The computational complexity of Stratego perfect strategy

By

Mitchell Donkers

Student number: 5720664

A master thesis for the master Artificial Intelligence, Utrecht University

August 15, 2024

First supervisor: Benjamin Rin

Second supervisor: Fan Yang

Abstract

In this thesis we look at some of the proofs of proven EXPTIME-complete games and give our own EXPTIME-completeness proof. We specifically look at chess and American checkers, for which we will describe their EXPTIME-hardness proofs. Furthermore for American checkers we will explain why the current proof is insufficient to prove any other checker variant and show an attempt at adapting it for international checkers. Next we give our own proof for perfect information Stratego being EXPTIME-complete. We do this by first proving a new game, that we call G2.5, to be EXPTIME-complete and making a reduction from that to stratego. The EXPTIME-hardness result within the EXPTIME-completeness proof can also be seen to apply to Stratego without perfect information. Lastly we will give some insights on how to improve both the checkers proof and our stratego proof.

Table of Contents

1 Introduction

1.1 The relevance of the field

Computational complexity in computer science is the study about the amount of resources an algorithm needs to be run. The two most prominent ones are time complexity and space complexity. The first one focuses on computation time. This is measured in a number of steps (or elementary operations) the algorithm takes. The other resource is the required amount of space, it is the number of bits an algorithm needs. In both fields people usually consider the worst case scenario (in the big O notation) relative to the size of the input and specify what kind of complexity the complexity classes of problem instances.

When we instead look at problems themselves and what kind of resources they need to be solved we are talking about computational complexity theory. This also puts the focus on relating complexity classes with each other.

1.2 Complexity of games

There has already been a lot of research done on games in computational complexity theory. For example, many puzzles, which we can say are actually 1-player games, are NP-complete [4][5][6][12][13][14].

Definition **Alphabet (Σ)**: An alphabet Σ is a finite, nonempty set of objects called symbols.

Definition **String**: A string is a finite list of symbols from an alphabet

Definition **Sigma star (Σ^*)**: Σ^* is the set of all possible strings in the alphabet Σ.

Definition **Language**: Given an alphabet Σ, a language is a subset of Σ^*

Definition **Decision Problem**: A decision problem is a yes/no question pertaining to the strings of a given alphabet Σ^* .

It is often the case that a decision problem is identified with the language containing precisely the strings whose answer is 'yes'. Hereafter, we treat a decision problem (or just problem, for brevity) and its corresponding language as essentially the same.

Definition **Turing Machine** [18]: A Turing machine is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where Q, Σ, Γ are all finite sets and …

1. Q is the set of states,
2. Σ is the input alphabet not containing the blank symbol ␣,
3. Γ is the tape alphabet, where $␣ \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R\}$ is the transition function,
5. $q_0 \in Q$ is the start state,
6. $q_{accept} \in Q$ is the accept state, and
7. $q_{reject} \in Q$ is the reject state, where $q_{reject} \neq q_{accept}$.

Definition **computable function**: A function $f : \Sigma^* \longrightarrow \Sigma^*$ is a computable function if some Turing machine M, on every input w, halts with just f(w) on its tape. [18]

Definition **mapping reducible**: Language A is mapping reducible to language B, written $A \leq_m B$, if there is a computable function $f : \Sigma^* \longrightarrow \Sigma^*$, where for every w, $w \in A \Longleftrightarrow f(w) \in B$. The function f is called the **reduction** from A to B. [18]

Definition **NP**: A decision problem is in NP if a solution to that problem can be verified in polynomial time.

Definition **NP-complete**: A problem is NP-complete if the problem is verifiable in polynomial time (it is in NP) and all NP problems are mapping reducible to it in polynomial time (it is NP-hard).

This is often the case for 1-player games (e.g. puzzles) because we can ask the question “is there a winning strategy for the player” and the answer to that is often yes that can be verified in polynomial time. Recently the author of this thesis has proven this for Arukone[3] and Bariasensa [7]. For 2-player games polynomial time is often not possible. Because to verify a winning move of player 1 we have to check that is true for every possible move player 2, and check for each of those moves for every possible move of player 1 if it true etc. This takes about the same amount of time as just checking the original move.

Definition **PSPACE**: A decision problem is in PSPACE if it is solvable by a deterministic Turing machine in polynomial space.

Definition **PSPACE-complete**: A problem is PSPACE-complete if it is solvable in polynomial space (it is in PSPACE) and all PSPACE problems are mapping reducible to it in polynomial time (it is PSPACE-hard).

Definition **EXPTIME**: A decision problem is in EXPTIME if it is solvable by a deterministic Turing machine in exponential time.

Definition **EXPTIME-complete**: A problem is EXPTIME-complete if it is solvable in exponential time (it is in EXPTIME) and every EXPTIME problem is polynomial time reducible to it (it is EXPTIME-hard).

Most 2-player deterministic open information turn based board games are either PSPACE-complete like Amazons [8], Hex [9, 10] and Othello [11] or EXPTIME-complete like Chess [2], Checkers [3]. For our research we focus on the latter complexity class.

A common strategy to prove EXPTIME-hardness for such games is to give an reduction from known EXPTIME-complete games. In section 2.2.1 we give some games from stockeyer and Chandra that are often used (g1 to G6) [1]. Later on, in sections 2.2.2 and 2.2.3, we give examples of EXPTIME-hardness reductions from those games to chess [2] and checkers [3] respectively.

1.3 Results

Our first observation is that Robson’s proof for American checkers does not work for international checkers. We explore why his proof does not work for these variants and explain what is necessary for these variants. We will first expand on Robson’s statement that checkers is in EXPTIME. In section 3 we will prove for multiple games (chess, checkers and Stratego) that they are in EXPTIME. Then we will give our view on checker variants and show why Robson’s proof does not suffice for other checker variants in section 4.

In this thesis our main theorem is going to be that the game Stratego is EXPTIME-complete for an N x N generalized Stratego board. For readers unfamiliar with the game we will explain the rules of regular 10 x 10 Stratego in section 5.1. The generalization to an N x N game is pretty straightforward for the most parts, but we will still make some choices regarding lakes, pieces and flags. We will explain our choices in section 5.2. For the proof that Stratego is EXPTIME-complete we will first prove a game, called $G_{2.5}$, to be EXPTIME-complete (see section 5.4). This will help us in our EXPTIME-hardness proof. Next we will give an global overview of our stratego reduction (see section 5.5). Then we will give our EXPTIME-hardness proof from our EXPTIME-complete game $G_{2.5}$ to Stratego (see section 5.6). In the final section we will discuss disastrous moves that a player can do that will give the opponent a massive advantage, in the sense that they will certainly not help the player and may even give the opponent a winning position (section 5.7).

Finally we will conclude or research (section 6) and give an extended future research section (section 7) where we explain what we believe is necessary to improve on our Stratego proof, but mainly explain our thoughts about how to improve the current checkers proof.

2 Background

2.1 algorithms

Definition **Zero-sum game**. A zero-sum game is a game whose payoff matrix is such that each participant's gain or loss of utility is exactly balanced by the losses or gains of the utility of the other participants. If the total gains of the participants are added up and the total losses are subtracted, they will sum to zero.

Definition **asymptotic upper bound (big-O).** Let f and g be functions $f,g : N \longrightarrow R^+$. Say that $f(n) = O(g(n))$ if positive integers c and n_0 exist such that for every integer $n \geq n_0$, $f(n) \leq c\, g(n)$. When $f(n) = O(g(n))$, we say that g(n) is an asymptotic upper bound (big-O) for f(n). [18]

A minimax algorithm, when used for two-player zero-sum turn-based boardgames with perfect information, can help in determining the optimal strategy for a player. To do this the algorithm will try to maximize the gains on the players turn and will try to minimize the gains on the opponents turn. For example, in Stratego there are three possible outcomes for the game: a player wins, a player loses or it will be a draw. To represent that value wise it would be +1 for a win, -1 for a loss and 0 for a draw. Note that the values need to equal 0 as it is a zero-sum game.

The algorithm works as follows: it will first determine the value of the terminal positions, these will become the leaves of the tree. Then it will, by backwards induction, determine the value of the nodes that reach them. The node will get the maximum(minimum) value of all the possible nodes/leaves that can be reached directly from it. We use this tree to determine an optimal strategy for a player, so a layer (or depth) in the tree where the player makes a move we will maximize the value of the nodes. On the other hand, a layer in the tree where the opponent makes a move we minimize the value of the nodes.

Once this has been done for all positions we have a game tree and can determine the optimal strategy. The time it takes is at worst $O(b^d)$, assuming connecting nodes takes $O(1)$, with d being the depth of the tree and b being the branching factor of the tree.

Most of the time constructing a game tree by going through all possible nodes is infeasible. Not only is it infeasible, but it is also suboptimal. A lot of the nodes do not influence the result of the branch. Again looking at the game of Stratego there are an exponential number of board states. Constructing the game tree would then take at least exponential time.

Here is where alpha-Beta pruning is useful. When the player has a move that gives them a score x, then they can ignore any branch with a score lower than x. This is because the player is trying to maximize their score. So x is a lower bound on the node(s) for the player. This also means that x is an upper bound on the node(s) of the opponent. After all, the opponent would never choose a move that is better than necessary for the player. Then the opponent can find a value y (that is lower than x) and ignores everything that is higher than y making y their upper bound and thus the lower bound for the player.

This is basically what alpha-Beta pruning does. It works like the minimax algorithm, but keeps track of lower bound α and upper bound β and prunes away any branch that has a value outside of those boundaries.

The time complexity of alpha-Beta pruning (in minimax) is in the worst case $O(b^d)$, the same as just the regular minimax algorithm. However with some smart design choices on how to construct the game, for example which branches it should check first, we can reduce the time complexity to $O(b^{d/2})$, though this is still exponential.

2.2 EXPTIME-completeness

2.2.1 Stockmeyer and Chandra

In this section we will explore previous proven EXPTIME-complete games.

The basis of most EXPTIME-complete proofs comes from the research of Stockmeyer and Chandra [1].

As there did not exist any game that was already proven to be EXPTIME-complete they decided to define their own games and prove that those are EXPTIME-complete.

They defined six games (G_k with $1 \leq k \leq 6$) based on propositional formulas. These formulas involve variables, binary connectives, unary connectives and parenthesis.

Below are two of the games, of Stockmeyer and Chandra, that are most often used for reduction proofs (G_2 and G_3) [3]. These games are set up pretty similarly, but their winning conditions are opposite of each other. In G_2 player I needs to fulfill the formula I-WIN to win, while in G_3 player I needs to prevent I-LOSE from being fulfilled because otherwise they lose. This is useful as some games are more dependent on a (wrong) move of the opponent while other are fully dependent on optimal play:

G_2: A position is a 4-tuple (τ, I-WIN (X, Y), II-WINX, Y), α) where $\tau \in \{1, 2\}$, I-WIN and II-WIN are formulas in 12DNF, and α is an $(X \cup Y)$-assignment. Player I (II) moves by changing the value assigned to at most one variable in X (Y). Player I (II) wins if the formula I-WIN (II-WIN) is true after some move of player I (II).

G_3: A position is a 4-tuple (τ, I-LOSE(X, Y), II-LOSE(X, Y), α) where $\tau \in \{1, 2\}$, I-LOSE and II-LOSE are formulas in 12DNF, and α is an $(X \cup Y)$-assignment. Player I (II) moves by changing the value assigned to exactly one variable in X (Y). Player I (II) loses if the formula I-LOSE (II-LOSE) is true after some move of player I (II).

Definition **Disjunctive Normal Form (DNF)**: A formula is said to be in disjunctive normal form, or DNF, if it is a disjunction of conjunctions of literals. A formula is in 12DNF if each conjunction has no more than twelve literals.

Definition **log-space computable**: A function f is log-space computable if there is a deterministic Turing machine which on input $a \in \sum^*$ outputs f(a) and uses at most $O(\log|a|)$ memory.

Definition **log-space reducible**: For problems $X, Y \in \sum^*$, X is log-space reducible to Y ($X \leq_{\log} Y$) iff there is a log-space computable function f such that for all $a \in \sum^*$, $a \in X$ iff $f(a) \in Y$.

Definition **log-complete**: A problem X is log-complete in class Z iff X is an element of Z and each problem Y from Z is log-space reducible to X ($X \in Z \land \forall_{Y \in Z}\, Y \leq_{\log} X$).

An even stronger notion of EXPTIME-completeness which is sometimes also possible is when a game is reducible in log-space. This is because given two languages A and B, if A $\leq_{log}$ B then A $\leq_p$ B. In words this means that if some game is log-space reducible to another game, then it is also polynomial time reducible. Stockmeyer and Chandra proved that G2 and G3 are EXPTIME-complete with the stronger stricter definition of EXPTIME-completeness. That is, the games are in EXPTIME and they are log-space reducible (and thus also polynomial time reducible) from a known EXPTIME-complete problem.

Some examples will make it clear how a reduction from one of these games (G_1 … G_6) would look. In the next section we will discuss the work of Fraenkel and Lichtenstein [2]. We will show their EXPTIME-hardness proof for chess. They reduce from the game of G_3. We will see how the gadgets in the chess paper exactly work and how a game would look if both players would make optimal moves. Later in this paper we will also look at the work of Robson [3]. We will look at his EXPTIME-hardness proof for American checkers and show that it not immediately works for other variants. We will assume that the reader both knows the rules of chess and checkers, but if the reader needs a recap we refer them to [17] and [16] for the rules of chess and checkers respectively.

2.2.2 Example chess

In this section we will give the EXPTIME-hardness reduction of chess from Fraenkel and Lichtenstein [2]. They do this by reducing the game G_3, from Stockmeyer and Chandra, to chess. This reduction works because, all pieces, except the ones mentioned in the reduction, are deadlocked. This means that it is not possible to do an legitimate chess move that is not part of the simulation of G_3. Thus if a player can win the game of chess below, they will also win the game G_3.

