already NP-complete even when instances are only allowed to contain rooks (which we call *Rook 2-SOLO CHESS*), bishops or queens, respectively [AMM22], [BDGL24]. Similarly, it is known that Knight 11-SOLO CHESS is NP-complete [BDGL24]. On the other hand, Pawn 2-SOLO CHESS can be decided in linear time [AMM22]. All of these results remain true for the generalized problem of ≤ 2 -SOLO CHESS, where each piece budget is *at most*, rather than exactly 2.

1.1. Contributions & Outline

We begin with the special cases of 1-SOLO CHESS where we extend existing work by giving a linear-time algorithm. We follow this up with one-dimensional SOLO CHESS (played on a chess board with only a single row). For this problem, we review the most general variant, allowing any combination of piece types and any upper bound B on piece budgets. We again extend existing work by giving an algorithm that solves one-dimensional $\leq B$ -SOLO CHESS instances of size n in time $\mathcal{O}(f(B) \cdot n)$ for some computable function f . In particular, this constitutes linear time for any fixed $B \in \mathbb{N}$.

Our main results concern the original 2-SOLO CHESS problem. We prove that King 2-SOLO CHESS, as well as Knight 2-SOLO CHESS are NP-complete. This yields a complete characterization of 2-SOLO CHESS. We also give an alternative proof for the NP-completeness of Rook 2-SOLO CHESS which extends to Bishop 2-SOLO CHESS and Queen 2-SOLO CHESS using known techniques.

For our reductions we introduce a new SAT variant which we call And-Or-(1,1)-SAT. In this variant, clauses consist of up to three literals combined either by up to two disjunctions or a disjunction and a conjunction. Additionally, each of the two literals of a variable is required to occur exactly once. We show that this problem is NP-complete. We then show the hardness results for King, Knight and Rook 2-SOLO CHESS by a reduction from And-Or-(1,1)-SAT. For this we directly place SAT instances on the chess board using gadgets for variable assignments, And and Or gates.

The remainder of this work is structured as follows: In Chapter 2, we define some general notation, as well as introduce the problem of And-Or-(1,1)-SAT. In Chapter 3, we give a formal definition of SOLO CHESS and introduce some related concepts. In Chapter 4, we discuss the special cases of 1-SOLO CHESS, as well as one-dimensional $\leq B$ -SOLO CHESS. In Chapters 5, 6 and 7, we show that King 2-SOLO CHESS, Knight 2-SOLO CHESS and Rook 2-SOLO CHESS are NP-complete, respectively. Finally, in Chapter 8, we summarize our work and give an outlook on remaining open problems.

1.2. Related Work

The complexity of SOLO CHESS was first studied by Aravind, Misra and Mittal who showed that deciding ≤ 2 -SOLO CHESS is NP-complete [AMM22]. To this end, they studied simplified versions of SOLO CHESS where only a single piece type is present. They showed that Rook ≤ 2 -SOLO CHESS is already NP-complete, via a reduction from Red-Blue Dominating Set (a variant of the Dominating Set problem). Furthermore, they gave reductions from Rook SOLO CHESS to Bishop SOLO CHESS and Queen SOLO CHESS, yielding analogous hardness results for Bishop ≤ 2 -SOLO CHESS and Queen ≤ 2 -SOLO CHESS. They also showed that Pawn ≤ 2 -SOLO CHESS can be decided in linear time. Finally, they gave an argument for the NP-completeness

of Queen 2-SOLO CHESS, where each piece starts out with a budget of *exactly* 2. However, while the statement itself holds true, their specific approach does not work. We elaborate on this in Appendix A. The authors also gave a short argument that 1-SOLO CHESS as well as one-dimensional Rook ≤ 2 -SOLO CHESS chess are in P. Finally, while they did not show results concerning Knight SOLO CHESS, they introduced a generalization of the game called *GRAPH CAPTURE*. In this game, pieces are placed on vertices of a graph and capture along edges. The authors show that this game played on either undirected graphs or DAGs is NP-complete for piece budgets of 2 by reductions from Colorful Red-Blue Dominating Set and 3SAT respectively.

Bilò, Di Donato, Gualà and Leucci later showed that Rook 2-SOLO CHESS is NP-complete via a reduction from Vertex Cover [BDGL24]. Thanks to the reductions from Rook SOLO CHESS outlined in [AMM22], this result extends to Bishop 2-SOLO CHESS and Queen 2-SOLO CHESS. They also showed that Knight 11-SOLO CHESS is NP-complete, via a reduction from a variant of the Hamilton path problem. The case of Knight 2-SOLO CHESS, as well as the problem of King 2-SOLO CHESS were left open.

Brunner, Chung, Coulombe, Demaine and Gomez instead studied a related problem where each piece has an unlimited budget [Bru+23]. They generalize the standard chess pieces to discuss arbitrary piece types that are *closed under submoves*. In this setting, they show that the single piece type problem is solvable in polynomial time. They also consider the scenario of a single uncapturable piece of some type S and some second piece type T . Since the single S piece cannot be captured, it remains as the final piece of any solving sequence. They showed that if the piece type T is symmetric and has a superset of legal moves of S , then this problem is solvable in polynomial time. For all remaining chess piece combinations except $S \in \{\text{Pawn, King}\}$, $T = \text{Rook}$ they show NP-completeness, reducing from variants of Hamilton Path and SAT respectively. These hardness results extend to the problem variants considering arbitrary configurations of the mentioned piece types (without any restrictions regarding capturability or number of occurrences of the piece).

2. Preliminaries

We provide some basic definitions and notation for graphs and the satisfiability (SAT) problem, as well as introduce a new variant of the SAT problem.

2.1. Graphs

An undirected graph G consists of a set of vertices V and a set of edges $E \subseteq \binom{V}{2}$. We shorten the notation of an edge $\{u, v\}$ to uv . An edge $uv \in E$ can be seen as a connection between u and v . Given such an edge uv , we say that v is a neighbor of u or v is adjacent to u and vice versa.

A *subgraph* of a graph G contains a subset of vertices and a subset of edges of G . For a graph $G = (V, E)$, we say that a graph $H = (V', E')$ is a subgraph of G , if $V' \subseteq V, E' \subseteq E \cap \binom{V'}{2}$. We call H an *induced* subgraph, if it retains all edges of G between its vertices, i.e., for every $u, v \in V'$ we have $uv \in E' \iff uv \in E$. We write $H \subseteq G$ or $H \subseteq_{\text{ind}} G$, respectively. We also say that G *contains* an (induced) H . Any subset of vertices $\tilde{V} = \{v_1, v_2, \dots, v_k\} \subseteq V$ uniquely defines an induced subgraph $G[\tilde{V}] := G[v_1, v_2, \dots, v_k] := (\tilde{V}, \tilde{E})$ with $\tilde{E} = E \cap \binom{\tilde{V}}{2}$.

We now define some common types of graphs that we need in the remainder of this work: A *path* is a graph with vertices $\{v_1, \dots, v_n\}$ and edges $\{v_i v_{i+1} \mid 1 \leq i \leq n-1\}$, i.e., E (only) contains a sequence of edges from v_1 to v_n . We denote a path by the sequence of its vertices, in this case (v_1, v_2, \dots, v_n) . We denote the length of a path as the number of edges it contains. In the above example, the path (v_1, v_2, \dots, v_n) has length $n-1$.

We call a graph $G = (V, E)$ *connected* if any two vertices $u, v \in V$ are connected by a path, i.e., if G contains as a subgraph a path containing both u and v . A *connected component* of a graph G is a (vertex-)maximal induced subgraph of G that is connected. In particular, every connected graph has exactly one connected component.

Let $G = (V, E)$ be a connected graph. We call a vertex $v \in V$ a *cut vertex* if removing it disconnects the graph, i.e., if $G[V \setminus \{v\}]$ has more than one connected component. Analogously, we call an edge $e \in E$ a *cut edge* or *bridge* if removing it disconnects the graph, i.e., if $G' = (V, E \setminus \{e\})$ has more than one connected component.

Antennae We define a non-standard type of subgraph that we use throughout our constructions, the *antenna*. An antenna consists of a sequence of vertices that form an induced path, such that only the final vertex of the sequence may be adjacent to vertices outside the sequence. More formally, for a graph $G = (V, E)$ we call vertices $\{v_1, \dots, v_k\}$ an antenna (of length $k-1$) if $G[v_1, \dots, v_k]$ is an induced path and only the final vertex of the path, v_k , is connected to vertices outside the path. Note that a path has no preferred direction. By the additional antenna condition, it may be the case that (v_1, \dots, v_k) forms an antenna while $(v_k, v_{k-1}, \dots, v_1)$ does not. The induced path in the graph shown in Figure 2.1 is an antenna of length 3, since only the right outer vertex is adjacent to vertices outside the path. We call the left-most vertex of such an antenna the “first” and the right-most vertex the “last” vertex of the antenna.

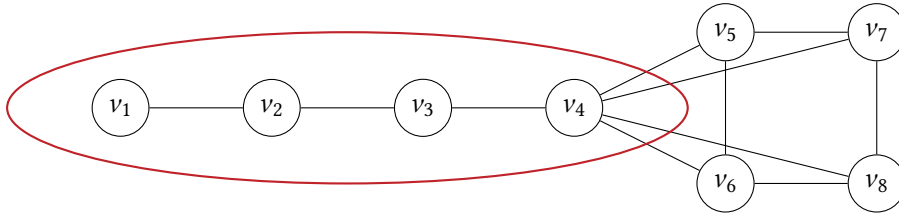


Figure 2.1.: The circled vertices form an antenna of length three, with first vertex v_1 and last vertex v_4 .

We observe that each edge of an antenna is a cut edge, since removing it disconnects the first from the last vertex of the antenna (this uses the fact that only the last vertex may be connected to vertices outside the antenna). For the same reason, every interior vertex of an antenna (i.e., all but the first and last vertex) is a cut vertex. Furthermore, if the graph G contains some further vertices outside an antenna, the antenna's last vertex is a cut vertex as well.

2.2. SAT

For our NP-hardness proofs, we reduce from a variant of the well-known SAT problem. SAT is the original NP-complete problem [Coo71]. It asks whether a given boolean formula has a satisfying assignment. The most common formulation is that of CNFSAT. In CNFSAT an instance consists of a set of *variables* U and a set of *clauses* C . A *literal* is a variable or its negation, for example for a variable $x \in U$ there is the positive literal x and the negative literal $\neg x$. A clause is the disjunction of a set of literals, for example $(x \vee \neg y \vee z)$. Finally, a formula in CNF is a conjunction of a set of clauses, for example $(x \vee \neg y \vee z) \wedge (\neg x \vee \neg y \vee \neg a) \wedge (y \vee z \vee a)$. The CNFSAT problem asks, given an instance $I = (U, C)$, whether there exists an assignment of truth values to the variables in U such that the conjunction over the clauses of C is satisfied, or equivalently, that every clause of C is satisfied.

A well known variant of CNFSAT is 3SAT, which requires each clause to contain (depending on the exact definition) either at most or exactly three literals. Either version of this problem is NP-complete as well, and we use it to show the NP-hardness of two further SAT variants.

2.2.1. And-Or-SAT

The first variant is based on a construction by Tovey [Tov84].

We define the $(2+1),3$ -SAT problem as a special case of the CNFSAT problem. A CNFSAT instance is a $(2+1),3$ -SAT instance if it contains at most three literals per clause and if each variable appears positive exactly twice and negative exactly once in the set of clauses. The decision problem asks whether a satisfying variable assignment exists. This problem is similar to the 3-SAT problem, however, it differs by the additional restrictions on the number of occurrences of the literals of each variable.

Lemma 2.1: *Deciding $(2+1),3$ -SAT is NP-hard.*

Proof. The first step of the proof is exactly analogous to the proof of Theorem 2.1 in [Tov84]. We reduce from 3-SAT. Let I be a 3-SAT instance. Let n_x be the number of occurrences of a variable x . Introduce n_x copies of the variable x , x_1 through x_{n_x} , and replace the i th occurrence of x with x_i . To ensure that all x_i are assigned the same truth value, we add clauses $x_i \vee \neg x_{i+1}$

for all $i = 1, \dots, n-1$, and $x_n \vee \neg x_1$. We call the resulting instance I' . In I' , each variable appears once in the original set of clauses (either positive or negative) and twice in the additional clauses (once positive and once negative). In a second step, for each variable that appears negative in the original set of clauses, we negate all its occurrences. The resulting instance I'' is a valid (2+1),3-SAT instance. Below is an example of the construction:

$C(I)$	$C(I')$	$C(I'')$
$(x \vee \neg y \vee z)$	$(x_1 \vee \neg y_1 \vee z_1)$	$(x_1 \vee y_1 \vee z_1)$
$(\neg x \vee \neg y \vee \neg a)$	$(\neg x_2 \vee \neg y_2 \vee \neg a_1)$	$(x_2 \vee y_2 \vee a_1)$
	$(x_1 \vee \neg x_2)$	$(x_1 \vee x_2)$
	$(x_2 \vee \neg x_1)$	$(\neg x_2 \vee \neg x_1)$
	$(y_1 \vee \neg y_2)$	$(\neg y_1 \vee y_2)$
	$(y_2 \vee \neg y_1)$	$(\neg y_2 \vee y_1)$
	$(z_1 \vee \neg z_1)$	$(z_1 \vee \neg z_1)$
	$(a_1 \vee \neg a_1)$	$(\neg a_1 \vee a_1)$

We show that I and I' are equivalent regarding satisfiability: Let ϕ' be a satisfying assignment for I' . Observe that if some x_1 is set to false, then by the first added clause, x_2 must be set to false as well and so forth. Similarly, if x_1 is set to true, the final clause forces x_n to be set to true and so forth. Thus, the cyclic structure of the added clauses ensures that $\forall i, j : \phi'(x_i) = \phi'(x_j)$. This allows us to define $\phi : x \mapsto \phi'(x_1) (= \dots \phi'(x_n))$ which satisfies all the original clauses and thus is a satisfying assignment for I .

Now let ϕ be a satisfying assignment for I . Then $\phi' : x_i \mapsto \phi(x)$ is a satisfying assignment for I' : Each of the original clauses is satisfied by construction, and each of the added clauses has exactly one true and one false literal. This shows that I and I' are equivalent.

Next, we show that I' and I'' are equivalent as well: Let ϕ' be a satisfying assignment for I' . Then ϕ'' defined as

$$x_i \mapsto \begin{cases} \phi'(x_i) & \text{if the } x_i \text{ literal in the original clause appeared positive,} \\ \neg \phi'(x_i) & \text{if the } x_i \text{ literal in the original clause appeared negative.} \end{cases}$$

is a satisfying assignment for I'' . The reverse direction works analogously. Thus, we have shown that I and I'' are equivalent and so the NP-hardness of (2+1),3-SAT follows. ■

Next, we define a new variant on the SAT problem, *And-Or-(1,1)-SAT*, which is loosely similar to 3-SAT but restricts the number of variable occurrences even further: An instance I of And-Or-(1,1)-SAT consists of a set of variables U and a set of clauses C . A clause can be a disjunction of up to three literals, i.e., of the form $l_1 \vee l_2 \vee l_3$ for literals l_1, l_2, l_3 . Additionally, a clause may also be of the form $(l_1 \wedge l_2) \vee l_3$. In the set of clauses, each literal appears exactly once, i.e., each variable appears once positive and once negative.

Theorem 2.2: *And-Or-(1,1)-SAT is NP-complete.*

Proof. We reduce from (2+1),3-SAT. Let I be an instance of (2+1),3-SAT. For each variable x , we introduce new variables x_1^+, x_2^+, x^- . We replace the two positive occurrences of x with x_1^+ and x_2^+ respectively, and the negative x literal with x^- . To ensure that x_1^+ and x^- are not set to true at the same time, we add an additional clause $c = (\neg x_1^+ \wedge \neg x_2^+) \vee \neg x^-$. In the resulting instance I' , each literal appears exactly once. In particular, the negative literal of each variable appears in some And-Or-clause, while the positive literal of each variable appears in one of the original disjunctive clauses. Once again, we give an example:

I	I'
$(x \vee \neg y \vee z)$	$(x_1^+ \vee y^- \vee z_1^+)$
$(\neg x \vee y)$	$(x^- \vee y_1^+)$
$(x \vee \neg z)$	$(x_2^+ \vee z^-)$
$(y \vee z)$	$(y_2^+ \vee z_2^+)$
	$((\neg x_1^+ \wedge \neg x_2^+) \vee \neg x^-)$
	$((\neg y_1^+ \wedge \neg y_2^+) \vee \neg y^-)$
	$((\neg z_1^+ \wedge \neg z_2^+) \vee \neg z^-)$

It remains to show that I' is satisfiable if and only if I is. Let ϕ be a satisfying assignment for I . Then for I' we set $\phi'(x_1^+) = \phi'(x_2^+) := \phi(x)$, $\phi'(x^-) := \neg\phi(x)$. By construction, each literal in the original clauses is assigned the same truth value under ϕ' as under ϕ . Thus, each of the original clauses is satisfied. The added And-Or clauses are satisfied too: If $\phi(x) = \text{true} = \phi'(\neg x^-)$, then the final operand of the And-Or clause is true. If $\phi(x) = \text{false} = \phi'(x_1^+) = \phi'(x_2^+)$, then both And operands are true and thus the And evaluates to true. In either case, the And-Or clause is satisfied.

Let ϕ' be a satisfying assignment for I' . We define $\phi(x) := \neg\phi'(x^-)$. Consider some clause c_k in I . This clause has a corresponding clause c'_k in I' which under ϕ' is satisfied by some literal x_j^+ (case (1)), or some literal x^- (case (2)). The relevant And-Or-clause $c' = (\neg x_1^+ \wedge \neg x_2^+) \vee \neg x^-$ is satisfied by ϕ' as well.

Case (1): By the definition of our transformation, c_k contains the positive literal x in this case. We know that $\phi'(x_j^+) = \text{true}$ since it satisfies c'_k . Thus, in c' the And operation evaluates to false. Since c' is satisfied by ϕ' , it follows that $\phi'(x^-)$ is set to false. Therefore, our definition sets $\phi(x) := \neg\phi'(x^-) = \text{true}$ and so c_k is satisfied.

Case (2): By the definition of our transformation, c_k contains the negative literal $\neg x$ in this case. We know that $\phi'(x^-) = \text{true}$ since it satisfies c'_k . Our definition of ϕ yields $\phi(x) := \neg\phi'(x^-) = \text{false}$. Thus, the literal $\neg x$ is set to true and c_k is satisfied. This shows that I' is satisfiable if and only if I is, which shows the NP-hardness of And-Or-(1,1)-SAT. ■

This constitutes an increase in difficulty compared to regular CNFSAT: Tovey showed that CNFSAT is in P when restricted to instances, in which each variable appears at most twice [Tov84].

2.2.2. And-Or Embeddings

In our construction, we place SAT instances on the chess board. To this end, we describe specific *And-Or Embeddings* of our SAT instances. Such an embedding is a drawing of the instance in the plane, similar to a logic circuit. Figure 2.2 shows an example drawing of the And-Or-(1,1)-SAT instance $I = (U, C) = (\{x, y, z\}, \{(x \wedge \neg y) \vee z, (\neg x \vee \neg z) \vee y\})$. We give a high level description of such an And-Or Embedding:

- 1 For each clause, its (up to) two logic gates are placed next to each other, as shown in the figure.
- 2 Clauses are placed in a single column with sufficient vertical distance between different clauses.
- 3 Variables are placed to the left of the clauses at sufficient horizontal distance to each other. Each variable is positioned vertically between the two clauses in which its two literals are used, aligning the positive and negative literal appropriately. (we assume without loss of generality that the two literals are not used in the same clause)

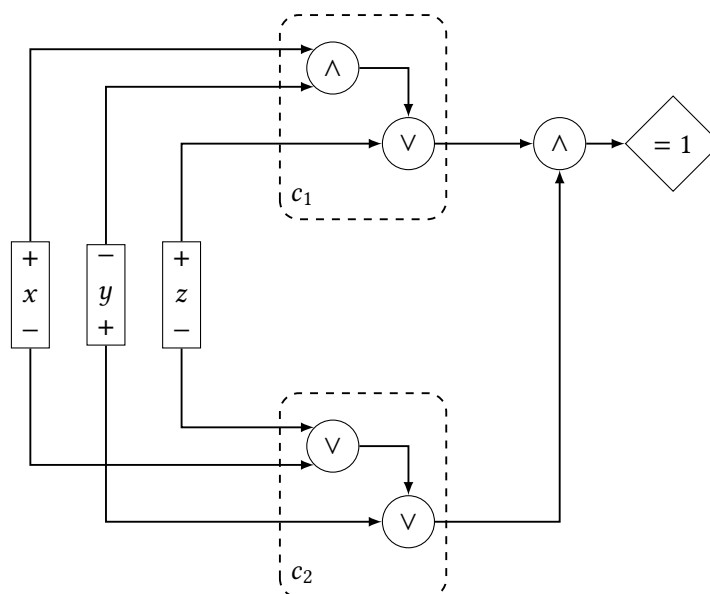


Figure 2.2.: An And-Or Embedding of the SAT instance $I = (U, C)$ with $U = \{x, y, z\}$ and $C = \{c_1, c_2\} = \{(x \wedge \neg y) \vee z, (\neg x \vee \neg z) \vee y\}$.

- 4 Outputs of variables and logic gates are connected to the inputs of the next logic gate, as specified by the SAT formula. Each of those connections consists of one or two axis-aligned line segments.
- 5 Optionally: The outputs of each clause are combined pairwise through further And gates, computing the value of the full SAT formula.

We note that such an And-Or Embedding can be created in polynomial time.

3. Solo Chess







We begin with describing the rules of SOLO CHESS.

3.1. Chess Board and Pieces

SOLO CHESS is played on a rectangular grid of squares. Usually the size of the grid is eight by eight squares, however, in this work we use arbitrary sized rectangles with N rows and M columns, $N, M \in \mathbb{N}$. We name the squares by their coordinates, i.e., we name the square in the top left corner $(1, 1)$ and the square in the bottom right corner (N, M) , i.e., row N and column M . Each square either contains one chess piece or is empty.

SOLO CHESS is a turn-based game. Every turn, the player makes a move according to the chess rules. Such a move consists of moving one piece P_1 from its current square z_1 to some other square z_2 . We denote this move as $(z_1 \rightarrow z_2)$. We may denote consecutive moves $(z_1 \rightarrow z_2), (z_2 \rightarrow z_3)$ as $(z_1 \rightarrow z_2 \rightarrow z_3)$. If the square z_2 was already occupied by some piece P_2 , this piece P_2 is removed from play, and we call the move a *capture*.

Each of the piece types permit a different way of moving. Broadly speaking, there are two groups of piece types:

- Long-range pieces which have unlimited range, namely the rook , the bishop  and the queen .
- Short-range pieces which have a constant range, namely the king , the knight  and the pawn .

The movement of all pieces except the pawn is symmetrical. This means that if a (non-pawn) piece can move from z_1 to z_2 it can also move from z_2 to z_1 . For simplicity, we define the movement of those pieces in one direction, the reverse moves are valid as well.

The *rook* moves arbitrarily far in the horizontal or vertical direction. It cannot jump over other pieces. More formally, a rook on square (i, j) may perform the move $((i, j) \rightarrow (i, k))$ if all squares $(i, j + 1), (i, j + 2), \dots, (i, k - 1)$ are empty. Analogously, the rook may perform the move $((i, j) \rightarrow (k, j))$ if all squares $(i + 1, j), (i + 2, j), \dots, (k - 1, j)$ are empty.

The *bishop* moves arbitrarily far diagonally. It too cannot jump over other pieces. More formally, a bishop on square (i, j) may perform the move $((i, j) \rightarrow (i + k, j + k))$ or $((i, j) \rightarrow (i + k, j - k))$ if for all $1 \leq \ell < k$ the squares $(i + \ell, j + \ell)$ or $(i + \ell, j - \ell)$ respectively, are empty.

The *queen* combines the movement of the rook and the bishop, i.e., it can move arbitrarily far in horizontal, vertical or diagonal direction. Like the other ranged pieces, it cannot jump over other pieces.

The *king* can move to any adjacent square, including diagonally adjacent squares. Due to its limited range, the problem of jumping over other pieces cannot arise. As a result, the king always has exactly eight legal moves, unless it is placed on the edge of the board, in which case some of these moves are not possible.

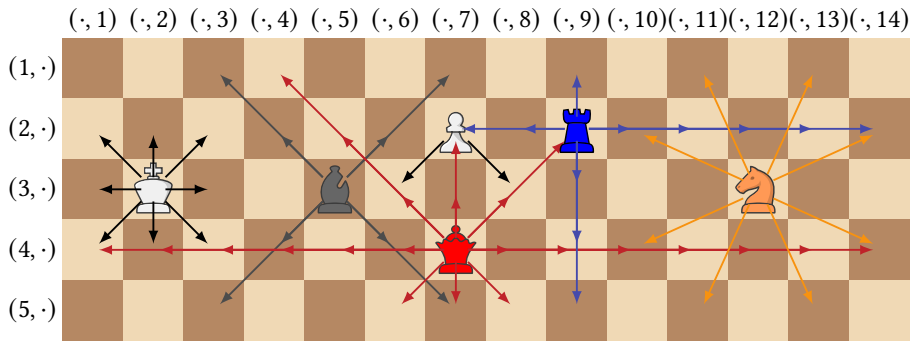


Figure 3.1.: A visualization of the movement of the chess pieces. The different piece colors are used for clarity.

The *pawn* is the only asymmetric piece, as well as the only piece that moves differently when capturing as compared to when not capturing. When capturing, the pawn moves one square forward diagonally. Thus, a pawn on square (i, j) may capture a piece standing on $(i + 1, j + 1)$ or $(i + 1, j - 1)$ but *not* one standing on $(i - 1, j + 1)$ or $(i - 1, j - 1)$. We don't need non-capturing moves and, thus, omit them here.

Finally, the *knight* has the most unusual movement. Its movement may be described as all squares a distance of two from the knight, which the queen cannot move to. More specifically, a knight on square (i, j) may move to squares $(i - 2, j + 1)$, $(i - 1, j + 2)$, $(i + 1, j + 2)$, $(i + 2, j + 1)$ as well as the reversed moves. It is the only piece that can jump over other pieces, meaning that unless it is near the edge of the board, the knight always has exactly eight legal moves.

Figure 3.1 visualizes the movement of the chess pieces. The following statement follows directly from the definition of the piece movements:

Lemma 3.1: *A piece placed on some row $j \neq i$ has at most three possible moves that end in row i for any $i \in \{1, \dots, n\}$. Analogous statements hold for columns and diagonals.*

Proof. We show the statement for the row case, the other cases are similar. We also assume that the edge of the board does not interfere with the piece's movements, otherwise the number of possible moves may be lower but never higher. We make a case distinction by the piece in question.

A knight can only move to row i if it is placed on rows $i - 2$, $i - 1$, $i + 1$ or $i + 2$. In all of these cases, it has exactly two moves that end in row i .

A queen not placed on row i always has exactly three moves ending in row i . For a queen placed on square (r, c) , those moves are $((r, c) \rightarrow (i, c))$, $((r, c) \rightarrow (i, c - r + i))$, $((r, c) \rightarrow (i, c + r - i))$.

The possible moves for pawns, kings, bishops and rooks are strict subsets of the possible moves for a queen. ■

3.2. Problem Definition

We define a decision problem for the game of SOLO CHESS. A configuration C consists of a set of pieces \mathcal{P} . Each piece $P \in \mathcal{P}$ has a piece type $t \in \{\text{King, Queen, Rook, Bishop, Knight, Pawn}\}$, a square $z \in \mathbb{N}^2$ and a remaining budget $c \in \mathbb{N}_0$. In the initial configuration of an instance of B -SOLO CHESS, every piece has the same budget B , for a problem parameter $B \in \mathbb{N}$. In the problem variant $\leq B$ -SOLO CHESS, each piece instead has an individual budget of at most B .

In SOLO CHESS, only moves that are also captures are allowed. A piece may move so long as its budget is greater than 0. Doing so reduces its budget by 1. The decision problem asks, given some instance, whether there exists a sequence of captures that reduces its configuration to a single piece.

In the next chapter we discuss different values for B . In all following chapters we consider the special case of $B = 2$. If a configuration contains only pieces of the same piece type, we call it *uniform*. This allows us to define a family of special cases for the SOLO CHESS problem, namely those that contain only uniform instances. For example, we define the Rook SOLO CHESS problem as the SOLO CHESS problem on uniform instances that contain only rooks.

Theorem 3.2: *SOLO CHESS is in NP.*

Proof. We describe how the problem can be solved in polynomial time by an oracle TM:

Guess a solving capture sequence. Note that the encoding of any instance with n pieces needs space $\Omega(n)$. At the same time, a solving capture sequence contains exactly $n - 1$ captures since each capture reduces the number of pieces by one and any instance needs to be reduced from n pieces down to one piece. Thus, encoding each capture by the coordinates of its origin and destination square yields a polynomially large encoding. Note that we assume without loss of generality that the size of the board is at most exponential in n , thus, each square coordinate can be encoded as a binary number of size $\mathcal{O}(n)$.

Furthermore, it is possible to test a capture sequence for validity in polynomial time: The validity of a single move can be tested by checking whether the origin and destination square differ by an amount permitted by the move rules of the respective piece. If the piece was a ranged piece, it also needs to be verified that the position of none of the other pieces interferes with the capture. This can be done by a linear scan over the set of pieces. To test the validity of the full sequence, validate and perform each move in order. This shows that the problem can be solved in polynomial time by an oracle TM, and as a result is in NP. ■

This statement extends to all the special cases that are discussed in this work.

3.3. General Notions of SOLO CHESS

In this section we describe a number of general notions that prove useful in the upcoming proofs.

Any solving capture sequence reduces a configuration to a single piece. We call this piece the *final piece* and its square the *final square*. Note that different solving capture sequences may have different final pieces / squares.

SOLO CHESS configurations adhere to a form of *monotony*: Given configurations C_1 and C_2 we say that $C_2 \geq C_1$ if they contain the same pieces on the same squares and if the budget of each piece in C_2 is greater or equal to that of the corresponding piece in C_1 . We observe that if $C_2 \geq C_1$ and if C_1 has a solution, i.e., a sequence of captures that clears it, then C_2 can be solved by that same sequence of captures.

We call a piece in a configuration *stranded*, if there is no sequence of captures, such that the piece leaves its square. For example, if a piece in a uniform configuration is not adjacent (according to the piece type's movement) to any other piece, it is stranded. We call a group of pieces stranded, if there is no sequence of captures such that every piece of the group leaves its square. For example, if two pieces in a uniform configuration are adjacent to each other but not to any other pieces, they are stranded. While either of them could make a capture and leave its square, it is not possible for both of them to leave their square.

One typical example of a stranded piece involves one with a budget of 0 (a *0-piece*). Since its budget is not greater than 0, it cannot directly perform a capture. Thus, any capture sequence clearing its square z_2 contains a pair of moves $(z_1 \rightarrow z_2 \rightarrow z_3)$ for some squares z_1, z_2, z_3 . In this case, squares z_1 and z_3 are not equal, since the first move vacates the square z_1 and so z_1 cannot be the destination of any future capture. This yields the following observation:

Observation 3.3: *Let the problem parameter B be at most 2 and let square z hold a 0-piece P . Then P is stranded, unless there exists a 2-piece P' , which*

- (a) *can move to z on an otherwise empty board and*
- (b) *can move from z to a square z' holding another piece P'' , on otherwise empty board.*

3.3.1. The Capture Graph

In some cases it can be useful to take a graph theory view on SOLO CHESS. To that end we define the concept of a *capture graph*.

Definition 3.4 (The capture graph): *For any uniform SOLO CHESS configuration that does not contain pawns we define its capture graph $G = (V, E)$. Its vertex set V is the set of all pieces of the configuration. Vertices u and v are connected by an edge $uv \in E$ if and only if piece u can capture piece v on an otherwise empty board.*

This capture graph contains the structure of an instance without depending on the exact placement on the board. Note that we only define the capture graph for uniform configurations, i.e., those that only contain one piece type excluding pawns. Due to the symmetry of the remaining piece types, if a piece u can capture a piece v , the converse holds as well, justifying the use of an undirected edge.

Note that for ranged pieces, not every sequence of captures on the capture graph can be translated into a sequence of captures on the normal chess board since ranged pieces cannot jump over other pieces. For example, if k rooks are placed in a row or column, their k vertices form a clique in the capture graph but only those rooks that are neighbors on the chess board can directly capture each other. However, any sequence of captures on the chess board can be translated to a sequence of captures in the capture graph. This implies in particular that if there exists no sequence of captures to clear the capture graph of a configuration then there also exists none on the chess board.

Capture graphs allow us to conveniently discuss some properties of (partial) configurations. If the capture graph of a configuration has more than one connected component, then for any capture sequence at least one piece in each component remains. It follows that the configuration cannot be solved. Consider now a configuration whose capture graph is connected, but has a cut vertex v . Then any capture by v splits the capture graph into multiple connected components, yielding an unsolvable configuration. It follows that any solving sequence clears all but one of these components before making a capture with v . The following lemma uses this insight in the scenario of an antenna whose length is equal to the budget c of its first vertex. An important special case to consider is that of a first vertex that has not yet moved and, thus, still has a “full” budget of B .

Lemma 3.5 (*B-Antennae*): *Let the capture graph of a configuration contain an antenna whose length is equal to the budget c of its first vertex. Then, in any solving capture sequence, the antenna is simplified to a 0-piece on its last square, unless the antenna contains the final square of the configuration.*

Proof. We show the claim by induction. If $c = 0$, the antenna consists only of a first vertex with a budget of 0. This vertex is already on the last square of the antenna, and so the claim holds.

Consider now the case of an arbitrary c . Let the final square of the configuration not be contained in the antenna for the given solving sequence. Then all $c + 1$ antenna squares are emptied by the sequence. However, all but the first vertex of the antenna are cut vertices. Thus, the only capture that does not disconnect the capture graph is from the first to the second vertex of the antenna. This creates an antenna of length $c - 1$ with a first vertex with a budget of $c - 1$. The claim follows by induction. ■

We now discuss the scenario of an antenna whose length is larger than the budget of its first vertex. One special case is an antenna of length 1 with a first vertex which has a budget of 0. Phrased differently, this scenario considers the case of a leaf in the capture graph with a budget of 0. Since a leaf has only one neighbor, by Observation 3.3 any *0-leaf* is stranded. We generalize this in the following lemma:

Lemma 3.6 (*B + 1-Antennae*): *Let the capture graph of a configuration contain an antenna whose length is greater than the budget c of its first vertex. Then the set of antenna vertices is stranded.*

Proof. Assume that the claim does not hold and that there exists a solving sequence with a final square outside of the antenna. Consider the first vertex of the antenna, and the c next vertices along the antenna. These $c + 1$ vertices form a c -antenna themselves, with a first vertex with budget c . Thus, by Lemma 3.5 this smaller antenna is simplified to a 0-piece on its last square. For the full antenna this translates to a 0-leaf as the first piece of an antenna of length greater than 0. As observed above, this 0-leaf is stranded, contradicting the assumption. ■

From the statement of the lemma it immediately follows:

Corollary 3.7: *Let the capture graph of a configuration contain an antenna whose length is greater than the budget c of its first vertex. Then, for any solving sequence, the antenna contains the final square of the configuration.*

As an example, consider again Figure 2.1 which shows an antenna of length 3. If each piece budget is equal to 2, for example in the initial configuration of an instance of 2-SOLO CHESS, then the set of antenna vertices is stranded. However, there exists a solving sequence, using antenna-vertex v_3 as the location for the final square of the configuration. It is reachable from v_1 in two captures. Also, all v_5 through v_8 can reach v_4 in one capture, allowing one more capture ($v_4 \rightarrow v_3$). This solves the instance, with the final square being contained in the antenna.

4. Special Cases

In this chapter, we review two special cases of SOLO CHESS that can be solved in linear time.

4.1. 1-SOLO CHESS

We begin with a simple version of SOLO CHESS. Recall that the problem parameter B of B -SOLO CHESS determines the initial budget of each piece. The smallest possible value of B is 1. Aravind, Misra and Mittal note that in this case the problem turns out to be easy [AMM22]. We expand on this by giving an algorithm that decides the problem in linear time.

Theorem 4.1: *B -SOLO CHESS with problem parameter $B = 1$ can be decided in linear time.*

Proof. Since every piece budget is 1, each piece may only capture once. Thus, after some piece P performs a capture ($z_1 \rightarrow z_2$), its budget becomes 0. By Observations 3.3 it follows immediately that P is stranded and z_2 can never be emptied since no (adjacent) 2-piece exists. Thus, z_2 already needs to be the final square of the configuration. Reducing such a configuration to a single piece is possible if and only if there exists a non-empty square that each other piece could legally move to on an otherwise empty board. Thus, it suffices to decide whether such a *center square* exists.

If we are given a correct center square, validating that it is indeed a valid center square can be done in linear time: Iterate over the list of pieces to check that there is a piece placed on the center square and that each other piece can move to the center square on an otherwise empty board (i.e., it is not necessary to consider possible interference with any of the other pieces).

It remains to find such a correct center square. Testing every square that holds a piece with the above method would necessitate checking n different squares and would lead to an $\Theta(n^2)$ algorithm. However, we achieve a $\Theta(n)$ running time by only checking a constant number of squares. We claim that there is always a set of at most 12 squares that can be found in linear time and that contains a center square if there exists one:

If the configuration contains a short-range piece (knight, king or pawn), then that piece has at most eight legal moves. Then any valid center squares is one of those eight destination squares or the square of the short-range piece itself. We find such a piece by a linear scan over the set of pieces, yielding a linear runtime as claimed.

The other case is when the configuration contains only long-range pieces. If every piece is in the same row or column, the instance is solvable if and only if there is at most one bishop (whose square would serve as the center square). Similarly, if every piece is in the same diagonal, the instance is solvable if and only if there is at most one rook. This can be decided with another linear scan.

Otherwise, we pick an arbitrary piece p_1 . Since not every piece is in the same row, column or diagonal, there exist pieces p_2 through p_5 not in the same row, column or diagonal as p_1 , respectively. Note that it is possible for some or all of the p_2, \dots, p_5 to be the same piece. Since the center square can be at most one capture away from p_1 , it must be either in the

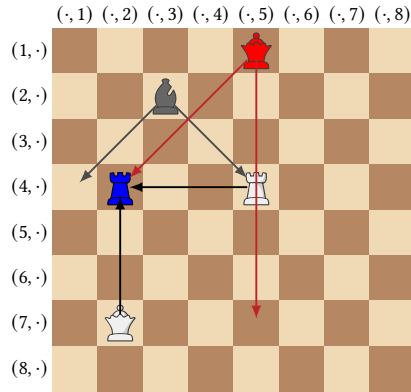


Figure 4.1.: Example configuration where the blue rook represents p_1 , the bishop p_2 , the white rook p_3 , the red queen p_4 and the white queen p_5 . Move arrows point towards the resulting candidate squares. Of the four candidate squares, $(4, 5)$ is a valid center square.

same row, the same column or the same diagonal as p_1 . If it is in the same row as p_1 , then by assumption, p_2 is not in the same row as the center square and, by Lemma 3.1, can move to at most three squares of that row. Thus, any center square in the same row as p_1 is one of those three squares. Analogous arguments for the column and diagonals yield a set of at most $4 \cdot 3 = 12$ candidate center squares. Thus, with another linear scan, we find a set of at most 12 squares to check. Figure 4.1 shows an example configuration. Overall, we conclude that 1-SOLO CHESS can be decided in linear time. ■

4.2. One-Dimensional SOLO CHESS

Another problem variant that can be decided efficiently is $\leq B$ -SOLO CHESS played on a one-dimensional board, i.e., a board containing only a single row. This section works towards proving the following claim:

Theorem 4.2: *We can decide any one-dimensional instance of SOLO CHESS, that has n pieces, each of which have a budget of up to B , in time $\mathcal{O}(f(B) \cdot n)$ for some computable function f . This constitutes linear time for any fixed B .*

We first discuss the special case in which only rooks are present. The following lemma is due to Aravind, Misra and Mittal [AMM22]:

Lemma 4.3 (1D Rook Lemma): *Any one-dimensional SOLO CHESS configuration that contains n rooks and no other pieces is solvable if and only if their budgets add up to at least $n - 1$. It has a solution with the final piece being the right-most or the left-most rook if and only if the budgets of the remaining rooks add up to at least $n - 1$.*

Proof. We discuss the first claim. First, observe that the condition is necessary: To reduce a configuration containing n pieces down to one containing only 1 piece, $n - 1$ captures are necessary. We show by induction that the condition is also sufficient:

For the base case we note that any configuration with one rook remaining is already solved.

Assume that the claim holds for any configuration with n rooks. Consider some configuration C with $n + 1$ rooks and a combined budget of at least n . We distinguish between two cases: If the configuration contains no 0-rook, then it is solvable by any sequence of captures all capturing towards the same (final) square. Thus, we now assume that there exists a 0-rook.

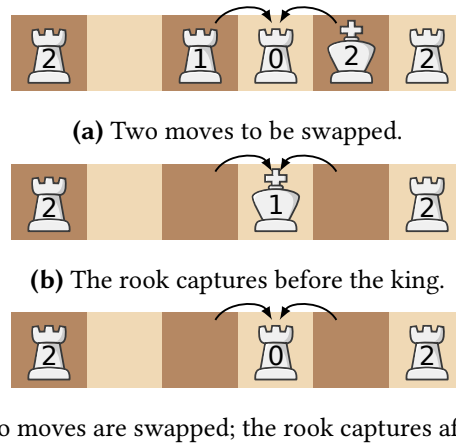


Figure 4.2.: Example position of one-dimensional SOLO CHESS

Since $n \geq 1$, the combined budget of our configuration is greater than 0. Therefore, there exists a rook with a budget that is greater than 0. With some number of 0-rooks and some number of rooks having a budget greater than 0, there exists some i -rook ($i > 0$) adjacent to some 0-rook. Then, capturing the 0-rook with the i -rook reduces the number of rooks as well as the combined budget by 1. By the induction hypothesis, this resulting configuration has a solution. Thus, the claim follows.

The other two claims can be shown by an analogous induction proof, which we omit here. ■

This lemma proves useful when discussing partial configurations on two-dimensional boards in the upcoming chapters. It also provides the basis for studying general one-dimensional SOLO CHESS. Thanks to the lemma we can see that one-dimensional Rook SOLO CHESS can be decided in linear time: It suffices to sum up n budget values and test that the result is at least $n - 1$.

We now consider scenarios with multiple different piece types, starting with kings and rooks. We first show that we can bring any solving sequence of such a configuration into a normal form which starts out with all the king moves:

Lemma 4.4 (1D Normal Form): *Let C be a one-dimensional SOLO CHESS configuration consisting of only kings and rooks. If there exists a solving capture sequence for C , then there exists one which consists of a sequence of king moves followed by a sequence of rook moves.*

Proof. Given a solving capture sequence, we follow a simple procedure: Whenever a king move is preceded by a rook move, swap the two moves. After completing this bubble sort type of approach, all king moves are sorted to the front.

It remains to show that any such swap does not invalidate any of the moves. To show this, we first deduce some facts about the capture sequence and the two moves to be swapped in particular. Let $(z_1 \rightarrow z_2)$ be the rook move and $(z_3 \rightarrow z_4)$ be the king move in question. Figure 4.2a shows an example position. The origin square of the king move, z_3 , cannot be the origin or destination square of the rook move. Otherwise, the king could no longer follow up said rook move since it would already have been captured. Similarly, the origin square of the rook move cannot be the destination square of the king move as that square is already vacated by the time the king move happens. If the two moves end on the same square $z_2 = z_4$, then

they have captured from opposite directions, since the rook cannot jump over the king. It follows that after performing the two moves (as shown in Figure 4.2b), either adjacent square of the destination square is empty. Thus, in this case we can conclude that there are no further moves by the king.

Consider now the capture sequence with swapped moves. Both the king and the rook move themselves stay valid: A king move cannot be blocked, and the rook move happens on a board containing a strict subset of filled squares as compared to the previous scenario. Also, the piece on the rook’s destination square has not moved away, so the move remains a capture.

If the two moves have different destination squares, then after the swap the same resulting position is reached. Otherwise, we observe that any future move cannot have been by the rook as that rook was captured in the original sequence. It also cannot have been by the king as observed above. Thus, any future move was by a different piece. In this case, the legality of any of the remaining moves is not impacted by whether square $z_2 = z_4$ holds a king or a rook. In particular, either could still be captured and neither could be jumped over. Overall, we see that such a swap is always possible without invalidating the solving sequence, and so the claim holds. ■

Being able to search for not just any solution but one in normal form greatly simplifies that search. We give an algorithm for deciding such configurations which can easily be adapted to also produce a solving sequence.

Lemma 4.5: *Let I be an instance of one-dimensional $\leq B$ -SOLO CHESS containing n pieces, each of which is either a rook or a king. The existence of a solving sequence for I can be decided in time $\mathcal{O}(f(B) \cdot n)$ for some computable function f .*

Proof. We assume without loss of generality that the instance is placed on a board containing only $\mathcal{O}(n)$ squares. In particular, since the king can only move to directly adjacent squares while the rook has unlimited range, any instance can be “compacted” such that between any two consecutive pieces there is at most one empty square. Furthermore, by the previous lemma we can assume that if there exists a solving sequence, then there also exists one in normal form.

A solving sequence in normal form is split into two phases: A king move phase and a rook move phase. In the latter phase the kings do not move and, therefore, are equivalent to 0-rooks. Thus, the problem becomes finding a sequence of king moves such that the remaining rook configuration is solvable, i.e., by the 1D Rook Lemma 4.3, such that some m pieces and at least $m - 1$ available rook captures remain.

To find such a sequence, we perform a linear scan (from left to right) over the configuration. We maintain an interface of all candidate partial solutions to the left of the current square. An element of this interface contains the B right-most squares of the partial solution, i.e., the B squares immediately to the left of the current square, as well as an integer t encoding the “invisible” remainder of the partial solution. It is defined as the difference $t = (\text{rook captures available} - \text{number of pieces present})$ in that remainder of the partial solution. Thus, it can be interpreted as the evaluation of the 1D Rook Lemma 4.3, where each king has been replaced by a 0-rook. We can discard dominated entries: Any entry with the same visible B squares, but a lower value t , provides no benefit for finding a solving sequence. Thus, at any time the number of elements of the interface is at most the number of configurations on B squares that contain kings and rooks with budgets between 0 and B , or empty squares. This gives a crude upper bound of $(2B + 3)^B$ elements, which only depends on B .

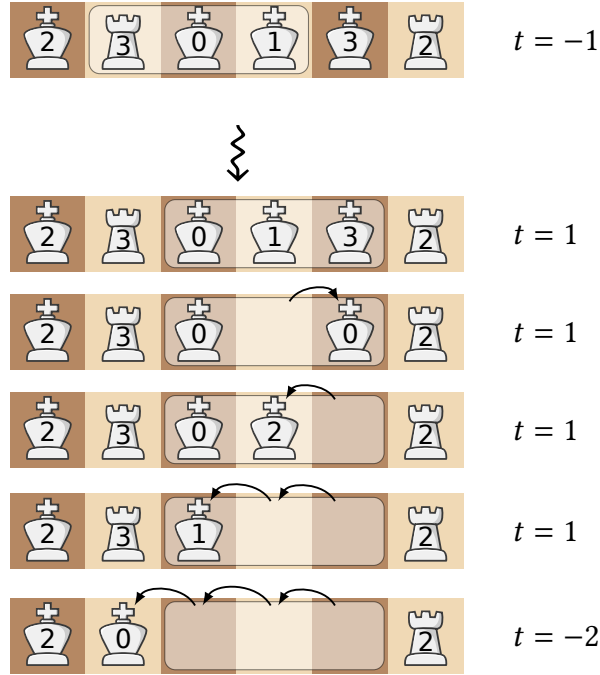


Figure 4.3.: An example configuration with problem parameter $B = 3$. An update step on the top interface element, highlighted in white, yields the set of bottom interface elements.

We then use this interface to generate the interface for the next square: If the new square in question is empty, we simply “shift” each interface element by one and discard any newly dominated entries. This shift removes the left-most square of the visible configuration and updates t appropriately (i.e., subtracts 1 if the removed square held a king, adds $i - 1$ if the removed square held an i -rook and keeps the value unchanged if it was empty), followed by adding the new square to the right. If the new square holds a rook and the right-most square of an interface element is a king with a budget greater than 0, the king can capture the rook. In this case, in addition to the shifted interface element, we also add an element with that king move performed before the shift. Finally, if the new square holds a king with a budget $c > 0$, it can perform up to c captures to the left (depending on the configuration). In this case, in addition to the shifted element and possibly the element containing a capture of the king from the right-most visible square, we also add elements with captures by the new king performed before the shift. Since dominated interface elements are discarded, we again end with at most $(2B + 3)^B$ elements. In particular, such an update step can always be done in time $f(B) \in \mathcal{O}(B \cdot (2B + 3)^B)$. Figure 4.3 shows an example of the update step on one interface element.

Once we reach the end of the configuration, after performing the above update step for the next B empty squares we remain with only one interface element: A visible partial configuration consisting of only empty squares, and its attached optimal value t , all other dominated interface elements having been discarded. Recall that this value encodes the difference $t = (\text{available rook captures} - \text{number of pieces present in the entire configuration})$. Thus, the configuration is solvable if and only if t is at least -1, by the 1D Rook Lemma 4.3.

Overall, we obtain a running time of $\mathcal{O}(f(B) \cdot n)$ which is FPT in B and, in particular, linear time for any constant B . ■

With this result at our disposal, proving the theorem becomes simple:

Proof of Theorem 4.2. Let I be an instance of $1D \leq B$ -SOLO CHESS. We transform it in linear time into an instance I' by replacing all bishops, pawns and knights with 0-rooks, and all k -queens with k -rooks. This results in an equivalent instance since on a 1D board, bishops, pawns and knights cannot perform any captures, and the queen only has rook moves available. We then solve I' in the desired running time as described above. ■

5. King 2-SOLO CHESS

After having reviewed some special cases of SOLO CHESS that turned out to be solvable in linear time, we now review the case $B = 2$ on a standard two-dimensional chess board. These parameters replicate those of the original game. In particular, since $B = 1$ is trivial as seen in the previous chapter, this is the smallest “interesting” value for B .

Under these parameters, the decision problem turns out to be NP-complete even when restricted to uniform instances. In particular, we show that 2-SOLO CHESS restricted to instances containing only kings is already NP-complete. We reduce from And-Or-(1,1)-SAT.

5.1. The Setup

We transform a given And-Or-(1,1)-SAT instance I into an equivalent King 2-SOLO CHESS instance: We choose an And-Or Embedding of I in the plane. We then directly translate the embedding into a corresponding chess position. Figure 5.1 shows the structure of the King 2-SOLO CHESS instance created from an example instance $I = (U, C)$ with variables $U = \{x, y, z\}$ and clauses $C = \{(x \wedge \neg y) \vee z, (\neg x \vee \neg z) \vee y\}$. It shows the creation of variables x, y, z (in green) and their respective positive and negative literals. These literals are connected by *wire* to the logic gates for each clause. These consist of Or-gates (in pink) and And-gates (in red). The outputs of the clauses are combined pairwise through And gates to produce a single final value. This value is then checked to be true with a *1-test*. We create gadgets for the variable assignment, the wire, the Or and And gates, and the 1-test. Not shown in the figure is the wire crossing gadget which is needed to implement crossings like the one in the bottom left of the example configuration.

Our transformation yields a SOLO CHESS instance that can be fully cleared, if and only if the SAT instance I is satisfiable. In the following sections, we describe how to represent logic values (true and false) as well as the implementation of each gadget, and discuss their correctness. When discussing a gadget, we may show which value it *evaluates* to. This is understood to denote which output value the gadget produces after it has otherwise been cleared completely (recall that a solving sequence clears every square bar one, which means that any square except the dedicated final square needs to be cleared at some point). More specifically, we create a variable assignment gadget which produces literals x and $\neg x$ and evaluates to one true and one false output. We create a wire gadget which propagates an input x from the beginning to the end of the wire. We create And and Or gadgets which evaluate inputs x and y to outputs $x \wedge y$ and $x \vee y$ respectively. Any solving sequence fully clears all of the above gadgets. Finally, we create a 1-test which is reduced to a single final piece if its input is true, and which cannot be reduced to a single piece if its input is false.

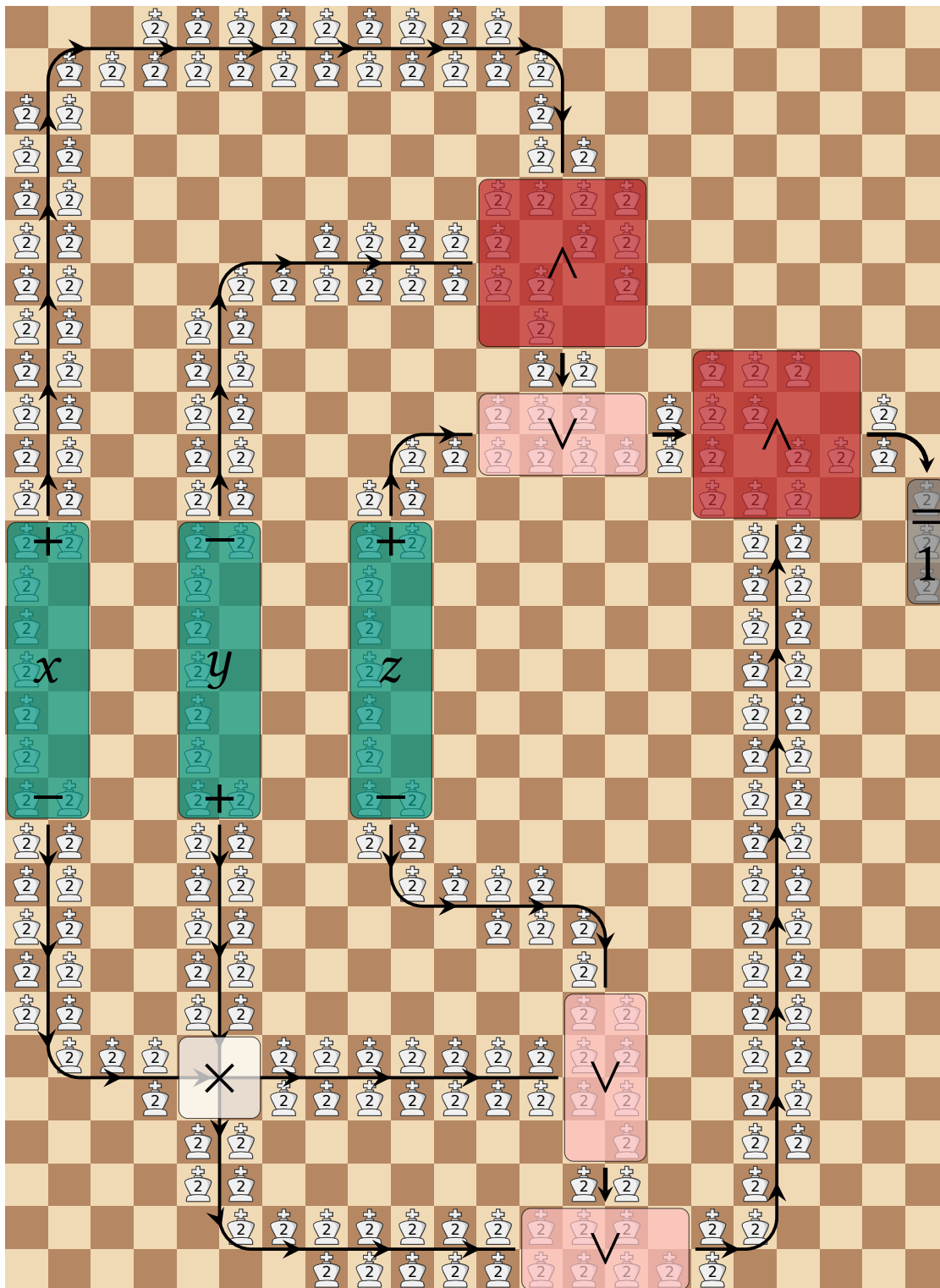


Figure 5.1.: A King 2-SOLO CHESS instance encoding the SAT instance $\{(x \wedge \neg y) \vee z, (\neg x \vee \neg z) \vee y\}$. For space constraint reasons, the wire crossing gadget \times is not placed on the board.

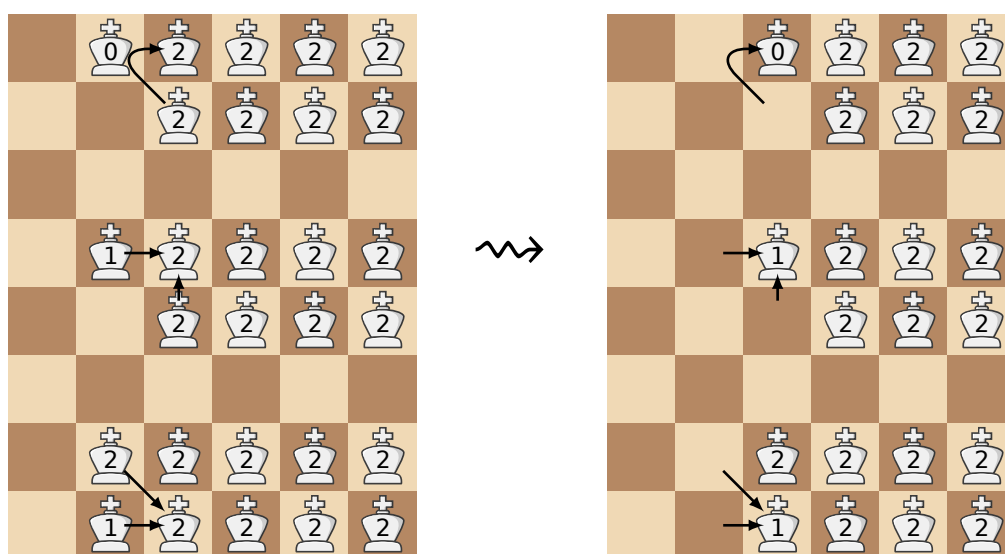


Figure 5.3.: Three wires holding values 0 through 2. Arrows indicate pairs of moves that propagate the signals by one column each.

5.2. The Wire

We begin with the wire, which provides the foundation for our construction. A wire is used to propagate a *signal* from one gadget to another. We implement a wire using two rows (or columns) of kings. Figure 5.2 shows an example of a wire.

We can propagate three different signal values through a wire, namely 0, 1 and 2. In our construction, 0 corresponds to the logic value *false*, while 1 corresponds to *true*. The value 2 is an auxiliary value used only within gadgets. In a wire, a 0-signal is represented by a column containing only a single 0-king, a 1-signal is represented by a column containing only a single 1-king and a 2-signal is represented by a column containing a 1-king and a 2-king. We call this column the *signal column* of the wire. Figure 5.3 shows examples of all three signals, as well as the intended way to propagate them by one column. As can be seen, propagating a signal “uses up” the wire. Thus, each wire can only be used once to propagate a single signal, after which the wire is no longer present. In particular, once one end of the wire holds a signal, this signal propagating through the wire gives that wire a *direction*. We mark this direction using repeated arrows, as can be seen in Figure 5.1.

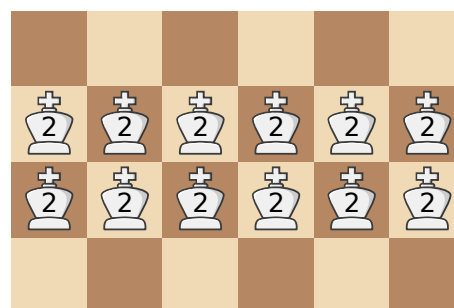


Figure 5.2.: A wire connecting the left and the right side of this configuration.

Note that the two kings in a column of a wire have equivalent moves: Each is adjacent exactly to the kings of the previous and the next column, as well as the other king of its own column. It follows, then, that the vertical orientation of the signal does not matter. In particular, the signal column having a single 0-king in the top row or the bottom row are equivalent representations of the 0-signal. More generally, an alternative view point of the

signal value is to view it as the number of excess captures of the signal column. If the column contains n kings that have a combined budget of c , the value of the signal is $s = c - n + 1$. This viewpoint immediately yields two further insights:

Observation 5.1: *There is a monotony on signal values. Every capture sequence that is possible with a lower signal value remains possible with a higher signal value, since the latter has at least as many (in fact, more) captures available.*

Based on this observation, when describing which value a gadget evaluates to, we always assume it to evaluate to the maximum possible value. The alternative viewpoint also explains why only signal values 0 through 2 are possible:

Lemma 5.2: *A wire cannot propagate signal values larger than 2 or smaller than 0.*

Proof. We first observe that the minimum value that the expression $s = c - n + 1$ can take for a column is -1, while the maximum value is +3. A hypothetical signal value of 3 would need to consist of a signal column of two 2-kings. However, when propagating a signal, at least one king of each column gets captured, after which it cannot have a budget of 2.

A hypothetical signal value of -1 would need to consist of a signal column of two 0-kings. Then, it is easy to verify that there is no sequence of moves that allows both those kings to leave the column; instead at least one of them remains stranded. ■

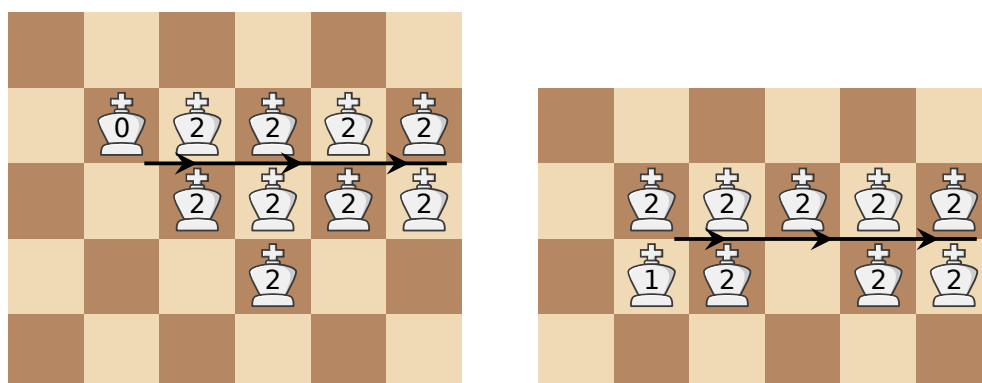
For convenience, we sometimes use the value -1 to indicate such a scenario of a stranded piece. For example, we say that a gadget evaluates to -1 for certain inputs, if it is impossible to clear the gadget completely. This implies in particular that the resulting configuration is unsolvable.

We show that the wire works correctly in the sense that it propagates values without modification. In particular, it is impossible to increase signal values; otherwise, it would be possible to turn the logic value *false* into *true*.

Lemma 5.3: *The signal value that is propagated through a wire neither decreases nor increases.*

Proof. We have already seen in Figure 5.3 that the signal value does not decrease (recall that we assume each gadget to evaluate to the maximum possible value). It remains to show that it is impossible to increase signal values. Using the above viewpoint, our claim amounts to saying that the value s is an invariant that cannot be increased while a signal traverses a wire. For captures within the signal column, this is clear: Each capture uses (at least) one unit of budget, while reducing the number of pieces by exactly one. For the signal to propagate, we also need to consider the next column. At least one of its pieces must be captured by one or more of the previous column's pieces. This captured piece then provides none of its budget to the capture sequence. Thus, we see that this column contributes at most two units of budget, while also adding two pieces itself. This means that the invariant does not increase when including the next column, which in turn shows that when propagating the signal by a column, its value cannot increase. ■

This invariant further suggests simple mini-gadgets to increase or decrease a signal value by 1 (up to the maximum of 2, or down to the error value -1). The Increment gadget consists of a wire with one specific column containing three instead of two kings, as seen in Figure 5.4a. The Decrement gadget consists of a wire with one specific column containing only one instead of two kings, as seen in Figure 5.4b. When part of a larger gadget, we may call such a wire column with only a single king a “defect” in the wire.



(a) The Increment gadget acting on a 0-signal. (b) The Decrement gadget acting on a 2-signal.

Figure 5.4.: Two possible modifiers for a wire propagating a signal.

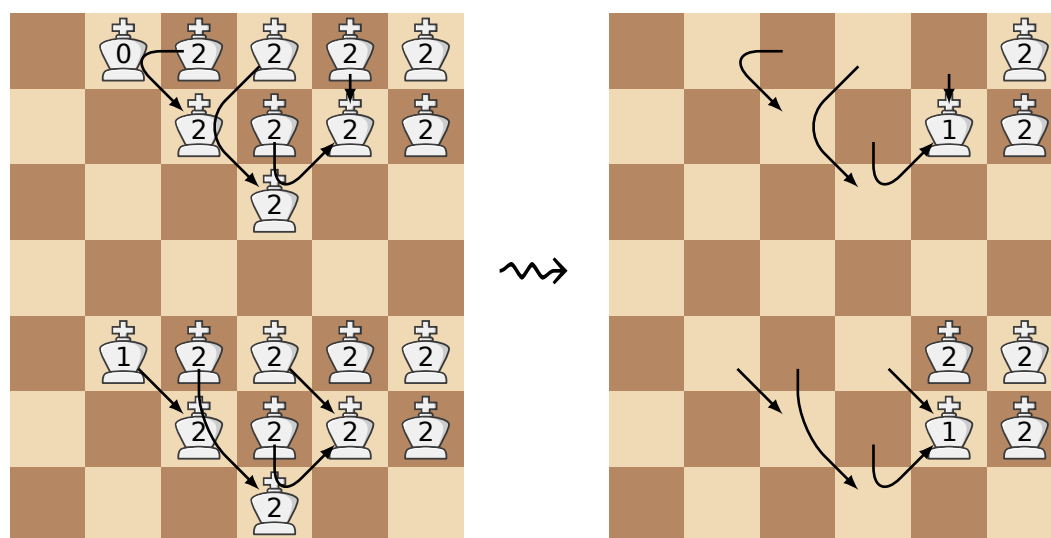


Figure 5.5.: Increasing a 0- and a 1-signal by 1 each. Moves happen left to right. Note that the budget of the extra king of the Increment gadget does not matter.

Lemma 5.4: *The Increment gadget increases the value of a passing signal by 1.*

Proof. We begin by giving a capture sequence achieving the claimed behavior: Figure 5.5 shows both a 0- and a 1-signal being increased by 1 by capturing through the extra king of the gadget. By monotony, a 2-signal stays unchanged under the Increment gadget. It is for example possible to follow the same capture sequence as the 1-signal, with one additional capture by the extra 2-king.

This is the best possible outcome: Consider again the invariant $s = c - n + 1$. The Increment column contains three kings, at least one of which must be captured and, thus, contributes none of its budget. Overall, the column contributes at most four units of budget while adding three pieces, a net gain of one, i.e., a signal increase of one. ■

Lemma 5.5: *The Decrement gadget decreases the value of a passing signal by 1.*

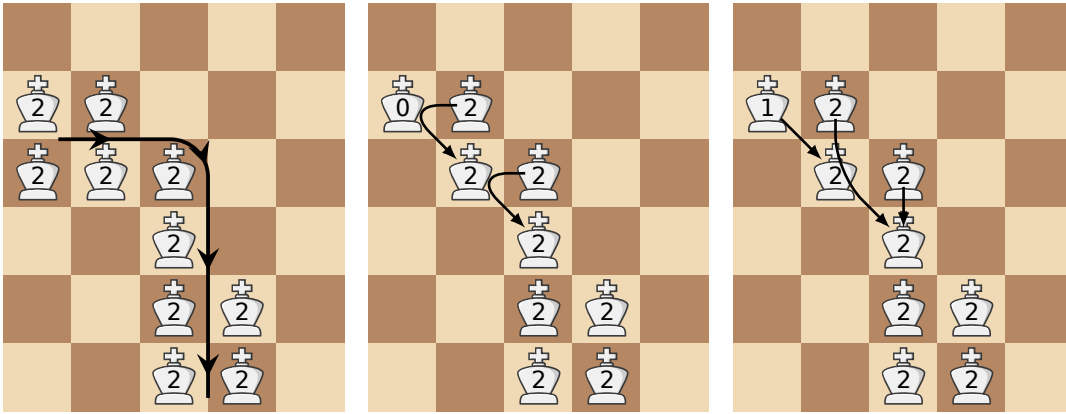


Figure 5.6.: A wire turning a corner from the left to the bottom. Both 0- and 1-signals are preserved.

Proof. Simply following the capture sequences for a 1- or 2-signal shown in Figure 5.3 yields a 0- or 1-signal, respectively, when passing through a Decrement gadget. This again is the best possible outcome by the invariant: The Decrement column contains only a single king, which has to be captured. Thus, it contributes zero units of budget and adds one piece, a net loss of one, i.e., a signal decrease by one. ■

Finally, we discuss the orientation of wires. Naturally, wires function the same whether they are oriented horizontally or vertically. However, to transition from horizontal to vertical movement or vice versa, it is necessary to turn a corner. Figure 5.6 shows a corner that is to be traversed from the left towards the bottom. As indicated by the two examples, the corner preserves both 0- and 1-signals. It does not preserve 2-signals, however, in our construction we only need corners in wires holding 0- or 1-signals.

Lemma 5.6: *A corner in a wire does not change the value of 0- or 1-signals.*

Proof. Figure 5.6 shows that it is possible to retain the signal value. It remains to show that the signal value cannot increase instead. The largest signal that can exit a corner is a 1-signal: The cut vertex in the fourth row is captured at some point, after which its budget is at most 1, corresponding to a 1-signal. Thus, it remains only to show that a 0-signal cannot be turned into a 1-signal. For this, it suffices to review the capture sequence given in Figure 5.6 and observe that either double-capture is necessary to not end with a stranded 0-king. Overall, 0- or 1-signals neither decrease nor increase, as claimed. ■

This concludes our discussion of wires. In the following sections we use these wire concepts to define more complicated gadgets which are, in turn, connected by said wires.