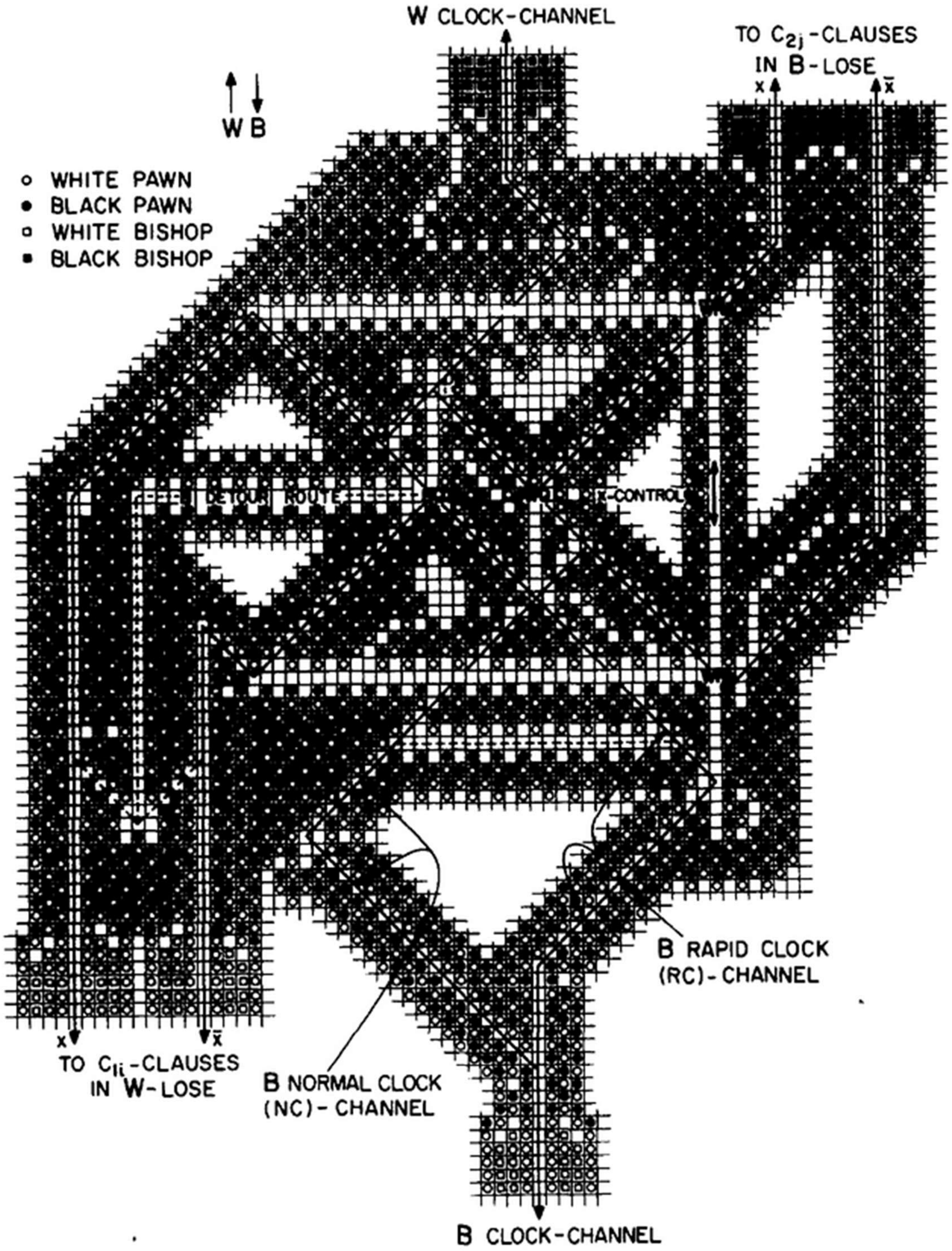


FIG. 1. White Boolean controller.

Figure 1 White Boolean Controller [2]

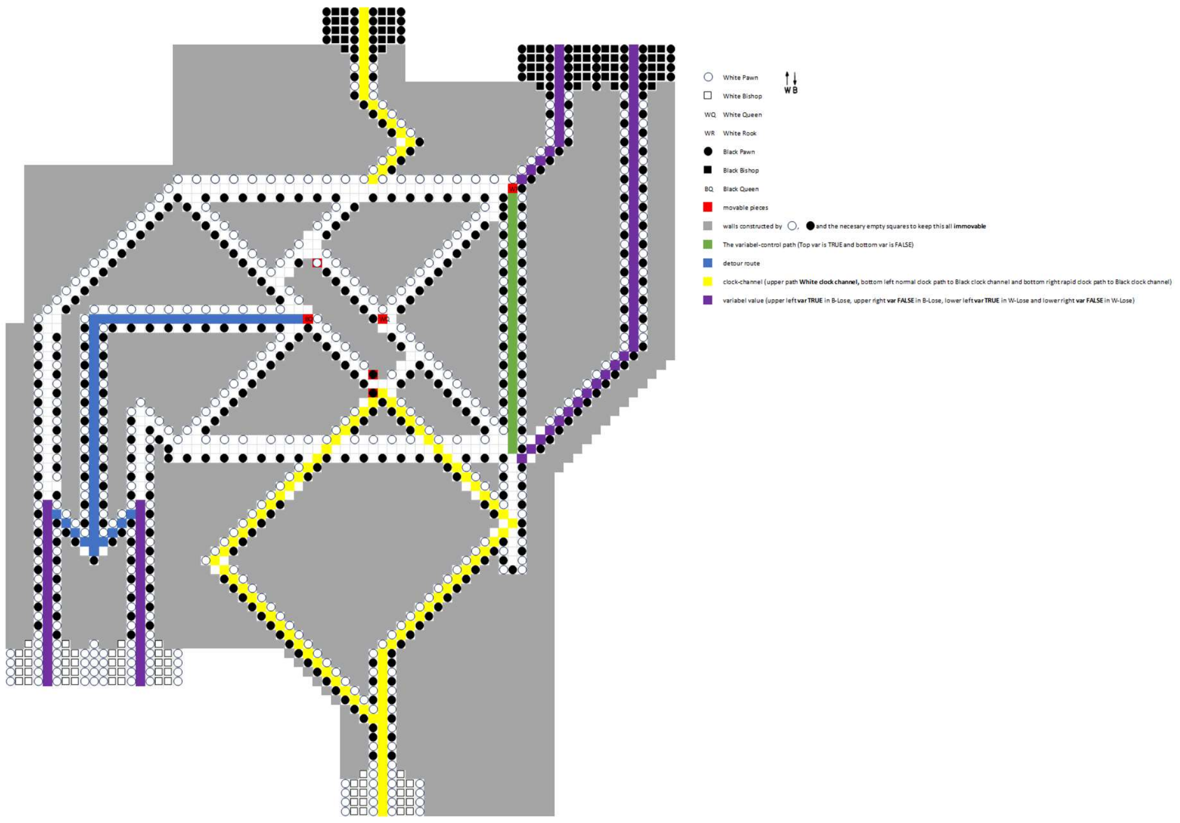

○ White Pawn
□ White Bishop
WQ White Queen
WR White Rook
● Black Pawn
■ Black Bishop
BQ Black Queen
W B
movable pieces
walls constructed by ○, ● and the necesary empty squares to keep this all immovable
The variabel-control path (Top var is TRUE and bottom var is FALSE)
detour route
clock-channel (upper path White clock channel, bottom left normal clock path to Black clock channel and bottom right rapid clock path to Black clock channel)
variabel value (upper left var TRUE in B-Lose, upper right var FALSE in B-Lose, lower left var TRUE in W-Lose and lower right var FALSE in W-Lose)

Figure 2 Legend White Boolean Controller

This is the main gadget. There are two versions of this gadget. The first one is as shown in the figures 1 and 2. They call this one the White Boolean controller (WBC). To get the other version, which is called the Black Boolean controller (BBC), when the gadget is turned 180 degrees and all colors are reversed. The gadgets are used by the players to take actions representing the changing of truth values of variables in G3. So there is one such gadget for each variable in the formula game. There are only six pieces that can move in this gadget. For the WBC that would be one black queen, one white queen, one white rook, two black pawns and one white pawn. Note that moving pawns is only useful in abnormal play. During regular play, moving them will only be disadvantages for the player making the move.

The gadget consists of walls (mostly) made by pawns placed in such a way that they cannot move. The path formed by these walls has three important parts.

First is the variable control column. When a player would change a variable from true to false (or vice versa) in G_3, they would move their rook in one of the Boolean controllers (the controller that corresponds to that variable) from the "true" position (top of the variable-control channel) to the "false" position (bottom of the variable-control channel) or vice versa.

Second the exits corresponding to specific truth-values (with possible detour). If a queen can exit through such a path, it means that that specific variable value, changed by moving the rook, is applicable, in the sense that the actual assignment of the variable corresponds to that pathway.

Finally are the normal/rapid clock paths. These paths are reachable by a player B (W) with a queen when the opposing players queen has left via a regular truth-values exit. These paths will guarantee a path towards the opposing king and are exactly 1 turn slower than reaching it with a regular winning strategy. In other words, if a queen leaves a Boolean controller while the player has not won the simulated G_3 game yet, the opponent will win via this path.

The regular winning strategy is as follows. When the game G_3 has been decided, and all rooks are in places that correspond to that truth-values, the winning player W(B) will start moving his/her queens through the corresponding variable channels that make the formula of B(W)-LOSE true. Note that each queen makes two moves in the Boolean controller plus one more to reach the next gadget (for a total of three moves). As soon as the winning player has moved his/her queen the opponent can start moving their queen to their clock channel. The purpose of the clock channel within the gadget is give the opponent a way to win the game within a certain number of moves, forcing the first player to find a win before that.

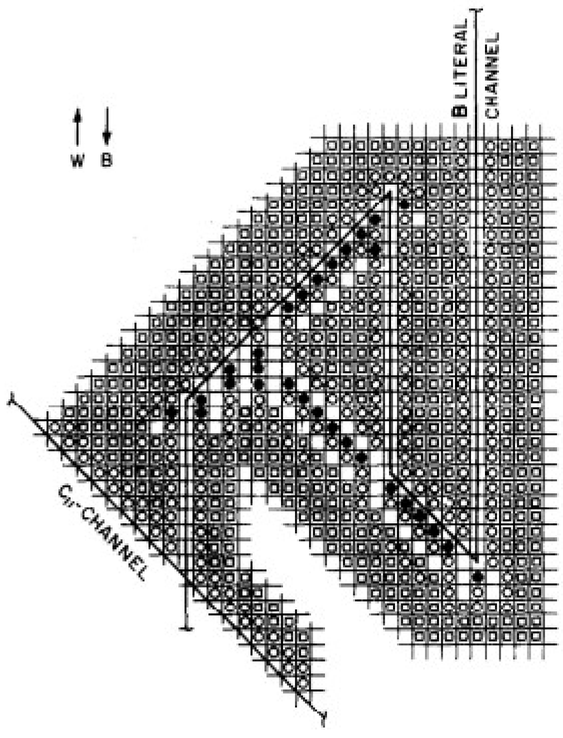


Figure 3 White one-way switch [2]

The next gadget is called a one-way switch. This gadget is reached when a queen leaves a Boolean controller through a variable-values exit (purple path). Again the walls are created by pawns that cannot move which force a specific path. The walls are hollow and filled with bishops of the same color. The amount of bishops, and thus the thickness of the wall, depends on the number of variables in the B(W)-LOSE of G_3. The wall needs to be sufficiently thick to discourage the player from not following the path with enough bishops to stop any queen after the first from passing through.

The moment a black (white) queen enters (from the Boolean controller) it reaches the bottom right corner (straight line from the top). Any other place along the way will get it captured by a pawn. In three moves the queen captures an opposing pawn in the middle of northeast-southwest path (see Figure 3). The queen will continue that direction and then exit on the bottom.

First note that the total number of turns the queen takes (without counting entering the gadget) is four turns within the gadget plus one turn exiting/entering the clause channel for a total of five turns. If combine it with the previous gadget we know that a queen, after G_3 has been decided, reaches a clause channel in eight turns. The second thing to note is that, just as the name suggests, it is impossible for the queen to go back up into this gadget. This is because the single pawn that gets removed when the queen passes through, makes it that the pawn beneath it can move one square up releasing enough bishops to fill the northwest-southeast path in the gadget. This would result in an immediate capture of the queen.

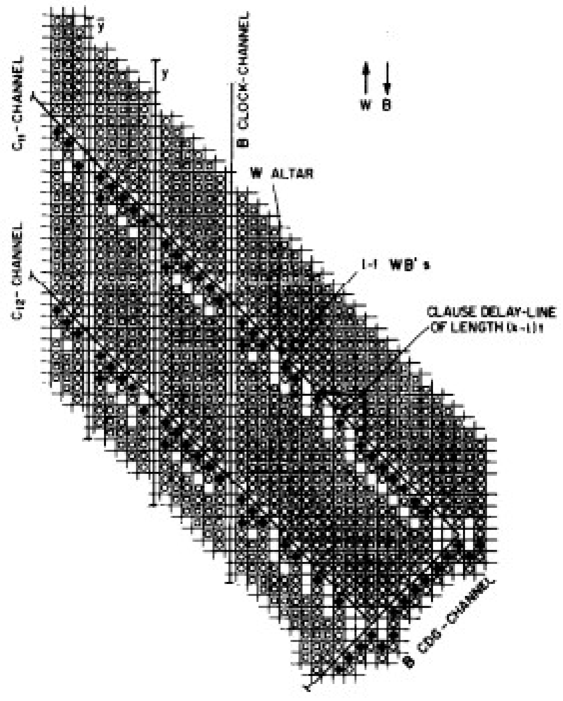


Figure 4 White Channel crossings, altars and delay lines [2]

The third gadget is a combination of multiple components and represents a clause in the formula. The components are channel crossings (a way to pass through walls), a number of altars (some structure that captures a queen) and a delay-line with a length dependent on the number of variables. The walls for each of these components are the same as in the previous gadget. This gadget connects to all switches (and thus indirectly the Boolean controllers) for all the possible truth-values for a specific player. The gadget has a number of clause-channels up to the number of clauses the formula in G_3 has.

In the regular winning strategy, once the winning player's queen enters through the variable values paths, it can only stop on intersections with clauses that contain that variable value. In figure 4, if the queen would enter the gadget via the $\bar{y}$-variable path, the queen can only stop on the intersection with the C_{11}-channel. On the intersection with the C_{12}-channel it would get captured by an opposing pawn.

When an queen enters the gadget via the clock-channel, it cannot stop anywhere in the gadget. It can, however, immediately move through the gadget in a straight line. This allows the player to skip the entire clause gadget. Remember that this path only used by the losing player during regular play, or used to win when a player has moved their queen from a Boolean controller before G_3 has been decided.