5.3. Variable Assignment

The assignment gadget for a variable x has zero inputs and two outputs, namely the x and $\neg x$ literals. Our implementation of the gadget can be seen in Figure 5.7. It consists of a single row of five kings, enclosed by a wire on either side. When the variable assignment gadget is resolved, these wires carry the positive and the negative literal, respectively. We show that the gadget sets exactly one literal to true and the other to false, as expected for a boolean variable.

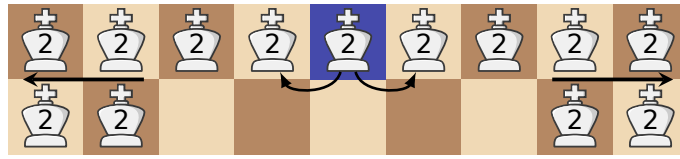


Figure 5.7.: A variable assignment gadget. The capture of the center king decides which of the two wires receives a 1- and which receives a 0-signal.

Lemma 5.7: *A variable assignment gadget produces one 1- and one 0-signal.*

Proof. We begin by discussing the center king, marked in blue. Assume that it gets captured at some point, without loss of generality, by the king to its left. Then this creates a king with a budget of 1 as the final piece of an antenna of length 2, connected to the wire on the right. By Lemma 3.6, this antenna is stranded. Since every solving sequence fully clearly every SAT gadget, the above does not happen in any solving sequence. Thus, the center king instead performs a capture itself. Such a capture creates an antenna of length 1 on either side of the center king. One of these antennae has as first vertex a 2-king, the other a 1-king due to the first vertex having been captured by the center king. Either antenna is resolved by a capture from its first vertex. This creates a 1-signal on the one wire and a 0-signal on the other wire. By capturing with the center king either to the left or to the right, the literals of a variable can take either of the two possible assignments. ■

As shown above, the only way to resolve the variable assignment gadget is to create a signal on each wire, which is then propagated away from the gadget. This gives both attached wires a direction, namely away from the gadget.

In our construction of the wire crossing gadget, we need two copies of the negative literal of a variable. For this purpose, we describe a Double Assignment gadget. It is a variable assignment gadget that produces two copies of the negative and one copy of the positive literal of a variable. Figure 5.8 shows a high level view of our implementation of this gadget. It consists of two variable assignment gadgets with one pair of outputs being combined with an And gadget (which we define in an upcoming section). This gadget has the desired behavior: Literal x is set to true if the And gadget evaluates to true, which is only the case if both x_1 and x_2 are assigned the value true. In this case, both $\neg x$ literals are set to false. Literal x is set to false if the And gadget evaluates to false. In this case, both x_1 and x_2 can be assigned the value false, resulting in both $\neg x$ literals being set to true. As with standard variable assignments, it is possible to set even fewer outputs to true. However, as before, we assume each gadget to evaluate to the maximum possible value.

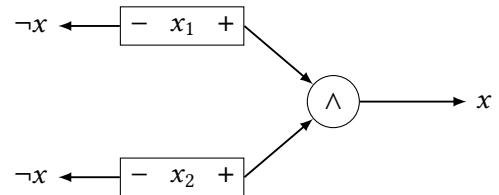
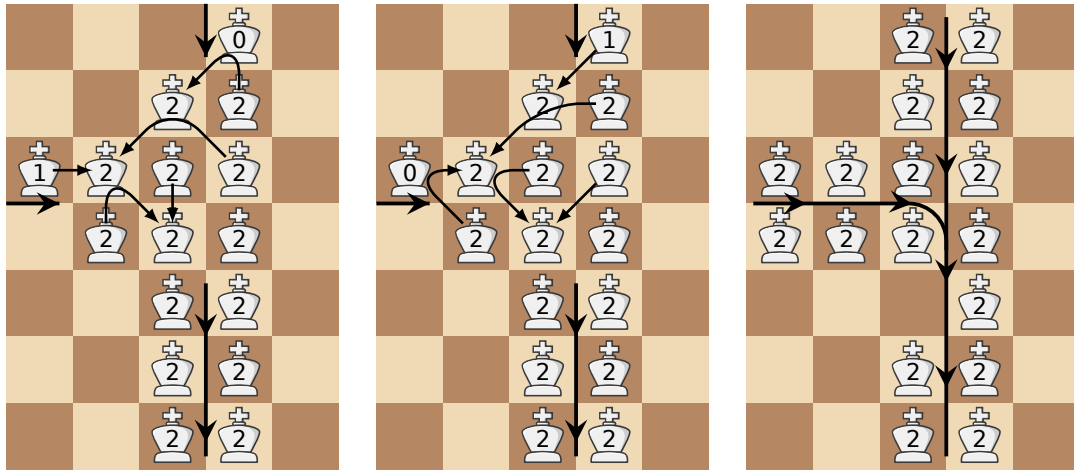


Figure 5.8.: A variable assignment producing one copy of the literal x and two copies of the literal $\neg x$.

This construction allows us to create multiple copies of the negative literal of a variable, so long as the positive literal is only needed once.



(a) A shifted sum with inputs 0 and 1 produces output 2. (b) A shifted sum with inputs 1 and 0 produces output 2. (c) The Or gadget combines a shifted sum and a Decrement.

Figure 5.9.: Two instantiations of a shifted sum gadget (left), and the implementation of the Or gadget (right).

5.4. The Or Gadget

The Or gadget has two inputs and one output. It evaluates to true if at least one of the inputs is true. Using signal values of 0 and 1, we can define the Or function as $x \vee y = \min\{1, (x + y)\}$. A useful intuition for our implementation of the Or gadget is the equivalent formulation $x \vee y = \min\{2, (x + y + 1)\} - 1$. To implement this function, we need a gadget that computes the *shifted sum* $x + y + 1$. Since a wire can only carry signal values up to a maximum of 2, the minimum part of the function is taken care of automatically. Thus, we only need to combine a shifted sum gadget with a Decrement gadget.

This shifted sum gadget is created by simply merging two wires. Figures 5.9a and 5.9b show the gadget under two different sets of inputs. If the left input of the shifted sum is at least 1, the left wire can be reduced by a sequence of captures so as to act as two consecutive Increment gadgets. This increases the top wire’s value by 2. If the left input is 0, the left wire can serve as a single Increment gadget, increasing the top wire’s value by 1. We show that the shifted sum gadget works correctly:

Lemma 5.8: *The shifted sum gadget produces an output of 1 if both inputs are 0, and an output of 2 otherwise.*

Proof. The sequences depicted in Figures 5.9a and 5.9b sketch how each of the claimed outputs can be reached. The only case where this does not yield the maximum possible value of 2 is when both inputs are 0. In this case, incrementing a 0-signal once yields a 1-signal as intended. We need to show that this is the best possible result. For this, we observe the following facts:

- 1 For two 0-signals as input, either 0-king can only be propagated by a pair of captures from and to the two adjacent kings.
- 2 Any 2-king in the left wire (before the “junction”) that is captured by a king of the right wire can, for the purposes of (1), only serve the second role of being captured towards.

- 3 As long as the left wire contains more than one column, none of its kings capture into the right wire, as that would introduce a defect into the left wire, leaving its 0-signal stranded.

We claim that any capture sequence that evaluates the gadget can be re-ordered to begin with all moves within the left wire. As in the 1D SOLO CHESS proof, we do this by a series of swaps similar to bubble sort. It again suffices to show that none of the swaps lead to an invalid capture sequence.

We begin with the scenario where the left wire contains more than one column, i.e., the condition of (3) is met. Consider moves m_1 and m_2 to be swapped, i.e., move m_2 being contained within the left wire while move m_1 is not. By (3), move m_1 is either contained within the right wire or moves from the right to the left wire. In the first case, the two moves are independent, and so swapping them always retains a valid sequence. Consider instead the second case, and a pair of moves that are not independent. By (2), the destination square of m_1 cannot be the origin square of m_2 . Thus, if they are not independent, the two moves share their destination square. In this case, m_2 is the second move in a pair of moves of (1). Then, the original sequence of m_1 followed by m_2 results in a 0-king, so swapping them results in a king with a budget that is no smaller. Therefore, by monotony, this swap retains a valid capture sequence.

Now, consider the scenario where the left wire contains only a single column. Since the left wire contained a 0-signal, this column consists of a single 0-king. In this case, there can be no further moves within the left wire, so there are no moves to be swapped in this scenario. The resulting configuration is exactly that of an Increment gadget, i.e., an increase of the top signal from 0 to 1, as claimed. ■

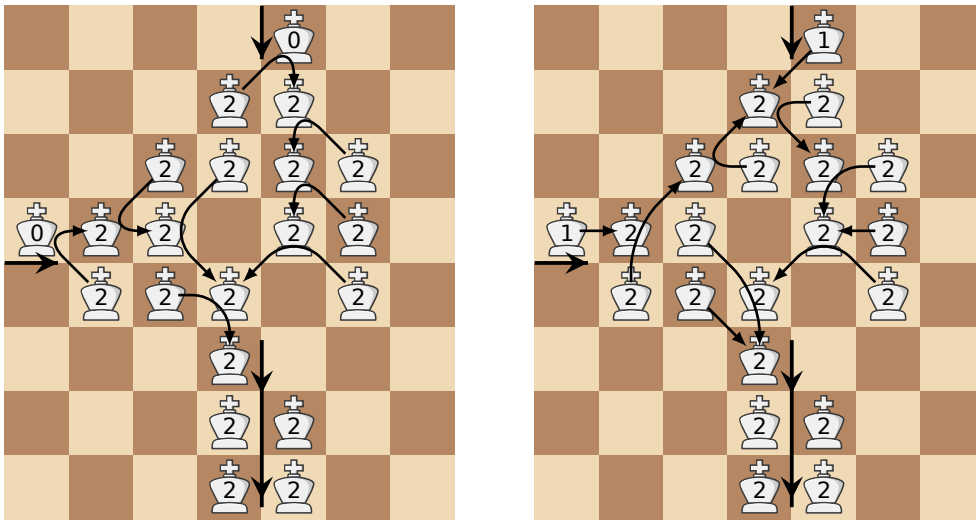
The full Or gadget consists of a shifted sum gadget followed by a Decrement gadget and can be seen in Figure 5.9c. Its correctness follows directly from the correctness of the shifted sum gadget and the Decrement gadget. Thus, we have shown:

Lemma 5.9: *The Or gadget evaluates to 0 if both inputs are set to 0. It evaluates to 1 for any other set of inputs.*

5.5. The And Gadget

The And gadget has two inputs and one output, just like the Or gadget. However, the requirements for the And gadget are slightly more complicated than for the Or gadget: If both inputs are 0, it evaluates to 0. Increasing either input value does not increase the output value, however, increasing both input values *does* increase the output value. The idea of our And gadget, shown in Figure 5.10, is to have for each input signal two different paths it can take. Choosing the top right path with the top signal and the bottom left path with the left signal is possible even when both input values are 0, and results in a 0-output. If both inputs values are 1, it is instead possible to choose the top right path for both signals while keeping the bottom left intact, resulting in a 1-output. We show that the And gadget works correctly for any set of inputs:

Lemma 5.10: *The And gadget evaluates to 1 if both inputs are set to 1. It evaluates to 0 for any other set of inputs.*



(a) Inputs 0 and 0 evaluate to an output of 0. (b) Inputs 1 and 1 evaluate to an output of 1.

Figure 5.10.: The And Gadget being evaluated under two sets of inputs.

Proof. The sequences depicted in Figure 5.10 show that it is possible to achieve the claimed output values. In particular, the sequence yielding a 0-output is, by monotony, valid for any set of inputs. It remains to show that it is impossible to achieve larger output values.

We begin by observing that the output wire begins with a defect, which shows that the output value is at most 1. This value is reached if both inputs are 1. We show that it is impossible to achieve a 1-output if at least one of the inputs is not 1. To this end, we first discuss how an output of 1 can be achieved in principle and then show that none of these scenarios are possible, unless both inputs are set to 1. We split the argument into multiple claims that we then show individually.

Claim 1: If the And gadget evaluates to 1, one of the intermediate configurations shown in Figures 5.12a, 5.13 and 5.14a has been reached first.

Proof. Figure 5.11 shows the desired configuration of a 1-output. We use the method of retrograde analysis: We work backwards from the desired result configuration to deduce what the final few moves were. Since each capture uses a unit of budget, each *un-capture* is played by a piece whose budget is smaller than 2. Thus, in a configuration where each piece has a budget of 2 except for the 1-king on (6, 4), the previous capture has been by a 2-king onto that (6, 4)-square. There are two possibilities for this: $((5, 4) \rightarrow (6, 4))$ or $((5, 3) \rightarrow (6, 4))$. If the first un-capture was $((5, 4) \rightarrow (6, 4))$, then the only square that could possibly hold a piece with a smaller budget than 2 is that same (6, 4) square. Thus, we deduce further un-captures $((x, y) \rightarrow (5, 3) \rightarrow (6, 4))$, yielding us an earlier intermediate configuration shown in Figure 5.12a. Here, the (x, y) -king is one of the three transparent kings with outgoing transparent move arrows.

If instead the first un-capture was $((5, 3) \rightarrow (6, 4))$, then once more the only possible king with a budget smaller than 2 is the king on (6, 4). Thus, we deduce another un-capture $((5, 4) \rightarrow (6, 4))$, preceded by either $((4, 3) \rightarrow (5, 4))$ or $((4, 5) \rightarrow (5, 4))$. In either case, the only possible un-capturer is the king on (5, 4). Depending on the above un-capture, the only remaining unused neighboring square is (4, 5) or (4, 3), respectively. This yields

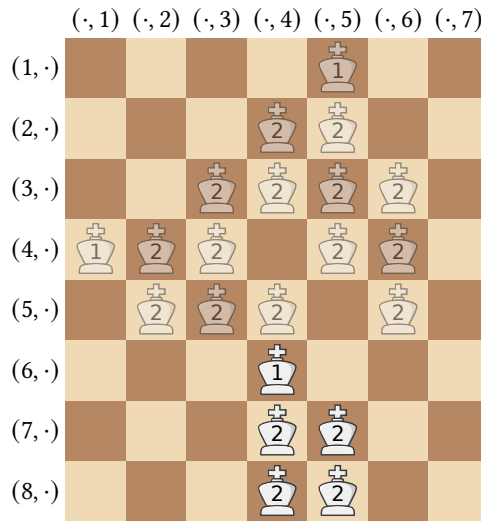


Figure 5.11.: The final configuration of an And gadget evaluating to 1. Transparent pieces indicate which kings were originally present in the gadget.

two further un-captures $((x', y') \rightarrow (4, 5) \rightarrow (5, 4))$ or $((x', y') \rightarrow (4, 3) \rightarrow (5, 4))$, respectively. Figures 5.13 and 5.14a show these two possible intermediate configurations. Since we covered all cases of possible un-captures from the 1-output configuration, one of these intermediate configurations has been reached, if the capture sequence then reached the 1-output configuration, which shows the claim.

We now show that none of the intermediate configurations is reached, unless both inputs are set to 1.

Claim 2: If the inputs of an And gadget are not both set to 1, then there is no capture sequence that reaches the configuration shown in Figure 5.12a.

Proof. Let at least one of the inputs of the And gadget be set to 0. Assume for contradiction that there exists a *resolving* capture sequence that reaches the configuration of Figure 5.12a. Then, in this sequence, all remaining (transparent) kings of the configuration (except the mystery (x, y) -square king) capture towards one of the visible kings to clear the remaining configuration. The resulting king has a budget of less than 2, therefore, it is the king on the $(5, 3)$ -square. We first discuss the case where the left input is a 0-signal and then the case where the top input is a 0-signal.

Let the left input be 0. If the 2-king on (x, y) is in the second column, $(4, 2)$ or $(5, 2)$, then the 0-king on $(4, 1)$ has only one transparent neighbor (that is allowed to capture or be captured by it), and, therefore, is immediately stranded. If instead $(x, y) = (4, 3)$, then for the 0-king to reach the $(5, 3)$ -square, any resolving sequence contains captures $((5, 2) \rightarrow (4, 1) \rightarrow (4, 2))$, $((3, 3) \rightarrow (4, 2) \rightarrow (5, 3))$, which requires a 2-king on $(3, 3)$. In this case, the remaining top right side of the configuration does not reach the $(5, 3)$ square and so the intermediate configuration is not reached, a contradiction.

Now, consider the case where the left input is 1 and the top input is 0. If $(x, y) = (4, 2)$, then any resolving sequence progresses the left 1-king through the capture $((4, 1) \rightarrow (5, 2))$. Furthermore, any resolving sequence propagates the resulting 0-king through captures $((4, 3) \rightarrow (5, 2) \rightarrow (5, 3))$, in which case the top right side does not reach

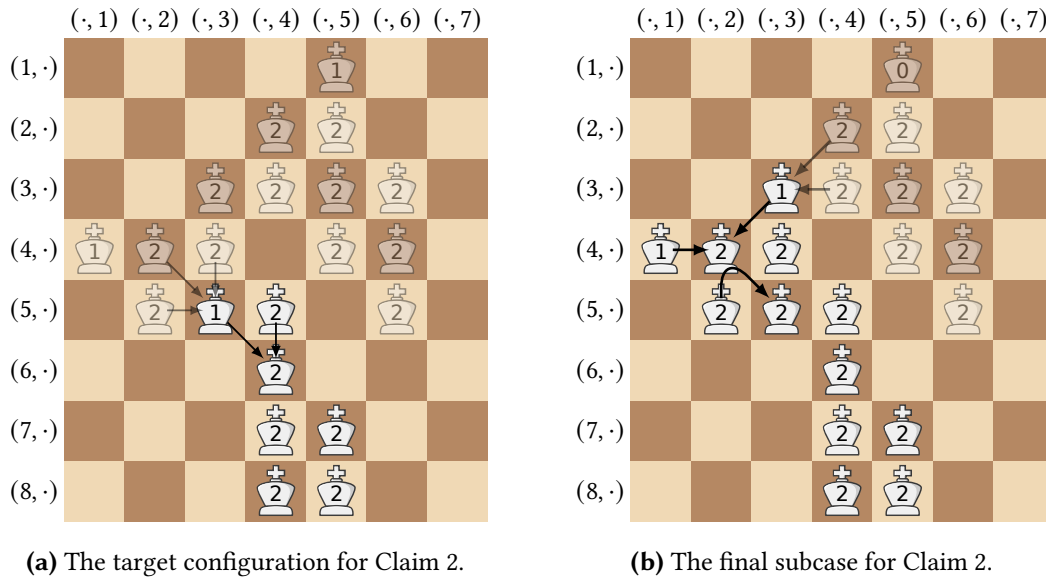


Figure 5.12.: The first option for possible intermediate configurations, discussed in Claim 2.

the (5, 3)-square. Next, consider the case that $(x, y) = (5, 2)$. After the initial capture $((4, 1) \rightarrow (4, 2))$, to not seal off the (5, 3)-square, any resolving sequence continues with captures $((3, 3) \rightarrow (4, 2) \rightarrow (5, 3))$. Then, for the (3, 4)-king to reach the (5, 3)-square, any resolving sequence contains further captures $((3, 4) \rightarrow (4, 3) \rightarrow (5, 3))$. This locks out the top right side of the configuration, and so the intermediate configuration is not reached in this way. Finally, consider the remaining case of $(x, y) = (4, 3)$. This creates a narrow path from the top signal to the (5, 3)-king. Both the (3, 3) and the (4, 2)-kings represent cut vertices. Thus, to cross this path, any resolving sequence contains some captures $((x'', y'') \rightarrow (3, 3) \rightarrow (4, 2))$, after all the remaining kings of the top right component have been cleaned up. This yields us a yet earlier intermediate configuration, shown in Figure 5.12b.

If the remaining top input was a 1-signal, this intermediate configuration could in fact be reached. However, with a 0-signal as input, propagating its 0-king requires using both neighboring second row kings. This rules out the (2, 4)-king to serve as the final capturing 2-king, which leaves $(x'', y'') = (3, 4)$. Thus, the remaining pieces reach (3, 3) via the (2, 4)-square. After captures $((2, 5) \rightarrow (1, 5) \rightarrow (2, 4))$, any resolving sequence propagates the resulting 0-king through captures $((3, 5) \rightarrow (2, 4) \rightarrow (3, 3))$. In this case, some kings on the right side of the configuration remain, meaning the intermediate configuration is not reached in this way, a contradiction. This concludes the final subcase and, thus, shows that the intermediate configuration of Figure 5.12a is not reached, contradicting the assumption and showing the claim.

Claim 3: If the inputs of an And gadget are not both set to 1, then there is no capture sequence that reaches the configuration shown in Figure 5.13.

Proof. Assume for contradiction that there exists a resolving sequence that reaches the configuration of Figure 5.13. We first discuss the case where the left input is 0, then the case where the top input is 0. Any resolving sequence contains the pair of captures

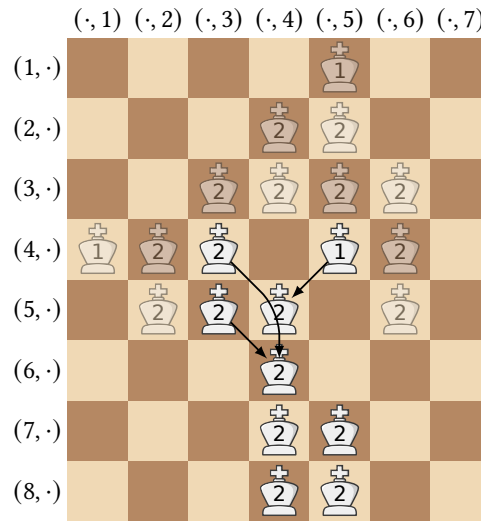


Figure 5.13.: The second option for a possible intermediate configuration, discussed in Claim 3.

$((4, 2) \rightarrow (4, 1) \rightarrow (5, 2))$ or $((5, 2) \rightarrow (4, 1) \rightarrow (4, 2))$. The resulting 0-king is stranded in the subgraph induced by the transparent pieces, a contradiction. In the latter case, this is because it is a 0-leaf in the subgraph.

Consider now the case that the left input is 1 and the top input is 0. To deviate from the above scenario is, any resolving sequence contains a capture by the 1-king of the left input itself. It is not the capture $((4, 1) \rightarrow (5, 2))$, as the resulting 0-king is a leaf in the subgraph induced by the transparent pieces, i.e., stranded. Thus, it is the capture $((4, 1) \rightarrow (4, 2))$, which creates an antenna of length 2. Any sequence resolves this antenna through captures $((5, 2) \rightarrow (4, 2) \rightarrow (3, 3))$. This results in two 0-kings, with only three 2-kings shared between them. Applying Observation 3.3 twice shows that the shared neighbor of the two 0-kings is the destination of both pairs of captures propagating the 0-kings. Thus, any resolving sequence contains the moves $((3, 4) \rightarrow (3, 3) \rightarrow (2, 4))$, $((2, 5) \rightarrow (1, 5) \rightarrow (2, 4))$. The resulting 0-king is a leaf and, thus, stranded. This final contradiction shows that the assumption is false and the claim holds.

Claim 4: If the inputs of an And gadget are not both set to 1, then there is no capture sequence that reaches the configuration shown in Figure 5.14a.

Proof. Assume for contradiction that there exists a resolving sequence that reaches the configuration of Figure 5.14a. Once more, we first discuss the case where the left input is 0, then the case where the top input is 0. If the left input is 0, then any resolving sequence includes captures $((5, 2) \rightarrow (4, 1) \rightarrow (4, 2))$. To propagate the resulting 0-king, while keeping the 2-kings of the intermediate configuration intact, any resolving sequence contains further captures $((3, 3) \rightarrow (4, 2) \rightarrow (4, 3))$. Then, for any resolving sequence, the mystery (x', y') -king is the king on $(3, 4)$, which again locks out the top right side of the configuration from reaching the $(4, 3)$ -square, a contradiction.

For the case of the left input being 1 and the top input being 0, we again discuss possible placings of the (x', y') -square, from which the final capture to $(4, 3)$ originates. It is not placed on $(4, 2)$, since after the capture $((4, 1) \rightarrow (5, 2))$, the resulting 0-king is stranded.

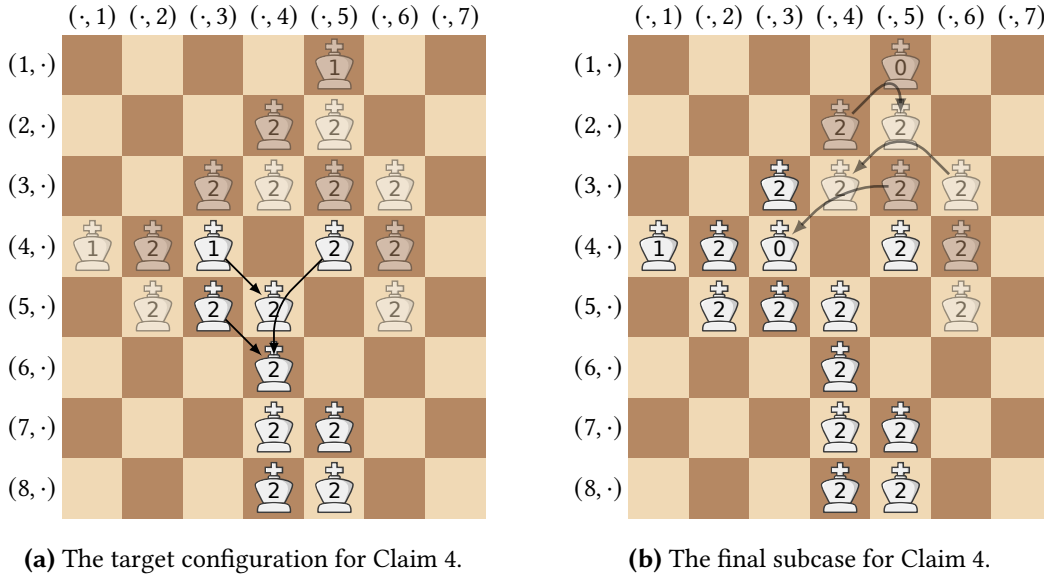


Figure 5.14.: The third option for a possible intermediate configuration, discussed in Claim 4.

Any resolving sequence with $(x', y') = (5, 2)$ can be transformed to one with $(x', y') = (3, 3)$ instead: Any resolving sequence of the former case contains the leaf capture $((4, 1) \rightarrow (4, 2))$, followed by propagating the resulting 0-king through captures $((3, 3) \rightarrow (4, 2) \rightarrow (4, 3))$. In this case, swapping the roles of $(5, 2)$ and $(3, 3)$ yields an equivalent resolving sequence with a mystery king on $(x', y') = (3, 3)$ instead.

If $(x', y') = (3, 4)$, this creates a narrow path from the top right side via squares $(2, 4)$ and $(3, 3)$, both of which are cut vertices. Thus, in this scenario, any cleanup of the top right side ends with captures $((x'', y'') \rightarrow (2, 4) \rightarrow (3, 3))$. Any resolving sequence propagates the resulting 0-king through captures $((4, 2) \rightarrow (3, 3) \rightarrow (4, 3))$. Then, the 1-king of the left signal is the first vertex of an antenna of length 2 and, thus, stranded.

Finally, we consider the case $(x', y') = (3, 3)$. This splits the configuration and separates the two signals. Since either component of the configuration is cleared separately, if there exists a resolving sequence, then (by reordering its moves) there exists one that reaches an earlier intermediate configuration shown in Figure 5.14b. Any resolving sequence propagates the top 0-king using the two second-row kings. This yields us an un-capture sequence from this earlier intermediate configuration of $((3, 5) \rightarrow (3, 4) \rightarrow (4, 3))$, preceded by $((3, 6) \rightarrow (2, 5) \rightarrow (3, 4))$, in turn preceded by $((2, 4) \rightarrow (1, 5) \rightarrow (2, 5))$. This once again leaves some kings on the far right side stranded. This contradiction concludes the final case and shows that the assumption is false. It follows that the claim holds.

To show the overall statement of the lemma, we combine the claims: Let at least one of the inputs not be set to 1. Then, by Claims 2, 3 and 4, none of the intermediate configurations is reached. It follows, by Claim 1, that the And gadget does not evaluate to 1. This shows the second part of the lemma and concludes the proof. ■

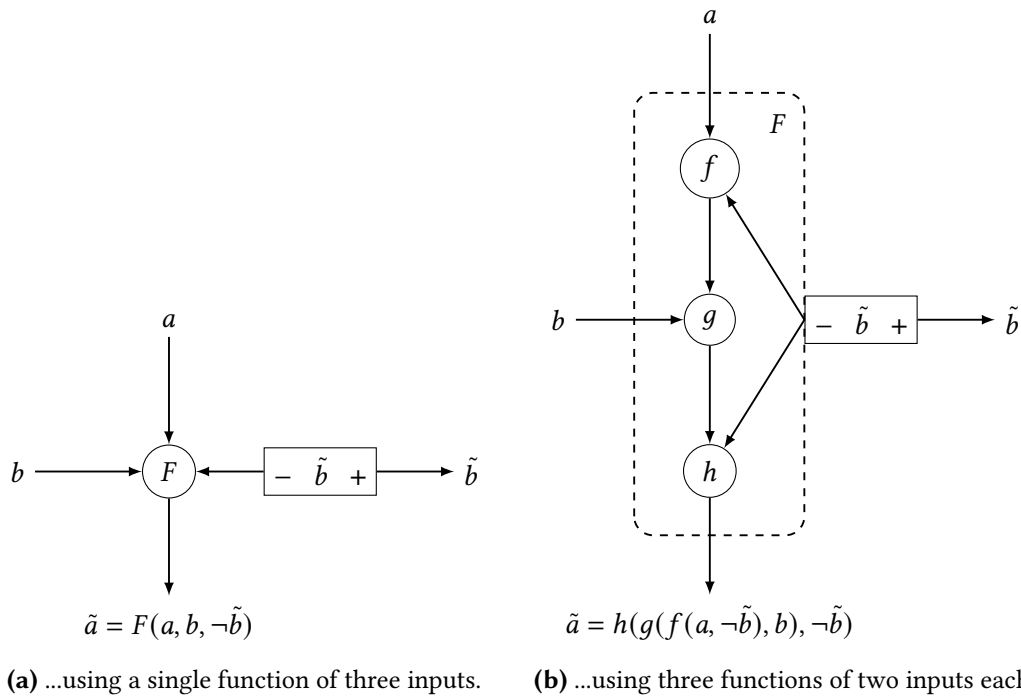


Figure 5.15.: A high level view on the workings of the wire crossing gadget...

5.6. Wire Crossings

As outlined in the beginning of this chapter, the embedding of our SAT instance may contain crossings. Since we directly model connections through wires, this necessitates a gadget to allow wires to cross. Naturally, simply intersecting two wires would allow the signals carried by those wires to interfere with each other. Instead, we define a gadget that takes two inputs a and b and produces two outputs $\tilde{a} = a$ and $\tilde{b} = b$ arranged in a way to allow for the crossing. A high level view of our approach is shown in Figure 5.15a. It consists of a variable assignment for a variable \tilde{b} that is meant to replicate the original b -signal on the other side of the wire that is carrying a . A suitable function F then ensures that b gets replicated correctly and — in this case — simply propagates through the a signal. The idea of such a function is to produce an error if $b \neq \tilde{b}$ and, otherwise, be the identity function on input a . Table 5.1 shows our function F . We remark that for inputs $b = 1, \neg\tilde{b} = 1$, the function F does not produce an error, even though in this case $b \neq \tilde{b}$. As usual, we assume that, due to monotony, any gadget produces the maximum possible output values. In this case, an input $b = 1$ can be matched by $\neg\tilde{b} = 0$, which yields an output of $\tilde{b} = 1$. Thus, the incorrect usage of the wire crossing gadget to downgrade an input b from 1 to 0 merely corresponds to the incorrect usage of a wire performing that same downgrade.

To avoid having to create a gadget for F that combines three inputs at once, we decompose F into multiple functions taking only two inputs each. Recall that it is possible to create two instances of the $\neg\tilde{b}$ literal using the Double

a	b	$\neg\tilde{b}$	$F(a, b, \neg\tilde{b})$
0	0	0	-1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	-1
1	0	1	1
1	1	0	1
1	1	1	1

Table 5.1.: Function F of the wire crossing gadget.

a	\tilde{b}	$f(a, \tilde{b})$	$f()$	b	$g(f(), b)$	$g()$	\tilde{b}	$h(g(), \tilde{b})$
0	0	0	0	0	0	0	0	-1
0	1	0	0	1	1	0	1	0
1	0	1	1	0	0	1	0	0
1	1	2	1	1	2	1	1	0
			2	0	2	2	0	1
			2	1	2	2	1	1

Table 5.2.: A decomposition of function F into three smaller functions.

Assignment gadget. This allows us to split F into three functions as shown in Table 5.2, and place these functions on the board as outlined in Figure 5.15b. Note that for all $a, b, \tilde{b} \in \{0, 1\}$ it holds that $h(g(f(a, \tilde{b}), b), \tilde{b}) = F(a, b, \tilde{b})$. We remark that because every function implemented in King 2-SOLO CHESS is monotone, this decomposition is essentially unique (see Appendix B). It remains to construct gadgets for each of the functions f, g and h .

5.6.1. Wire Crossing Functions f and g

Observe that for all $x, y \in \{0, 1\}$ it holds that $f(x, y) = g(y, x)$. Thus, for the gadget of function f , we simply use the gadget of function g with flipped inputs. Our gadget for function g is shown in Figure 5.16. It is based on the same idea as the And gadget, where each input signal has two different paths it can take. Depending on the set of inputs, different path choices are possible, producing different outputs. We show that this gadget represents a correct implementation of the function g .

Lemma 5.11: *For any set of inputs, the function g gadget evaluates to the value given in Table 5.2.*

To maintain readability, we present only the primary line of reasoning and omit many details across the various cases. We remark, however, that this gadget – like all the other gadgets – has been computer verified to work correctly.

Proof sketch. As usual, we first give capture sequences which show that the correct function values can be attained, then argue that the gadget does not evaluate to larger values. Figure 5.17 gives capture sequences that achieve the values $g(0, 0) = 0, g(0, 1) = 1, g(1, 1) = 2$ and $g(2, 0) = 2$. By monotony, capture sequences for the remaining sets of inputs exist as well. It remains to show that no larger output values can be reached.

To show correctness, due to monotony, it suffices to show that $g(1, 0)$ does not evaluate to a value larger than 0 and $g(0, 1)$ does not evaluate to a value larger than 1. To this end, we discuss the different possible paths that the two input signals can take through the gadget. We show that any path option either does not yield a too large output value or that it is impossible under the relevant pair of inputs.

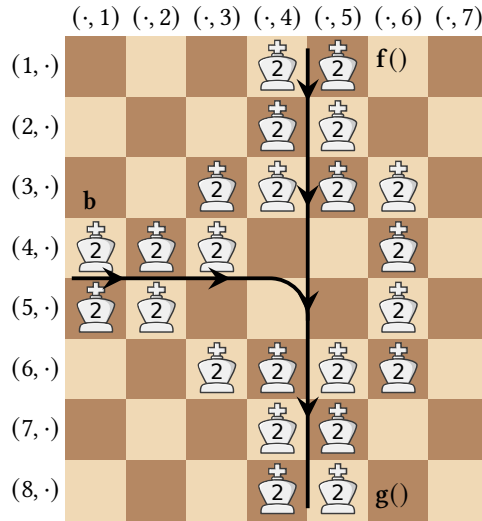


Figure 5.16.: Function g of the wire crossing gadget

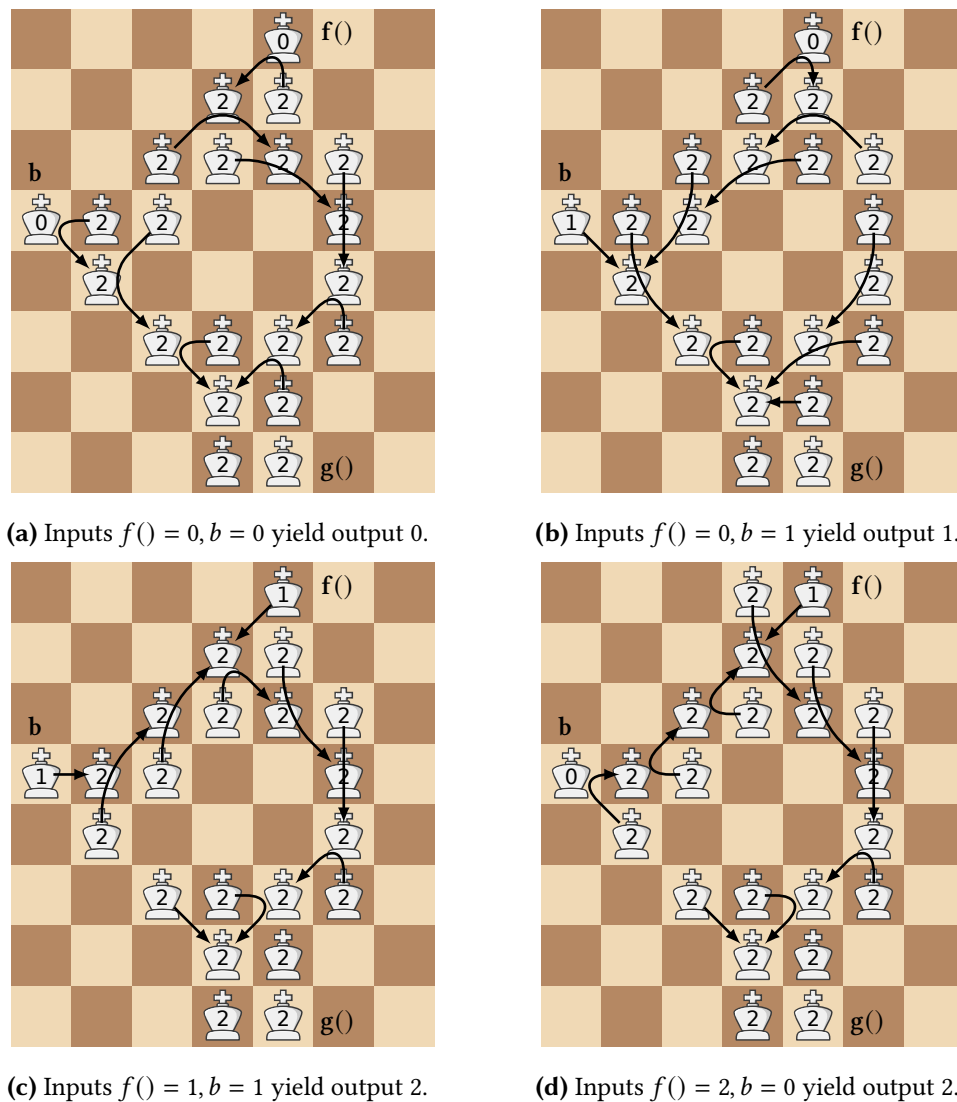


Figure 5.17.: Evaluation of the function g gadget under different sets of inputs.

Claim 1: For any set of inputs, the function g gadget evaluates to an output no larger than given in Table 5.2, if the top signal travels via the right path and the left signal travels via the bottom left path of the gadget.

Proof. If the top signal travels via the right path and the left signal travels via the bottom left path of the gadget, then both $(5, 2)$ and $(4, 6)$ are cut vertices along the respective paths. Thus, by the time the top half of the gadget is cleared, both have been captured and have a budget of at most 1, independent of the inputs to the gadget. This scenario is shown in Figure 5.18a. Any clearing sequence contains the captures $((4, 6) \rightarrow (5, 6)), ((6, 6) \rightarrow (5, 6) \rightarrow (6, 5))$ and $((5, 2) \rightarrow (6, 3))$, as otherwise, some pieces end up stranded. This leads to a familiar configuration of two 0-kings and only three 2-kings between them. As usual, these two 0-kings are propagated towards a

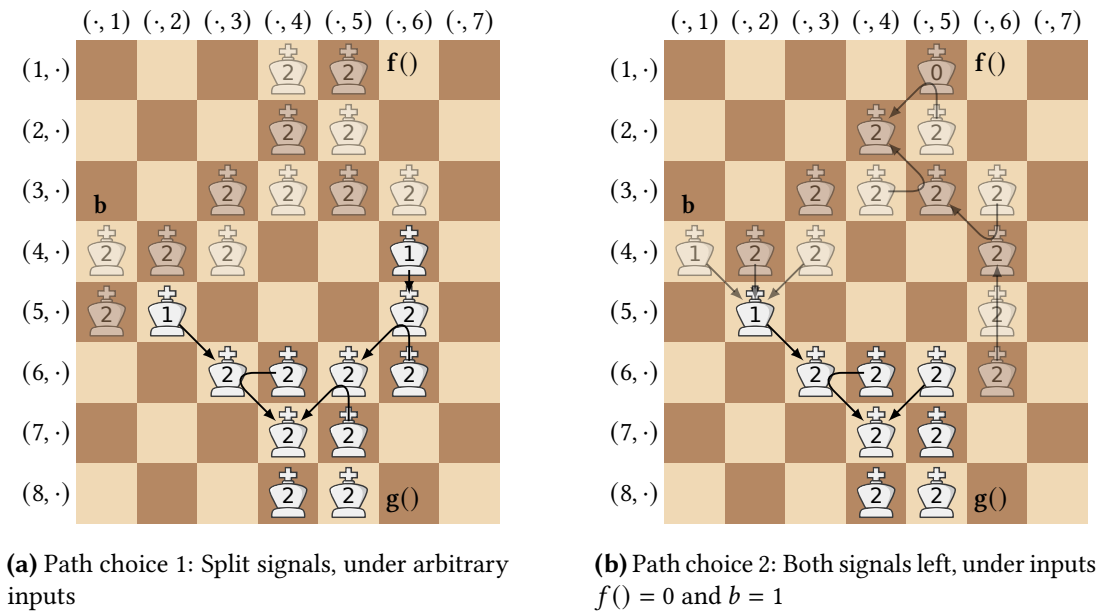


Figure 5.18.: Some potential intermediate configurations for different path choices.

shared neighboring square, specifically, towards the (7, 4) square as seen in Figure 5.18a. This results in a 0-output, which shows that with split signal paths, the gadget does not evaluate to a too large output.

Claim 2: For any set of inputs, the function g gadget evaluates to an output no larger than given in Table 5.2, if both signals travel via the bottom left path of the gadget.

Proof. If both signals travel via the bottom left path of the gadget, then (5, 2) is a cut vertex along the shared path. Thus, in any solving sequence, it captures only once both input signals have reached it. We discuss the two relevant pairs of inputs $(f(), b)$, namely (1, 0) and (0, 1):

We begin with the scenario where the left input b is a 0-signal. Any solving sequence propagating the 0-king on (4, 1) via the bottom left path contains the captures $((4, 2) \rightarrow (4, 1) \rightarrow (5, 2)), ((4, 3) \rightarrow (5, 2) \rightarrow (6, 3))$. However, this cuts off the other signal, showing that with these inputs this path choice is impossible.

Next, consider the scenario of inputs $f() = 0$ and $b = 1$. This corresponds to evaluating $g(0, 1)$. Thus, we need to show that in this case, the gadget does not evaluate to an output of 2. To do so, we combine forward and retrograde analysis. Recall that in a solving sequence, the cut vertex on (5, 2) captures only once both signals have reached it. Observe that the move arrows of the non-transparent pieces in Figure 5.18b depict the only possible capture sequence to reach a 2-output from a configuration with a 1-king on (5, 2), using the captures $((5, 2) \rightarrow (6, 3)), ((6, 4) \rightarrow (6, 3) \rightarrow (7, 4)), ((6, 5) \rightarrow (7, 4))$. In particular, the configuration consisting of the non-transparent pieces is a necessary intermediate configuration to reach a 2-output. We show that this configuration is impossible to reach under the given inputs: Any capture sequence that reaches the configuration, propagates the (6, 6)-king all the way around the gadget, beginning with captures $((6, 6) \rightarrow (5, 6) \rightarrow (4, 6)), ((3, 6) \rightarrow (4, 6) \rightarrow (3, 5))$. This results in a familiar configuration with two 0-kings on (3, 5) and (1, 5) and only three 2-kings between them.

Thus, as usual, any solving sequence propagates the two 0-kings to a shared neighboring square, e.g. through moves $((2, 5) \rightarrow (1, 5) \rightarrow (2, 4))$, $((3, 4) \rightarrow (3, 5) \rightarrow (2, 4))$. Either choice of a shared neighboring square leaves the resulting 0-king stranded. Thus, we conclude that with the given inputs and path choice it is impossible to achieve a 2-output, as claimed.

Claim 3: For any set of inputs, the function g gadget evaluates to an output no larger than given in Table 5.2, if both signals travel via the right path of the gadget.

Proof. If both signals travel via the right path of the gadget, then $(4, 6)$ and $(5, 6)$ are cut vertices through which the signals pass. Thus, any solving sequence taking this path contains the move $((4, 6) \rightarrow (5, 6))$. We deduce the moves prior to the above capture. Any solving sequence contains a prior capture of either $((3, 6) \rightarrow (4, 6))$ or $((3, 5) \rightarrow (4, 6))$. In the latter case, after extending the capture sequence by a prior $((2, 5) \rightarrow (3, 6) \rightarrow (4, 6))$, both signals are blocked off from reaching the right path of the gadget. Thus, any solving sequence instead contains the moves $((3, 6) \rightarrow (4, 6) \rightarrow (5, 6))$, preceded by $((x, y) \rightarrow (3, 5) \rightarrow (4, 6))$ and, before that, some sequence of captures that propagates both signals onto said $(3, 5)$ -square. Figure 5.19 depicts a possible intermediate configuration using $(x, y) = (2, 5)$. We briefly outline why it is impossible to reach this intermediate configuration under two relevant pairs of inputs $(1, 0)$ and $(0, 1)$.

We begin with inputs $f() = 1$ and $b = 0$ and discuss the different choices for the (x, y) -square. If $(x, y) = (2, 4)$, then the only possible capture sequence under which the 1-king of the top input reaches the $(3, 5)$ -square is $((1, 5) \rightarrow (2, 5))$, $((3, 4) \rightarrow (2, 5) \rightarrow (3, 5))$. This blocks off the left input from reaching the right path of the gadget. The case $(x, y) = (2, 5)$ is the one depicted in Figure 5.19. It shows a transparent capture sequence which falls one capture (or one unit of budget) short of reaching the $(3, 5)$ -square with both signals. Manual checking yields that there is no capture sequence, with or without utilizing captures by the $(6, 3)$ -king, such that both signals actually reach the $(3, 5)$ -square. The case for $(x, y) = (3, 4)$ is similar.

Next, consider the scenario of $f() = 0$ and $b = 1$. In this case, any solving sequence propagates the 0-king of the first row through captures from and to the two second-row kings. This leaves only the $(3, 4)$ -king as possible (x, y) -king. We then observe that the $(3, 3)$ -king represents a cut vertex on the left signal's path to the right side. After being captured by the left signal, this $(3, 3)$ -king with a resulting budget of at most 1 cannot reach past the $(2, 4)$ -square, after which it is impossible for both 0-kings $(1, 5)$ and $(2, 4)$ to reach the square $(3, 5)$ with only a single 2-king on $(2, 5)$ available to propagate them. This shows that with the given inputs, this path choice is impossible.

Combining the statements of the three claims, we see that it is impossible to achieve an output larger than outlined in Table 5.2 for inputs of either $g(1, 0)$ or $g(0, 1)$, under any choice of paths. The upper bounds for the remaining sets of inputs follow by monotony. This shows the correctness of the function g gadget. ■

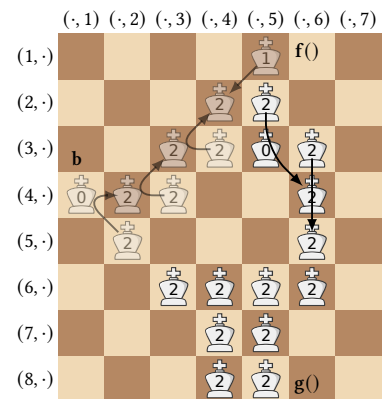
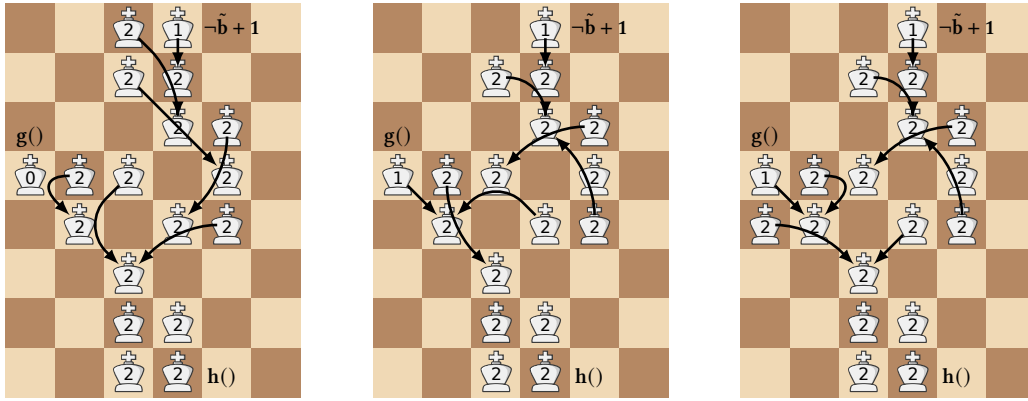


Figure 5.19.: Path choice 3: Both signals right, under inputs $f() = 1$ and $b = 0$, using $(x, y) = (2, 5)$



(a) Function h with inputs $g() = 0, -\tilde{b} = 1$ and output 0 (b) Function h with inputs $g() = 1, -\tilde{b} = 0$ and output 0 (c) Function h with inputs $g() = 2, -\tilde{b} = 0$ and output 1

Figure 5.21.: Evaluation of the (reduced) wire crossing function h under different sets of inputs. Note that the top Increment has already been evaluated.

5.6.2. Wire Crossing Function h

Our gadget for function h is shown in Figure 5.20. It consists of an Increment gadget on the top input $-\tilde{b}$, followed by the remaining gadget that combines the resulting top signal $-\tilde{b} + 1$ with the input $g()$. We show that this gadget represents a correct implementation of the function h .

Lemma 5.12: *For any set of inputs, the function h gadget evaluates to the value given in Table 5.2.*

Proof. Figure 5.21 gives capture sequences which show that for inputs $g() = 0, -\tilde{b} = 1$ or $g() = 1, -\tilde{b} = 0$ or $g() = 2, -\tilde{b} = 0$ the correct output value can be achieved. As usual, by monotony, the remaining sets of inputs can use the same capture sequences. It remains to show that no larger output values can be reached.

Observe that the output wire begins with a defect. It follows that the maximum possible output value is 1 which is reached if the input $g()$ is equal to 2. We further show that for inputs $g() = 0$ and $-\tilde{b} = 0$ the gadget cannot be cleared and for inputs $g() = 1$ and $-\tilde{b} = 1$ an output of 1 is impossible. The remaining cases then follow by monotony. We begin with the first scenario:

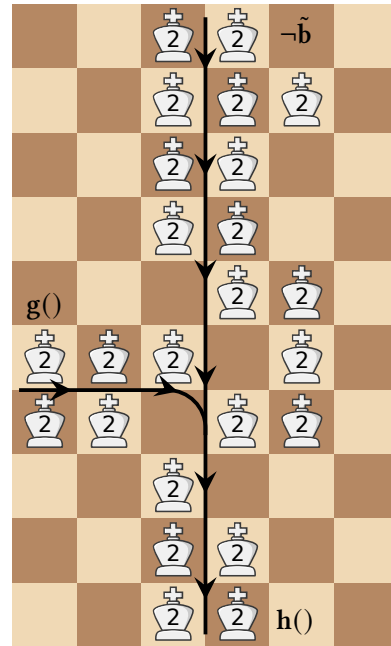


Figure 5.20.: Function h of the wire crossing gadget

For inputs $g() = 0$ and $-\tilde{b} = 0$ (i.e., $-\tilde{b} + 1 = 1$) the left 0-king can only be propagated by two double captures, as seen in Figure 5.22. Any other set of captures either leaves the 0-king stranded or disconnects the configuration. The capture graph of the remaining configuration contains the set of vertices of a wire with two defects and a subset of its edges (enclosed by the dashed lines in the figure). A wire with two defects turns an input value of 1 into an output value of -1, i.e., it is impossible to clean up all the pieces. Since the actual configuration offers only a subset of possible captures, any capture sequence in it would also be legal in the “normal” wire. It follows from the above that it is also impossible to clean up said remaining configuration.

The other case is that of inputs $g() = 1$ and $-\tilde{b} = 1$ (i.e., $-\tilde{b} + 1 = 2$). For this case we once more employ the method of retrograde analysis: Assume the output is a 1-signal as shown in Figure 5.23. The final move cannot have been $((5, 2) \rightarrow (6, 3))$ since that would leave an earlier antenna of length 2 ending in a 1-king on $(4, 1)$, which by Lemma 3.6 cannot have been resolved. Thus, the final move was $((5, 4) \rightarrow (6, 3))$ as indicated in the figure. Then the only possible un-capture piece is a 0-king on $(6, 3)$. If the previous double move had been $((4, 3) \rightarrow (5, 2) \rightarrow (6, 3))$, this would leave the top right side locked out from reaching the cut vertex on $(6, 3)$. Thus, the previous double move instead was $((4, 2) \rightarrow (5, 2) \rightarrow (6, 3))$ and at some earlier time $((4, 1) \rightarrow (5, 2))$. In this configuration, the only possible un-capturer is the king on $(5, 2)$. The only remaining possible un-captures are $((3, 4) \rightarrow (4, 3) \rightarrow (5, 2))$, which again locks out the top right side. Thus, we reach a contradiction and see instead that for these inputs we cannot achieve a 1-output.

By monotony, this result extends to the inputs of $g() = 0$, $-\tilde{b} = 1$ and $g() = 1$, $-\tilde{b} = 0$. This shows the correctness of the function h gadget. ■

5.6.3. The Full Wire Crossing Gadget

Using the above tools, we now construct the full wire crossing gadget. It can be seen in Figure 5.24.

Lemma 5.13: *Given inputs a and b , the wire crossing gadget produces outputs $\tilde{a} = a$ and $\tilde{b} = b$.*

Proof. As shown above, functions f , g and h combine to compute the function F . We discuss the different possible inputs for F . Any solving sequence matches an input of $b = 0$ with an output of $\tilde{b} = 0$: Assigning \tilde{b} the value 1 would assign $-\tilde{b}$ the value 0, which, by definition of F , leads to an error case, meaning that the gadget cannot be cleared. Outside this error case,

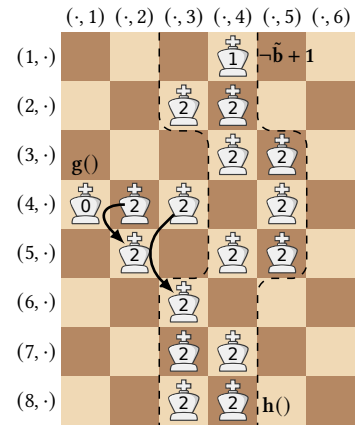


Figure 5.22.: The reduced configuration from scenario 1 is impossible to resolve.

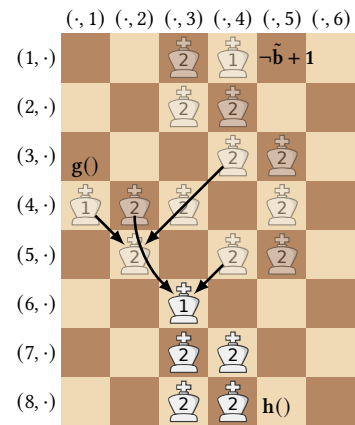


Figure 5.23.: The goal configuration for scenario 2 is impossible to reach.

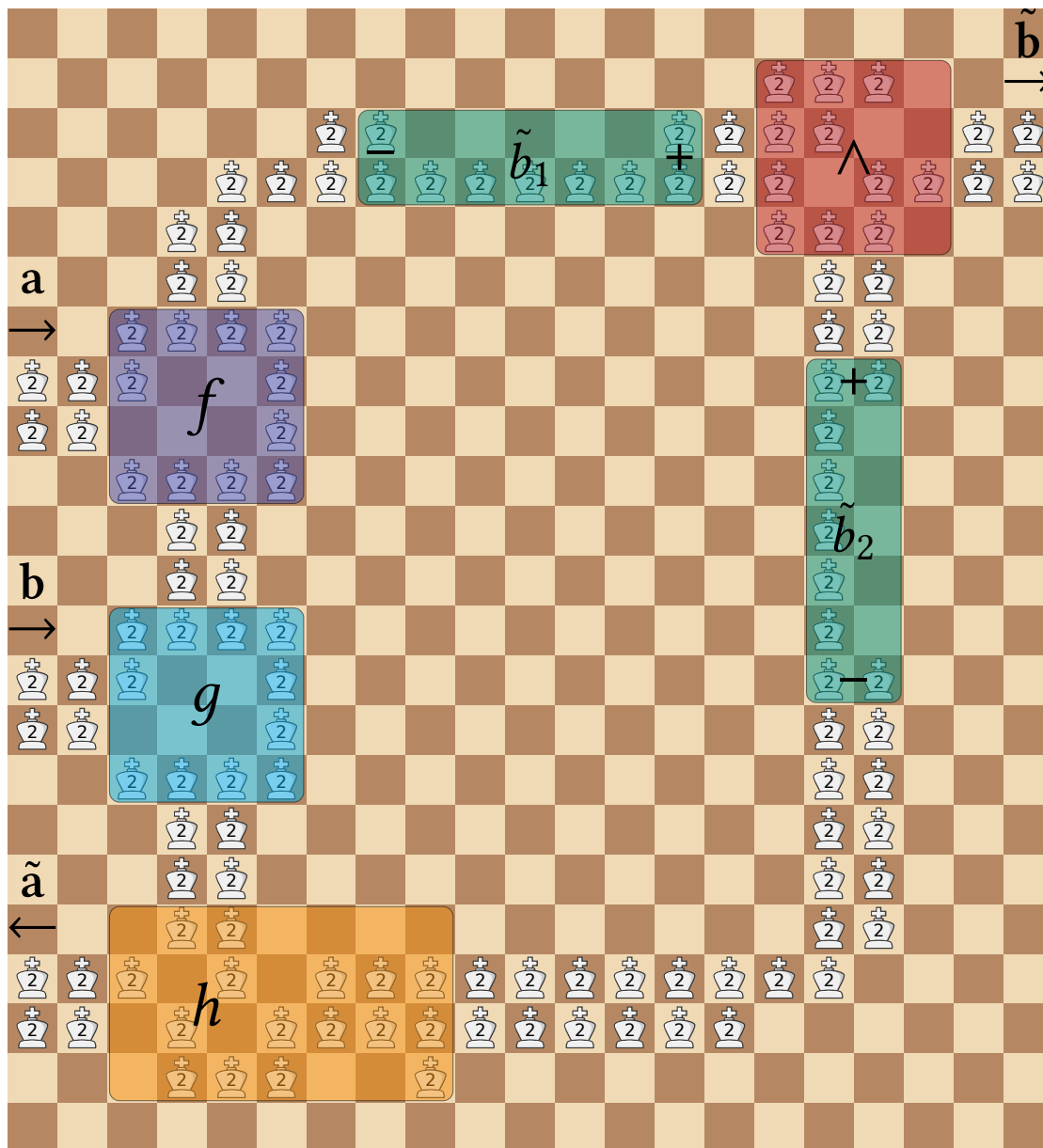


Figure 5.24.: The full wire crossing gadget.

function F is the identity on a . Thus, it correctly produces an output of $\tilde{a} = a$, as intended. The remaining scenario is that of an input of $b = 1$. In this case, both $\tilde{b} = 0$ and $\tilde{b} = 1$ are valid outputs that do not lead to an error. As usual, we assume that any gadget evaluates to the maximum possible output value. Thus, in this case, the wire crossing gadget produces an output of $\tilde{b} = 1 = b$ and so the input b is correctly reproduced on the other side of the wire. ■

5.7. The 1-Test Gadget

The 1-test gadget serves two purposes: It contains the final square of the instance, thereby ensuring that every other gadget needs to be evaluated and fully cleared. Furthermore, it ensures that the full set of clauses is satisfied, by only being reducible to a single square under an input of at least 1. It has one input and no outputs, and consists of an antenna of length 2 connected to an incoming wire, as seen in Figure 5.25.

Lemma 5.14: *The 1-test gadget can be reduced to a single piece if its input is at least 1. Otherwise, it is impossible to reduce the gadget to a single piece.*

Proof. Figure 5.25 shows a possible capture sequence that achieves the first claim. To show the second claim, we observe that the kings of the final column combine with one of the wire kings to form an antenna of length three. By Corollary 3.7, we conclude that this antenna contains the final square of the instance. In particular, for any capture sequence, the bottom left king does not leave the final column. Thus, to reduce the gadget to a single piece, any solving sequence propagates the wire signal to that final column. In case of a 0-signal, there is no such capture sequence, since propagating the signal results in a penultimate column holding only a 0-king, which remains stranded. ■

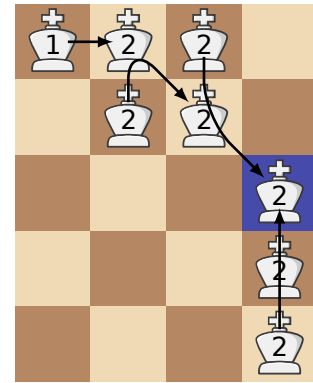


Figure 5.25.: The 1-test gadget reduces to a single piece if and only if it has an input of (at least) 1. The final square is highlighted in blue.

This concludes our discussion of the various gadgets. It remains to correctly put the pieces together.

5.8. The Final Reduction

We now describe our transformation of an And-Or-(1,1)-SAT instance into a King 2-SOLO CHESS instance. Let an And-Or-(1,1)-SAT instance $I = (U, C)$ be given. Choose an And-Or Embedding of I that includes the optional conjunction of the outputs of the clauses. For every variable $u \in U$, place a variable assignment gadget on the board at sufficient distance to all other variable assignment gadgets. For each clause, place the respective logic gadgets on the board according to the embedding, and place wire on the board to connect their inputs with the correct outputs of variable assignments or other logic gadgets. Place the remaining And gadgets that combine the outputs of the clauses, according to the embedding. Where necessary, place wire crossing gadgets to allow wires to cross. Finally, place a 1-test gadget and connect it to the output of the final And gadget. An example of such a placement on the board can be seen in Figure 5.1. Using this construction, we now prove:

Theorem 5.15: *King 2-SOLO CHESS is NP-hard.*

Proof. We reduce from And-Or-(1,1)-SAT. We transform a given instance as described above. Note that this transformation runs in polynomial time. Let n be the number of variables and m be the number of clauses of the SAT instance. For each variable, a single gadget is placed on the board, for a total of n gadgets. For each clause, at most two logic gate gadgets are

placed on the board, for a total of at most $2m$ gadgets. To combine the outputs of all clauses, $m - 1$ And gadgets are placed on the board. Including the final 1-test, the number of placed gadgets is in $\mathcal{O}(n + m)$. In addition, in our And-Or Embeddings, each of the $\mathcal{O}(n + m)$ wires connecting two gadgets has length $\mathcal{O}(n + m)$ and crosses at most $\mathcal{O}(n + m)$ other wires. Each gadget consists of only a constant number of kings. Thus, the total number of kings is in $\mathcal{O}((n + m) + (n + m)^2)$, which is polynomial in the input size. Furthermore, the transformation can be computed in time proportional to its output size. It remains to show that the two instances are in fact equivalent.

SAT instance solvable \implies Chess instance solvable: Let Φ be a satisfying assignment for the SAT instance. Then a solving sequence consists of the following steps: For each variable assignment gadget, assign the value 1 to the literal that is set to true by Φ and the value 0 to the other literal (see Lemma 5.7). Propagate the signal of each literal through its wire and possible wire crossing gadgets to the logic gadget of its clause. By Lemmata 5.3, 5.6 and 5.13 each signal remains unchanged in this process. For each clause, evaluate each logic gadget and propagate the resulting signal through the outgoing wire to the input of the next logic gadget. Since each clause was satisfied by Φ , by Lemmata 5.9 and 5.10, each outgoing wire from the final gadget of a clause receives a 1-signal. Then, evaluate each of the additional And gadgets which combine the different clause outputs. Since each clause evaluated to 1, the sequence of And gadgets by Lemma 5.10 once more yields an output of 1. Finally, propagate the resulting 1-signal to the 1-test gadget, which is reduced to a single final piece (see Lemma 5.14). Recall that whenever any other gadget is evaluated, none of its pieces remain. Thus, only the final king of the 1-test gadget remains, meaning that this is a valid solving sequence.

Chess instance solvable \implies SAT instance solvable: Let a solving capture sequence for the chess instance be given. Then the following procedure yields a satisfying assignment Φ for the SAT instance: For each variable assignment gadget, check which of the output wires is assigned a 1-signal. If the wire corresponds to some literal $\ell = x$, set $\Phi(x) := \text{true}$. If it corresponds to some literal $\ell = \neg x$, set $\Phi(x) := \text{false}$. If neither of the two wires is assigned a 1-signal (through some “suboptimal” capture sequence), the value of the variable does not matter, so we can for example set $\Phi(x) := \text{true}$. Then Φ is a satisfying assignment:

By Lemma 5.7, at most one of the wires of each variable assignment gadget is assigned a 1-signal. Thus, Φ is a well-defined. We also see that every clause is satisfied: The solving capture sequence reduces the configuration to a single piece. Thus, by Lemma 5.14 the 1-test gadget has received an input of (at least) 1. By Lemma 5.10 it follows that each of the And gadgets that combine the outputs of clauses has received two 1-signals as inputs. This means that each clause in the SOLO CHESS instance has evaluated to 1. Then, by Lemmata 5.9 and 5.10, under the variable assignments given by Φ , each clause evaluates to 1, i.e., true. It follows, that in the SAT instance, each clause is satisfied under Φ .

Overall, we conclude that the two instances are equivalent, which shows the NP-hardness of King 2-SOLO CHESS. ■

6. Knight 2-SOLO CHESS

In this chapter, we extend our study of 2-SOLO CHESS on a two-dimensional board. We show that the NP-hardness result for the king case also holds for instances containing only knights. The overall reduction is similar to that of King 2-SOLO CHESS. We reduce from And-Or-(1,1)-SAT and directly translate an And-Or Embedding into a corresponding chess position. To this end, we once again present gadgets for wires, wire crossings, Or-gates, And-gates and a 1-test. As before, any solving sequence fully clears all gadgets except the 1-test.

6.1. The Wire

We begin with the wire. It is constructed similarly as in the King 2-SOLO CHESS reduction, however, due to the “strange” movement of the knight, it has a few peculiarities. Figure 6.1 shows the capture graph of a wire as well as its placement on the chess board. We define pairs of consecutive vertices sharing the same square color, such as 1 and 2 or 3 and 4, as (dark-squared or light-squared) *wire-columns*. Note that a knight always moves from a dark square to a light square, or from a light square to a dark square. As a result, the square colors of consecutive wire-columns alternate. We call vertices which (originally) have degree four *inner vertices* (or *inner knights*) and vertices which have degree two *outer vertices* (or *outer knights*). Every wire-column consists of an inner and an outer vertex. Note that while the wire is being cleared, the degree of its vertices changes. However, we keep referring to vertices as inner or outer vertices based on their original degree. In our vertex numbering, those of the form $4k + 1$ or $4k + 4$ for some $k \in \mathbb{N}_0$ are outer vertices, and those of the form $4k + 2$ or $4k + 3$ are inner vertices.

The wire is once again used to propagate signals. These signals differ from the King 2-SOLO CHESS signals in that they have two components: We define signal values as elements of $\{0, 1\}^2$ with $(0, 0)$ representing false, $(1, 0)$ representing true and auxiliary values $(0, 1)$ and $(1, 1)$. In Figure 6.1 we call the wire-column containing vertices 1 and 2 the signal column if all previous wire-columns have been cleared. Note that for parity reasons, only every other wire-column, namely every dark-squared one, is a signal column at some point. Similar to King 2-SOLO CHESS, we can read off signal values in the signal column. The value of the first

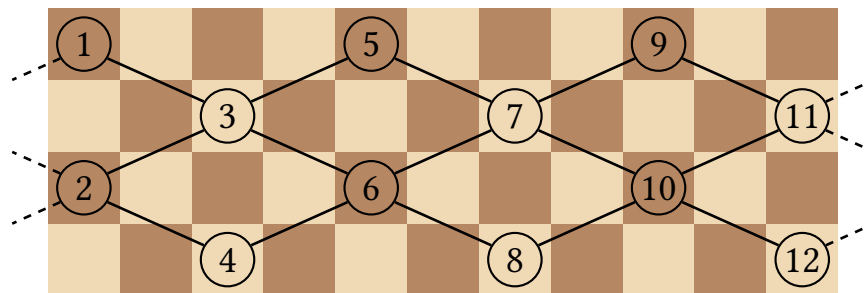


Figure 6.1.: The Knight Wire placed on a chess board.