Note that for the winning strategy the player will first move all of the queens to the clause-channel that is the one that made W-LOSE (B-LOSE) true. Otherwise, if the player first moves one queen, the bishops that were stopped at the altar can start moving in the clause and disrupt the winning play.

Once in the clause-channel the queens can move through the channel till they reach the altar structure and capture a pawn. Here all but one of the queens will be captured by bishops in the wall.

Let us call the number literals in a clause l, then there are l – 1 bishops waiting there. This is why we need at least l queens to pass the clause. The final queen will continue to move through the gadget until it reaches the delay-line. The delay consists of a pawn placed in such a way that the queen must capture it to progress (and thus waste a turn for each pawn it needs to capture). This delay equalizes all the clauses so that there is no difference after passing it. The length of the delay line is (k – l)t with k being the size of the largest possible clause in W-LOSE (B-LOSE), l being the actual size of the clause (same as above) and t being the number of turns it took to reach the clause. Note that for regular play that number t would be 8. These are the 8 turns from the previous gadgets till this point.

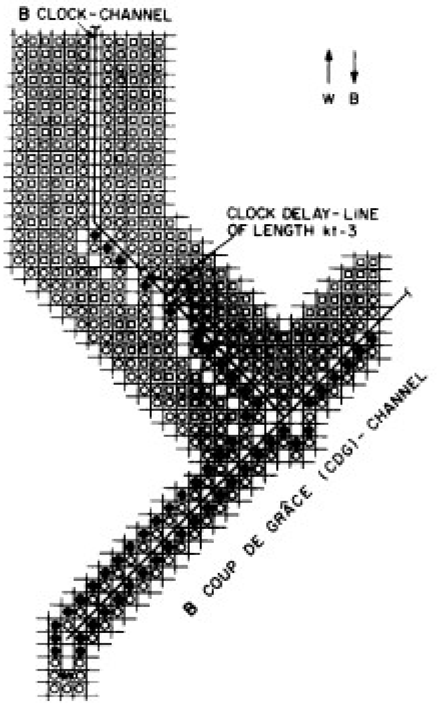

B CLOCK-CHANNEL
W
B
CLOCK DELAY-LINE OF LENGTH kt-3
B COUP DE GRÂCE (CDG)-CHANNEL

Figure 5 White Coup de grâce channel [2]

The final gadget is the Coup de grâce. Here is where it will end as it is the gadget where we can checkmate the king.

Once the winning player's queen reaches this gadget in regular play it will stop at the bottom of the gadget and checkmate the king. Then, in the final turn, it will move the last three places and capture the king.

When the queen reaches it via the clock-channel it will reach a delay-line (which is separate from the previous one). Recall that the clock channel pathway skips the entire previous gadget. To make sure that the winning player can actually win, this pathway will also need some delay. The way it works is identical to the previous one, but the length of the line differs. Here there are kt – 3 pawns that will delay the queen (with k still being the largest possible clause and t still the number of turns to reach the clause). The queen then reaches the Coup de grâce-channel and needs two more moves to checkmate the king.

Time complexity

For chess to be EXPTIME-complete it needs to fulfill two criteria. First the language (i.e. does the White(Black) player have a winning strategy) needs to be in EXPTIME. This means that it should be solvable in exponential time (i.e. $O(2^{p(n)})$). Second the reduction above needs to be computable in polynomial time. We will discuss the first condition in a later section. For the second condition let us assume the number of disjuncts in W-LOSE and B-LOSE of G_3 be of order n. Then computing the reduction from G_3 as shown will certainly take no more than O(p(n)). Thus, as it is computable in polynomial time, chess is EXPTIME-hard.

2.2.3 Example checkers

In this section we will show the EXPTIME-hardness part of Robson's proof. Initially Robson intended to use G_3 for his reduction, the same as is used in the chess paper. The only difference being that they renamed II-LOSE to WWIN and I-LOSE to BWIN.

However there were some problems with using regular G_3 for checkers. Recall that G_3 does not allow any passing. In the game of checkers there are no stationary pieces (like the bombs in Stratego) or ways to deadlock pieces (like the wall of pawns in the chess reduction). So there are always pieces that can make a (legal checkers) move that is not part of the simulation.

To solve this problem he uses a modification of a sub problem of G_3. Note that when we reduce problem A to problem B, the proof actually shows that an arbitrary instance of A can be reduced to a specific kind of instance of B. The range of the reduction function may be a proper subset of the whole set B. So the reduction proof shows that problem A is reducible to this subset itself. The sub problem of G_3 used in Robson's proof is the range of the reduction in the EXPTIME-hardness proof to G_3 from Stockmeyer and Chandra [1].

Robson's reduction will reduce checkers from this sub problem of G_3, which is similar to G_3 but with the winning conditions being WWIN V WGONE and BWIN V BGONE for White and Black respectively, where WGONE and BGONE are defined as below. As mentioned above, WWIN (BWIN) corresponds to II-LOSE (I-LOSE) in the original game of G_3. The formula's for WGONE and BGONE are as follows:

$$\text{WGONE} = \bigvee_{i=1}^{2m+2} (a'_{i+1} \wedge b'_i), \qquad \text{BGONE} = \bigvee_{i=1}^{2m+2} (a'_i \wedge b'_i)$$

Figure 6 Formula's WGONE and BGONE [2]

Here $a_{i'}$ and $b_{i'}$ are sub expressions over X_1 to X_m and Y_1 to Y_m respectively. Any move that White makes (except one that result in an immediate loss) will set exactly one a'_I true and all other a' to false. Then the next turn, when Black makes a move (except one that result in an immediate loss), they will make b'_I true (same i) and make all other b' false. Note that at this point there is a clause of BGONE true. Next whit is White's turn again to make a move, changing a'_{i+1} to true and making all other a' false. Note that now there is a clause in WGONE true and all clauses in BGONE are false again. This will repeat cyclically so that a players winning condition is true after they make a move,

We now have a way to deal with moves outside of the simulation of G_3. Even if White makes a move outside of the simulation (except one that result in an immediate loss) it would still fulfill WGONE. If Black makes a move that does result in an immediate loss (i.e. next turn White wins) in checkers, they do not falsify WGONE and thus make it so White wins G_3 next turn.

So Robson reduces G_3 with WWIN V WGONE and BWIN V BGONE to checkers, which will have the same results as regular G_3. However this variant does have an answer for (potential) moves outside of the simulation.

reduction

In figure 6 in the left image we see an overview of the checker reduction. The reduction consists of two main components, namely a spiral border and the part that simulates the known EXPTIME-complete game, which is in this proof a special version of G_3. The part that is used to simulate G_3 is surrounded by the spiral border. The distance of the spiral border is far enough from the rest of the reduction that no player will consider reaching it from the simulation of G_3.

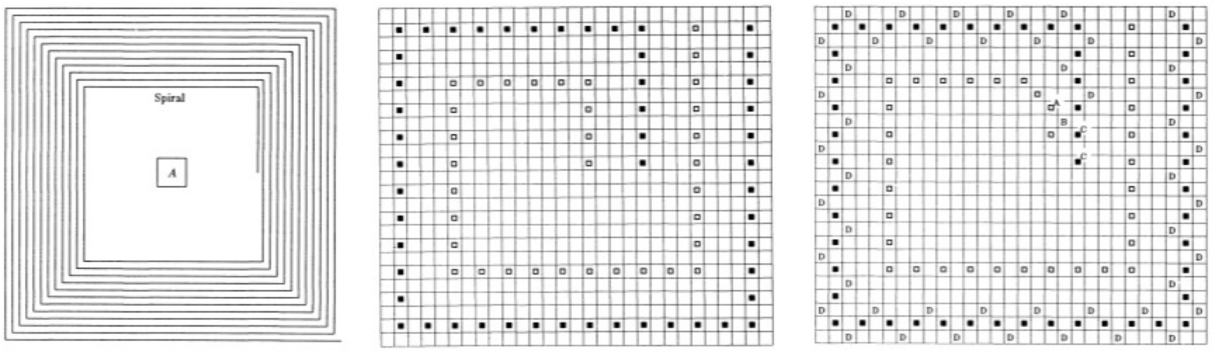


Figure 7 Spiral border: overview (left), close up (middle) and attacking the spiral (right) [3]

The boundary is made up of two paths of kings (both colors) and is in the shape of a spiral, hence the name spiral border. These paths are placed with exactly three squares between them and placed in such a way that player I (White) can capture all opposing kings of player II (Black) in the spiral in a limited number of moves, regardless of the length of the path (see Figure 7 middle image). This is because the opponent has a limited number of free moves where they can move a piece away from the spiral resulting in the fact that the player may need some more moves to capture the entire spiral.

Once G3 has been decided White (Black) can capture all the opposing kings as explained above. This is because the simulation of G_3 in the center ends with a (huge) number of forced captures by Black. In the turns it takes to resolve those forced captures White can launch an attack on spiral border like in figure 6 right image. White moves the piece marked with an A into position B. Regardless if Black captures that piece, with the one marked C, the next turn White can capture most, if not all, of the opposing kings in the spiral border.

With this White has such a massive material advantage that we say that they forced a win. They can just shrink their border till they can surround all the pieces leftover in the center.

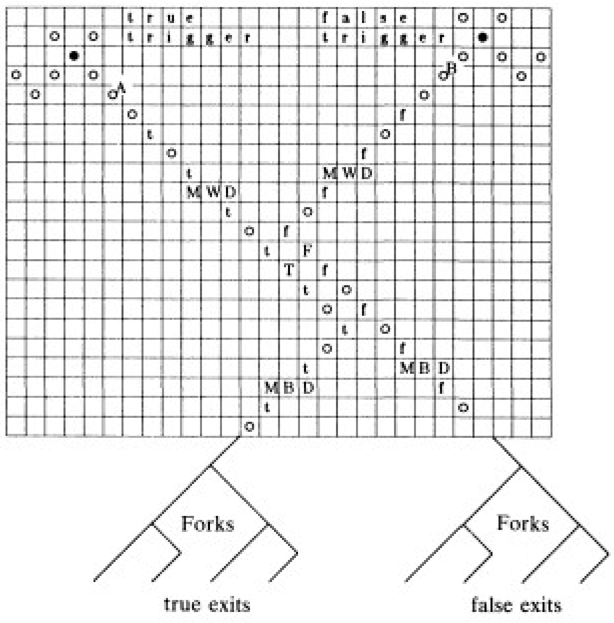


Figure 8 White Boolean Controller [3]

Just as with the chess paper the simulation of G_3 starts with a Boolean controller. Also the same is the fact that there are two versions namely the White and Black Boolean controllers, which differ by turning the Boolean controller 180 degrees and swapping the colors. The only difference, with the chess paper, is that it ends in multiple fork (sub-)gadgets that let the player whose controller it is (i.e. Black or White) choose the direction in which the capturing continues.

When White would change a variable in G_3 from True to False, he can simulate that in their respective Boolean controller (a White Boolean controller for White and a Black Boolean controller for Black) by moving a king between the T and F squares on the board (see Figure 7 above). Both players will keep doing this, on different Boolean controller, as long as the game G_3 is undecided.

As soon as all variable have an value assigned such that WWIN (BWIN) is fulfilled White (Black) can start the winning strategy for regular play. White can now move their piece marked with A (B) to force Black to start a forced capture sequence along the path marked with t (f) if a variable is true (false) in the fulfilled clause (see Figure 7). While continuing this path it passes multiple white delays (MWD) and multiple black delays (MBD), each of which contains a piece that the other player must spend a move to capture. The path eventually reaches a fork.

Note that the MWD and the MBD cancel each other out under normal play. However, if a player deviates from the intended strategy, only the deviating player has a massive delay that is enough to lose them the game. Also note that if this forced capture sequence is opposite of the variable value (e.g. variable is true and the opponent captures along the f path), then White will only encounter a MWD which delays enough to let White lose the game because Black can then attack the spiral border.

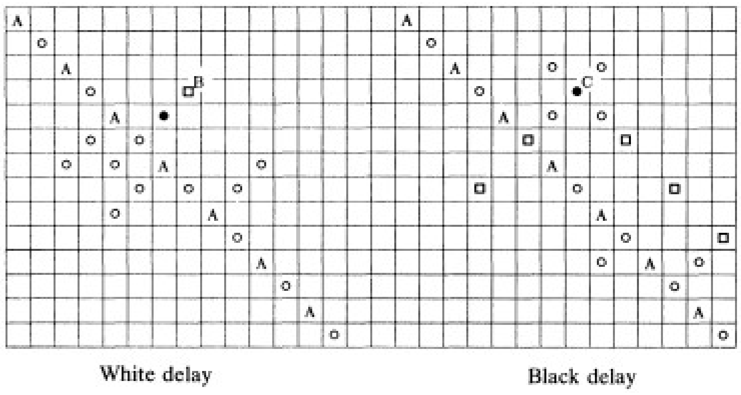


Figure 9 Delays [3]

Here the capturing comes temporarily to an end as both players need to resolve their respective multiple delays as there are multiple forced captures in each delay (see Figure 8).

The Boolean controller is connected to at least three gadgets called forced fork gadgets and some number of gadgets we call regular forks. A forced fork is used to direct a capturing sequence while a regular fork leaves the choice to the opponent. These gadgets are shown in figure 10 and connect further to paths shown in figure 12. The reason for the forced forks is to either to reach a defense path, attack paths marked with 1, attack path marked with 2 or attack path marked with 3. Otherwise we use regular forks for multiple attack paths marked 1 if the number of total variables is greater than 3. The reason for this will be discussed later in more detail.

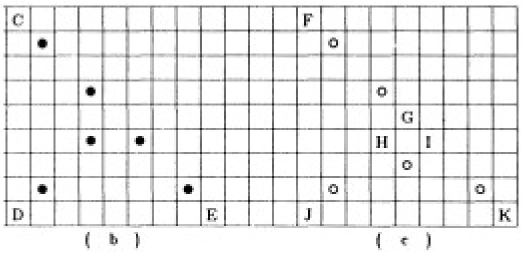


Figure 10 Regular fork (left) and forced fork (right) [3]

After all the delays are resolved the White player needs to make a move in the fork(s) to continue the capturing process. White can decide to not move a piece and stop the capture sequence here. However, if White moves his/her piece to H then the capturing continues to left and if the piece is moved to I the capturing continues to right (see Figure 10 image c). As mentioned above the choices for the capture direction are made based on the fact that that the capturing can either be intended as an attack/validate a winning clause (e.g. from White Boolean controller to an attack zone representing a clause that let White win) or to defend/invalidate a wrong clause (e.g. from White Boolean controller to an attack zone representing a clause that let Black win).