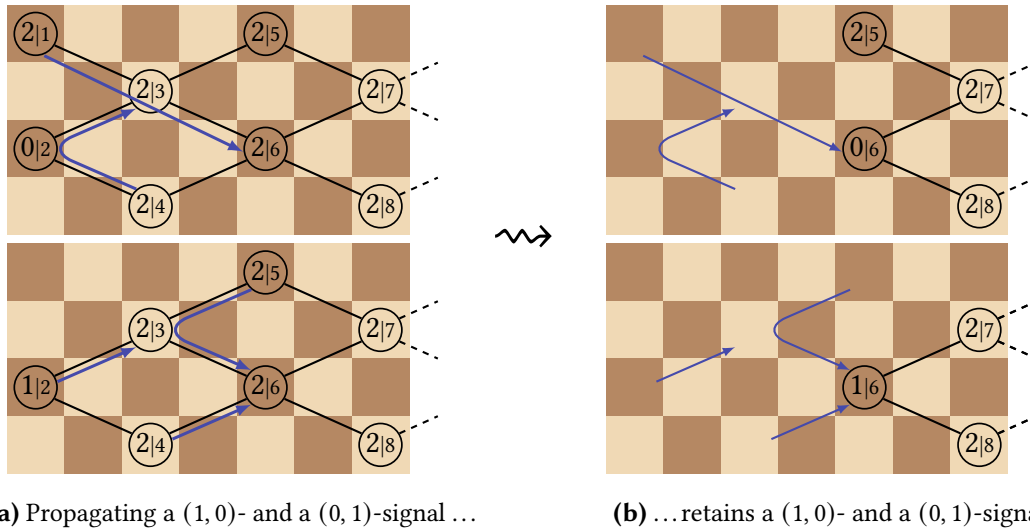


Figure 6.2.: Two Knight Wires propagating a signal each. Each vertex shows its budget followed by its vertex number.

component depends on the state of the outer vertex, while the value of the second component depends on the state of the inner vertex of the signal column. Figure 6.2 shows a $(1, 0)$ - and a $(0, 1)$ -signal. In general, the value of a signal can be read off as follows: If the outer vertex of the signal column is present with a budget of 2, this corresponds to a first signal component of 1. If it is not present at all, this corresponds to a first signal component of 0. Meanwhile, under correct use the inner vertex of the signal column is always present, either with a budget of 0 or 1. The second signal component is the value of this inner vertex budget, so again either 0 or 1. We show below that these are the only possible signals that a wire can propagate.

A signal is being propagated by the outer vertices of the wire capturing towards the inner vertices of the wire. Figure 6.2 shows a $(1, 0)$ - and a $(0, 1)$ -signal being propagated by two wire-columns each. More generally, if the first component of a signal is 1 (as in the top wire) the outer vertex of the signal column captures forward, otherwise (bottom wire), it captures backward. Analogously, if the second component of a signal is 1 (bottom wire), the outer vertex of the next wire-column captures forward, otherwise (top wire) it captures backward. Similarly, a $(1, 1)$ -signal is propagated by both outer vertices capturing forward, while a $(0, 0)$ -signal is propagated by both outer vertices capturing backward. It follows that once more there is a monotony on signal values: Using the example of the bottom wire of Figure 6.2, by immediately capturing with vertex 4 through captures $(4 \rightarrow 2 \rightarrow 3)$, this $(0, 1)$ -signal can be turned into a $(0, 0)$ -signal. Analogously, for the top wire, the first component can be decremented from 1 to 0 by capturing $(4 \rightarrow 2 \rightarrow 3)$, $(1 \rightarrow 3)$, $(5 \rightarrow 3 \rightarrow 6)$. Note, however, that while it holds that $(0, 0) \leq (1, 0) \leq (1, 1)$ and $(0, 0) \leq (0, 1) \leq (1, 1)$, the signal values $(1, 0)$ and $(0, 1)$ are incomparable. When discussing gadgets, we still assume that any gadget evaluates to the maximum possible value. This is ill-defined if a gadget can evaluate to either $(1, 0)$ or $(0, 1)$, however, this case does not occur in our construction. We show that the wire works correctly in the sense that it propagates values without modification.

Lemma 6.1: *Propagating a signal through a wire does not change its value.*

Proof. We have outlined in Figure 6.2 how to propagate a signal value such that it does not decrease. To see that the signal value cannot increase, we remark that the invariant presented in the King 2-SOLO CHESS reduction works analogously for Knight 2-SOLO CHESS. In particular, the value $s = c - n + 1$ for a wire-column containing n knights with a combined budget of c counts precisely the number of 1-components of the signal value. As before, every wire-column contributes at most two units of budget and adds two knights, thus, the value of the invariant never increases. This shows that a signal value never increases. It remains to show that it is impossible to transform the value $(1, 0)$ into the value $(0, 1)$ and vice versa. For this, we first show:

Claim 1: If an inner vertex captures an outer vertex of a wire, that wire cannot be cleared.

Proof. Assume for contradiction that there exists a clearing capture sequence (without loss of generality, from left to right) where an inner vertex captures an outer vertex. Fix the left-most inner vertex v that captures an outer vertex w . Consider the case that at the time of the capture v has a budget of less than 2. Then after capturing w , said vertex w becomes a 0-leaf and thus stranded, a contradiction. Next, consider the case where at the time of the capture v has a budget of 2. Since the wire is cleared left to right, at least one vertex of the wire-column to the left of v captures a vertex of the wire-column of v . Since v is the left-most (inner) vertex that captures an outer vertex, the vertices of the wire-column to its left do not capture the outer vertex of the wire-column of v . However, based on its budget of 2 at the time of capturing, v itself has not been captured either. We again reach a contradiction, which shows that the assumption is false. It follows directly that in any clearing capture sequence, no inner vertex captures an outer vertex.

Consider now the capture sequence of a $(1, 0)$ -signal, as shown in Figure 6.2. We show that there is no deviation from this sequence that results in a $(0, 1)$ -signal: Vertex 2 can only leave its square through being captured by one of its neighbors, followed by capturing the other neighbor. By Claim 1, the second of these captures cannot be towards an outer vertex. Thus, the given pair of captures $(4 \rightarrow 2 \rightarrow 3)$ is the only one that does not leave a piece stranded. It results in vertex 3 having a budget of 0. This vertex 3 can proceed by the pair of captures as shown in the figure, which restores a $(1, 0)$ -signal. Performing instead a pair of captures $(5 \rightarrow 3 \rightarrow 6)$ leaves vertex 1 stranded. Finally, performing a single capture $(1 \rightarrow 3)$ and then a pair of captures $(5 \rightarrow 3 \rightarrow 6)$ merely decrements the $(1, 0)$ -signal to a $(0, 0)$ -signal. This covers all possible clearing capture sequences. It follows that a $(1, 0)$ -signal cannot be turned into a $(0, 1)$ -signal.

Next, we show that there is no deviation from the capture sequence of a $(0, 1)$ -signal that results in a $(1, 0)$ -signal: In a $(0, 1)$ -signal, the signal column consists of only a single inner 1-vertex. Propagating this 1-vertex through a pair of captures $(4 \rightarrow 2 \rightarrow 3)$ again decrements the signal to a $(0, 0)$ -signal. Otherwise, by Claim 1, vertex 2 captures $(2 \rightarrow 3)$. As above, the resulting 0-vertex can only be propagated through a pair of captures $(5 \rightarrow 3 \rightarrow 6)$. What remains is a 2-leaf in vertex 4 and a 0-vertex in vertex 6. Capturing $(8 \rightarrow 6)$ followed by $(4 \rightarrow 6 \rightarrow 7)$ once more decrements the signal to a $(0, 0)$ -signal. The remaining possibility of immediately capturing $(4 \rightarrow 6)$ restores the original $(0, 1)$ -signal. Thus, it is impossible to turn a $(0, 1)$ -signal into a $(1, 0)$ -signal which concludes the final case and finishes the proof. ■

We observe that there are no other signals than the ones described above: An outer vertex can be present with a budget of 2 or not be present at all. Since it is never captured, it is never present with any other budget. Conversely, from the previous proof, we know that every inner vertex is captured by a vertex to its left. Thus, when it is part of a signal column, it has a budget 0 or 1.

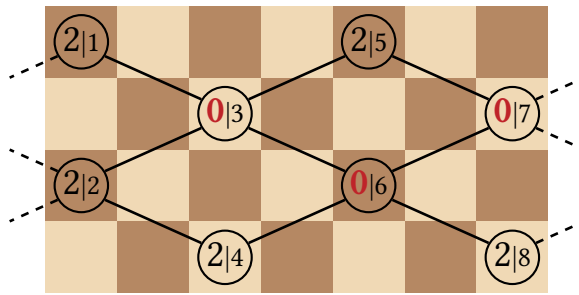


Figure 6.3.: A wire with reduced vertex budgets.

Since each inner vertex is captured, and so has its previous budget replaced by that of the capturing vertex, we observe:

Observation 6.2: *Reducing the budget of an inner vertex outside the signal column does not change the functionality of the wire. In particular, the wire shown in Figure 6.3 propagates a signal without changing its value.*

This proves useful when constructing larger gadgets that contain a wire.

We give a number of possible placements of the capture graph on the chess board. We have already seen in Figure 6.1 the “standard” placement of a (*thin*) wire. We give further placements of the capture graph which serve different purposes. Figure 6.4 shows a wire turning a corner. Since the capture graph of the wire corner is the same as that of a standard wire, it behaves exactly like a standard wire and does not change the value of the signal it propagates.

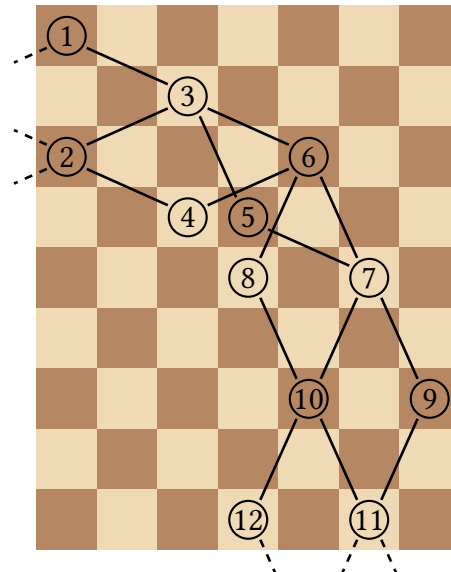
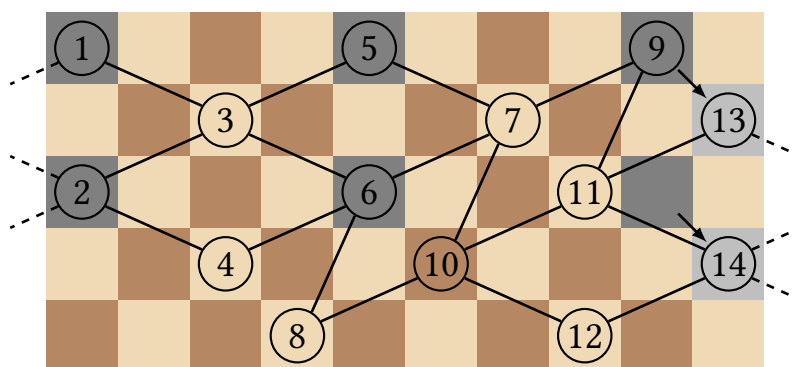


Figure 6.4.: A wire turning a corner.

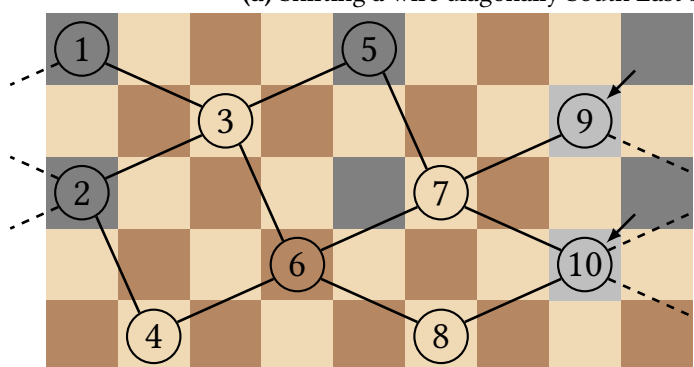
Figures 6.5a and 6.5b show further placings, which once more have the same capture graph. These placings (and their mirrored versions) allow us to shift the wire as needed so that it is aligned correctly with the inputs to the yet to be defined function gadgets. More specifically, the first of these offsets the wire diagonally down and to the right by one square relative to the standard wire. The latter offsets the wire diagonally down and to the left by one square relative to the standard wire. Note that since wire-columns always alternate in color, it is impossible to offset a wire only by one square to the right or only by one square down, as that would change a signal column’s color.

Figure 6.6 shows a mirrored wire. It too has the same capture graph as the standard wire, however, it ends with a mirrored version of the wire where even-numbered pieces are on top and where the light-squared wire-columns are one row further up than the dark-squared wire-columns. To maintain the color parity, this includes a shift by one square to the right.

Next, we briefly discuss wire crossings. To this end we first show a placement of the wire that is stretched in the vertical dimension, a *thick* wire as seen in Figure 6.7. Using this thick wire, we can create a wire crossing by simply placing the two wires to be crossed appropriately, as seen in Figure 6.8. Observe that none of the knights of the blue wire can capture any knight



(a) Shifting a wire diagonally South East by one.



(b) Shifting a wire diagonally South West by one.

Figure 6.5.: Two wire shifts. Dark gray squares show the (would-be) positions of the signal columns in a standard wire. The light gray squares show the actual position of the right-most signal column of the placement.

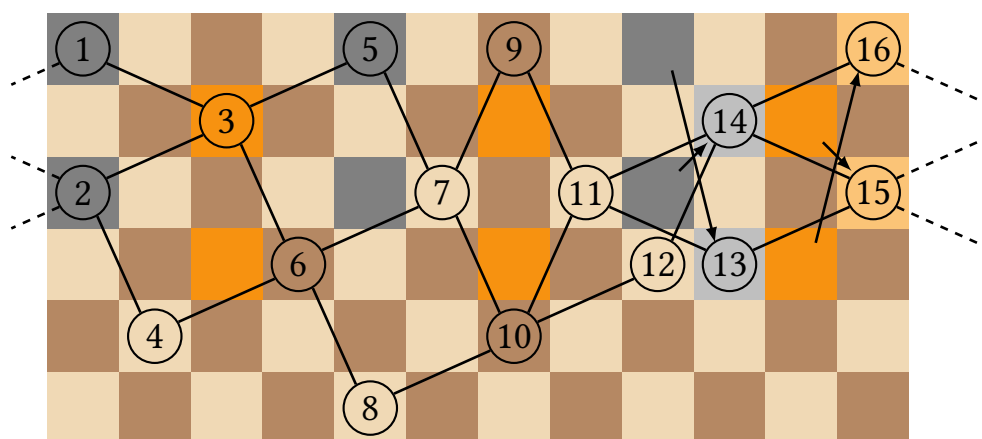


Figure 6.6.: A mirrored wire, ending with even numbers on top and odd numbers at the bottom. Dark gray and orange show the (would-be) positions of the signal and non-signal columns, respectively.

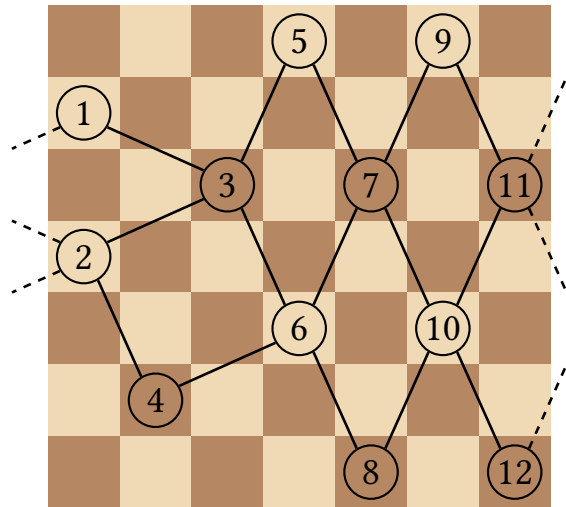


Figure 6.7.: Transitioning from the standard wire placement to the alternative *thick* wire.

of the red wire and vice versa. Thus, the two wires do not interact in any way and simply propagate their values as usual.

This finishes our discussion on wires. We have seen various placements of the wire capture graph on the board to create a standard wire, corners, shifts and mirrored wires. By combining these, it is possible to propagate a signal to a signal column in arbitrary position and orientation on the board. This allows us to create a wire crossing gadget without the need for auxiliary functions, as well as construct various further gadgets.

6.2. Auxiliary Gadgets

In this section, we describe a number of auxiliary gadgets which are later composed to create the desired SAT gadgets.

6.2.1. Signal manipulation gadgets

We begin with gadgets that manipulate the value of a signal. Analogous to King 2-SOLO CHESS we describe the Increment and Decrement gadgets. Figures 6.9 and 6.10 show an Increment and a Decrement gadget for the first signal component. The corresponding second component Increment gadget instead adds a knight on a light square that is adjacent to a dark inner vertex. Similarly, the second component Decrement gadget removes the outer knight of a light-squared wire-column.

Lemma 6.3: *The Increment gadget sets a specific component of a signal to 1, regardless of its previous value. In particular, the dark-squared Increment gadget acts on the first, while the light-squared Increment gadget acts on the second component of the signal.*

Proof. We show the claim for the dark-squared Increment gadget. We first show that it is possible to set the first component of the signal to 1, then, that this is the best possible result. Consider the case where the first component of the incoming signal is 0, shown in Figure 6.11. Then the standard propagation of the signal contains the move pair $((1, 5) \rightarrow (2, 3) \rightarrow (3, 5))$.

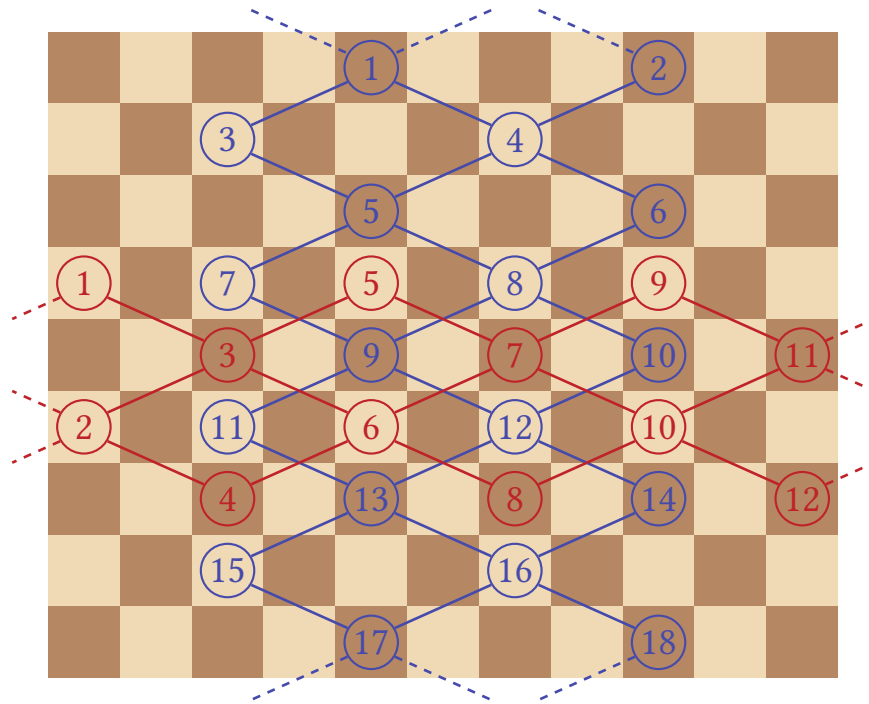


Figure 6.8.: A wire crossing of a red and a blue wire. Note that no red piece can capture any blue pieces and vice versa.

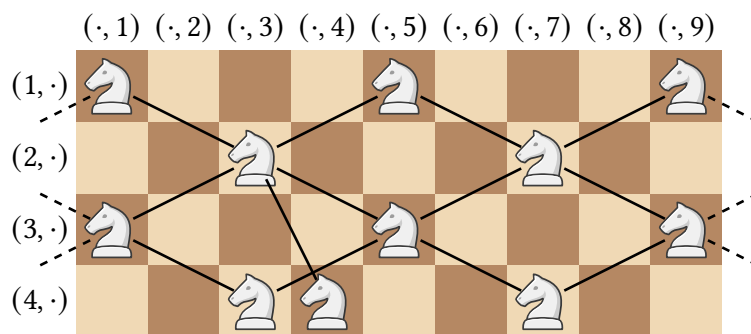


Figure 6.9.: The dark-squared Increment gadget sets the first component of the signal to 1.

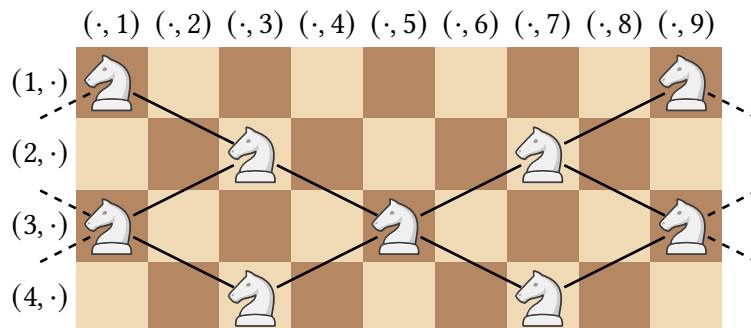


Figure 6.10.: The dark-squared Decrement gadget decrements the first component of the signal.

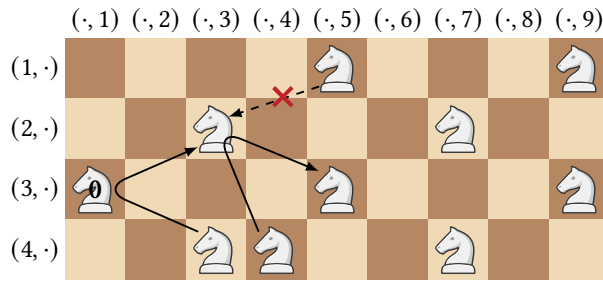


Figure 6.11.: Incrementing the first component of a $(0, 0)$ -signal.

Replacing it by $((4, 4) \rightarrow (2, 3) \rightarrow (3, 5))$ sets the first component of the signal to 1 by retaining the $(1, 5)$ -knight. Note that the second component is unchanged by this. By monotony, this also holds when the first component of the signal is already set to 1.

To show that this is the best possible result, by monotony, it suffices to show that a dark-squared Increment gadget cannot turn a $(1, 0)$ -signal into a $(x, 1)$ -signal for any $x \in \{0, 1\}$. Observe that it still holds that no outer vertex can be captured. Thus, any clearing capture sequence contains the pair of moves $((4, 3) \rightarrow (3, 1) \rightarrow (2, 3))$. Then, both the $(1, 1)$ - and the $(4, 4)$ -knights are leaves, and capture the $(2, 3)$ -knight at some point to clear the wire. Once the eventual 1-knight on $(2, 3)$ captures the next inner knight on $(3, 5)$, this results in a 0-knight, i.e., a second component of 0. Thus, it is impossible to increment the second component of a signal with a dark-squared Increment gadget.

The proof for the light-squared Increment gadget is similar. ■

Lemma 6.4: *The Decrement gadget reduces a specific component of a signal by 1, that is, it turns a 1-component into a 0-component while turning a 0-component into a -1 -component, i.e., leads to a stranded piece. The dark-squared Decrement gadget acts on the first, while the light-squared Decrement gadget acts on the second component of the signal.*

Proof. We show the claim for the dark-squared Decrement gadget of Figure 6.10. If the incoming signal has a first component of 1, then once reaching the Decrement column, the first component becomes 0 since that signal column does not contain an outer knight in the first place. Note also that there is no capture sequence that increases the second component in return, as otherwise this sequence would also be valid in a standard wire. If the incoming signal has a first component of 0, then regardless which third wire-column knight the $(3, 1)$ -knight captures, the resulting 0-knight is a leaf and, thus, stranded. Similar to King 2-SOLO CHESS, we denote the resulting signal as $(-1, x)$ to indicate an error state.

Once again, the proof for the light-squared Decrement gadget is similar. ■

The Increment gadget sets a component of a signal to 1, regardless of its previous value. We create the converse Zeroing gadget, which sets a component of a signal to 0. In particular, it differs from the Decrement gadget by not turning a 0-component into an error state -1 . Instead, to construct the Zeroing gadget, we combine the Increment and the Decrement gadget, as seen in Figure 6.12. After setting the component to 1 using the Increment gadget, it is safe to use the Decrement gadget to set it to 0, as desired. In figures, we denote the Zeroing gadget that sets the first component of a signal to 0 by “ $\cdot(0, 1)$ ”. The converse $\cdot(1, 0)$ -gadget is created by combining a $+ (0, 1)$ - and a $- (0, 1)$ -gadget. We have seen:

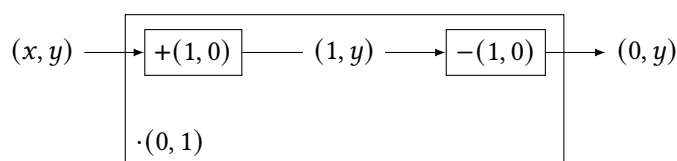


Figure 6.12.: The dark-squared Zeroing gadget sets the first component of a signal to 0.

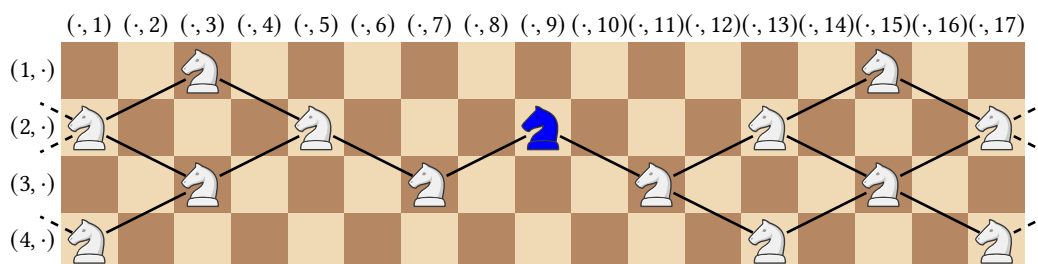


Figure 6.13.: The three valued production gadget produces left and right outputs $(1, 1) \leftrightarrow (0, 0)$ or $(1, 0) \leftrightarrow (0, 1)$ or $(0, 0) \leftrightarrow (1, 1)$.

Lemma 6.5: *The dark-squared Zeroing gadget sets the first component of a signal to 0, regardless of its previous value. The light-squared Zeroing gadget sets the second component of a signal to 0, regardless of its previous value.*

6.2.2. Value Production Gadgets

We use value production gadgets to populate the wires with values. We begin with a three valued production gadget (*3-Val* for short), shown in Figure 6.13.

Lemma 6.6: *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.*

Proof. Consider the central knight marked in blue. If it is captured by the knight to its right, this creates a knight with a budget of at most 1 as the final piece of an antenna of length 2. By Lemma 3.6, this antenna is stranded. Thus, since every solving sequence fully clears every SAT gadget, the blue knight is never captured by the knight to its right. However, it can be captured by the knight to its left through a capture $((3, 7) \rightarrow (2, 9))$ which turns it into an antenna of length 1 with a 1-leaf. After resolving the antenna with another leaf capture, there remains a $(1, 1)$ -signal on the left and a $(0, 0)$ -signal on the right, 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 $((2, 9) \rightarrow (3, 7) \rightarrow (2, 5))$, this creates a $(0, 0)$ -signal on the left and a $(1, 1)$ -signal on the right by an analogous argument. If it captures to the right through $((2, 9) \rightarrow (3, 11))$, this creates a $(0, 1)$ -signal on the right. Then, after a leaf capture on the left, 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. ■

Building on the 3-Val gadget, we create an asymmetric value production gadget, shown in Figure 6.14.

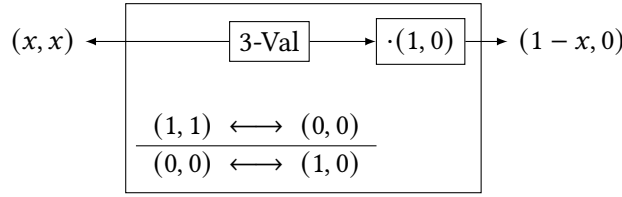


Figure 6.14.: The asymmetric value production gadget produces left and right outputs $(1, 1) \leftrightarrow (0, 0)$ or $(0, 0) \leftrightarrow (1, 0)$.

Lemma 6.7: *The asymmetric value production gadget produces a $(1, 1)$ -signal to the left and a $(0, 0)$ -signal to the right, or a $(0, 0)$ -signal to the left and a $(1, 0)$ -signal to the right. An alternative version of the gadget produces a $(1, 1)$ -signal to the left and a $(0, 0)$ -signal to the right, or a $(0, 0)$ -signal to the left and a $(0, 1)$ -signal to the right.*

Proof. Consider the different options for evaluating the 3-Val gadget. It can produce outputs $(1, 1) \leftrightarrow (0, 0)$. In this case, zeroing the second component of the right output does not modify it, as such this yields the first of the two claimed possible outputs. The 3-Val gadget can produce outputs $(1, 0) \leftrightarrow (0, 1)$. In this case, zeroing the second component of the right output turns it into a $(0, 0)$ -signal. This yields an overall output of $(1, 0) \leftrightarrow (0, 0)$ which is smaller than the first option and thus, by monotony, is not chosen. Finally, the 3-Val gadget can produce outputs $(0, 0) \leftrightarrow (1, 1)$. In this case, zeroing the second component of the right output turns it into a $(1, 0)$ -signal, yielding the second possible output.

Modifying our construction by using a $\cdot(0, 1)$ gadget instead of a $\cdot(1, 0)$ gadget yields an analogous gadget producing outputs $(1, 1) \leftrightarrow (0, 0)$ or $(0, 0) \leftrightarrow (0, 1)$. ■

6.2.3. The Maximum Gadget

Next, we describe the Maximum gadget seen in Figure 6.15. It computes the component-wise maximum of the two input signals.

Lemma 6.8: *The Maximum gadget computes the component-wise maximum of its two input signals. I.e., for inputs (u, v) and (w, x) it produces the output $(\max\{u, w\}, \max\{v, x\})$.*

Proof. First, observe that the white knights on $(7, 4)$, $(9, 5)$, $(8, 7)$ form an antenna of length 2, which by Lemma 3.5 is reduced to a 0-knight on $(7, 4)$. We now show:

Claim 1: For a top (blue) input of $(0, 0)$, the Maximum gadget is the identity on the left (orange) input.

Proof. Figure 6.16 shows a suitable capture sequence. The resulting configuration consists of the orange wire with one inner knight turned into a 0-knight. By Observation 6.2 this modification does not change the value of the orange signal, thus, the wire acts as the identity function on the orange input.

To see that this is the best possible output, consider the alternative capture sequences. First, observe that after the two pairs of captures by the blue knights, the white 0-knight only has one neighbor that has a budget of 2. Thus, any clearing sequence contains the third pair of captures, as otherwise the white 0-knight would end up stranded. Next, we discuss the blue knight captures themselves, in particular, the path of the 0-knight on $(1, 5)$. It is possible to propagate it to $(5, 7)$ instead of $(5, 3)$, through captures $((3, 4) \rightarrow$

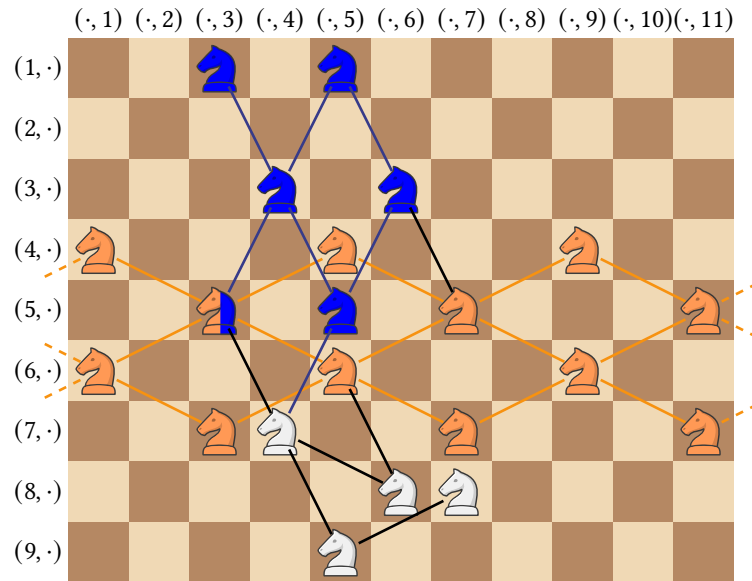


Figure 6.15.: The Maximum gadget computes the component-wise maximum of the blue and the orange signal.

$(1, 5) \rightarrow (3, 6)$, $((5, 5) \rightarrow (3, 6) \rightarrow (5, 7))$. Then, if the white 0-knight is propagated to $(8, 6)$, through captures $((5, 3) \rightarrow (7, 4) \rightarrow (8, 6))$, this creates a 0-leaf which is stranded. Otherwise, after propagating it to $(5, 3)$, as in the intended sequence, there again remains an orange wire that propagates the orange signal unmodified. Propagating the $(1, 5)$ -knight to $(5, 5)$ via the $(3, 4)$ or $(3, 6)$ -square produces a 0-leaf that is stranded. This covers all cases and shows the claim.

Next, we discuss the case where the blue input is greater than $(0, 0)$. We show:

Claim 2: For a blue input of $(1, 0)$, the Maximum gadget is a first component Increment on the orange input.

Proof. Figure 6.17 shows a suitable capture sequence. This sequence propagates the $(1, 5)$ -knight to $(5, 3)$ without needing to use the $(5, 5)$ -knight. This in turn allows the $(5, 5)$ -knight to take the role of the white 2-knight, freeing up that white knight to serve as a dark-squared increment.

To show that this is the best possible output, we need to show that there is no capture sequence that uses a blue $(1, 0)$ -input to increment the second component of the orange signal. We again consider alternative capture sequences. Deviating from the first pair of moves and propagating the blue 0-knight to $(3, 6)$ instead of $(3, 4)$, through captures $((3, 4) \rightarrow (1, 5) \rightarrow (3, 6))$, leaves the extra $(1, 3)$ -knight stranded. Thus, any clearing capture sequence propagates the 0-knight to $(3, 4)$ as in the given sequence. Propagating this $(3, 4)$ -knight to $(1, 3)$ or $(3, 5)$ leaves the resulting 0-knight stranded. Thus, any clearing capture sequence propagates the $(3, 4)$ -knight to $(5, 3)$. Deviating from the second pair of moves by interjecting the capture $((1, 3) \rightarrow (3, 4))$ followed by $((5, 5) \rightarrow (3, 4) \rightarrow (5, 3))$ merely transposes to the capture sequence for a blue $(0, 0)$ -signal which, as we have seen, does not increment the second component of the orange signal. Finally, we discuss deviating from the given sequence after the first two pairs of moves. Propagating the

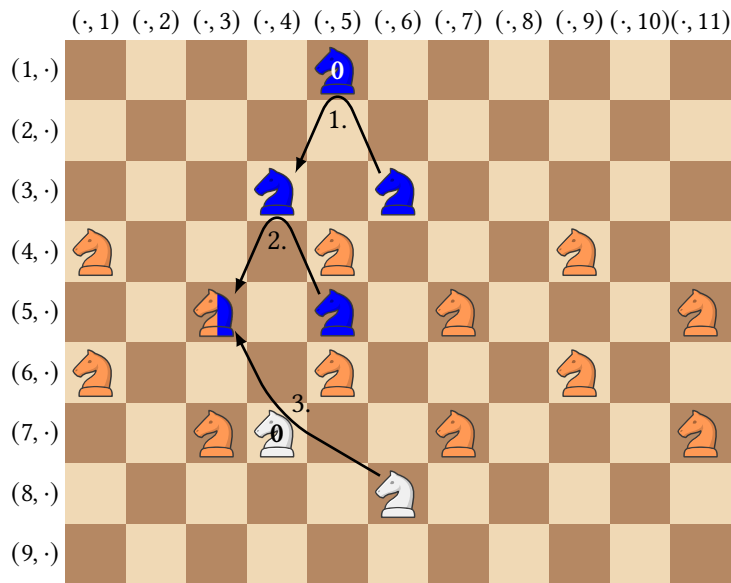


Figure 6.16.: The Maximum gadget with a top input of (0, 0).