As we discuss regular play here or the winning strategy we assume that White is trying to show that WWIN is fulfilled. So the capturing sequence is intended as an attack. We temporarily disregard what number this attack is, but say that White chose each fork in such a way that the capturing sequence is going in the right direction.

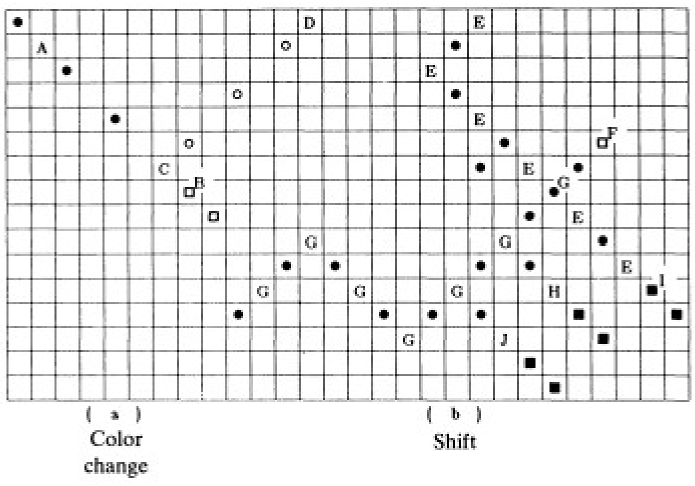


Figure 11 Color change gadget and shift gadget [3]

After leaving the fork(s) Black will continue their capture towards an attack zone.

If a capturing sequence is intended as an attack, the path it takes consists of regular captures with, where necessary, a shift gadget and a single color change gadget. All the attacks in the attack zone gadget (see below) need to be of the same color as the clause it represents (so a WWIN clause needs to be reached by an capturing sequence of the White player).

Remember that we currently are forcing Black to continuously capture, so somewhere along the way the capture sequence changes from Black to White. This happens in the color change gadget (see Figure 10a). The forced capture sequence of Black enters from position D and continues capturing till position C. Then White can start a capture sequence from position B and exit this gadget in the direction of A. Note that if it was a capture sequence to defend we do not need to change the color, as defending in a Black attack zone requires capturing from Black.

The shift gadget is only used when there is a problem with connecting the path to the attack zone. White captures along the path marked with E. This forces a single capture by the black king marked with I. Finally the white king marked with F can now capture along the path marked with G and exit shift. This displaces the path, from the line of the initial entrance, by exactly one square diagonally. Note that with 2 of these gadgets any desired direction can be achieved to continue the path.

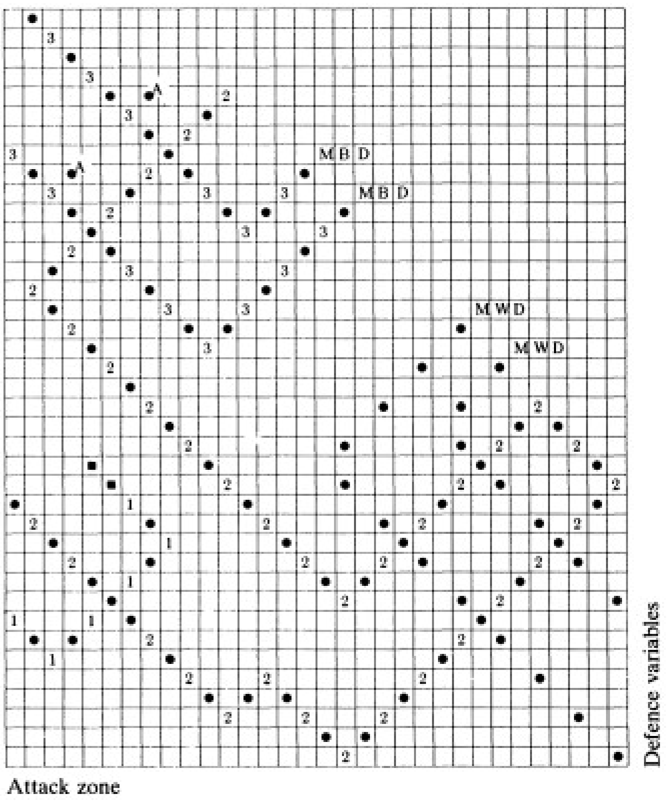


Figure 12 White Attack zone [3]

When the player has reached the attack zone it can either be as attacker or defender. An attack zone can be seen as a representation of a clause. A White (Black) attack zone represents a clause in WWIN (BWIN).

In a White attack zone there is a defense path for each Black controlled variable in the WWIN clause it represents. In figure 11 we see two paths, so we know the clause has two Black controlled variables. The number of attack paths in a White attack zone is equal to the number of White controlled variables in the clause plus one. That extra one is a backup in case of abnormal moves of the opposing player (for more information we refer to Robson [3, 7. Abnormal play]). Again, in figure 11 we see three attack paths, so we know that there are two White controlled variables in the clause.

Once the first n – 2 attacks (with n being a natural number between 2 and 12) reach the attack zone they will start capturing the attack paths marked with 1's. These result in a forced capture for the opponent at the end. Because of this, the next player with a free move is again the attacking player and they can continue their attack. Note that if there are only two player controlled variables there will be no paths marked with 1's. Also note that if we did not direct the forks the correct way we would get stuck (path marked with 2) or worse get captured (path marked with 3).

After the n – 2 attacks the next attack, so the (n – 1)th attack, will follow along the path marked with 2's. This is now possible as the capturing sequences in the previous attacks removed the blocked parts in the path marked with 2. First note that this capturing sequence does not end with an opposing capture, thus the opponent is free in their move the next turn.

Next notice that this attack has opened up the defense paths of the opponent. If the opponent (Black) has a defense variable lying in wait, they can now start the capturing sequence, activating MWD which creates enough delay that they can attack the spiral border, which results in the White players loss. This would happen if the WWIN clause was not fulfilled, so in regular play this would not happen.

Finally, assuming that Black did not have a defense variable, the White player can launch its nth attack. First note that there are two paths marked with 3's. This is because it is possible for the opponent to interrupt one of them during abnormal play. Second, this attack will reach an MBD which creates enough delay that they can attack the spiral border, which results in the Black player's loss. This means that the clause in WWIN was indeed fulfilled and White won via regular play. Thus White has also won the game G_3.

Time complexity

For checkers to be EXPTIME-complete it needs to fulfill again the previously mentioned conditions. The language (i.e. does a player have a winning strategy) needs to be solvable in EXPTIME. Second the reduction above needs to be polynomial time computable. We will explain the first condition together with the one for chess in the next section. For the second condition, it being polynomial time computable, it is pretty clear. The parts simulating G_3 can certainly be built in O(p(n)). The addition of the border makes it at most O(p(n)) * O(n) which is still O(p(n)).

3 Games in EXPTIME

In this section we will explain why chess and checkers are in EXPTIME. Both in the papers of A. S. Fraenkel and D. Lichtenstein [2] and J. M. Robson [3] the authors mention that, because there are an exponential number of unique board states, it is simple to prove that respectively chess and checkers are in EXPTIME and just omitted the proof. We believe that it is not that simple for everyone to see that it is in EXPTIME and thus decide to explain it below. We will explain it for checkers, but a similar argument holds for chess.

If we do not consider the number of pieces each player has access to we can determine that an N x N board in checkers can be filled $5^{(N^2)/2}$ ways. The explanation for that is as follows: each square can be in five different states: empty, white man, white king, black man or black king. There are an N^2 number of such squares, but we can only use half of them (we can only place/move pieces on the dark squares).

If we want to distinguish between whose turn it is to play we get $2 * 5^{(N^2)/2}$ different board states. Although Robson in his proof does not consider any drawing rules in official checker tournaments there is a rule that declares a draw when a board state is repeated for the third time. So let us also differentiate board states based on the how often it has been repeated. As this is at most three times we get $3 * 2 * 5^{(N^2)/2}$.

We can, theoretically speaking (because in reality it is way too complex to do so for a computer in reasonable amount of time), create a game tree by taking the game states as nodes and draw a vertex between each node if a legal move in the first game state can lead to the second game state.

Let us define n = N x N.

The depth of each branch is then no more than $6 * 5^{n/2}$. This is because if the branch is any longer than that there would certainly be a repeating board state. We do not have to check any further as whatever the result is in another branch we can reach it earlier. Now If we use the minimax algorithm with alpha beta pruning we can construct the game tree and obtain a winning strategy in

$O(b^{d/2})$ time. However if we keep track of all the board states that complexity can be lowered. Once we know for a board state what the maximum (minimum) score is that player I (II) can receive in a branch we just store that and do not have to calculate it again in another branch. This will lower the complexity from $O(b^{d/2})$ to $O(6 * 5^{n/2})$ which is exponential. However we still need to consider that checking for a repetition takes exponential time and looking up if a board state has already been calculated takes also polynomial time. In total this would take $O(6 * 5^{n/2})$ times $O(2^{p(n)})$ times $O(1)$ which exponential time. So checkers (and also chess) is in EXPTIME.

For Stratego the arguments are really similar to what we discussed above (see section 5.3).

4 Checkers variants

4.1 Shortcomings

While reading Robson's proof we have noticed some differences between the rules for what we know as checkers and the rules that are used in the proof. This is because there are a lot of different rulesets for checkers. According to Wikipedia there are at least 25 different variants [19]. We have verified that these are actual playable variants based on the checker rules. These variants can be broadly divided into 3 classes based on what the pieces are allowed to do.

The first class of checker variants all have the following rules:

- The kings can only move and capture a single square in any of the diagonal directions.
- Men cannot move or capture backwards.

The game that Robson describes is from this class. He never mentions it, but it is highly likely that the game he based his proof on is American checkers. Aside from the rules mentioned above, there is one other rule that mentions that any capturing sequence can be chosen but that all possible captures must be made in that sequence. This rule is crucial for the workings of the proof as the gadgets force the game from one sequence to the next sequence. If the player could stop halfway through or choose not to capture then the proof would not work.

The second category are variants that include the so called "flying kings". The base rules are as follows:

- The kings can move any amount of squares in any of the diagonal directions.
- Men cannot capture backwards.

This is pretty similar to the rules in Robson's proof, but the rule that the kings can move any number of squares has some major consequences. First of all because of the unlimited distance the connection between gadgets becomes a bit tricky. It may be possible that the kings can go to another gadget if we connect them the wrong way. This could also possibly force a capture without doing any moves in the reduction. Because of this we can also no longer guarantee that the damage (e.g. unwanted captures) is only localized (which is a claim made in Robson's proof). Even without the difficulty in connecting the gadgets we can easily see the proof go wrong in at least two places. First there is a possible (forced) capture in all color change gadgets that can start the captures towards the attack zone. This basically means the first person that plays wins the game because they can immediately "fulfill" a clause. A second place where the proof goes wrong is in the spiral border. This one may be even worse as the player can capture the opponents entire spiral in one turn and then, in the same turn, continue capturing in other gadgets. Depending on how the gadgets connect this may possibly finish the game in that single turn. At the very least it will give the player an major material advantage.

The final category contains variance that also have the flying kings but additionally the men can capture backwards:

- The kings can move any amount of squares in any of the diagonal directions.
- Men can also capture backwards.

Of course, these variants have the same problems as the first category because they also have flying kings. The additional rule however brings with it its own problems. Whenever a single men is behind another piece of the opposite color an unintended forced capture can happen. A good example of this is in the Boolean controller. The piece that is surrounded at the top of the true/false trigger has, even before any movements have been made, a backwards capture possibility. The moment the opponent does that (either forced or by choice) the entire Boolean controller no longer works. When it is no longer possible to force the opponent in a specific direction it also becomes impossible to reach the attack/defend zone as intended.

We want to mention that our findings are in no way intended to disregard or devaluate the work of Robson for American checkers. It helped greatly in our understanding of EXPTIME-completeness proofs and the reader is actually encouraged to read the original paper. We do have, however, an issue with the statement that checkers is EXPTIME-complete. Since the proof seems to be only about American checkers, the statement can be problematic for international readers. For the majority of people checkers is still not proven EXPTIME-complete as they play an entire different version of checkers (including the author of this thesis). In the future research section we will discuss possible adjustments and/or additions to improve upon this proof.

4.2 International checkers

For our research we attempted to adjust Robson's proof for American checkers to also include International checkers. International checkers is included in the final category of variants we discussed above, kings are flying kings and men can capture backwards.

Below we will explain the two changes to the gadgets of Robson's proof that, when fully implemented (e.g. for all pieces in all gadgets), are enough to let it also prove EXPTIME-completeness of international checkers and, with some additional rules, can even be used for a generalized checker variants proof.

1. Replace all kings with regular men.

   As the main reason for using kings was for their ability to capture backwards we can change them to men. With the international rules men have the same capturing ability as the kings in the previous ruleset. If we left them kings however we would have a problem as kings in international checkers have way more mobility which can lead to kings in one gadget reaching another gadget without passing through the intended path.

2. Fortify (e.g. make sure that capture or movement is not possible) places where we don't want a capture to take place.

   If we change the rules to the ones from international checkers men can also capture backwards. This can lead to some unintended captures which in turn lead to the entire proof structure collapsing. So there need to be some changes in some of the gadgets to prevent this.

When applying these two rules we succeeded in most of the gadgets, but in one specific gadget we did not find a way to successfully do so, namely the Boolean controller.

In figures 13 and 14 we see some working adaptations of Robson's gadgets for international checkers. Like we mentioned, we swapped out all the kings and replaced them with men, as capturing direction are no longer an issue in international checkers. Next, in the place where opposing colored pieces were directly next to each other we fortified, or even created, some walls. This was done by putting two pieces of the player's color in each of the 4 orthogonal direction surrounding a lone opposing piece, basically isolating it (see for example Figure 14).