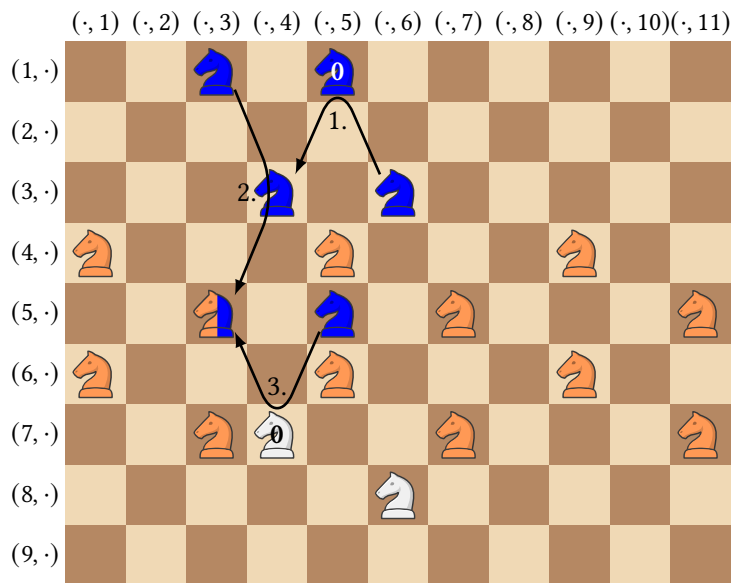


Figure 6.17.: The Maximum gadget with a top input of (1, 0).

white 0-knight from (7, 4) to (5, 5) or (8, 6), rather than to (5, 3), leaves the resulting 0-knight stranded. Propagating it to (5, 3) using captures ((8, 6) → (7, 4) → (5, 3)) leaves the (5, 5)-knight stranded. Finally, interjecting the capture ((5, 5) → (7, 4)) followed by captures ((8, 6) → (7, 4) → (5, 3)) once more transposes to the capture sequence for a blue (0, 0)-signal. This covers all the possibilities to deviate from the given sequence that do not result in stranded pieces. It follows that there is no clearing capture sequence given this blue input that increments the second component of the orange signal.

We show an analogous result for the second component:

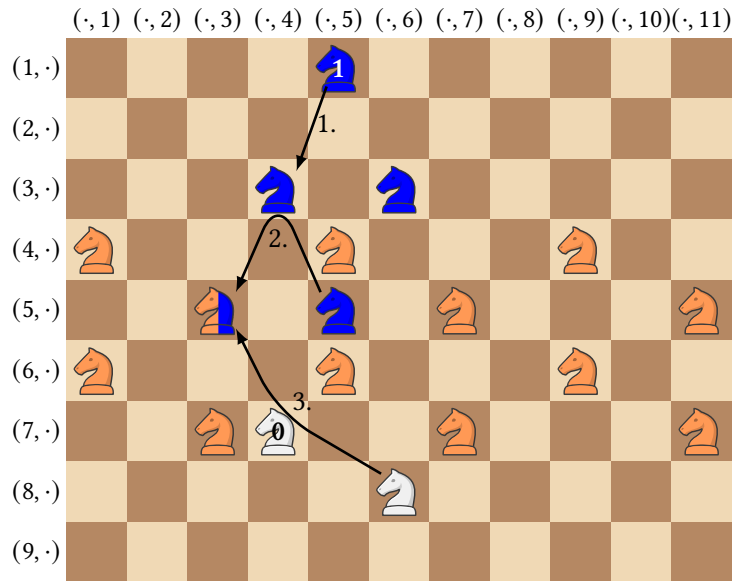


Figure 6.18.: The Maximum gadget with a top input of $(0, 1)$.

Claim 3: For a blue input of $(0, 1)$, the Maximum gadget is a second component Increment on the orange input.

Proof. Figure 6.18 shows a suitable capture sequence. This sequence propagates the $(1, 5)$ -knight to $(5, 3)$ without needing to use the $(3, 6)$ -knight. This allows the latter to serve as a light-squared increment.

To show that this is the best possible output, we need to show that there is no capture sequence that uses a blue $(0, 1)$ -input to increment the first component of the orange signal. We consider alternative capture sequences. We first discuss deviating from the second pair of moves. Propagating the 0-knight from $(3, 4)$ to $(5, 5)$ instead of $(5, 3)$, through moves $((5, 3) \rightarrow (3, 4) \rightarrow (5, 5))$ results in a 0-knight on $(5, 5)$. To not end stranded, this 0-knight is propagated by its only neighbor with a budget of 2, through moves $((3, 6) \rightarrow (5, 5) \rightarrow (7, 4))$. This results in a 0-leaf and, thus, a stranded piece. Next, we discuss deviating from the first move. Capturing to the right instead of to the left, through $((1, 5) \rightarrow (3, 6))$, results in an effectively symmetric scenario. After captures $((5, 5) \rightarrow (3, 6) \rightarrow (5, 7))$, the $(3, 4)$ -knight takes the role of the light-squared Increment. Other alternative continuations are equivalent to those given in the prior case. It is also possible to propagate the $(1, 5)$ -knight through a pair of captures. In this case moves $((3, 6) \rightarrow (1, 5) \rightarrow (3, 4))$ transpose to the case of a blue $(0, 0)$ -signal which as we have seen does not increment the first component of the orange signal. Here too, the alternative pair of moves $((3, 4) \rightarrow (1, 5) \rightarrow (3, 6))$ results in a symmetric, equivalent scenario. This covers all the options to propagate the $(1, 5)$ -knight. Finally, for the final pair of moves, the white 0-knight has only two neighbors that it can possibly be propagated to. Propagating it to the remaining white knight (e.g. in the above-mentioned symmetric scenario) results in a stranded 0-leaf. The remaining option is the one given in the example sequence. It follows that there is no clearing capture sequence given this blue input that increments the first component of the orange signal.

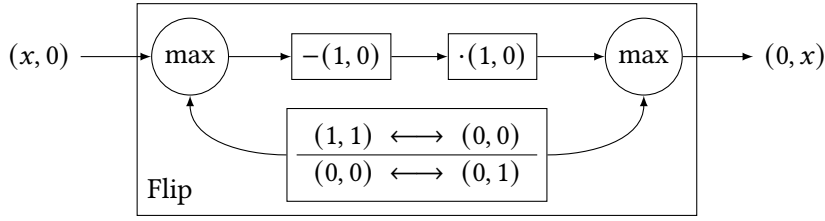


Figure 6.19.: The Flip gadget flips the two components of a signal.

If the blue input is $(1, 1)$, then combining the two previous approaches increments both components of the orange wire signal. Overall, we see that the Maximum gadget is the identity on the orange wire signal if the blue wire signal is $(0, 0)$, and that any blue signal component set to 1 sets the corresponding component of the orange signal to 1, thereby correctly implementing the maximum function. ■

6.2.4. The Flip Gadget

Finally, we describe the Flip gadget, which flips the two components of a signal. Figure 6.19 shows a high level view of the gadget. Specified in more detail:

Lemma 6.9: *The Flip gadget turns a $(1, 0)$ -signal into a $(0, 1)$ -signal. It leaves a $(0, 0)$ -signal unchanged.*

Proof. The idea of the Flip gadget is to use the asymmetric value production gadget to differentiate the two possible inputs. It produces its first possible pair of output values for a gadget input $(x, 0) = (0, 0)$, and its second possible pair of output values for a gadget input $(x, 0) = (1, 0)$.

We verify all cases of possible inputs and produced value pairs:

- Input $(0, 0)$, pair $(1, 1) \leftrightarrow (0, 0)$: $(0, 0) \xrightarrow{\max} (1, 1) \xrightarrow{-(1,0)} (0, 1) \xrightarrow{\cdot(1,0)} (0, 0) \xrightarrow{\max} (0, 0)$
- Input $(0, 0)$, pair $(0, 0) \leftrightarrow (0, 1)$: $(0, 0) \xrightarrow{\max} (0, 0) \xrightarrow{-(1,0)} (-1, 0) \rightsquigarrow \text{Error}$
- Input $(1, 0)$, pair $(1, 1) \leftrightarrow (0, 0)$: $(1, 0) \xrightarrow{\max} (1, 1) \xrightarrow{-(1,0)} (0, 1) \xrightarrow{\cdot(1,0)} (0, 0) \xrightarrow{\max} (0, 0)$
- Input $(1, 0)$, pair $(0, 0) \leftrightarrow (0, 1)$: $(1, 0) \xrightarrow{\max} (1, 0) \xrightarrow{-(1,0)} (0, 0) \xrightarrow{\cdot(1,0)} (0, 0) \xrightarrow{\max} (0, 1)$

By monotony, for each input the maximum possible output is chosen. The claim follows. ■

This concludes our discussion of the auxiliary function gadgets.

6.3. SAT Gadgets

In this section, we describe how to construct the necessary gadgets for our high level SAT construction. For this, recall that we encode the logic value true as $(1, 0)$ and the logic value false as $(0, 0)$.

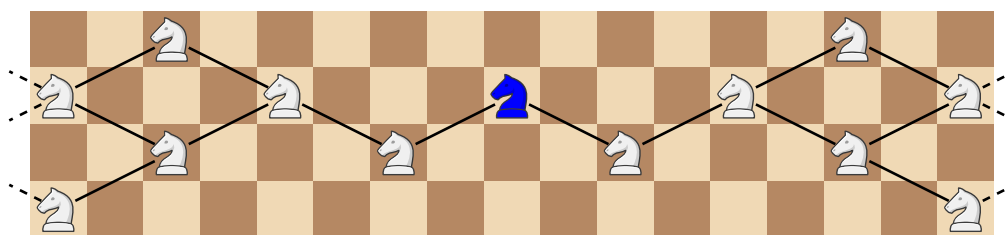


Figure 6.20.: The variable assignment gadget produces left and right outputs $(1, 0) \leftrightarrow (0, 0)$ or $(0, 0) \leftrightarrow (1, 0)$.

6.3.1. The Variable Assignment Gadget

We begin with the variable assignment gadget seen in Figure 6.20. It works analogously to the King 2-SOLO CHESS variable assignment gadget: It consists of a path of length five enclosed by a wire on either side.

Lemma 6.10: *A variable assignment gadget produces one $(1, 0)$ - and one $(0, 0)$ -signal.*

Proof. The proof is effectively analogous to the king case. The center knight of the path, marked in blue, cannot be captured as it would otherwise end up stranded. Thus, it needs to make a capture itself. If it captures to the left, this creates a $(0, 0)$ -signal on the left wire and a $(1, 0)$ -signal on the right wire, if it captures to the right the two signals are swapped. ■

Translated into the language of logic values, this shows that the variable assignment gadget produces one true and one false literal, as expected.

6.3.2. The Or Gadget

For the Or gadget, we use the unmodified Maximum gadget. If both inputs are false – i.e., $(0, 0)$ – it produces the output false since $\max\{(0, 0), (0, 0)\} = (0, 0)$. If one or both inputs are true – i.e., $(1, 0)$ – it produces the output true since for any $x \in \{0, 1\}$: $\max\{(1, 0), (x, 0)\} = (1, 0)$. Thus, the Maximum gadget correctly implements the Or function.

6.3.3. The And Gadget

Next, we describe the And gadget, seen in Figure 6.21. Note that the $-(1, 1)$ gadget simply consists of a first component, followed by a second component Decrement gadget.

Lemma 6.11: *The And gadget receives two inputs, $(x, 0)$ and $(y, 0)$. It produces an output of $(1, 0)$ if both x and y are 1, and an output of $(0, 0)$ otherwise.*

Proof. The idea of this construction is similar to that of the Flip gadget. The asymmetric value production gadget differentiates two classes of inputs, namely the case where $x = y = 1$, and the case where at least one of x or y is 0. We first compute the left-most Maximum gadget: $\max((x, 0), \text{Flip}((y, 0))) = \max((x, 0), (0, y)) = (x, y)$ as indicated in the figure. Next, we verify the cases of possible inputs and produced value pairs:

- Input $(x, y) = (1, 1)$, value pair $(1, 1) \leftrightarrow (0, 0)$:

$$(x, y) = (1, 1) \xrightarrow{\max} (1, 1) \xrightarrow{-(1,1)} (0, 0) \xrightarrow{\max} (0, 0)$$

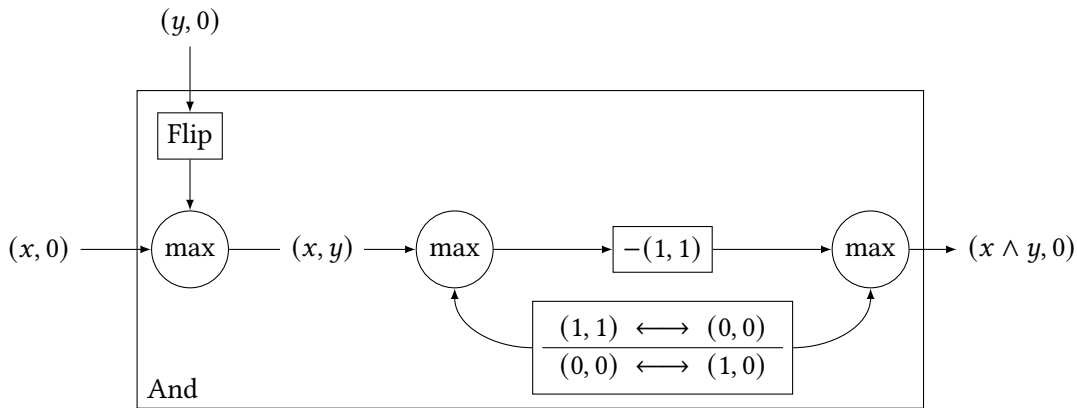


Figure 6.21.: The And gadget computes the logical conjunction of its two inputs.

- Input $(x, y) = (1, 1)$, value pair $(0, 0) \leftrightarrow (1, 0)$:

$$(x, y) = (1, 1) \xrightarrow{\max} (1, 1) \xrightarrow{-(1,1)} (0, 0) \xrightarrow{\max} (1, 0)$$

- Input $(x, y) < (1, 1)$, value pair $(1, 1) \leftrightarrow (0, 0)$:

$$(x, y) \xrightarrow{\max} (1, 1) \xrightarrow{-(1,1)} (0, 0) \xrightarrow{\max} (0, 0)$$

- Input $(x, y) < (1, 1)$, value pair $(0, 0) \leftrightarrow (1, 0)$:

$$(x, y) \xrightarrow{\max} (x, y) \xrightarrow{-(1,1)} (x - 1, y - 1) \rightsquigarrow \text{Error, since } (x, y) < (1, 1)$$

By monotony, for each input the maximum possible output is chosen. The claim follows. ■

Translated into the language of logic values, this shows that the And gadget evaluates to true if both inputs are true, and evaluates to false, otherwise.

6.3.4. The 1-Test Gadget

Finally, we describe the 1-Test gadget, shown in Figure 6.22. As in the King 2-SOLO CHESS case, it is used at the very end to ensure that the SAT formula evaluates to true.

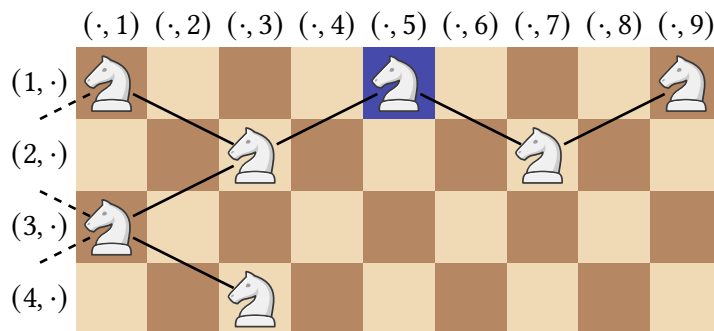


Figure 6.22.: The 1-Test reduces to a single piece if its input is $(1, 0)$. If its input is $(0, 0)$, it cannot be reduced to a single piece.

Lemma 6.12: *The 1-Test gadget reduces to a single piece on the final square marked in blue, if and only if the first component of its input is 1.*

Proof. Consider the case that the first component of the signal is set to 1, i.e., the 2-knight on $(1, 1)$ is present. Then, after captures $((4, 3) \rightarrow (3, 1) \rightarrow (2, 3))$, either end of the resulting path of length five captures towards the blue finale square through captures $((1, 1) \rightarrow (2, 3) \rightarrow (1, 5))$, $((1, 9) \rightarrow (2, 7) \rightarrow (1, 5))$, resulting in a single remaining piece as claimed.

Next, consider the case that the first component of the signal is set to 0, i.e., the 2-knight on $(1, 1)$ is not present. Then, after the eventual capture $((4, 3) \rightarrow (3, 1))$, we remain with a path of length five once more, however, with the left leaf having a budget of at most 1. Thus, the left leaf cannot reach the blue (final) square, while the right leaf cannot reach past the blue square, meaning that the two cannot reach a common square. It follows that more than one piece remains, as claimed. ■

6.4. The Final Reduction

We now describe our transformation of an And-Or-(1,1)-SAT instance into a Knight 2-SOLO CHESS instance. Let an And-Or-(1,1)-SAT instance $I = (U, C)$ be given. Choose an And-Or Embedding of I that includes the optional conjunction of the outputs of the clauses. For every variable $u \in U$, place a variable assignment gadget on the board at sufficient distance to all other variable assignment gadgets. For each clause, place the respective logic gadgets on the board according to the embedding and place wire on the board to connect their inputs with the correct outputs of variable assignments or other logic gadgets. Place the remaining And gadgets that combine the outputs of the clauses, according to the embedding. Where necessarily, place wire crossing gadgets to allow wires to cross. Recall that thanks to the various wire placements, it is always possible to align the different wires and gadgets appropriately. Finally, place a 1-test gadget and connect it to the output of the final And gadget. Using this construction, we now prove that Knight 2-SOLO CHESS is NP-hard. The proof is effectively analogous to that of the NP-hardness of King 2-SOLO CHESS.

Theorem 6.13: *Knight 2-SOLO CHESS is NP-hard.*

Proof. We reduce from And-Or-(1,1)-SAT. We transform a given instance with n variables and m clauses as described above. This transformation runs in polynomial time: Similar to the reduction for King 2-SOLO CHESS, the transformed instance consists of $\mathcal{O}(n + m)$ gadgets, each consisting of a constant number of pieces. Additionally, we place $\mathcal{O}(n + m)$ wires of $\mathcal{O}(n + m)$ pieces each. In total, our instance consists of $\mathcal{O}((n + m) + (n + m)^2)$ pieces and can be computed in the same running time. It remains to show that the two instances are equivalent.

SAT instance solvable \implies Chess instance solvable: Let Φ be a satisfying assignment for the SAT instance. A solving sequence for the chess instance consists of the following steps: For each variable assignment gadget, assign the value $(1, 0)$ to the literal that is set to true by Φ and the value $(0, 0)$ to the other literal. Since Φ satisfies each clause, by the correctness of the various gadgets, we obtain a capture sequence that yields a $(1, 0)$ -output from the final gadget of each clause. The conjunction of all gadget outputs once more yields a $(1, 0)$ -output, allowing the 1-test gadget to reduce to a single final piece. Every other gadget has been cleared completely, so only this single piece remains, meaning that this is a valid solving sequence.

Chess instance solvable \implies SAT instance solvable: Let a solving capture sequence for the chess instance be given. Then the following procedure yields a satisfying assignment Φ for the SAT instance: For each variable assignment gadget, check which of the output wires is assigned a $(1, 0)$ -signal. If that wire corresponds to some literal $\ell = x$, set $\Phi(x) := \text{true}$. If it corresponds to some literal $\ell = \neg x$, set $\Phi(x) := \text{false}$. If neither of the two wires is assigned a $(1, 0)$ -signal, set $\Phi(x) := \text{true}$. Then Φ is a satisfying assignment:

At most one wire of each variable assignment gadget is assigned a $(1, 0)$ -signal. Thus, Φ is a valid assignment. The solving capture sequence reduces the instance to a single piece. Thus, the 1-test gadget received a $(1, 0)$ -signal as input. It follows that each of the extra And gadgets received two $(1, 0)$ -signals as inputs. This means that each clause in the SOLO CHESS instance has evaluated to $(1, 0)$. Then, under the variable assignment given by Φ , each clause evaluates to $(1, 0)$, i.e., true. It follows that each SAT clause is satisfied under Φ .

We conclude that the two instances are equivalent, which shows the NP-hardness of Knight 2-SOLO CHESS. ■

7. Rook 2-SOLO CHESS

In this chapter, we present another result for 2-SOLO CHESS on a two-dimensional board. We give a proof of the NP-hardness of Rook 2-SOLO CHESS. This result has independently been shown by Bilò, Di Donato, Gualà and Leucci, via a reduction from Vertex Cover [BDGL24]. We give an alternative proof, reducing from And-Or-(1,1)-SAT.

7.1. The Setup

Similar to our reductions for King and Knight 2-SOLO CHESS, we directly translate an And-Or Embedding into a corresponding chess position. However, this reduction has some notable differences: The rook has unlimited range. As such, no wire is required to propagate logic values. Consequently, we implement logic values differently. Furthermore, the satisfaction of each clause is checked individually. In particular, given a variable assignment, each unsatisfied clause leaves behind stranded pieces. Figure 7.1 shows the structure of the Rook 2-SOLO CHESS instance created from the example And-Or-(1,1)-SAT instance $I = (U, C)$, with variables $U = \{x, y, z\}$ and clauses $C = \{(x \wedge \neg y) \vee z, (\neg x \vee \neg z) \vee y\}$. It shows the creation of variables x, y, z (in green) and their respective positive and negative literals. These literals are aligned with the logic gates for each clause. These consist of And-gates (in red), *left* Or-gates (in blue) and *right* Or-gates (in pink). The latter two differ in that the left Or-gate produces an output that is used to evaluate the remaining parts of the clause, while the right Or-gate – the final gate of each clause – produces no output. All variables and clauses are enclosed by the *shell* (in light and dark gray), whose purpose is explained below. We create gadgets for each of the above concepts.

Our transformation yields a SOLO CHESS instance that can be reduced to a single piece, if and only if the SAT instance I is satisfiable. In the remainder of this chapter, we describe the representation of logic values, as well as the implementation of each gadget, and discuss their correctness. More specifically, we create a variable assignment gadget which produces literals x and $\neg x$ and evaluates to one True and one False output. We create And and left Or gadgets which evaluate inputs x and y to outputs $x \wedge y$ and $x \vee y$, respectively. These three gadgets are fully cleared, when being evaluated. This stands in contrast with the right Or gadget. It has inputs x and y and can be fully cleared when at least one of x or y is True, however, it cannot be fully cleared when both x and y are False. Furthermore, it produces no output. In addition to these SAT gadgets, we create some extra cleanup gadgets: The *conveyor belt* collects left-over rooks from the evaluation of the clauses and guides them towards the *tomb*, which, similarly to the 1-test of previous chapters, contains the final square of the instance.

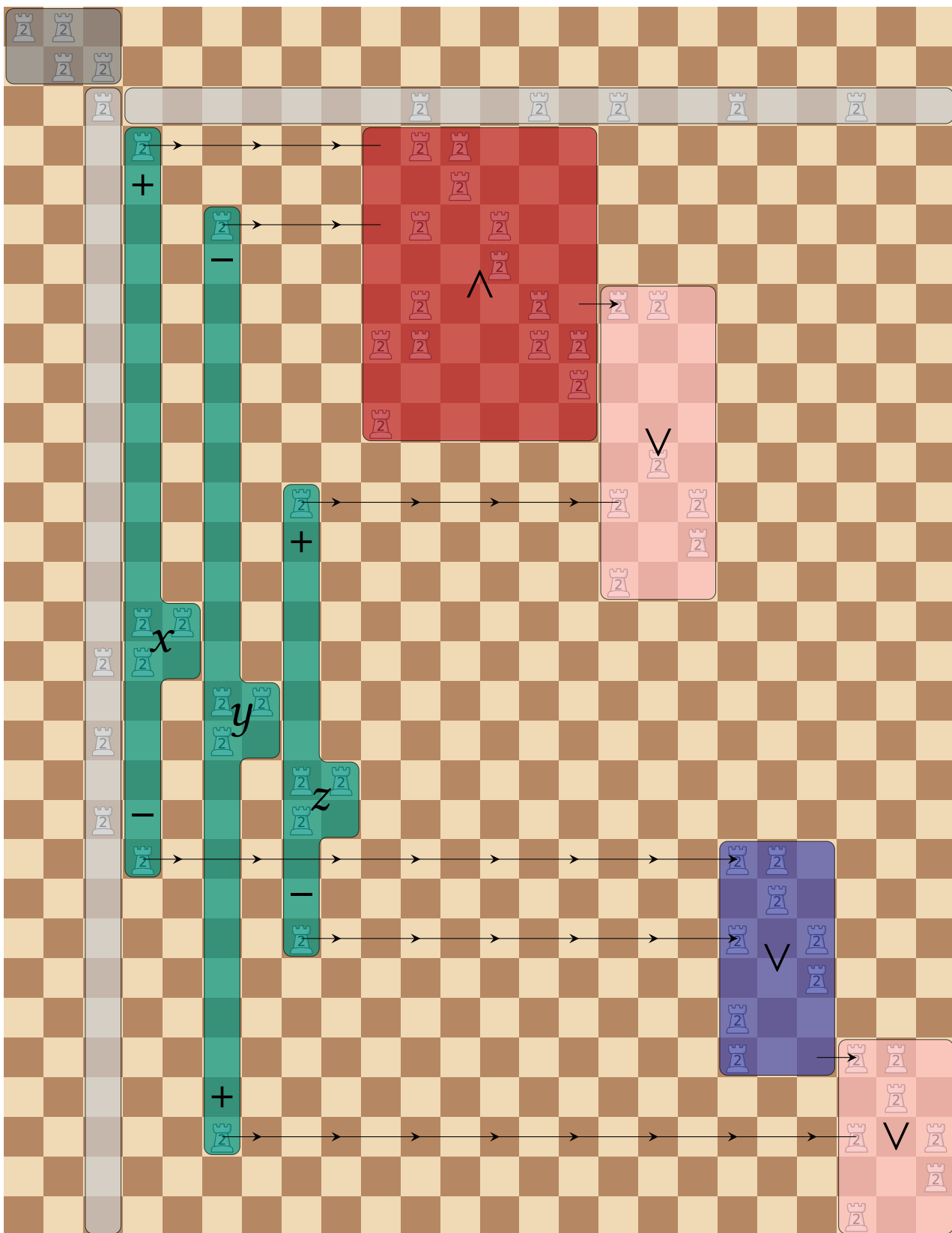


Figure 7.1.: A Rook 2-SOLO CHESS instance encoding the SAT instance $[\{x, y, z\}, \{((x \wedge \neg y) \vee z), ((\neg x \vee \neg z) \vee y)\}]$. For space constraint reasons, the cleanup rooks are not shown.

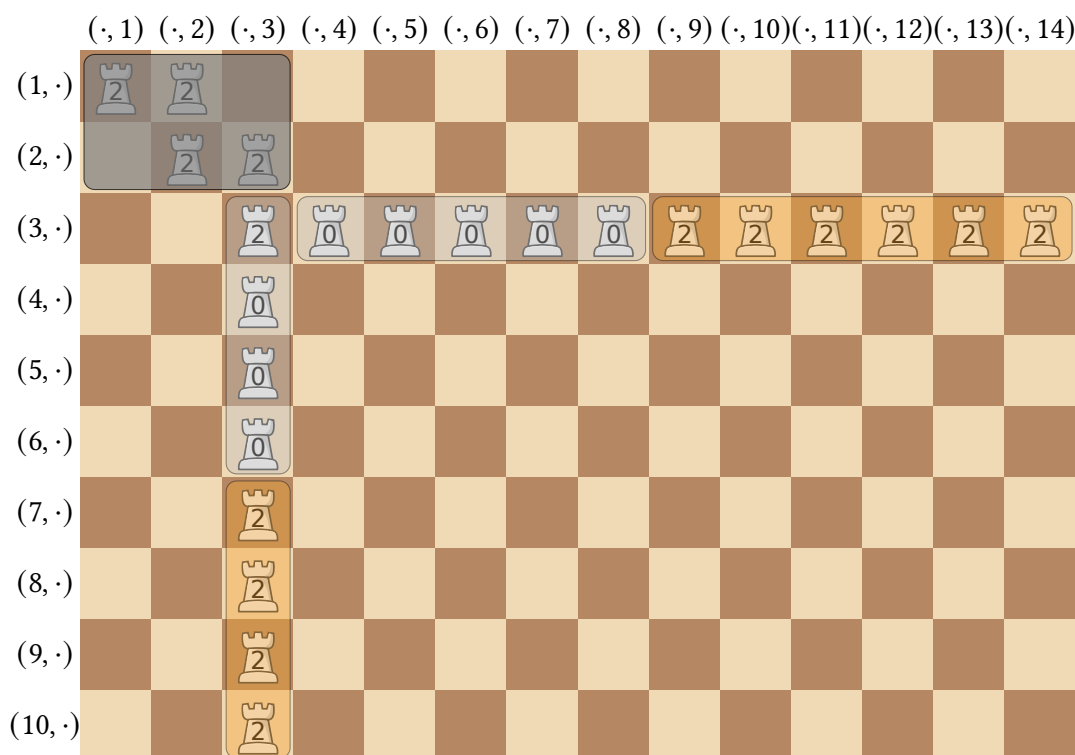


Figure 7.3.: The shell of an instance is reduced to a single final piece in the tomb.

7.1.1. The Cleanup Phase

We first discuss the extra cleanup gadgets, which are used in the final phase of a solving capture sequence. The very top left of Figure 7.1 shows in dark gray the *tomb* gadget, which contains the final resting place of the final piece of the instance. It is implemented as an antenna of length 3. Figure 7.2 shows our implementation in Rook 2-SOLO CHESS with the antenna marked in gray and some rooks outside the antenna present.

By Corollary 3.7, for any solving sequence, the final square of the instance is contained in the tomb. We give a capture sequence that reduces the configuration of Figure 7.2 to a single piece. It consists of first clearing the third row with captures $((3, 5) \rightarrow (3, 3) \rightarrow (2, 3))$, then clearing the third column with captures $((5, 3) \rightarrow (2, 3) \rightarrow (2, 2))$, before completing the sequence with captures $((1, 1) \rightarrow (1, 2) \rightarrow (2, 2))$.

Zooming out a bit, Figure 7.3 shows a compressed version of the outer *shell* of the example instance. It contains the tomb (in dark gray), as well as the *conveyor belt* (in light gray and orange). The conveyor belt itself is divided into a set of *capture destinations* (in light gray) and a set of *cleanup rooks* (in orange). The latter were omitted from Figure 7.1 for space constraint reasons. The capture destinations are captured towards in the process of clearing the

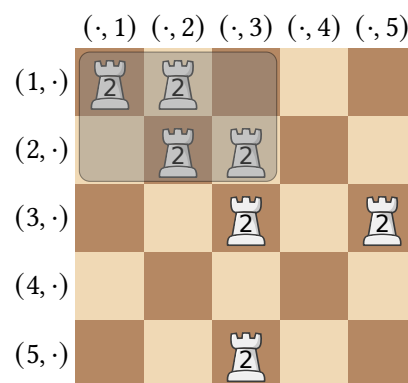


Figure 7.2.: The tomb gadget ensures that any solving sequence has its final square on $(2, 2)$.

various yet to be defined SAT gadgets. As a result, these rooks end up with budgets as small as 0. The count of light gray rooks of the third column and row is matched by an equal number of orange cleanup rooks.

We describe a solving sequence for the shell. It uses the same approach as outlined above, of first clearing the third row, then the third column. By the 1D-Rook Lemma 4.3, it is possible to clear the partial configuration of all pieces of the third row, except the right-most orange rook, towards the left-most rook of the row. Then, the row is fully cleared through the pair of captures $((3, 14) \rightarrow (3, 3) \rightarrow (2, 3))$ from that right-most orange rook. Note that clearing the third row like this reduces the number of gray rooks in the third column by one, since the $(3, 3)$ -rook is no longer present. Once more, by the 1D-Rook Lemma 4.3, it is possible to clear the partial configuration of all pieces of the third column, except the bottom orange rook, towards the top rook of the column. Then, resolving the tomb through captures $((10, 3) \rightarrow (2, 3) \rightarrow (2, 2)), ((1, 1) \rightarrow (1, 2) \rightarrow (2, 2))$ completes the solving sequence.

We call a Rook 2-SOLO CHESS configuration *well-formed*, if it consists of a shell as well as a (possibly empty) set of SAT gadgets. The latter are placed such that they are not adjacent to the tomb or any of the cleanup rooks, however, they may be adjacent to some capture destination rooks. Then, since the tomb contains the final square of the configuration, any solving sequence fully clears all the SAT gadgets. Conversely, if there is a capture sequence that fully clears all the SAT gadgets, then by the above procedure, the remaining shell can be reduced to a single piece, i.e., the configuration can be solved. Overall, we have shown:

Lemma 7.1: *A well-formed configuration has a solving sequence, if and only if there exists a capture sequence that fully clears each of its SAT gadgets. The final square of such a solving sequence is contained in the tomb.*

7.1.2. Logic Squares

We describe how to represent truth values in Rook 2-SOLO CHESS. Recall that our construction for Knight 2-SOLO CHESS encoded the value of the first component of its signal by the presence or absence of a 2-piece on a specific square (the outer vertex of a signal-column). We use this idea in Rook 2-SOLO CHESS with what we call *logic squares*. A logic square represents the value True, when it holds a 2-rook, and False, when it holds no rook, i.e., is empty. A logic square z holds the output of a gadget A_1 and at the same time the input of the next gadget A_2 . This can be seen in Figure 7.1 where neighboring gadgets share some rows so that the output square of a gadget on the left is in line with a gadget on the right to serve as input square. Initially, all logic squares are filled and, thus, set to True. However, throughout a solving sequence, the 2-rook on output square z may make a capture within the gadget A_1 . In this case, it is no longer present on z , and so the corresponding input of gadget A_2 is False. Figure 7.4 shows an example of a left Or gadget with both, one, or none of the inputs set to True. We mark inputs (i.e., logic squares, or the rooks placed on them) in blue and outputs in orange; capture destinations of the shell are marked in gray.