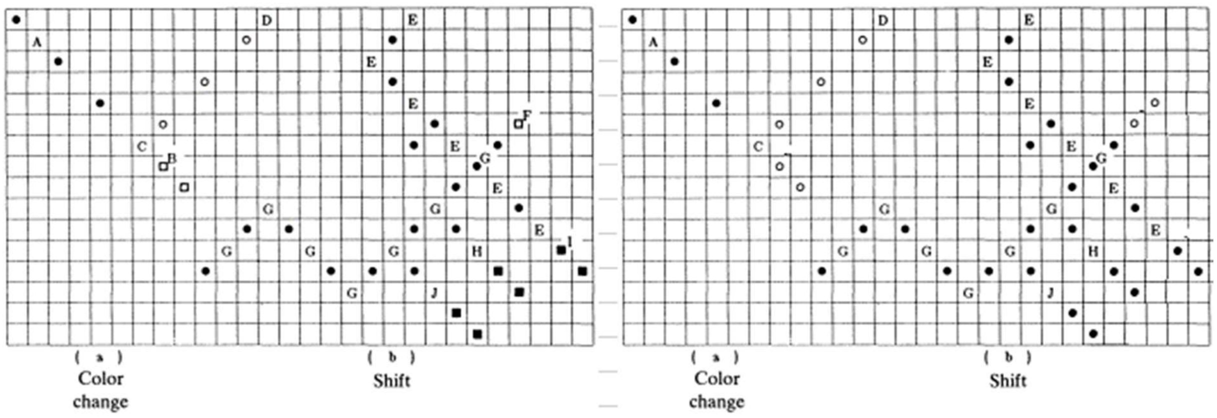


Figure 13 Color change and Shift gadget for American checkers (left) and International checkers (right) [3]

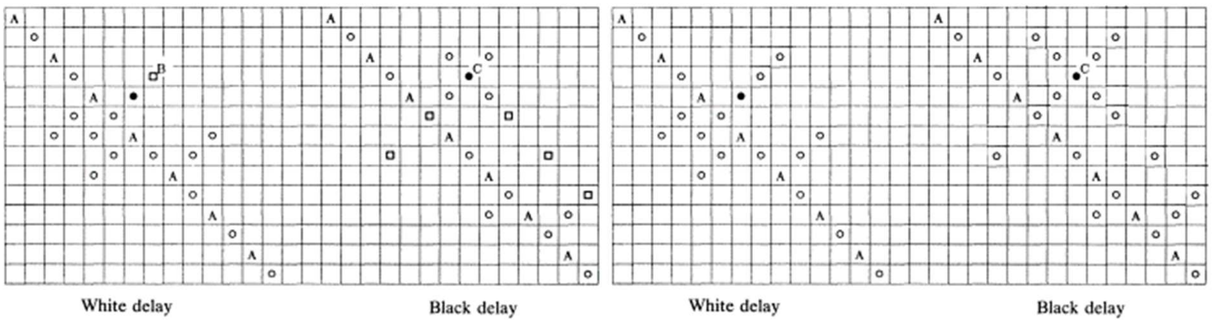


Figure 14 White/Black delay for American checkers (left) and International checkers (right) [3]

When we tried to apply these rules for the Boolean controller there were some complications. Applying the first does work. We can make sure that the opponent does not capture backwards. However, when trying to replace the king in the Boolean controller, we realized that doing that would is not possible. Recall the king in this gadget is used to move between the T and F marked on the board. We cannot simply replace this with a man, because the man can capture but not move backwards like an American king. So our second attempt was to create walls (e.g. two pieces of the same color) in such a way that the king could only move between the squares marked F and T and no further. However this did not work as, regardless of the color of the wall, the White player himself can break down the wall to free the king and create a massive advantage. This happens either because White can just move part of the or move another piece close to the wall, so that the opponent is forced to capture, resulting in the wall breaking down.

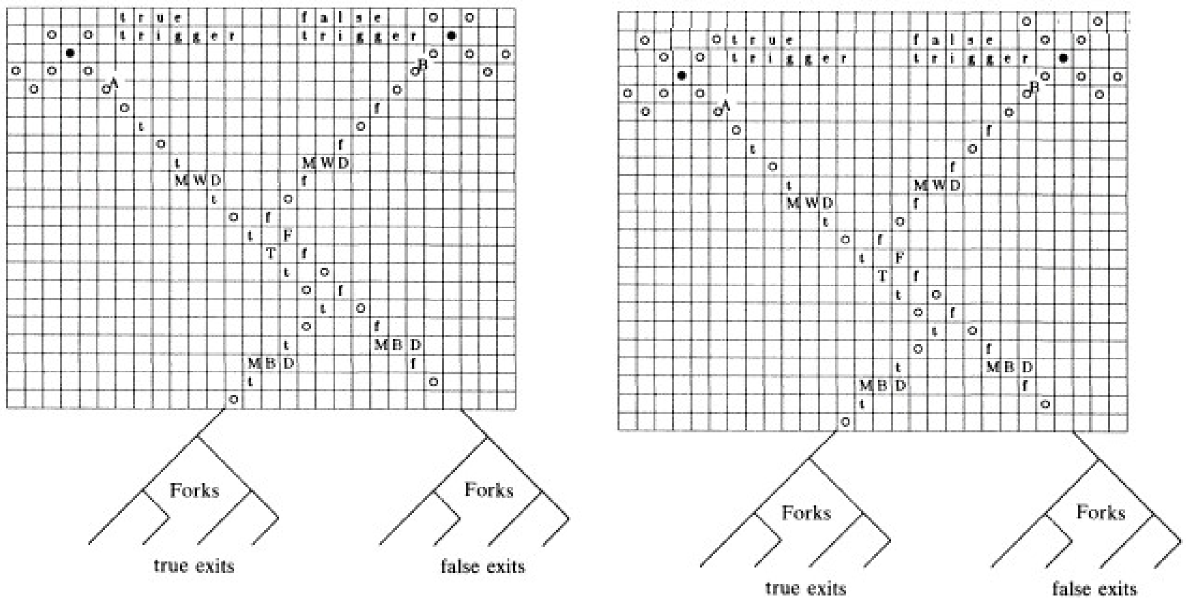


Figure 15 An unsuccesfull translation of the boolean controller [3]

We could not come with an answer to this problem and leave this for future research together with the question if it is even possible for a generalized proof for all checker variants.

5 Stratego

In this section we will look at the game of Stratego. We will first take a look at the rules for regular Stratego (section 5.1) and explain our considerations for generalizing the game to a N x N board (section 5.2). We give a proof for Stratego perfect information being in EXPTIME (section 5.3). After that we will give our proof for a new EXPTIME-complete game that we call $G_{2.5}$ (section 5.4). We give a global overview of the gadgets we use in the Stratego EXPTIME-hardness proof (section 5.5) and walk through our proof while giving the winning strategy (section 5.6). We end this part by explaining what moves the moves a player should never consider as those moves will guarantee that they will eventually lose (section 5.7). Unlike the result for the upper bound, Stratego perfect information being in EXPTIME, this result also holds for Stratego without perfect information. This because this is a superset of perfect information Stratego

5.1. Stratego game rules

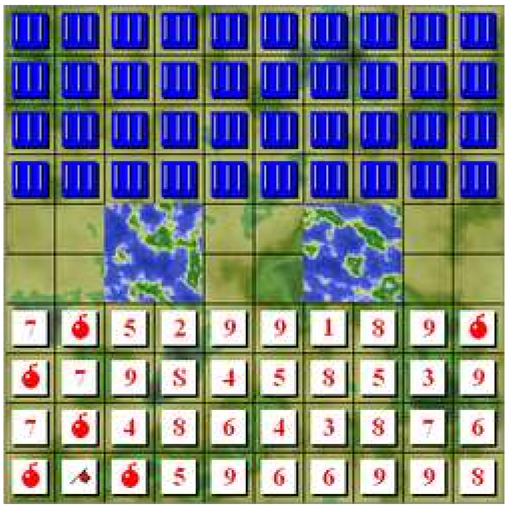


Figure 16 (American version) Stratego board

Stratego is played on a 10 x 10 board with in the center rows two puddles of water of size 2 x 2 (see Figure 16). These two zones are impassable (i.e. pieces cannot move on or over them). At the start of the game both players will place their 40 pieces in the first 4 rows with the image towards themselves. The pieces are:

- A flag
    - Cannot move. If this is attacked, the player loses.
- A marshal rank 1 (10 in European version)
    - This is the highest ranking piece.
- A general rank 2 (9 in European version)
- 2 colonels rank 3 (8 in European version)
- 3 majors rank 4 (7 in European version)
- 4 captains rank 5 (6 in European version)
- 4 lieutenants rank 6 (5 in European version)
- 4 sergeants rank 7 (4 in European version)
- 5 miners rank 8 (3 in European version)
    - Miners diffuse bombs.
- 8 scouts rank 9 (2 in European version)
    - Scouts can move as far as they want in a straight line.
- 1 spy rank S (1 in European version)
    - This is piece weaker than all other pieces, but can defeat a Marshal.
- 6 bombs
    - Cannot move. Defeat everything that attacks it except the miner.

The goal of the game is to capture the opponents flag or all their movable pieces. After both players have placed their pieces the first player can start his/her turn.

Rules for moving

- Each turn a player must move one of their pieces one square either horizontally or vertically (unless stated otherwise above).
    - A piece cannot move diagonally
- A piece cannot jump over other pieces or share a square with another piece.
    - If it tries to do so it is seen as an attempt to attack (see rules for attacking)

- A piece cannot move onto a lake square.
- A piece cannot move between the same two squares three consecutive turns (Two-Squares rule).
    - For a scout this would also include all the squares he steps over.
- A piece is not allowed to endlessly chase an opponent's piece (More-Squares rule).
    - According to [15] this means: If a chasing move (e.g. a move that threatens to capture an opponent's piece) would lead to a position on the board which has already taken place ("board repetition") this potential move is considered illegal and the player should perform another move.

When a piece moves towards a square that contains an opposing piece they will battle. Depending on the opponents piece the results differ.

Rules for attacking:

- If a piece is higher in rank than the opposing piece, the opposing piece is removed and the piece takes its place.
- If it was the same as the opponents, both pieces are removed.
- If it was lower, the piece is removed.
- If a spy attacks a marshal, the marshal is removed.
- If anything except a miner attacks a bomb, the attacking piece is removed.
- If a miner attacks a bomb, the bomb is removed.
- If a players piece attacks the opposing flag, the player wins

5.2 N x N Stratego

Below we explain or choices (or lack off choices in some cases) for generalizations when we work with an N x N version of Stratego.

- Ending the game
    - In the Stratego rules, both in the game ruleset and in the official international Stratego federation rules, there is no guaranteed way to end a game when board positions start to be repeated . The closest rule related to this is that is the More-Squares rule, but that still does not guarantee an end for moves unrelated to chasing. For the proof that Stratego is in EXPTIME, we will assume the additional rule that if a board state is repeated for the third time (and thus there has never been a threatening move on a repeated board state) the game ends in a draw. This assumption is not required for the proof of hardness.
- Information
    - Although Stratego is an imperfect information game, we assume that we reached a state where everybody has all information (like an perfect information game). This is necessary for our upper bound proof, as Stratego with imperfect information, is not in EXPTIME [20]. All pieces are reachable and are assumed to have been attacked at least once. This gives the players information about the ranking and locations of the pieces. Walls, in our reduction, are made up of bomb pieces and are hollow on the inside. We could also have claimed that all pieces of a single rank were hidden in the walls (this can still be known to the player by deduction as these are the only

pieces unreachable without destroying the walls) to create unbreachable walls. However for our current proof that was not necessary.

- The lakes
    - To generalize the lakes two options come to mind. First we can say that every couple of cells there is a 2x2 lake (just as in the regular game). An N x N board then looks like multiple Stratego boards placed next to each other. The other option is to declare that there are two big MxM water areas somewhere in the center. Than the N x N game looks like an upscaled version of the small board.
      For our proof it does not matter which way we generalize it. Because our proof works with scouts that can move unlimited distances we can just adapt the distances in such a way that we can always move around the lakes regardless of the way we generalize it.
- The pieces
    - We have chosen to scale the number of pieces with the board size. We did not find it reasonable to work with 40 pieces on a N x N board. The exact number of pieces is irrelevant to our proof, but we assume that there are enough to build our gadgets.
- The flags
    - Unfortunately, we work in our proof with multiple flags per player. In our reduction we work with a path that gives players a certain win if the opponent does not follow (our simulated actions of) G2.5 (see section 5.5). There need to be multiple starting points for that path. Because of the size of the reduction we did not see any way to connect all these starting points in a single path. So we created individual paths with individuals flags.
      However this does not influence the way the game is played. In our reduction it is still sufficient to just capture a single flag to win the game.

5.3 Stratego Perfect information is in EXPTIME

Here we will show that Stratego Perfect information is in EXPTIME. This works similar to what we showed earlier for chess and checkers.

Each square on the board can have at most 25 different states: one for each of the 12 possible pieces that player I controls, one for each of the 12 possible pieces player II controls and 1 for it being empty. In regular Stratego there are 100 square, but a player can only place pieces on 92 of those (there are 8 lake squares). Thus in a N x N Stratego game, where the lakes scale with the board size, the number of possible game states would be: $25^{(N\^2)*0,92}$. Just like with checkers if we decide to keep track of whose turn it is we get $2 * 25^{(N\^2)*0,92}$ unique board states.

Originally there is no rule that forces a limit on how long a game can take, but, as we mentioned earlier, we decided that if a board state repeats itself for a third time the game ends in a draw. So if we keep track of that we get $3 * 2 * 25^{(N\^2)*0,92}$. So far this is the same as for chess and checkers.

In Stratego however there is the Multi-Square rule. Which mentions that it is illegal to continuously try to capture a piece in a repeated board state. This will be explained in more detail in section 4.1. What is relevant here is that we need to differentiate the board states some more. We need to differentiate if, in a board state, the player starts threatening (e.g. moved an piece next to an opponent's piece). Also we need to know if that same player threatened the turn before. So, as both of these questions can be answered with yes or no, we get a total of 2 * 2 * 3 * 2 * $25^{(N\char`^2)*0,92}$ = 24 * $25^{(N\char`^2)*0,92}$ unique board states.

Again let us define n = N x N.

The depth of each branch is then no more than 24 * $25^{n*0,92}$. We use the minimax algorithm with alpha beta pruning to construct the game tree and obtain a winning strategy in $O(b^{d/2})$ time. Once we know for a board state what the maximum (minimum) score is that player I (II) can receive in a branch we just store that and do not have to calculate it again in another branch. This will lower the complexity from $O(b^{d/2})$ to O(24 * $25^{n*0,92}$) which is exponential. We can disregard that looking up if a board state has already been calculated takes constant time. As in total this would take O(24 * $25^{n*0,92}$) times O(1) which is still exponential time. So in conclusion Stratego is also in EXPTIME.