The intuition behind our implementation of truth values is that the presence of a 2-rook on a logic square provides additional units of budget. The typical use case of such an input 2-rook is to propagate a 0-rook of the gadget towards a capture destination. If the 2-rook is not present, then a solving sequence may instead need to utilize the output rook of the gadget to propagate the 0-rook. In this way, the value of the input of a gadget can affect the value of the output of a gadget. As in the Knight 2-SOLO CHESS construction, there is a monotony on input values: An input of True (a present 2-rook) is no worse than an input of False (an

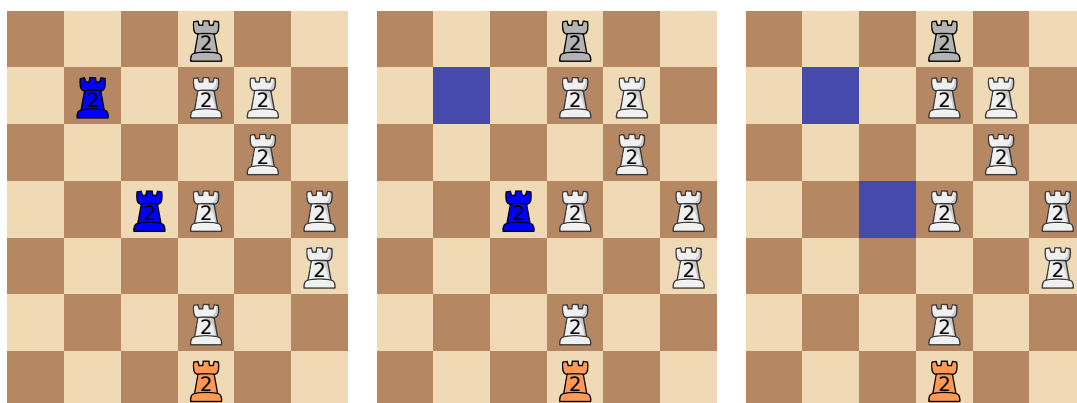


Figure 7.4.: A left Or gadget with both, one, or no input set to True.

absent rook), which in turn is no worse than an “invalid” input consisting of either a 1-rook or a 0-rook on the input square. We formalize this intuition in the following subsection. As usual, we assume that any gadget evaluates to the maximum possible output value.

In the following sections, we describe the various SAT gadgets. Recall that, by Lemma 7.1, each solving sequence fully clears each SAT gadget. Thus, when describing what each of the gadgets evaluates to, we only consider sequences which fully clear the gadget. This excludes gray shell rooks which can remain, since they are cleaned up in the cleanup phase. Furthermore, the logic squares containing the gadget’s outputs can contain rooks after the gadget is fully cleared, since they are also inputs and, thus, part of the next gadget.

7.1.3. Gadget inputs

A common occurrence in our SAT gadget constructions is the *triangle*. Shown in Figure 7.5, it is an implementation of a *B*-antenna in Rook 2-SOLO CHESS. Note that due to the rook’s unlimited range, a triangle can include any number of empty rows or columns between its rooks. This does not change its structure and the corresponding capture graph.

We give a brief intuition on the use of triangles / antennae in our construction. By Lemma 3.5, any solving sequence reduces such an antenna to a single 0-piece on its last square. In a solving sequence, this 0-piece is propagated away from its current square by being captured by some 2-rook. Thus, it serves as a check that a specific 2-rook is present to be able to propagate the 0-rook. Note, however, that for Lemma 3.5 to apply, the triangle needs to be an antenna in the graph of the full instance. In our constructions this is usually not the case. Instead, we show that the statement of the lemma holds conditionally and then show for each gadget that there is a solving sequence that adheres to the condition.

We now present a mini-gadget that is part of each of the gadgets for logic gates, the *gadget input*. Shown in Figure 7.6, it combines the concepts of an input logic square and a triangle. The red arrows indicate directions that contain no further rooks in the entire configuration (this is relevant due to the rook’s unlimited range). Furthermore, by convention, we depict capture destination rooks (in gray) in the first row. In particular, if there is no rook on a square of the first row, this is understood to mean that there is no capture destination rook in

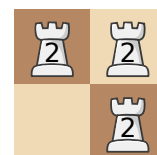


Figure 7.5.: A triangle is an antenna of length 2.

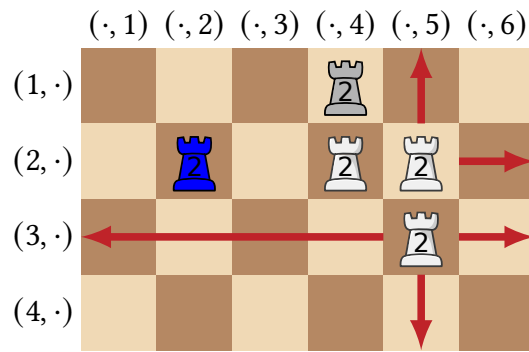


Figure 7.6.: A gadget input is used in each of the gadgets for logic gates. Red arrows indicate directions that contain no further rooks in the entire configuration.

the corresponding column. In the full instance, a gadget may be placed many rows below a capture destination of the same column. However, due to the rook’s unlimited range, this is equivalent to being placed just below.

As indicated above, the triangle of white rooks does not form an antenna in the full configuration, since the (2, 5)-rook is adjacent to the blue input square rook, which is not part of the antenna. Thus, we can only show the statement of Lemma 3.5 conditionally.

Lemma 7.2: *Let a well-formed configuration C contain a gadget input. Let a solving sequence for C not reach an intermediate configuration with a 0-rook to the left of the gadget input’s triangle and no triangle rooks present anymore. Then it instead reduces the triangle to a 0-rook on the triangle’s last square.*

Proof. By Lemma 7.1, any solving sequence fully clears the gadget input. Thus, it contains the capture $((3, 5) \rightarrow (2, 5))$, as otherwise the leaf on (3, 5) would remain stranded. This results in a 1-rook on (2, 5), which in turn performs a capture at some point. If it captures the (2, 4)-rook, this results in a 0-rook on the last square of the triangle, as claimed. Otherwise, it captures a rook further to the left (after the (2, 4)-rook has moved out of the way), which results in a 0-rook to the left. Since the (2, 4)-rook has already moved out of the way, none of the triangle rooks are present anymore, breaking the condition of the lemma. ■

This leaves open the possibility of capturing towards the left. While such a capture to the left may be part of a solving sequence, it does not occur *after* the previous gadget on the left has been fully cleared (apart from the possible output rooks).

Lemma 7.3: *Let the input square of a gadget input in a well-formed configuration hold a rook that is not adjacent to any rooks outside the gadget input. Then there exists no solving sequence for the configuration that includes a capture from the gadget input to the input square rook.*

Proof. Recall that any solving sequence has its final square in the tomb. Consider a capture sequence including a capture from the gadget input to the input square rook. This capture is either performed by the rook on (2, 4), or by the rook on (2, 5) once the (2, 4)-rook has moved out of the way. In either case, the set of remaining gadget input rooks and the input square rook are disconnected from the rest of the configuration and, thus, stranded. ■

It immediately follows:

Corollary 7.4: *Let the input square of a gadget input in a well-formed configuration hold a 0-rook that is not adjacent to any rooks outside the gadget input. Then the configuration cannot be solved.*

Proof. Consider an arbitrary capture sequence. If it contains a capture from one of the gadget input rooks to the 0-rook, then, by the previous lemma, it is not a solving sequence. If it does not contain such a capture, then the 0-rook remains stranded, thus, it is not a solving sequence either. ■

Corollary 7.5: *Let the input square of a gadget input in a well-formed configuration hold a 1-rook that is not adjacent to any rooks outside the gadget input. If there exists a solving sequence for this configuration, then there also exists one for the configuration in which the input square is empty instead (i.e., an input of False).*

Proof. By Lemma 7.3, no solving sequence contains a capture to the 1-rook. Thus, any solving sequence instead contains a capture from the 1-rook to the gadget input. This capture replaces a rook with some budget greater or equal than 0, with one having budget exactly 0. This solving sequence is also valid for the configuration having an empty input square (excluding the capture by the no longer present 1-rook). ■

The remaining possibility to be considered is that of a 2-rook on the input square.

Lemma 7.6: *Let the input square of a gadget input in a well-formed configuration be empty and not be adjacent to any rooks outside the gadget input (i.e., an input of False). If there exists a solving sequence for this configuration, then there also exists one for the configuration in which the input square holds a 2-rook instead (i.e., an input of True).*

Proof. Observe that the condition of Lemma 7.2 is met. Thus, by the lemma, any solving sequence reduces the triangle of the gadget input to a 0-rook on its last square. This solving sequence remains valid under an input of True, by initially interjecting a capture from the rook of the input square to the last square of the triangle. ■

The three previous results all apply to the outputs of fully cleared gadgets, as those output squares are only adjacent to the gadget inputs of the following gadgets. Thus, we see that the claimed monotony does hold, justifying the assumption that gadgets evaluate to the maximum possible value.

All the above results consider a previous gadget that is fully cleared. However, a solving sequence may still contain a capture from a gadget input to the left, so long as the previous gadget is not yet fully cleared.

Lemma 7.7: *Let a well-formed configuration containing a gadget input be given. Furthermore, let a solving sequence contain a capture to the left from a rook of the gadget input's triangle. Then, the solving sequence contains a capture to the left from the final remaining rook of the triangle that results in a 0-rook on the capture's destination square (i.e., is performed by a 1-rook).*

Proof. Since the solving sequence fully clears the triangle, it does include the leaf capture $((3, 5) \rightarrow (2, 5))$. Consider the case that the $(2, 4)$ -rook performs a capture to the left. Then, the $(2, 5)$ -rook is the final rook of the triangle, and its only neighbors are to its left. Thus, the solving sequence contains a capture by the $(2, 5)$ -rook to the left, resulting in a 0-rook on the capture's destination square.

Consider now the case that the $(2, 4)$ -rook does not perform a capture to the left. Then, by assumption, the $(2, 5)$ -rook performs a capture to the left, once more resulting in a 0-rook on the capture's destination square. ■

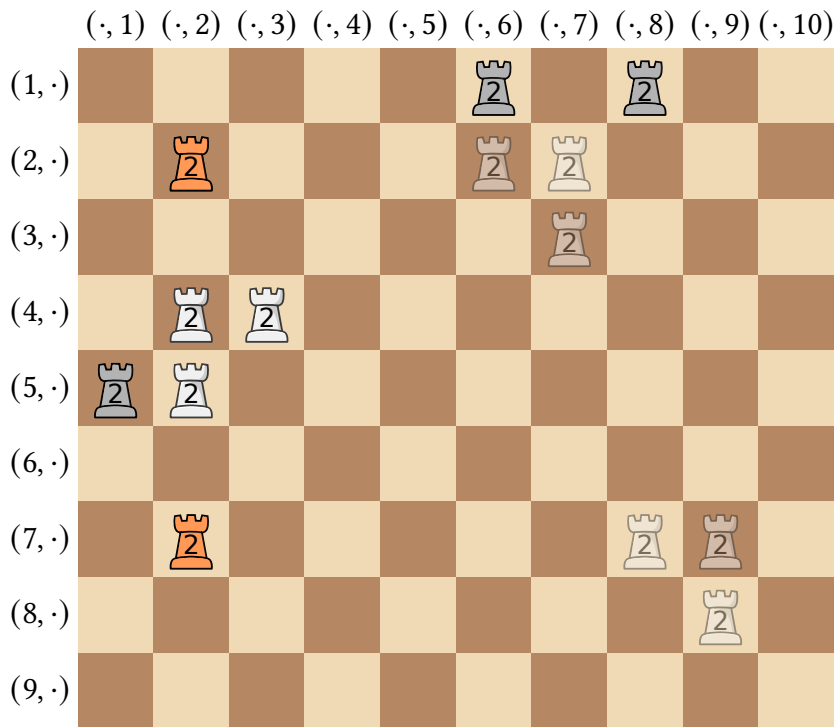


Figure 7.7.: The variable assignment gadget (on the left) assigns truth values to the logic squares holding the orange rooks. The transparent rooks (on the right) depict the gadget inputs of the next gadget in the configuration.

With these tools at our disposal, we now present all the required SAT gadgets. As seen in Figure 7.1, they consist of a variable assignment gadget, an And gadget and a left and a right Or gadget. The general approach to clearing these gadgets is always the same: Based on the inputs, all rooks except possibly the rooks on the output squares are cleared by a sequence of captures. To retain a valid output of either a 2-rook or an empty square, some of the clearing captures are made towards adjacent capture destination rooks. Once only rooks on output squares remain, they serve as inputs for the next gadget, which is then cleared in the same fashion.

For space constraint reasons, we may compress diagrams of SOLO CHESS configurations in some of the figures. In that case, the row and column numbering may not be contiguous. In particular, we denote each square by the coordinates it would have on an uncompressed board.

7.2. The Variable Assignment Gadget

We begin with the variable assignment gadget, seen in Figure 7.7. It has zero inputs and two outputs, namely the x and $\neg x$ literals. The logic squares holding orange rooks, (2, 2) and (7, 2), hold the truth values of the two literals. We call those logic squares *output squares* of the gadget, or *outputs* for short. The remainder of the gadget consists of a triangle, with its last square aligned with a gray shell rook.

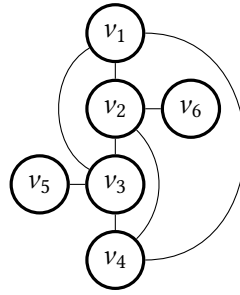


Figure 7.8.: The capture graph of the variable assignment gadget.

Before reviewing the intended way to resolve a variable assignment gadget, we first discuss an alternative scenario: Since the output squares of the variable assignment are part of the gadget inputs of the next gadget, it is possible to capture from those gadget inputs to the variable assignment's output squares. We show, however, that this does not occur in any solving sequence.

Lemma 7.8: *Given a well-formed configuration containing a variable assignment whose outputs are aligned with the gadget inputs of further gadgets. Then there exists no solving sequence containing a capture from one of the gadget input's triangle rooks to the variable assignment gadget.*

Proof. Assume for contradiction that there exists a solving sequence containing a capture from a gadget input's triangle to an output square of a variable assignment. By Lemma 7.7, this results in a 0-rook on the output square with no other rooks of the gadget input remaining. Recall that any capture sequence on the chess board is also a valid sequence in the capture graph. Thus, it suffices to show that there is no clearing sequence in the capture graph, shown in Figure 7.8. In this graph, the two output vertices v_1 and v_4 cannot be distinguished, thus, it suffices to show the claim for vertex v_1 being captured into.

The final square of any hypothetical solving sequence is contained in the tomb. Thus, by Observation 3.3, 0-vertex v_1 is propagated by a pair of captures from a 2-vertex w to another vertex. This vertex w cannot be vertex v_2 , as otherwise, vertex v_6 remains stranded. If $w = v_3$, then after a capture ($v_3 \rightarrow v_1$), the gadget is no longer connected to the shell vertex v_5 and can only be cleared via the second output, cut vertex v_4 . However, any sequence clearing the remaining vertices of the gadget towards vertex v_4 results in a 0-vertex on that output square. By Corollary 7.4, this cannot be extended to a solving sequence. If instead $w = v_4$, then after the capture ($v_4 \rightarrow v_1$), the gadget can only be cleared via the shell vertex. But once again, any sequence clearing the remaining vertices of the gadget results in a 0-vertex on the square of v_3 , which, being a 0-leaf, remains stranded. This contradicts the assumption and shows the claim. ■

In our construction, the only adjacency between the rooks of the variable assignment gadget (excluding the gray shell rook) and the remaining configuration is the discussed scenario of outputs being aligned with gadget inputs. Using this fact, we now show:

Lemma 7.9: *The variable assignment gadget evaluates to one True and one False output.*

Proof. We give a capture sequence achieving the claimed result, then show that there is no capture sequence achieving a better result. To reach the claimed result, clear the triangle with captures $((4, 3) \rightarrow (4, 2) \rightarrow (5, 2))$. Then, propagate the 0-rook towards the shell rook with

captures $((x, 2) \rightarrow (5, 2) \rightarrow (5, 1))$ for $x = 2$ or $x = 7$, to retain the bottom or the top literal as True, respectively. The other logic square is emptied in the process, and so the other literal is set to False. As noted previously, this constitutes a cleared gadget, as both the gray shell rook and the output logic squares are part of other gadgets.

We show now that this is the maximum possible output value. This effectively states that, for a variable x , it is impossible to set both its literals to True at the same time. Any capture sequence that keeps both outputs set to True, does not contain a capture within the gadget from or to one of the output square's rooks. The remainder of the gadget is an antenna of length 3 which, by Observation 3.3, is stranded. Thus, it is not possible to fully clear the gadget with both outputs set to True. ■

7.3. The Left Or Gadget

We now discuss the left Or gadget, shown in Figure 7.9. It has two inputs and one output. We distinguish between the left and the right Or gadget. The latter is the final gadget of each clause. Since each clause combines at most three literals, it uses at most two logic gates. Thus, with the right Or gadget being the final gadget of each clause, each left Or gadget is the first gadget of its clause. This means, in particular, that both of its inputs are literals straight from a variable assignment gadget.

Like all other gadgets for logic gates, the left Or gadget contains two gadget inputs aligned with the logic squares holding its inputs. We show that the gadget evaluates to True, if and only if at least one of its inputs is True.



Figure 7.9.: The left Or gadget with both inputs set to True.

Lemma 7.10: *If at least one of its inputs is set to True, the left Or gadget evaluates to True. If none of its inputs are set to True, it evaluates to False.*

Proof. We first give a capture sequence for the first claim, then one for the second claim. Let the top input be set to True. Then, a clearing sequence is given by first reducing each triangle to a 0-rook on its last square, followed by propagating each 0-rook towards the gray shell rook, as shown in Figure 7.10. This fully clears the gadget, with the output being set to True. The case for the second input or both inputs being set to True works analogously.

Next, let none of the inputs be set to True. Then a clearing sequence is given by again reducing the triangles to 0-rooks, followed by propagating each 0-rook towards the gray shell rook using the orange output square 2-rook instead of the blue input square 2-rook. This fully clears the gadget, with the output being set to False.

We show now that these are the maximum possible output values. For this, it suffices to show that the gadget does not evaluate to True if both inputs are set to False. We consider the different possible capture sequences. Observe that, by Lemma 7.8, none of the gadget inputs capture towards the variable assignment gadgets to the left of the gadget. It follows from Lemma 7.2, that each of the triangles of the gadget inputs is reduced to a 0-rook on its last square. Since each of the inputs is False, either 0-rook is only adjacent to the other rooks of the column. This column consists of a gray 2-rook in the shell, two white 0-rooks, a

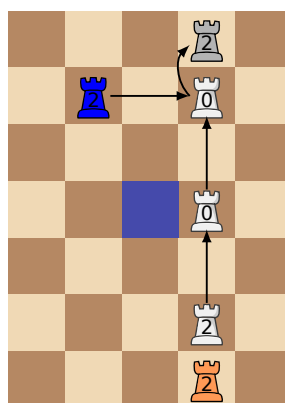


Figure 7.10.: A partially cleared left Or gadget with one True and one False input.

white 2-rook, and an orange 2-rook on the output square. The only two rooks of the column that are adjacent to rooks outside the column are the gray capture destination rook and the orange output rook. Thus, any solving sequence propagates the two 0-rooks to one of those two colored rooks, as they would otherwise remain stranded. Propagating them to the gray capture destination rook requires captures from the other two 2-rooks of the column and matches the sequence above. Propagating them to the orange output rook instead, requires captures from the white and gray 2-rooks and results in a gadget containing only a 0-rook on the orange output square. This orange output square is the input square of the next gadget. By Corollary 7.4, the resulting configuration cannot be solved. Overall, we see that there is no solving capture sequence that evaluates the gadget to True, given two inputs of False. ■

Excluding the input square rooks, the output square rook is the only rook of the left Or gadget that is adjacent to other gadgets. In particular, it is part of the gadget input of the next gadget, whose triangle rooks are adjacent to it. Thus, it remains to discuss the scenario of a capture from a triangle rook of the next gadget to the output rook of this gadget.

Lemma 7.11: *Let a well-formed configuration containing a left Or gadget, whose inputs are both set to False, be given. Then there exists no solving sequence that contains a capture from the gadget input of the next gadget towards the left Or gadget.*

Proof. Assume there exists a solving sequence containing a capture from a gadget input to the output square of the left Or gadget. By Lemma 7.7, this results in a 0-rook on the output square, with no rooks of the gadget input remaining. After reducing the left Or gadget's triangles to 0-rooks, we once more remain with a one dimensional sub-configuration, this time consisting of two 2-rooks and three 0-rooks. Of these, only the gray shell rook is adjacent to rooks outside the gadget. By the generalized 1D Rook Lemma 4.3, there is no capture sequence clearing this column towards that gray shell rook. Thus, some non-shell rooks of the gadget remain stranded. ■

Note that there may exist solving sequences that do contain a capture from a gadget input to the output of a left Or gadget, so long as both of the latter's inputs are set to True. However, in this case, there also exists a solving sequence that does not contain such a capture. We discuss this scenario in the section on the final reduction.

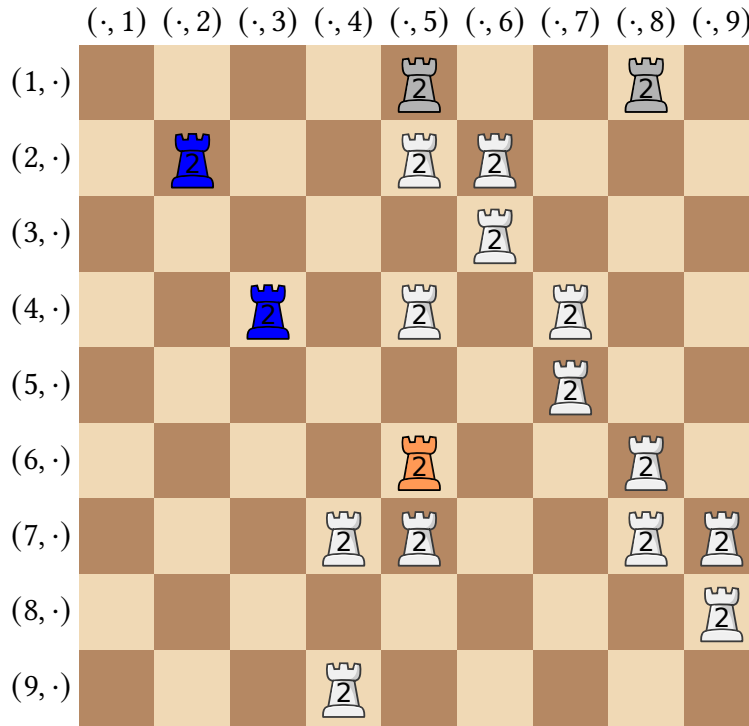


Figure 7.11.: An And gadget with two blue inputs and one orange output

7.4. The And Gadget

The And gadget, shown in Figure 7.11, has two inputs and one output. If both inputs are False, it evaluates to False. Changing one of the inputs to True does not change the output, however, changing both inputs to True does change the output to True. Similar to our reduction for King 2-SOLO CHESS, this behavior is achieved by an And gadget that permits a different way of clearing it when both inputs are set to True, as compared to when only one or none are. We show separately that there exist clearing sequences that achieve the correct outputs, and that these are the maximum possible output values.

Lemma 7.12: *If both of its inputs are set to True, it is possible to clear the And gadget with its output being set to True. For any other set of inputs, it is possible to clear the And gadget with its output being set to False.*

Proof. Consider the case of both inputs being set to True, i.e., the configuration shown in Figure 7.11. Then, a clearing sequence is given by first reducing the triangles of the two gadget inputs, as well as the triangle in the bottom right of the configuration, yielding the intermediate configuration shown in Figure 7.12a. This is followed up by using the two blue input rooks to propagate the two top 0-rooks towards the gray capture destination, as well as performing a leaf capture in the bottom left, as indicated by the arrows in the previous figure, yielding a later intermediate configuration shown in Figure 7.13. After performing the highlighted two pairs of captures among the remaining white rooks, the output rook of the otherwise cleared gadget remains.

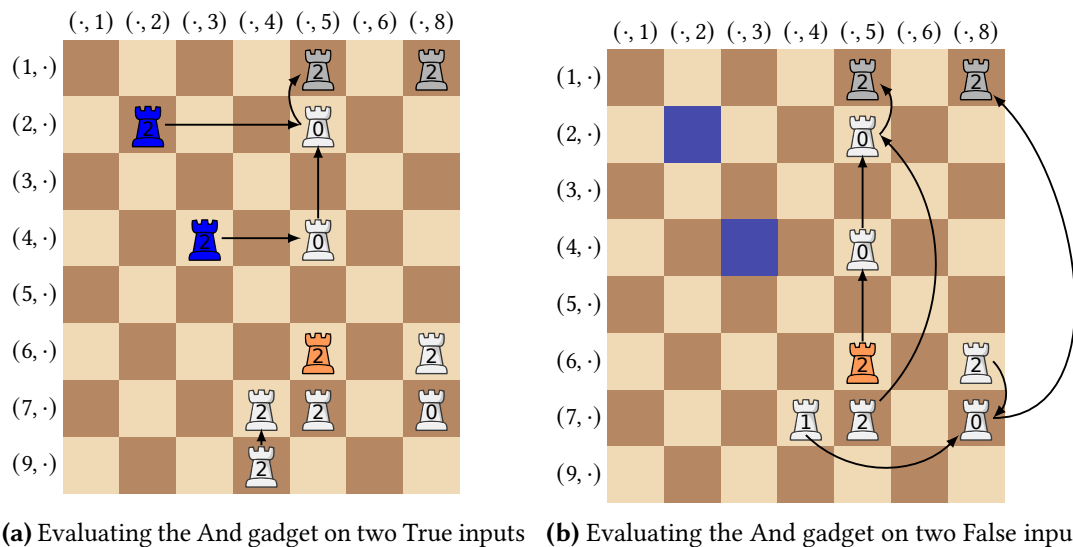


Figure 7.12.: Some (compressed) intermediate configurations reached throughout the evaluation of the And gadget; note the noncontiguous row and column numbering

Consider now the case of none of the inputs being set to True. A clearing sequence is given by again reducing the three triangles and performing the leaf capture, yielding the intermediate configuration shown in Figure 7.12b. The sequence is completed by first cleaning up the two top 0-rooks, before resolving the remaining 0-rook, as indicated in the figure. This fully clears the gadget, with the output being set to False, as claimed. In the case of a single input being set to True, the gadget can be cleared similarly. ■

Before showing that these are the maximum possible output values, we briefly give some intuition on the workings of the gadget. The key idea of the gadget is to use the fact that rooks cannot jump over other rooks. We utilize this fact by creating a sort of cyclic dependency of necessary rook captures that can each only be performed after some other rook capture has been performed. Consider a possible intermediate configuration for a scenario with one True input, shown in Figure 7.14b. If rooks were allowed to jump, the capture sequence indicated by the arrows would fully clear the gadget while retaining a True output, violating the definition of the And function. However, the 1-rook capture $((7, 4) \rightarrow (7, 8))$ is only legal, once the blocking 2-rook has performed the capture $((7, 5) \rightarrow (2, 5))$.

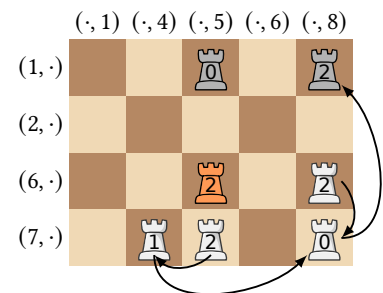


Figure 7.13.: Evaluating the And gadget on two True inputs, part 2

This, in turn, is only possible, once the orange output rook has moved out of the way, capturing to the right towards the next gadget. This capture, in turn, is blocked by the 2-rook on $(6, 8)$, which first needs to perform its pair of capture $((6, 8) \rightarrow (7, 8) \rightarrow (1, 8))$. Finally, both of these two captures are played after the initial 1-rook capture $((7, 4) \rightarrow (7, 8))$: If both captures were played prior to the 1-rook capture, then the latter's destination square would be vacated and, thus, it would not be a (valid) capture. If the first of the two captures was played prior to the 1-rook capture, then the second capture would no longer be valid, as the rook's budget would have become 0. Overall, there is no possible ordering of the captures such that each is

a valid capture. We give a more formal version of this argument in the proof of the following lemma. Nevertheless, for the sake of brevity, we do omit some details. As before, we note that this gadget, like all others, has been computer verified to work correctly.

Lemma 7.13: *For any set of input values for the And gadget, the output values given in Lemma 7.12 are the maximum possible output values.*

Proof. For two True inputs, the gadget evaluates to the maximum possible output value of True. It remains to show that it is not possible to achieve a True output with only one or no True inputs. We show this statement for the case of a second input of True, the other cases are similar. We discuss some properties of possible solving sequences.

Claim 1: Any solving sequence reduces the triangle of each gadget input to a 0-rook on its last squares.

Proof. By Lemma 7.8, none of the gadget inputs capture towards the variable assignment gadgets to the left of the gadget. The claim then follows from Lemma 7.2.

Claim 2: If there exists a solving sequence, then there also exists one which begins with leaf captures $((9, 4) \rightarrow (7, 4))$ and $((8, 9) \rightarrow (7, 9))$.

Proof. Let a solving sequence be given. We discuss the first part of the claim, on the leaf capture $((9, 4) \rightarrow (7, 4))$. Since the rook on $(9, 4)$ is a leaf, the solving sequence does contain the capture at some point. Consider the reordered sequence that begins with the capture $((9, 4) \rightarrow (7, 4))$. If every capture of the sequence is still valid, then this is a solving sequence with the claimed property.

Otherwise, there exists a move $(z_1 \rightarrow z_2)$ that is the first move which is no longer valid in the reordered sequence. The move is played on a board containing a subset of present pieces as to compared when it was played in the original sequence. Thus, it is not blocked by any piece. Furthermore, since the $(9, 4)$ -rook was a leaf, we can deduce that it was not adjacent to the z_1 -rook, i.e., it was not the destination of the capture. The remaining option to no longer be valid is if the rook had a greater than 0 budget in the original sequence, but now has a budget of 0. The only rook whose budget is different in the reordered sequence compared to the original sequence is the rook on $(7, 4)$. In the original sequence, it performs a capture *after* the $((9, 4) \rightarrow (7, 4))$ capture, thus, if its capture is still valid in the reordered sequence, this results in a 0-rook in either sequence, leading to identical configurations. The remaining option is that its capture is no longer valid, which happens precisely in the scenario that the original sequence contains a capture $(z_3 \rightarrow (7, 4))$ resulting in a 0-rook, followed by the leaf capture, which increases the budget of the $(7, 4)$ -rook to 1. In this case, in the reordered sequence, the $(7, 4)$ -rook remains with a budget of 0 instead of 1, and cannot perform the capture $((z_1 = (7, 4)) \rightarrow z_2)$. Observe that in a solving sequence, z_2 is not the right-most rook on $(7, 9)$, since otherwise this rook would be stranded (possibly together with the $(8, 9)$ -rook). Thus, due to the rook's non-jumping nature, we deduce that $z_3 = (7, 5)$ and $z_2 = (7, 8)$. In this case, replacing the subsequence containing captures $((7, 5) \rightarrow (7, 4)), ((9, 4) \rightarrow (7, 4)) \rightarrow (7, 8)$ with captures $((9, 4) \rightarrow (7, 4)), ((7, 5) \rightarrow (7, 8)), ((7, 4) \rightarrow (7, 8))$ transforms the original solving sequence into one that starts with the leaf capture. The second part of the claim holds by an analogous (mirrored) argument. We omit the details here.

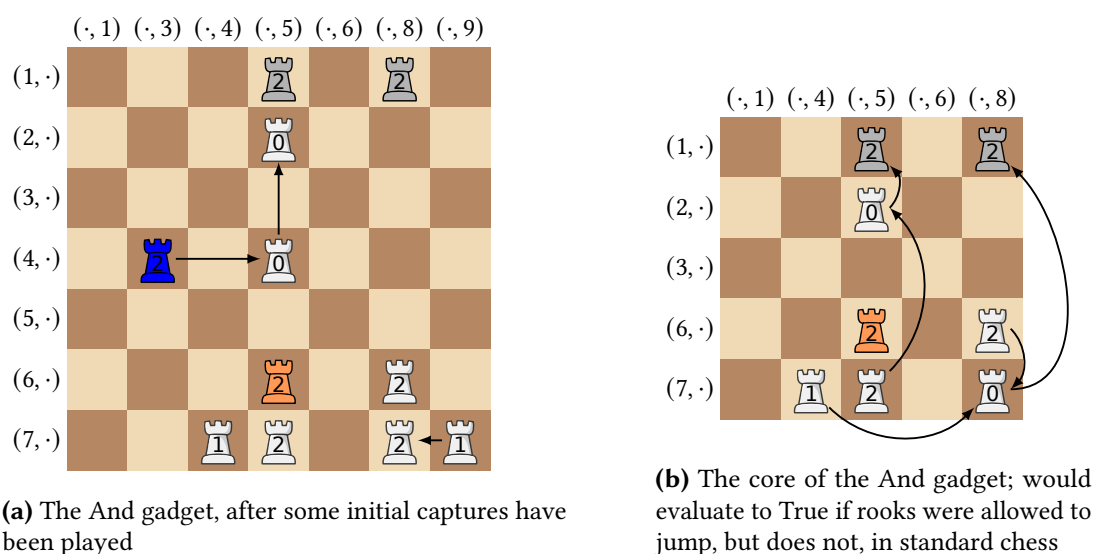


Figure 7.14.: Some intermediate configurations reached throughout the evaluation of the And gadget with one True and one False input; neither configuration evaluates to an output of True.

After reducing the triangles to 0-rooks and performing the leaf captures, we are left with the configuration shown in Figure 7.14a. For the next claims, we make the simplifying assumption that some solving sequence contains the pair of captures $((4, 3) \rightarrow (4, 5) \rightarrow (2, 5))$.

Claim 3: Any solving sequence reaching the configuration shown in Figure 7.14a and containing the pair of captures $((4, 3) \rightarrow (4, 5) \rightarrow (2, 5))$, also contains the capture $((7, 9) \rightarrow (7, 8))$.

Proof. Consider the 1-rook on $(7, 9)$. To not end stranded, it performs a capture $((7, 9) \rightarrow z_4)$ at some point. Observe that $z_4 \neq (7, 4)$, as otherwise, the resulting rook would be isolated and, thus, stranded.

Assume for contradiction that the solving sequence contains the 1-rook capture $((7, 9) \rightarrow (z_4 = (7, 5)))$ (after the blocking $(7, 8)$ -rook has either captured it or moved out of the way). Then, after the leaf capture $((7, 4) \rightarrow (7, 5))$, the resulting rook has a budget of 0. By Observation 3.3, any solving sequence contains a capture of a 2-rook towards $(7, 5)$, followed by a capture from $(7, 5)$ to another square. This 2-rook is not the gray shell rook, as otherwise the resulting rook would be isolated and stranded. If it is instead the orange output rook, then after the capture $((6, 5) \rightarrow (7, 5))$, the only rook of the column adjacent to rooks outside the column is the gray shell rook. However, by the 1D Rook Lemma 4.3, there exists no capture sequence that clears the column towards the gray shell rook. Thus, there remain stranded pieces, contradicting the assumption.

By method of exclusion, the solving sequence instead contains the capture $((7, 9) \rightarrow (7, 8))$, as claimed.

As pointed out before, the resulting configuration of Figure 7.14b could be fully cleared while retaining an output of True, if rooks were allowed to jump over other pieces. However, since rooks are not allowed to jump over other pieces, we show:

Claim 4: There is no solving sequence for the configuration shown in Figure 7.14b that retains an output of True.