5.4 $G_{2.5}$

Let us remind the reader about the (relevant) G games from Chandra and Stockmeyer:

G_2: A position is a 4-tuple (τ, I-WIN (X, Y), II-WINX, Y), α) where $\tau \in \{1, 2\}$, I-WIN and II-WIN are formulas in 12DNF, and α is an ($X \cup Y$)-assignment. Player I (II) moves by changing the value assigned to at most one variable in X (Y). Player I (II) wins if the formula I-WIN (II-WIN) is true after some move of player I (II).

G_3: A position is a 4-tuple (τ, I-LOSE(X, Y), II-LOSE(X, Y), α) where $\tau \in \{1, 2\}$, I-LOSE and II-LOSE are formulas in 12DNF, and α is an ($X \cup Y$)-assignment. Player I (II) moves by changing the value assigned to exactly one variable in X (Y). Player I (II) loses if the formula I-LOSE (II-LOSE) is true after some move of player I (II).

There are two key differences between these two games. The first one is that in G2 player I (II) immediately wins after they make I-WIN (II-WIN) true, while in G3 player I (II) only wins after the opponents turn in which they fulfilled II-LOSE (I-LOSE). The second difference is that in G2 the definition includes "at most one" whereas the definition of G3 includes the words "exactly one". This implies that in the game of G2 the player is allowed to pass by not changing any variable value.

For our EXPTIME-hardness proof for Stratego G2 and G2 are slightly difficult to use for the reduction. In G2 player I (II) wins immediately when, after making a move, I-WIN (II-WIN) becomes true. In our corresponding Stratego gadgets it is such that when player I (II) makes a move that satisfies the simulated I-WIN (II-WIN), the opponent has one more turn to falsify that simulated formula, preventing a win. Therefore we looked at G3, as in this game the opponent is allowed one turn to falsify a winning formula. However, unlike G2, G3 does not allow passing and our Stratego gadgets always allow players to make moves that are equivalent to passing.

Therefore we decided to define a new game, called G2.5, where a player is allowed to pass and has an extra turn to invalidate an opponent's winning move. Below we will prove that, just like G2 and G3, this game is EXPTIME-complete.

$G_{2.5}$: A position is a 4-tuple (τ, I-LOSE(X, Y), II-LOSE(X, Y), α) where $\tau \in \{1, 2\}$, I-LOSE and II-LOSE are formulas in 12DNF, and α is an ($X \cup Y$)-assignment. Player I (II) moves by changing the value assigned to at most one variable in X (Y). Player I (II) loses if the formula I-LOSE (II-LOSE) is true after some move of player I (II).

Proposition: G2.5 is EXPTIME-complete

To prove that G2.5 is also EXPTIME-complete we will reduce it to G2. We will do this for player I. To get the proof for player II just replace I-WIN with II-WIN, II-LOSE with I-LOSE and X with Y.

Proof:

We will transform a arbitrary instance of G2 to an instance of G2.5 as follows.

First, alpha and tau remain the same.

Second, the formula I-WIN from G2 is transformed into a formula II-LOSE in G2.5 according to the following rules:

- Any clause in I-WIN of the form X1 ^ ... ^ Xn or the form Y1 ^ ... ^ Yn (for natural number n between 1 and 12) will be kept the same in II-LOSE in G2.5.
- All clauses in I-WIN from G2 with the form X1 ^ ... ^ Xj ^ Y1 ^ ... ^ Yk (for natural numbers j and k between 1 and 11) will be replaced by j separate clauses in II-LOSE in G2.5 with the form X1 ^ ... ^ Xi-1 ^ xi+1 ^ ... ^ Xj ^ Y1 ^ ... ^ Yk.

Claim 1: if player I (II) can win in G2.5, then player I (II) can also win in G2 at most one turn later.

Note that player I winning in G2.5 happens when player II makes a move on their turn that satisfies II-LOSE. Given the formula I-WIN in G2, each clause in the corresponding formula II-LOSE in G2.5 has one of three forms:

- The clause is in the form of X1 ^ … ^ Xn
    - In this case there exists a clause that is exactly the same in I-WIN in G2. Making this clause true in II-LOSE in G2.5 would also make it true in I-WIN in G2. If player I wins in G2.5 by satisfying II-LOSE, then player I would already have won in G2 a turn earlier as the clause only has variables that player I controls.
- The clause is in the form of Y1 ^ … ^ Yn
    - In this case there exists, again, a clause that is identical in I-WIN in G2. Making this clause true in II-LOSE in G2.5 would again also make it true in I-WIN in G2. So if player I wins in G2.5 because player II satisfies II-LOSE, then player I can win a turn later by simply passing.
- The clause is in the form of X1 ^ ... ^ Xi-1 ^ xi+1 ^ ... ^ Xj ^ Y1 ^ ... ^ Yk.
    - In this case, a clause of the form X1 ^ ... ^ Xj ^ Y1 ^ ... ^ Yk exists in I-WIN. So, for any G2.5 circumstance in which player II makes X1 ^ ... ^ Xi-1 ^ xi+1 ^ ... ^ Xj ^ Y1 ^ ... ^ Yk true, the corresponding G2 circumstance would enable player I to make X1 ^ ... ^ Xj ^ Y1 ^ ... ^ Yk true on the next move by either passing or changing Xi.

Claim 2: if player I (II) can win in G2, then player I (II) can also win in G2.5 at most one turn later.

Proof:

We again give a proof by cases. Given the formula I-WIN in G2, we know that each clause in G2 is in one of 3 forms:

- The clause is in the form of X1 ^ … ^ Xn
    - In this case there exists a clause that is exactly the same in I-WIN in G2.5. Making this clause true in I-WIN in G2 would also make it true in II-LOSE in G2.5. If player I wins in G2 by making a move that fulfills I-WIN, then player I would win in G2.5 a turn later

as the clause only has variables that player I controls and the opponent cannot falsify it again.

- The clause is in the form of Y1 ^ … ^ Yn
  - In this case there exists, again, a clause that is identical in II-LOSE in G2.5. Making this clause true in I-WIN in G2 would again also make it true in II-LOSE in G2.5. So if player I wins in G2 because player II satisfies I-WIN and player I does not (and cannot) falsify it, then player I would already have won in G2.5 a turn earlier as the clause only has variables that player II controls.
- The clause is in the form of X1 ^ ... ^ Xj ^ Y1 ^ ... ^ Yk.
  - In this case, a clause of the form X1 ^ ... ^ Xi-1 ^ xi+1 ^ ... ^ Xj ^ Y1 ^ ... ^ Yk exists in II-LOSE. So, for any G2 circumstance in which player I satisfies X1 ^ ... ^ Xj ^ Y1 ^ ... ^ Yk, there is a clause of the form X1 ^ ... ^ Xi-1 ^ xi+1 ^ ... ^ Xj ^ Y1 ^ ... ^ Yk in G2.5 which was already fulfilled one turn before.

With the examples we have shown that player I wins in G2 if and only if player I wins in G2.5. The only difference between I-WIN in G2 and II-LOSE in G2.5 is that II-LOSE has, at most, eleven times the number of clauses (as m, the number of variables the player can change, is at most 11 in the mixed clauses) as I-WIN. This number is small enough that certainly if G2 is in EXPTIME, then G2.5 is also in EXPTIME.

5.5 Global overview

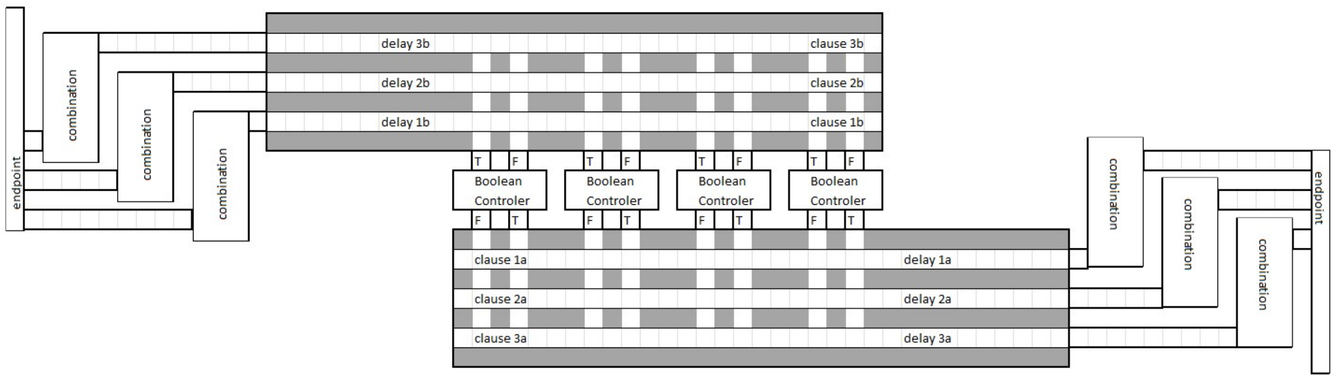


Figure 17 Stratego overview

Our reduction consists of four major components which, when correctly connected together, will give an player a way to win the game of Stratego if and only if that player would also win the EXPTIME-complete game G2.5.

The first component is a Boolean controller. The Boolean controller is the gadget that represents giving an value to a variable in G2.5. There are Boolean controllers equal to the number of different variables over all clauses. Part of them are controlled by the Blue player and the rest are controlled by the Red player. Do note that the distribution does not necessarily have to be equal (e.g. Red can have more Boolean controllers than Blue and vice versa). There are two versions of this gadget: one for the Red player (red Boolean controller) and one for the Blue player (Blue Boolean controller), with the only differences being that it is turned 180 degrees and all Red pieces becoming Blue pieces (and vice versa). The top left path from the players perspective represents the variable being TRUE while the top right path represents that variable being false.

Directly above or below the Boolean controller, connected with those paths, are the clause gadgets. Clause gadgets represent the different clauses in G2.5. All clauses are connected to each other top to bottom. A player can reach a specific clause by crossing them vertically. Each clause gadget is a single corridor that can only be entered (i.e. start moving horizontally in them) when the variable value of the path the player came from corresponds to the variable value in that clause.

At the leftmost end of each corridor is some delay which is different for each clause, but depends on the number of variables and its position towards the rest of the clauses.

Definition **intended flag**: The intended flag is the flag that is captured by the winning player when both players play according to the simulation of $G_{2.5}$.

After the delay is the combination gadget. Here we check if an entire clause is fulfilled. It is basically a corridor leading to a chamber with a width equal to the number of variables in the connected clause. Let us call this z. Then in that room there are z – 1 sergeants (rank 4). The sergeants are there to prevent scouts of the player to reach the endpoint gadget where the intended flag is. When z scouts have gathered (one for each variable) at least one scout can then cross the room in one turn to reach the endpoint gadget.

The endpoint gadget is where the intended flag is for the player to capture. The gadget itself is just one big corridor with the intended flag at the end. It is connected to all combination gadgets so that if one clause is fulfilled (and checked) the intended flag can be captured in one turn (or two if we count entering the gadget as well).

For a total overview of the structure see figure 17.

5.6 Winning strategy

In this section we explain how the strategy in Stratego looks for the player who has won the G2.5 simulation. Arbitrarily we assume this is the Blue player. Therefore we associate Blue with player I in G2.5 . However, as is standard in Stratego, the figures are made with Red at the bottom.

We discuss the regular flow of play in accordance with the simulation of G2.5. For moves that deviate from this, we discuss them in section 5.6 Illegal moves.

We also want to inform the reader that all paths shown in the following gadgets show an eagle eye view. This means that although the path may seem to be going in a straight line, this is a really zoomed out view. Unless specifically mentioned all path can twist and turn and take multiple turns to pass through even with a scout.

Boolean controller

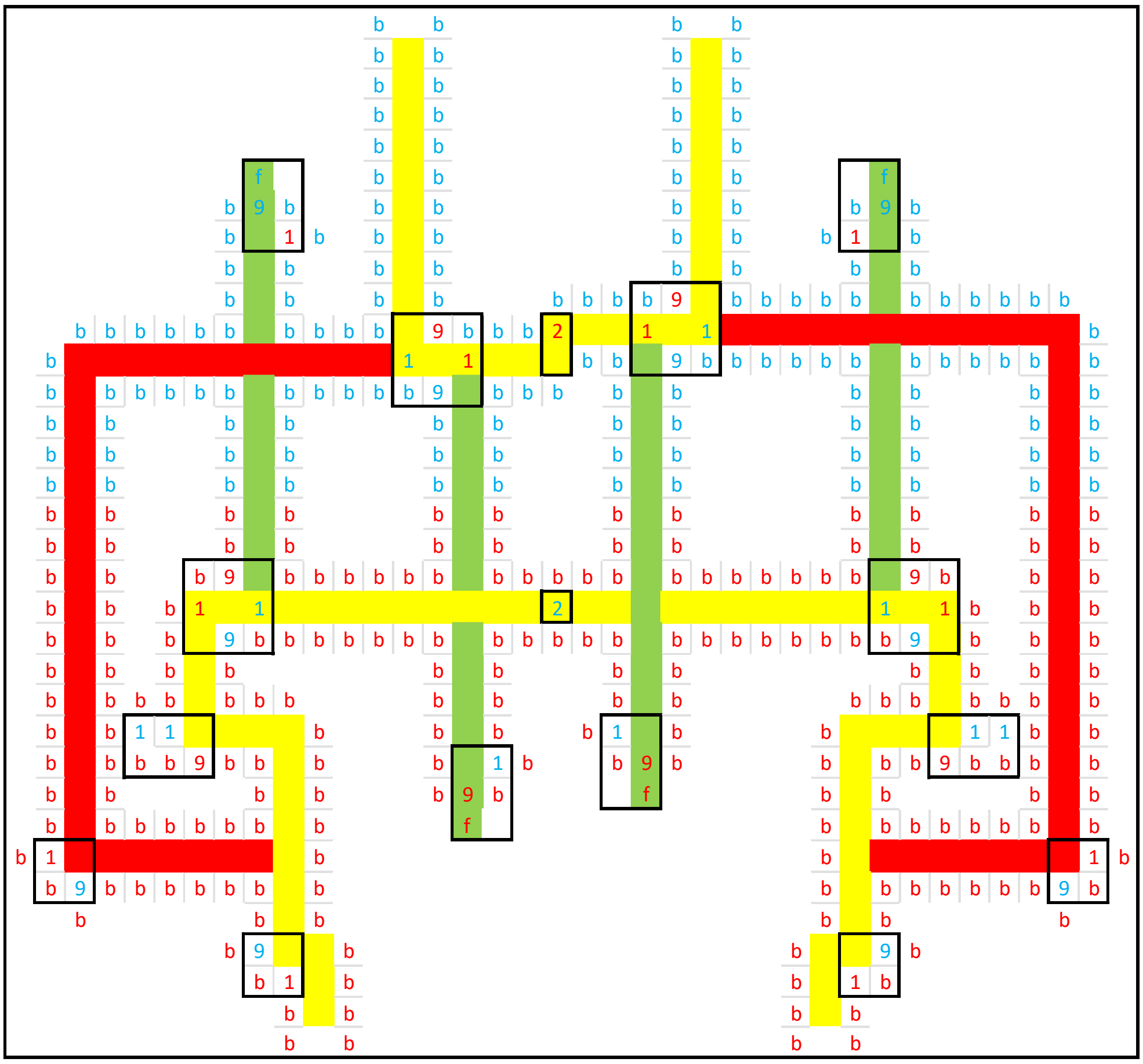


Figure 18 Red (player II) Boolean controller. It has the regular path (yellow), the cheat path (green) and the opponent's defense path (red). The darkened boxes are just for ease of reference.

Recall that a boolean controller is a gadget that is used to simulate the changing of a variable in $G_{2.5}$. In $G_{2.5}$ a player can either change a truth value (for one of the variables they are allowed to change) or skip their turn. To simulate changing a truth value in Stratego the player would be moving their scout up (down) one space in a Boolean controller, changing the value to true (false). We will see later that other types of moves will either allow the opponent a forced win or accomplish nothing, and it is the latter that simulate $G_{2.5}$ passing in Stratego. When, at some point in time, all variable values are corresponding to the variable values in a specific clause, with which we mean that all Boolean controllers represent the correct truth value, player I can start his winning strategy. This means that it is possible for a scout of player I to reach the intended flag and guarantee a win.

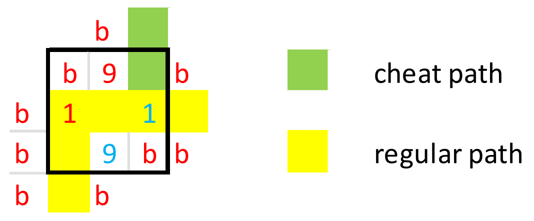


Figure 19 Red (player II) cheat path blockage

In Figure 19 we see a 3 x 3 cheat path blockage. Note that all the pieces keep each other in check. If one of them moves early (i.e. before $G_{2.5}$ has been decided) it will negatively affect that player. Player I will first need to move his spy in a cheat path blockage in a boolean controller (corresponding to a variable path in the winning clause) and either attack the bomb below or move into the cheat path. This will open the path for player I to move his scout, entering from the right side in one move, and capturing the opposing spy the next turn. At the same it will also open the cheat path for the player II (the Red player). This means that player II can now safely move their Marshal into the cheat path.

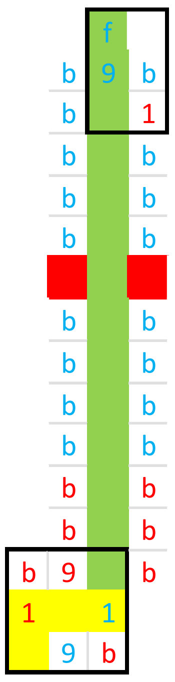


Figure 20 Red (player II) cheat path

Note that, starting now, player II can (try) to reach the flag, with a Marshal, at the end the cheat path that just opened up. Each cheat path has a length equal to the regular winning strategy. If player II would go for the intended flag then, even if player II also has fulfilled their formula, player I can reach it faster as they already started their winning strategy.

Player II has three viable moves that they can do. One they move their Marshal, from the cheat blockage, into the cheat path trying to go for the flag. Two they move their spy, from the cheat blockage, into the opponents pathway creating an extra turn of delay. Three they try to invalidate the formula clause by moving their scout up (down) in another (relevant) Boolean controller, keeping the option open to intercept player I later on.

Note that if the opponent does either move two or three it will not result in any turn advantage. If player II does option two, moving their spy, it is 1 turn in which they do not go for the flag while player I is also delayed by 1 turn. If player II does option three, changing the variable value in another clause, they can eventually intercept player I via the red path. However they did a single move to invalidate the clause, which is one turn they did not go for the flag, which cost player II the 1 turn advantage that they get from intercepting.

Player I will continue moving their scout as far as possible along the yellow path (see Figure 18) in the same boolean controller as the one player II makes a move in, as long as the Boolean controller is relevant for the winning strategy. Otherwise player I will make a move in any relevant Boolean controller. This will continue till player II can capture the scout with their Marshal in the 2 x 3 breakthrough structure (see Figure 21).

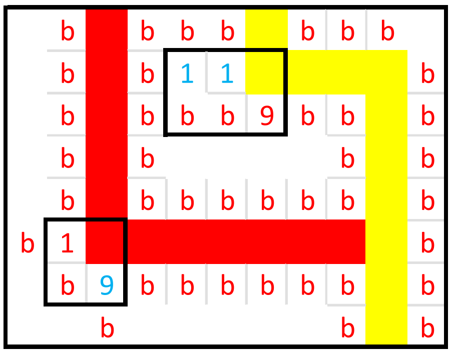


Figure 21 Interception path and 2 x 3 breakthrough structure

Once the scout has been captured by the Marshal, player I is free to move their spies that were being guarded. Player I will now capture the opposing Marshal with their first spy and will continue to move with the spy whenever player II makes a move in that Boolean controller. Player I will continue to do this till their spy reaches the cell just before the intersection with the red path. At this point there are two possibilities. First if player II did not make any moves towards intercepting player I, the spy can just continue onwards passing the intersection with the red path. If, however, the spy of player II is also at the last red square before or on the intersection, player I (and player II also) will lose their spy. Player I will then start moving their second spy to continue on the yellow path.

By intercepting player I, player II can gain a one turn advantage if and only if their scout was already in the correct position that simulated that variable value. If the variable value was changed after the winning strategy of player I has started, that one turn advantage from intercepting has been lost. Note that, if player II freed their spy on the red path, the scout of player II was also captured by the guarding Marshal of player I.

The spy of player I will continue till it reaches a stand-off between their Marshal and an opposing spy. Player I will attack the opposing spy with their spy, destroying them both, and making sure that their Marshal is free to move. The Marshall of player I will then continue onwards towards the next gadget.

An important note here is that, although both the interception path and the regular can twist and turn, the distance from the Marshal from the defense path and the first Marshal of the breakthrough structure to the intersection of the paths needs to be equal. In Figure 21 that distance is shown as 8 squares, but it can be more or less.

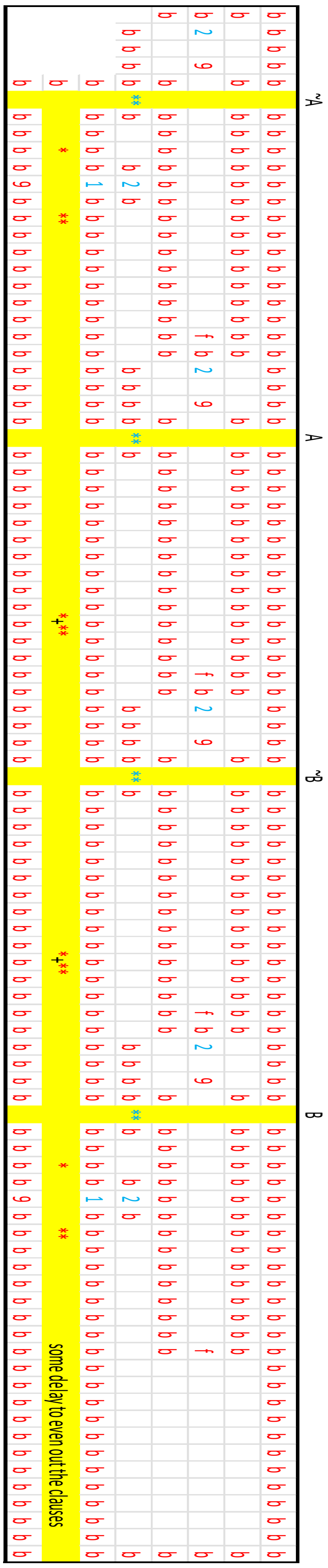


Figure 22 Blue (player I) clause gadget

Clause-gadget

This gadget (see Figure 22) represents a single 12DNF clause in G2.5. Multiple of these gadgets can be connected to each other by connecting the top and bottom entrances/exits with each other.

The Marshal of player I that left the Boolean controller will enter the gadget from the top. It will continue downwards (passing paths with a length of **) till it reaches the simulated clause in G2.5 that is fulfilled. Note that although player II has a Marshal guarding the pathways it has no incentive to capture the Marshal of player I by mutual destruction. This is because the scouts of player I, that are locked up in the rooms, can then move towards the path opposite of the room and capture a flag. On the hand, player I also has no incentive to go for the Marshal of player II. If player I does attempts this, both Marshal's will enter the room, which results in the capture of the scout of player I and a delay to the regular winning strategy of player I.

Note that the path towards the flag is entirely straight and can be traversed in 1 turn with a scout. Also note that this flag is not viable with other pieces after the G2.5 has been decided, as it is too long, and cannot be reached with other pieces before G2.5 has been decided (for the reason why see section 5.6)

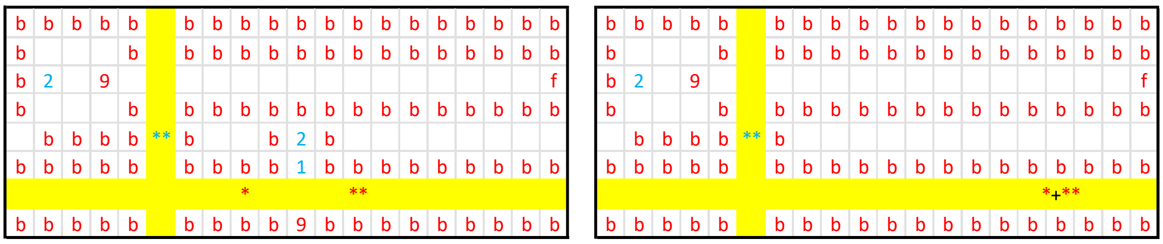


Figure 23 Variable present (left) or absent (right) in a clause

Once the Marshal from player I reaches the relevant clause, it will start moving towards the right till it reaches the place where player I's scout is being held (see Figure 23 left image). This can be passed in a doable number of turns (this is marked with * in Figure 23). Note that the length of * is a minimum of 12 x (number of clauses – 1) * (**– 2) – 1. This is because we don't want the possibility for a Marshal to first free a scout and then walk back and also pass an extra passage and, again possibly, create an extra advantage by suiciding with the guarding Marshal and allow future scouts to pass through.

Once reached player I trades Marshal's with player II, let their spy attack a bomb and moves their scout to the clause path. In total it takes player I three turns to free the scout and get it safely (i.e. without being immediately captured) in the horizontal clause path. Note that if the variable value was not present in the clause there would not be any scout of player I within a reasonable distance (marked with either ** or *+** in Figure 23). With this we mean a scout that cannot be reached before player II wins via the cheat path. It is important to mention that the parts marked with red stars are actually straight (e.g. no twists, no turns). A scout needs to be able to pass these areas in a single turn.

Finally the player I's scout, that is now on the horizontal path, can move in one turn towards the delay section of the clause. Again note that if player I tries to move that distance with any other ranking piece the number of turns it would take would be so much that player II can win via the cheat path. The delay itself will differ from clause to clause and will be explained in the last section of this paragraph.

Combine gadget

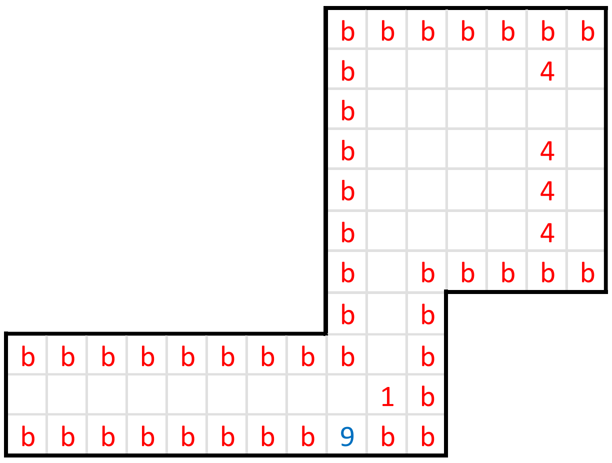
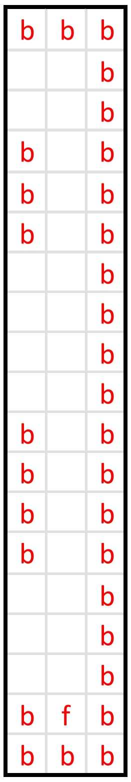


Figure 24 Blue (player I) combine gadget

We would, again, like to remind the reader that although there seems to be one straight path it is not an one turn move with a scout. The path towards the room can curve and make turns. Figure 24 just shows an eagle eye view. Once the scout of player I has left the delay from the clause gadget it will, eventually, reach a corner where it captures an spy from player II. From here player I will move the scout as far to the top as possible. If it would not do that it would block future scouts from getting into the room. Once player I has a scout for every variable in the clause in the room (this can be any number from 1-12), one of them can reach the exit on the right side in one move. This is because the room has a height equal to the number of variables in the clause, but there are height – 1 number of sergeants (rank 4) of player II guarding the room. So there is at least one path that is not covered by the sergeants at any point in time. Notice that the room here is, indeed, a rectangular room. So a scout can indeed pass through it in one turn.

Endpoint gadget



b	b	b
		b
		b
b		b
b		b
b		b
		b
		b
		b
		b
b		b
b		b
b		b
b		b
		b
		b
		b
b	f	b
b	b	b

Figure 25 Blue (player I) endpoint gadget

Finally, once a scout of player I has passed the room of the combine gadget, they will reach a long corridor called the endpoint gadget. The length of the corridor needs to be enough to connect all combine gadgets. The final turns Player I will move the scout as far downwards as possible each move, capturing the intended flag of player II and winning the game. Note that, with optimal play, player II could have captured the flag of player I, at the end of the first cheat path that opened, exactly 1 turn later.