Proof. Consider the 0-rook on (7, 8). By Observation 3.3, any solving sequence contains a capture from a 2-rook to the 0-rook, followed by a capture away from the square. This pair of captures is not performed by the (7, 5)-rook, as that would leave the 1-rook on (7, 4) isolated and stranded. It is also not performed by the gray shell rook on (1, 8) (after the (6, 8)-rook has moved out of the way), since after a pair of captures $((1, 8) \rightarrow (7, 8) \rightarrow (7, 5))$, the resulting configuration cannot be cleared as seen in the proof of the previous claim. Thus, instead, the solving sequence contains the pair of captures $((6, 8) \rightarrow (7, 8) \rightarrow (1, 8))$. It follows, in particular, that the (6, 8)-rook does not serve as an input of True for the next gadget.

Consider now the (5, 7)-rook below the orange output square rook. It makes a capture at some point. If it captures to the right, then the 0-rook on (2, 5) is cleaned up by the orange (6, 5)-rook, which, therefore, is not retained as an output of True, as claimed. If it captures the orange rook itself, once more, no output of True is retained. Finally, if it captures a rook above the orange rook, then it does so after the orange rook has moved out of the way. However, it does so before the 2-rook on (6, 8) has performed its pair of captures, as otherwise the 1-rook on (7, 4) would remain stranded. Thus, the orange rook has moved out of the way while the 2-rook to its right was still blocking its path, showing that it did not serve as an input of True for the next gadget, as claimed. This concludes the case distinction and shows the claim.

We combine the shown claims. Let a solving sequence be given that evaluates an And gadget with a second input of True. By Claim 1, this sequence reduces each gadget input triangle to a 0-rook on its last square. By Claim 2, the sequence can be reordered to continue with the leaf captures of $((9, 4) \rightarrow (7, 4))$ and $((8, 9) \rightarrow (7, 9))$, reaching the configuration of Figure 7.14a. Under the assumption that the solving sequence contains the pair of captures $((4, 3) \rightarrow (4, 5) \rightarrow (2, 5))$, Claims 3 and 4 then combine to show the statement of the lemma. It remains to argue that if there is a solving sequence, then there is one that contains the assumed pair of captures. For this, we only give a high level argument.

Any solving sequence contains the blue rook capture $((4, 3) \rightarrow (4, 5))$. Consider alternatives to the follow-up capture $((4, 5) \rightarrow (2, 5))$: The solving sequence does not contain the capture $((4, 5) \rightarrow (1, 5))$ instead, since it is blocked by the 0-rook on (2, 5), which cannot be propagated out of the way. If the solving sequence contains a capture from (4, 5) to one of the rooks below it, this reduces that 2-rook's budget to 0. Then, a similar proof to the one given above shows that the solving sequence does not produce an output of True. Finally, if the solving sequence does not contain a capture by the (4, 5)-rook before that one is captured itself, then this effectively adds an extra 0-rook to the configuration discussed above. Then, a similar proof as above shows that there is no solving sequence that produces an output of True. This shows the statement of the lemma for an And gadget with a second input of True. The proofs for And gadgets with a second input of False and a first input of True or False, are once more analogous to the above proof. ■

For the And gadget, excluding the input square rooks, the only rooks adjacent to rooks of other gadgets are the ones in the sixth row, i.e., the orange output square rook and the white 2-rook to its right. They are in line with the gadget input of the next gadget. Thus, it again remains to discuss the scenario of a capture from a triangle rook of the next gadget to a sixth row rook of the And gadget.

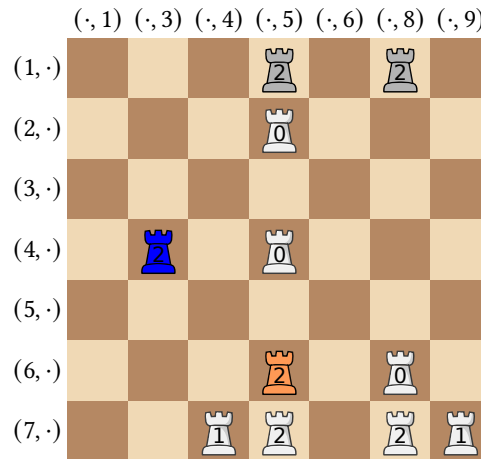


Figure 7.15.: An And gadget, being captured into from the right

Lemma 7.14: *Let a well-formed configuration containing an And gadget with at most one input set to True be given. Then, there exists no solving sequence containing a capture from the gadget input of the next gadget to the And gadget.*

Proof. Once more, we consider the case of the second input being set to True. Assume for contradiction that there exists a solving sequence containing a capture from a gadget input into the And gadget, i.e., into one of the two rooks in the row of the output square. By Lemma 7.7, this results in a 0-rook on the square of that rook, with no rooks of the gadget input remaining. For either possibility of the position of that 0-rook, we observe that Claims 1 and 2 from the previous lemma hold with identical proofs. We now consider the first scenario of the (6, 8)-rook being captured towards, yielding the configuration shown in Figure 7.15.

Claim 1: Any solving sequence of the configuration shown in Figure 7.15 contains the move $((7, 4) \rightarrow (7, 5))$.

Proof. Assume for contradiction that there exists a solving sequence that does not contain the move. The solving sequence contains a capture by the 1-rook on (7, 4), since the rook otherwise stays stranded. The solving sequence does not contain the capture $((7, 4) \rightarrow (7, 9))$, since the resulting 0-rook would be isolated and stranded. Assume that the solving sequence contains the capture $((7, 4) \rightarrow (7, 8))$. This results in two 0-rooks in the eighth column. Applying Observation 3.3 twice, we see that the solving sequence contains captures $(z_1 \rightarrow (6, 8) \rightarrow z_2)$ and $(z_3 \rightarrow (7, 8) \rightarrow z_4)$ in some order, for some z_1, z_2, z_3, z_4 of which at most z_2 and z_4 are the same. Observe that the only possible choice under these restrictions is the sequence of captures $((6, 5) \rightarrow (6, 8) \rightarrow (7, 8)), ((1, 8) \rightarrow (7, 8) \rightarrow (7, 9))$. The resulting 0-rook on (7, 9) is isolated and stranded, a contradiction to the assumed capture $((7, 4) \rightarrow (7, 8))$. By method of exclusion, the claim holds.

Claim 2: The configuration of Figure 7.15 followed by the move $((7, 4) \rightarrow (7, 5))$ has no solving sequence.

Proof. Assume for contradiction that there exists a solving sequence. Then it contains a capture by the (7, 8)-rook. Assume that it contains the capture $((7, 8) \rightarrow (7, 5))$ or the capture $((7, 8) \rightarrow (7, 9))$. In either case, the 1-rook on (7, 9) is the first vertex of an antenna of length 1, which, by Lemma 3.5, is reduced to a 0-rook on (7, 5). Applying

Observation 3.3 thrice, by a similar argument as above, shows that at least one of the 0-rooks ends up stranded, contradicting the assumed captures $((7, 8) \rightarrow (7, 5))$ or $((7, 8) \rightarrow (7, 9))$. Next, assume that the solving sequence contains some capture $((7, 8) \rightarrow (x, 8))$ towards the top. Then, after the $((7, 9) \rightarrow (7, 5))$ -capture, the same argument shows that at least one of the 0-rooks ends up stranded, a contradiction. By method of exclusion, the claim holds.

Combining the two claims, we see that there is no solving sequence for the configuration shown in Figure 7.15.

Consider now the second scenario of a capture from the right towards the $(6, 5)$ -rook. Since rooks cannot jump over other pieces, this happens after the $(6, 8)$ -rook has moved out of the way. It results in three 0-rooks in the fifth column and three adjacent 2-rooks. Once more, applying Observation 3.3 thrice shows that at least one of the 0-rooks ends up stranded, the final contradiction. We conclude that the assumption is false, which shows the statement of the lemma. The proofs for a second input of False and a first input of True or False are, again, analogous to the above proof. ■

7.5. The Right Or Gadget

The final gadget is the right Or gadget, shown in Figure 7.16. It closely resembles the left Or gadget, however, while it does have two inputs, it does not have an output. Instead, being the final gadget of each clause, it can be fully cleared if the Or function evaluates its inputs to True, and it cannot be fully cleared if the Or function evaluates its inputs to False. For the latter statement, to account for the possibility of capturing to the left, we show a slightly weaker statement:

Lemma 7.15: *If at least one of its inputs is set to True, it is possible to fully clear the right Or gadget. Conversely, if there exists a solving sequence for a well-formed configuration (which fully clears each right Or gadget), then there exists one that sets at least one of the inputs for each right Or gadget to True.*

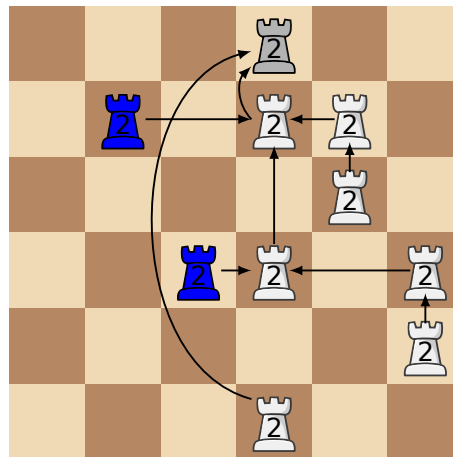


Figure 7.16.: A right Or Gadget with two blue inputs and no output

Proof. For the first claim, let at least one of the inputs be set to True. Then a clearing capture sequence is given by first reducing each triangle to a 0-rook on its last square, followed by propagating each 0-rook to the gray shell rook, using the 2-rook(s) on the input squares, as well as the remaining white 2-rook.

Now, let a solving sequence be given which sets none of the inputs of a right Or gadget to True. Assume for contradiction that the sequence does not contain any captures from the right Or gadget to the left. Then, by Lemma 7.2, the triangle of each gadget input is reduced to a 0-rook on its last square. The resulting configuration of a single column containing two

2-rooks and two 0-rooks can, by the 1D Rook Lemma 4.3, not be cleared towards the gray shell rook. This means that some pieces remain stranded, a contradiction. Thus, we deduce that the capture sequence contains a capture from the right Or gadget towards the left.

By Lemma 7.8, the capture to the left is not towards a variable assignment gadget. Thus, the capture is either towards a left Or gadget or an And gadget. In the first case, by Lemma 7.11, the left Or gadget had at least one input of True. Then, by Lemma 7.10, there exists a clearing sequence, which evaluates the left Or gadget to True. Replacing the subsequences of the left and the right Or gadget being cleared with one that evaluates the left Or gadget to True and then fully clears the right Or gadget, yields a solving sequence which sets one of the inputs for the right Or gadget to True. In the second case, by Lemma 7.14, the And gadget had both inputs set to True. Then, by Lemma 7.12, there exists a clearing sequence which evaluates the And gadget to True. The same replacement as before yields a solving sequence which sets one of the inputs for the right Or gadget to True.

Repeating this for each right Or gadget yields an overall solving sequence with the claimed properties. ■

7.6. The Final Reduction

We now describe our transformation of an And-Or-(1,1)-SAT instance into a Rook 2-SOLO CHESS instance. Let an And-Or-(1,1)-SAT instance $I = (U, C)$ be given. Choose an And-Or Embedding of I that does *not* include the optional conjunction of the outputs of the clauses. Adjust the vertical positions of the variable assignments and the horizontal position of the clauses such that they are not aligned with other variable assignments and clauses (accounting for the rook's unlimited range). Then, based on the adjusted embedding, place for each variable $u \in U$ a variable assignment gadget on the board at sufficient distance to all other variable assignment gadgets. For each clause, place the respective logic gadgets on the board according to the embedding, aligning the outputs and inputs of the SAT gadgets as required from the SAT formula. Place the shell around the bounding box that contains all SAT gadgets, containing capture destination rooks placed as indicated in the various gadgets, a sufficient number of cleanup rooks, as well as the tomb. An example of such a placement on the board is shown in Figure 7.1. Using this construction, we now prove:

Theorem 7.16: *Rook SOLO CHESS is NP-hard.*

Proof. We reduce from And-Or-(1,1)-SAT. We transform a given instance as described above. This transformation runs in polynomial time (in fact, in linear time): Let n be the number of variables and m be the number of clauses of the SAT instance. For each variable, a single variable assignment gadget is placed on the board, for a total of n gadgets. For each clause, at most two logic gate gadgets are placed on the board, for a total of at most $2m$ gadgets. Each of these gadgets contains only a constant number of rooks. For each gadget, a constant number of shell rooks are placed on the board. Finally, the tomb consists of a constant number of rooks. In total, $\mathcal{O}(n + m)$ rooks are placed on the board. Furthermore, the transformation can be computed in time proportional to its output size, i.e., linear time. It remains to show that the two instances are in fact equivalent.

SAT instance solvable \implies Chess instance solvable: Let Φ be a satisfying assignment for the SAT instance. Then a solving sequence consists of the following steps: For each variable assignment gadget, assign the value True to the literal that is set to True by Φ , and the value False to the other literal (see Lemma 7.9). For each clause, evaluate each logic gadget.

Since each clause is satisfied by Φ , by Lemmata 7.10 and 7.12, at least one input of the right Or gadget of the clause is True. Then, each of the variable assignment gadgets and each of the clause gadgets are cleared fully. By Lemma 7.1, there is a solving sequence for the remaining configuration of the shell.

Chess instance solvable \implies SAT instance solvable: Let the chess instance have a solving sequence. Then, by Lemma 7.15, there exists one that sets at least one input for each right Or gadget to True. On this solving sequence, the following procedure yields a satisfying assignment Φ for the SAT instance: For each variable assignment gadget check which of the two outputs is set to True, i.e., for which of the two output square rooks a capture to the right, as a 2-rook, is contained in the sequence. If that output corresponds to some literal $\ell = x$, set $\Phi(x) := \text{True}$. If it corresponds to some literal $\ell = \neg x$, set $\Phi(x) := \text{False}$. If neither of the two outputs is set to True, the value of the variable does not matter, so we set $\Phi(x) := \text{True}$. Then Φ is a satisfying assignment:

By Lemma 7.9, at most one output of each variable assignment is set to True. Thus, Φ is well-defined. We also see that every clause is satisfied: By definition, for each clause, at least one input of its right Or gadget is set to True. If that input is the output of a variable assignment gadget, then the corresponding literal satisfies the clause in the SAT instance under Φ . Otherwise, it is the output of a left Or gadget or an And gadget, in which case, by Lemma 7.10 or Lemmata 7.12 and 7.13, respectively, it follows again that the SAT clause is satisfied under Φ .

Overall, we conclude that the two instances are equivalent, which shows the NP-hardness of Rook 2-SOLO CHESS. ■

8. Conclusion

In this thesis, we analyzed the complexity of the SOLO CHESS decision problem. We began by introducing the And-Or-(1,1)-SAT problem and showed its NP-completeness. This variant of SAT is unique in that each literal only occurs once, making it useful in deriving other hardness results. In particular, we reduced from And-Or-(1,1)-SAT to show the NP-completeness of different uniform 2-SOLO CHESS variants. For the short-range pieces of the king and the knight, we presented a method for embedding SAT formulas onto the chess board using specific gadgets for variable assignments, wires and logic gates. This construction allowed us to directly translate the SAT instance into an equivalent SOLO CHESS instance. For the long-range piece of the rook, we encoded each clause independently, using gadgets for variable assignments and logic gates. Thanks to the rook’s unlimited range, the satisfaction for each clause could be checked individually and without the need for an additional wire gadget.

The new results for King 2-SOLO CHESS and Knight 2-SOLO CHESS, combined with the previous results for Rook, Bishop, Queen and Pawn 2-SOLO CHESS [AMM22], [BDGL24], complete the computational characterization of (uniform) 2-SOLO CHESS. They also extend to the more general problem of (uniform) ≤ 2 -SOLO CHESS, whose computational complexity is now also completely characterized.

Additionally, our study of $\leq B$ -SOLO CHESS played on a one-dimensional board yielded a linear-time algorithm for any constant B . The algorithm performs a linear scan through the configuration, maintaining an interface whose size depends solely on B . This result holds true for any combination of piece types and any (constant) upper bound B for piece budgets, providing a complete resolution of the one-dimensional variant.

Future Work

Pawn 2-SOLO CHESS is the only uniform 2-SOLO CHESS variant which is solvable in polynomial time (assuming $P \neq NP$) [AMM22]. Further research could explore whether this holds true for larger piece budgets. Similarly, the complexity increase from (uniform) 1-SOLO CHESS, solvable in linear time, to (uniform) 2-SOLO CHESS, which is NP-complete, suggests that (uniform) B -SOLO CHESS remains NP-complete for any constant $B > 2$. For the special case $B = 11$ with knights this has been shown [BDGL24]. One possible approach to show a general result for any $B > 2$ would be to generalize the gadgets of our SAT reductions to work with any piece budgets. Recall that antennae of length B can simulate 0-pieces, however, this approach does not work to simulate pieces with budgets greater than 0 (see also Appendix A).

On the other hand, uniform SOLO CHESS with piece budgets equal to the number of pieces n is solvable in polynomial time [Bru+23]. It would be interesting to study different variants with large piece budgets, say $\frac{n}{c}$ or $n^{\frac{1}{c}}$ for some $c > 1$, and determine at which point the problem becomes solvable in polynomial time.

Other extensions include investigating various other piece types such as the golden and silver general from Shogi (Japanese chess), the cannon from Xiangqi (Chinese chess), or various fairy chess pieces. More generally, it would be interesting to characterize which movement properties lead to a uniform SOLO CHESS problem being solvable in polynomial

time or, alternatively, being NP-complete. Modifying the game rules, for example allowing pieces to move without capturing, may lead to new complexity insights. Under this rule some problem variants, such as Rook 2-SOLO CHESS, become trivial as any rook can reach any square of the board within two moves. However, problem variants combining multiple different piece types may still turn out to be hard.

Further research could focus on parameterized versions of 2-SOLO CHESS. These include 2-SOLO CHESS played on a board with a bounded number of rows (but an unlimited number of columns). We have shown that for a board containing only a single row, this problem is solvable in linear time, while for an unlimited number of rows it is NP-complete. However, the complexity of intermediate cases remains unknown. It would for example be interesting to study whether 2-SOLO CHESS parameterized by the number of rows is fixed-parameter tractable (FPT).

All chess images in this paper use the piece set 'staunty' by sadsnake1 under CC BY-NC-SA 4.0 Deed, of the website lichess.org.

Bibliography

- [AD21] Francesco Arena and Miriam Di Ianni. “Complexity of Scorpion Solitaire and applications to Klondike”. In: *Theoretical Computer Science* Volume 890 (2021), pp. 105–124.
- [AKI87] Hiroyuki Adachi, Hiroyuki Kamekawa, and Shigeki Iwata. “Shogi on $n \times n$ board is complete in exponential time”. In: *Trans. IEICE* Volume 70 (1987), pp. 1843–1852.
- [AMM22] N.R. Aravind, Neeldhara Misra, and Harshil Mittal. “Chess Is Hard Even for a Single Player”. In: *11th International Conference on Fun with Algorithms (FUN 2022)*. Edited by Pierre Fraigniaud and Yushi Uno. Vol. 226. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, 5:1–5:20. ISBN: 978-3-95977-232-7. DOI: [10.4230/LIPIcs.FUN.2022.5](https://doi.org/10.4230/LIPIcs.FUN.2022.5).
- [BDGL24] Davide Bilò, Luca Di Donato, Luciano Gualà, and Stefano Leucci. “Uniform-Budget Solo Chess with Only Rooks or Only Knights Is Hard”. In: *12th International Conference on Fun with Algorithms (FUN 2024)*. Edited by Andrei Z. Broder and Tami Tamir. Vol. 291. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024, 4:1–4:19. ISBN: 978-3-95977-314-0. DOI: [10.4230/LIPIcs.FUN.2024.4](https://doi.org/10.4230/LIPIcs.FUN.2024.4).
- [BDHW20] Josh Brunner, Erik D Demaine, Dylan Hendrickson, and Julian Wellman. “Complexity of retrograde and helpmate chess problems: Even cooperative chess is hard”. In: *arXiv preprint arXiv:2010.09271* (2020).
- [Bru+23] Josh Brunner, Lily Chung, Michael Coulombe, Erik D. Demaine, Timothy Gomez, and Jayson Lynch. *Complexity of Solo Chess with Unlimited Moves*. 2023. arXiv: [2302.01405](https://arxiv.org/abs/2302.01405).
- [Coo71] Stephen A. Cook. “The complexity of theorem-proving procedures”. In: *Proceedings of the Third Annual ACM Symposium on Theory of Computing*. Shaker Heights, Ohio, USA: Association for Computing Machinery, 1971, pp. 151–158. ISBN: 9781450374644. DOI: [10.1145/800157.805047](https://doi.org/10.1145/800157.805047).
- [FL81] Aviezri S Fraenkel and David Lichtenstein. “Computing a perfect strategy for $n \times n$ chess requires time exponential in n ”. In: *International Colloquium on Automata, Languages, and Programming*. Springer, 1981, pp. 278–293.
- [Hea05] Robert A. Hearn. *Amazons is PSPACE-complete*. 2005. arXiv: [cs/0502013](https://arxiv.org/abs/cs/0502013).
- [Hel03] Malte Helmert. “Complexity results for standard benchmark domains in planning”. In: *Artificial Intelligence* Volume 143 (2003), pp. 219–262.
- [IK94] Shigeki Iwata and Takumi Kasai. “The Othello game on an $n \times n$ board is PSPACE-complete”. In: *Theoretical Computer Science* Volume 123 (1994), pp. 329–340.
- [LM09] Luc Longpré and Pierre McKenzie. “The complexity of solitaire”. In: *Theoretical Computer Science* Volume 410 (2009), pp. 5252–5260.

- [LS78] David Lichtenstein and Michael Sipser. “Go is pspace hard”. In: *19th Annual Symposium on Foundations of Computer Science (sfcs 1978)*. IEEE Computer Society. 1978, pp. 48–54.
- [Rei80] Stefan Reisch. “Gobang ist PSPACE-vollständig”. In: *Acta Informatica* Volume 13 (1980), pp. 59–66.
- [Rei81] Stefan Reisch. “Hex ist PSPACE-vollständig”. In: *Acta Informatica* Volume 15 (1981), pp. 167–191.
- [Rob83] John Michael Robson. “The complexity of Go”. In: *Proc. IFIP, 1983* (1983).
- [Rob84a] John Michael Robson. “Combinatorial games with exponential space complete decision problems”. In: *Mathematical Foundations of Computer Science 1984: Proceedings, 11th Symposium Praha, Czechoslovakia September 3–7, 1984* 11. Springer. 1984, pp. 498–506.
- [Rob84b] John Michael Robson. “N by N checkers is Exptime complete”. In: *SIAM Journal on Computing* Volume 13 (1984), pp. 252–267.
- [RS24] Benjamin G Rin and Atze Schipper. “Arimaa Is PSPACE-Hard”. In: *12th International Conference on Fun with Algorithms (FUN 2024)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. 2024.
- [RW86] Daniel Ratner and Manfred Warmuth. “Finding a shortest solution for the N x N extension of the 15-puzzle is intractable”. In: *Proceedings of the Fifth AAAI National Conference on Artificial Intelligence*. 1986, pp. 168–172.
- [sad] sadsnake1. “The piece set ‘staunty’ on the website lichess.org”. Licensed under CC BY-NC-SA 4.0 <https://creativecommons.org/licenses/by-nc-sa/4.0/>. URL: <https://github.com/lichess-org/lila/tree/master/public/piece/staunty>. (accessed: 2024.05.21).
- [Ste11] Jesse Stern. *Spider Solitaire is NP-Complete*. 2011. arXiv: 1110.1052.
- [Sto83] James A Storer. “On the complexity of chess”. In: *Journal of computer and system sciences* Volume 27 (1983), pp. 77–100.
- [Tov84] Craig A. Tovey. “A simplified NP-complete satisfiability problem”. In: *Discrete Applied Mathematics* Volume 8 (1984), pp. 85–89. ISSN: 0166-218X. DOI: 10.1016/0166-218X(84)90081-7.
- [UI90] Ryuhei Uehara and Shigeki Iwata. “Generalized Hi-Q is NP-complete”. In: *IEICE TRANSACTIONS (1976-1990)* Volume 73 (1990), pp. 270–273.
- [Zha19] Zhujun Zhang. *A Note on Hardness Frameworks and Computational Complexity of Xiangqi and Janggi*. 2019. arXiv: 1904.00200.
- [Zha22] Zhujun Zhang. “A Note on the Computational Complexity of Selfmate and Reflexmate Chess Problems”. In: *arXiv preprint arXiv:2208.05376* (2022).

A. Counter Example for the Original Queen 2-SOLO CHESS Proof

In this chapter we discuss a claim made in the seminal paper on SOLO CHESS by Aravind, Misra and Mittal [AMM22]. They give a reduction from the NP-complete problem of Red-Blue Dominating Set (RBDS) to Rook ≤ 2 -SOLO CHESS. Later, they describe how to transform the resulting Rook ≤ 2 -SOLO CHESS instances into equivalent Queen ≤ 2 -SOLO CHESS instances. Figures A.1a and A.1b give an example, transforming a No-instance of RBDS into a No-instance of Queen ≤ 2 -SOLO CHESS.

They then sketch how to modify this transformation to produce an equivalent instance of Queen 2-SOLO CHESS. To this end, they describe a method to simulate 1-queens using pairs of 2-queens. Figure A.2 shows the resulting instance after performing the suggested transformation on the above example. Note that for the sake of clarity, we keep the 0-queen on (1, 5) intact. It can be simulated by an antenna of length 2. The idea of the modified transformation is that the added 2-queens in the bottom right quadrant capture their partner queens in the top left quadrant to yield the desired 1-queens. However, by not immediately performing these captures, it is possible to gain an additional unit of budget later on. A specific solving sequence is given through the moves $((2, 3) \rightarrow (2, 1)), ((8, 7) \rightarrow (2, 1) \rightarrow (1, 1)), ((6, 6) \rightarrow (1, 1) \rightarrow (1, 5))$. This shows that the transformed instance is a Yes-instance, despite the original instance being a No-instance. We conclude that the transformation does not produce equivalent instances, and so, the NP-completeness of Queen 2-SOLO CHESS cannot be shown in this way. However, we remark that the result itself still holds. Since Rook 2-SOLO CHESS has since been shown to be NP-complete [BDGL24], the transformation from Rook SOLO CHESS to Queen SOLO CHESS yields the desired hardness result.

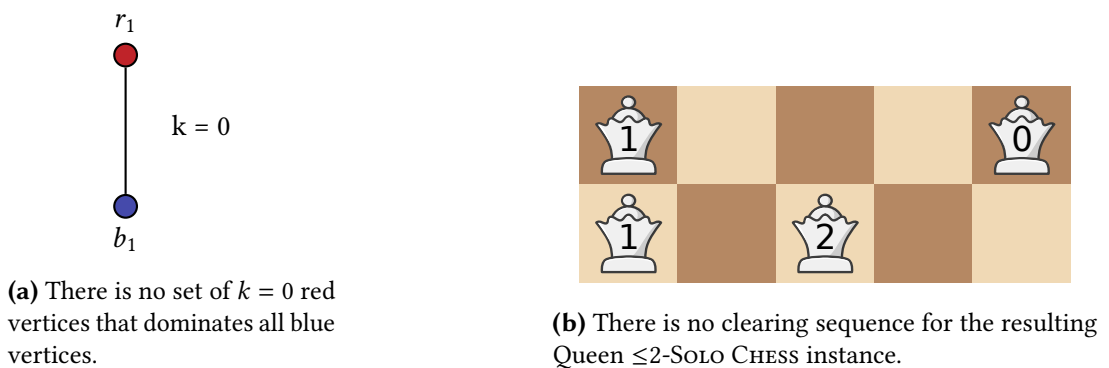


Figure A.1.: A transformation from RBDS to Queen ≤ 2 -SOLO CHESS.

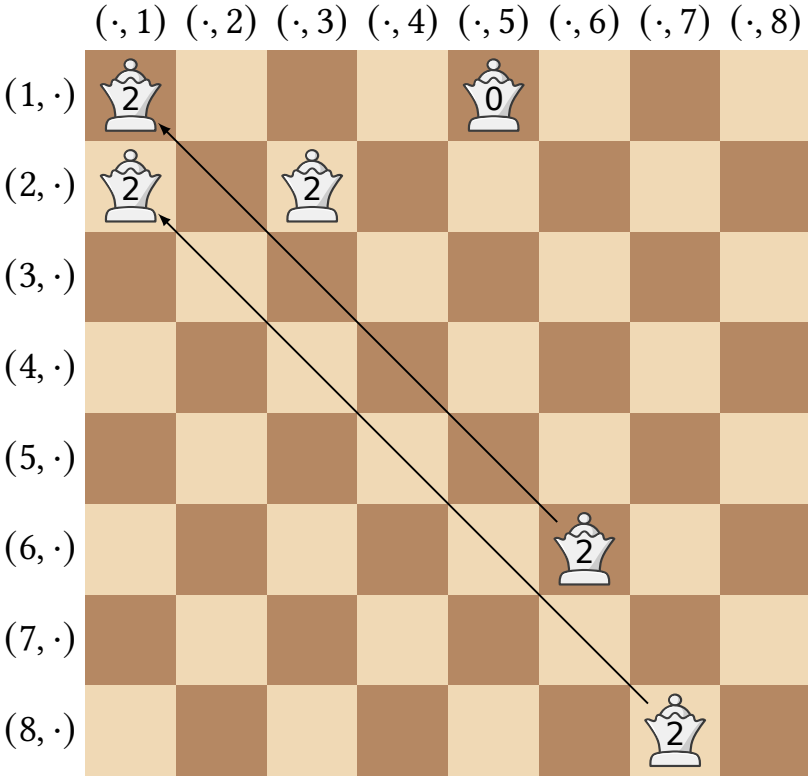


Figure A.2.: Simulating 1-queens using pairs of 2-queens. This SOLO CHESS instance is solvable even though the original RBDS instance was not solvable.

B. King SOLO CHESS Wire Crossing

In this chapter, we elaborate on the decomposition of function F of the wire crossing gadget introduced for King 2-SOLO CHESS.

We decompose the function $F(a, b, \neg\tilde{b})$ described in Table B.1 into functions with two parameters each. We show below that under the monotony requirement imposed by King 2-SOLO CHESS, there exists no decomposition into just two functions. Instead, we give a decomposition into three functions of the form $F(a, b, \neg\tilde{b}) = h(g(f(a, \neg\tilde{b}), b), \neg\tilde{b})$. This is possible by creating multiple copies of the $\neg\tilde{b}$ literal using the Double Assignment gadget.

a	$\neg\tilde{b}$	b	$\neg\tilde{b}$	$F(a, b, \neg\tilde{b})$	Row
0	0	0	0	-1	(1)
0	1	0	1	0	(2)
0	0	1	0	0	(3)
0	1	1	1	0	(4)
1	0	0	0	-1	(5)
1	1	0	1	1	(6)
1	0	1	0	1	(7)
1	1	1	1	1	(8)

We note that under the given restrictions, this type of decomposition gives the maximum expressive power that is possible with two-parameter functions: The only input value of which multiple copies can be created is that of $\neg\tilde{b}$, using a Double Assignment gadget (or a more general Multi Assignment gadget). Due to the planar geometry, the wire carrying the a -signal is a divide between the variable \tilde{b} and the incoming b -signal. It follows that the decomposition cannot utilize a function receiving inputs b and $\neg\tilde{b}$. Instead, every function is placed along the path of the a -signal and receives as inputs the modified value of a and either b or $\neg\tilde{b}$. Utilizing more than one function before or after the function receiving the incoming b -signal does not give additional expressive power: Any sequence of monotone functions receiving a copy of $\neg\tilde{b}$ each, can be collapsed to a single function which is still monotone. Thus, one function on either side of the “additional information” b already gives the maximum in expressive power.

Table B.1.: Function F to be decomposed.

We now construct functions f , g and h that compose to F . We number the different input tuples according to Table B.1. We use the fact that each of the functions is monotone in each argument to compare pairs of tuples and deduce inequalities between different intermediate values:

(4) + (7) give us that $g(f(1, 0), 1) > g(f(0, 1), 1)$, which implies that $f(1, 0) > f(0, 1) \geq^{\text{mon.}} f(0, 0)$.

(3) + (5) give us that $g(f(0, 0), 1) > g(f(1, 0), 0)$.

(4) + (6) give us that $g(f(1, 1), 0) > g(f(0, 1), 1) \geq^{\text{mon.}} g(f(0, 0), 1) > g(f(1, 0), 0)$, which yields that $f(1, 1) > f(1, 0) > f(0, 1) \geq^{\text{mon.}} f(0, 0)$.

Note that every set of inputs to f can be extended to a set of inputs for F that does not produce an error value. Thus, f itself does not produce an error value for any set of inputs, which restricts its output to values between 0 and 2. Thus, using the above chain of inequalities, we deduce that f is defined as shown in Table B.2

a	$\neg\tilde{b}$	$f(a, \neg\tilde{b})$
0	0	0
0	1	0
1	0	1
1	1	2

Table B.2.: Definition of function f

We now discuss the function g , shown in Table B.3. Once again, we deduce from Table B.1.

(3) + (5) give us that $g_1 > g_2$.

(4) + (6) give us that $g_4 > g_1$.

(4) + (7) give us that $g_3 > g_1$.

(2) gives us that $0 \leq g_0 \leq^{\text{mon.}}$

g_2 , i.e., neither are the error value -1.

This yields $0 \leq g_0 \leq g_2 < g_1 < g_4 \leq^{\text{mon.}} g_5 \leq 2$ and $g_1 < g_3 \leq^{\text{mon.}} g_5 \leq 2$. We deduce that function g is defined as shown in Table B.4.

Possible values for $(a, -\tilde{b})$	$f(a, -\tilde{b})$	b	$g(f(), b)$
$(0, 0)/(0, 1)$	0	0	g_0
$(0, 0)/(0, 1)$	0	1	g_1
$(1, 0)$	1	0	g_2
$(1, 0)$	1	1	g_3
$(1, 1)$	2	0	g_4
$(1, 1)$	2	1	g_5

Table B.3.: The setting for function g .

$f()$	b	$g(f(), b)$
0	0	0
0	1	1
1	0	0
1	1	2
2	0	2
2	1	2

Table B.4.: Definition of function g .

Finally, function h is defined as shown in Table B.5, with h_0 being determined by (1), h_1 by (2), h_2 by (3), h_3 by (4), h_4 by (7) and h_5 by (6).

$g()$	$-\tilde{b}$	$h(g(), -\tilde{b})$
0	0	$h_0 = -1$
0	1	$h_1 = 0$
1	0	$h_2 = 0$
1	1	$h_3 = 0$
2	0	$h_4 = 1$
2	1	$h_5 = 1$

Table B.5.: Definition of function h .

This shows that there is at most one way to decompose F into three two-parameter functions. At the same time, it is easy to verify that this decomposition is in fact valid, which shows that there is exactly one way to decompose F into three two-parameter functions. Since neither f nor h are the identity function, it follows, in particular, that a decomposition into two functions is impossible.