Delay

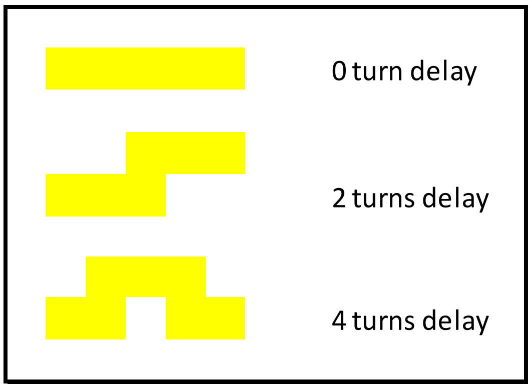


Figure 26 Delay is always even on horizontal pathway

As we mentioned before, the delay at the end of the clause is there to even it out with any other possible clause. For example we do not want that a clause with three variables let player I win faster than a clause with twelve variables. Or that all clauses from player I take longer to reach the intended flag than any of player II's clauses.

To determine the delay for each clause we would need to calculate 4 steps. In the first three we determine what the total delay is and in the fourth step we show how to determine the delay for each clause specifically.

First we need to determine the total number of turns it would take to win without the delay. This would be the minimum to which we would equal a path towards the intended flag (i.e. it can be more, but not less). This delay will be used to set the number of turns to pass a clause equal to the worst case scenario. The worst case scenario (without delay) would be a clause with twelve variables and the fulfilled clause would be the bottom most clause (because than the Marshals need to pass all scout capture paths). This would result in a worst-case scenario of:

12 x steps in boolean controller + (total number of clauses – 1) x ** (the number of paths between clauses) + 12 x (*+1) (this includes freeing the scout and it passing the clause) + 12 * going up + 1 (going up in combination gadget and one scout goes to the end) + 1 (capturing flag in end gadget).

The second step is determining what the formula for the total delay would look like for each clause. The necessary delay = worst case – actual case. The delay would then be enough to add a number turns to equal the worst case. So the formula for the necessary delay would be:

(12 – number of Boolean controllers) x steps in Boolean controller + (total number of clauses – clause number) x ** (this would compensate for the scout capture paths) + (12 – number of Boolean controllers) x (*+1) (compensates freeing scouts and going towards combination gadget) + (12 – number of Boolean controllers) x going up (compensates scouts moving up in combination gadget).

For the third step we would have to compensate for the number of variables in a clause. Notice that the delay will not be passed only one time. It will be passed a number of times equal to the number of variables in the clause. So to compensate for this we have to make sure that, instead of blindly using the minimum, the total number of turns for the winning strategy is in the form of: something x 12! ≥ minimum. This works because this way the delay can always be divided by the number of scouts that need to pass through. So it does not matter how many variables there are in the clause we know we can always divide it equally over all the scouts that pass through. This also fulfills the final step of actually being able to build the delay. As we want an horizontal entrance and exit to the delay it needs to be of even length (see Figure 26). As any number multiplied by 12! is already even this has already been solved.

So in conclusion: let a be the number of Boolean controllers, b be the steps in a Boolean controller, c be **, d be *, e be total number of clauses, f clause number and g is the steps going up in the combination gadget.

The necessary delay of a clause is then equal to: (12 – a) * b + (e – f) * c + (12 – a) * (d+1) + (12 – a) * g. Note that we have influence over the length of b, c, d and g (e.g. by adding extra twist or turns to the path). As is seen in the formula each part of the summation is influenced by a variable we have control over. By choosing the numbers of b, c, d and g intelligently we can guarantee that the delay is divisible by 12!. So each delay of a clause is then equal to ((12 – a) * b + (e – f) * c + (12 – a) * (d+1) + (12 – a) * g)/a which we know is possible as the necessary delay is divisible by 12!.

5.7 Illegal moves

$G_{2.5}$ is a game where the player has the option to pass their turn and in our reduction we have some pieces that can be moved that would count as passing a turn. These moves below however will count as cheating and either guarantee that the player will lose or will give the opponent a massive advantage. Although we believe most of the moves are obvious we will still mention them including the consequences of said move. We will reason using the same images that we used for section 5.5 and will refer to them where possible.

Note that each gadget or part of a gadget has an opposing version where all colors are reversed. The same arguments work for those, just replace player I with player II and vice versa.

Boolean controller

The cheat blockages (Figure 19):

- Moving the spy from player I blocking the cheat path entrance before the game of G2.5 has been decided.
    - Player II can directly start moving into the cheat path with the Marshal and win the game after a fixed number of turns.
- Moving the spy from player II not blocking the cheat path entrance before the game of G2.5 has been decided.
    - Player I will move their Marshall one square upwards. Their next turn either the spy or the Marshal of player I will capture the Marshal of player II. This will stop any future use of the cheat path for player II.
- Moving the Marshal from player I not next to the cheat path entrance before the game of G2.5 has been decided.
    - Player II will move their spy one square towards the right.
        - If player I does not move their spy, player II will capture it the next turn with their spy and the turn after that will start moving in the cheat path with their Marshall. Player II will then win in a fixed number of turns.
        - If player I does move their spy, player II will immediately move their Marshal in the cheat path and win the game after a fixed number of turns.
- Moving the Marshal next to the cheat path entrance before the game of G2.5 has been decided.
    - Player I will immediately capture it with their spy. This will stop any future use of the cheat path.

The cheat paths (Figure 20):

- Moving the spy of player II either before or after G2.5 has been decided.
    - The spy will immediately get captured by the Marshal of player I.
        - The first consequence of this is that the cheat path for player II becomes blocked or at least very difficult to use, as it requires now at least pieces two pieces (a Marshal and at least one other piece) to capture that flag.
        - The second consequence is that, given enough time, the Marshal can open up the cheat path for player I. The Marshal will go via the defense path of player II (the red path) towards a cheat blockage and disrupt it. This will result (after some turns) in that the Marshall of player I in the cheat blockage can safely start capturing the flag.
- Moving the Marshal of player I either before or after G2.5 has been decided.

    - The Marshal will immediately get captured by the spy. Then the spy will capture the player I's flag 2 turns later.

The 2 x 3 breakthrough structure (Figure 21):

- Moving the first spy of player I either before or after G2.5 has been decided.
    - The spy will immediately get captured by the Marshal. Then the Marshal can then keep the second spy of player I in check. If, later on, player II can intercept player I (see winning strategy), then instead of delaying player I it will stop player I from continuing.
- Moving the Marshal of player II either before or after G2.5 has been decided.
    - The Marshal will immediately get captured by the first spy of player I. The spy of player I will then continue and accelerate the loss of player II. If G2.5 has not been decided, this will give enough turn advantage that player I can now play as if a clause (their choice) that contains the variable value, where the breakthrough structure is, is fulfilled.

The Marshal-spy standoff in the defense path (Figure 21):

- Moving the spy of player II either before or after G2.5 has been decided.
    - That spy will immediately get captured by the Marshal of player I. The Marshal can then continue onwards, accelerating the loss of player II the same way as mentioned above in the breakthrough section.
- Moving the Marshal of player I either before or after G2.5 has been decided.
    - The Marshal will immediately get captured by the spy of player II. Player II can now always intercept the spy in the corresponding variable value of player I. When that happens instead of only a one turn advantage, player II gets a more turn advantage (e.g. everything player II would need to do to reach the standoff). This effectively makes winning with any clause that contains that variable value for player I impossible.

The Marshal-spy standoff in the regular path (bottom Figure 18):

- Moving the spy of player II either before or after G2.5 has been decided.
    - The spy will immediately get captured by the Marshal of player I. The Marshal of player I can now continue moving towards the clause gadget. This will accelerate the loss of player II the same way as mentioned above in the breakthrough section.
- Moving the Marshal of player I either before or after G2.5 has been decided.
    - That Marshal will immediately get captured by the spy of player II. The spy can then, either go up and interfere with the breakthrough gadget or staying and blocking any Marshal or spy from player I to exit the Boolean controller. Either way, the path that contains that variable value for player I becomes impossible.

Clause gadget

Marshal guarding a scout in passages between clauses (vertical path Figure 23):

- Moving the scout of player I in the (locked) room.
    - This is mostly safe, but if the scout moves to close to the guarding Marshal of player II it will get captured. This makes it so that the Marshal no longer has to guard there (it may still want to). Now it can interfere with a scout being freed in a clause or stop an opposing Marshal from passing through to the clause below. Regardless, all scenarios negatively impact player I.
- Moving the Marshal of player II that is guarding a room with a scout.
    - If the Marshal moves more than a couple of steps away from the room the scout of player I can exit. In one turn it can reach the flag opposite of the room, immediately winning player I the game.

Marshal-spy standoff with locked scout (horizontal path Figure 23):

- Moving the spy of player I that is guarding the scout.
    - The spy will immediately get captured by the Marshal of player II. This will make the related variable value impossible for player I to use either by blocking the scout of player I from reaching the clause or straight up capture that scout. Furthermore, player II can now keep the Marshal in the clause path prevent other pieces, of player I, from passing and reaching the end of the clause.
- Moving the Marshal of player II.
    - The Marshal will immediately get captured by the spy of player I. Two turns later player I can have their scout safely in the clause path and, in the next turn, pass the clause gadget. This will give player I enough turn advantage that they can now continue playing as if that clause is fulfilled.

Combine gadget

Marshal-spy standoff (Figure 24):

- Moving the spy of player II.
    - The spy will immediately get captured by the Marshal of player I. If $G_{2.5}$ has not been decided yet, the Marshal of player I can go for the intended flag, go towards the last Boolean controller (e.g. the closest one) and disrupt the opponents cheat path or kickstart the cheat path of player. Otherwise it will still prevent the sergeants of player II from exiting the gadget.
- Moving the Marshal of player I.
    - The Marshal will immediately get captured by the spy of player II. The spy and the sergeants of player II can now freely exit the combine gadget. If $G_{2.5}$ has not yet been decided, the sergeants can, possibly, even block the smaller part of the combine gadget, preventing player I from entering with a scout. This prevent any future regular winning strategy.

Below we want to give one special case scenario, in which the winning player wins with a one-move difference in regular play.

- Player I moving any piece, not mentioned in section “the winning strategy”, after G2.5 has been decided.
    - If the opponent has been moving their Marshal, as much as possible, in the first cheat path that opened up, they can now win by continuing moving in that path. This is because a single misplay creates enough delay for the opponent to reach the flag earlier than the player does.

6 Conclusion

We first proved that there exists a game, that we called G2.5, that is EXPTIME-complete. We did this by showing a reduction from G2 to G2.5. If one wins in the game of G2, they would have already won in G2.5 or would do that the same turn. If one would win in the game of G2.5, then they will also win in G2 at most one turn later. So G2.5 is also EXPTIME-complete.

Next as we have shown that perfect information Stratego is in EXPTIME and gave a reduction from a game known to be EXPTIME-complete, namely G2.5, we have completed our proof for the main result that Stratego is EXPTIME-complete. Although this proof works for N x N generalized Stratego, it does not say anything about 10 x 10 regular Stratego. This is because even one Boolean controller is already bigger than the entire board. Also, as mentioned earlier, our EXPTIME upper bound result only holds for perfect information Stratego. That result also assumes a three-time repetition rule that is similar to the one for chess, but is not officially present in standard 10 x 10 Stratego.

We have also shown that Robson’s proof about checkers, while it does prove EXPTIME-completeness of American checkers, it fails to do the same for other variants. We have made an attempt to translate the gadgets so that they work for international checkers and partially succeeded. In the future research section we will give a setup to continue research into this area.

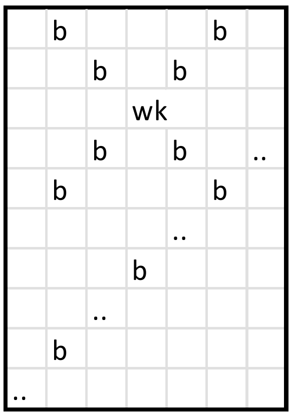
7 Future research

Originally we set out to prove the hardness of Ultimate Lightning Stratego, which is a variant of Stratego. We believed this one to be easier to prove. With our current result, Stratego being EXPTIME-complete, we have a basis for proving that Ultimate Lightning Stratego is EXPTIME-complete. Our current proof would also work for Ultimate Lightning Stratego except that scouts in Ultimate Lightning Stratego can also remove bombs (which would destroy our walls). We can adapt the proof by changing our walls. In our current proof we have hollow walls (i.e. the outer border is made of bombs and the inside is hollow). We can fill these walls with pieces of the same rank and still have perfect information. This is because a player can deduce what pieces are missing on the field, and if that is only one kind of piece, they can know what is in the wall. Filling the walls this way, can prevent scouts in Ultimate Lightning Stratego from passing through. The change may (or may not) affect some of the gadgets (most likely the guarding rooms) and will require some small changes. Our research leaves this as an open question for (possible) future research.

An important question is if, without our three-time repetition rule assumption, Stratego is EXPTIME-complete or just EXPTIME-hard. We could prove Stratego being in EXPTIME because with this rule we could limit the length of a branch in a game tree to an exponential number. If, however, the maximum length of a branch is limitless, how do we prove the game to be in EXPTIME?

Another interesting research topic, namely what is the time complexity of imperfect information Stratego. There are proofs that show that imperfect information games have a time complexity of doubly exponential [20]. This means that there should be a possible reduction for imperfect information Stratego in double exponential time, but this has not been done yet as far as we know. By actually constructing a proof for this we can gain a better understanding in imperfect information games and may also get insight in ways to improve on our proof.

For checkers it is interesting to see if it is possible to adapt Robson's checker proof in such a way that it works for all possible checker classes. We have described what goes wrong and also mentioned some rules to generalize Robson's proof for the variant group which includes international checkers. For example we can fortify the walls. This means that for each direction we do not want a piece to move, we put 2 men of the opposing color (or 1 if there is already a piece) so that the opponent cannot move in that direction or jump over it. This would solve both the flying kings problem and the backwards capturing of men problem (with the exception of the spiral border gadget). There is however one big problem with this. The opponent can move their pieces in a fixed number of turns to remove that wall. We would need to find a way to create enough distance/delay such that removing the wall is no longer a feasible strategy.



b = black man

wk = white king

.. = capturing path

Figure 27 Locked flying king for ending game

An even bigger question is then, if it is even possible make a generalized proof for all variants of checkers. It will then be impossible to replace the kings with men without redesigning entire parts of the gadgets. Another option would be to just fortify all other orthogonal directions around the kings to force them into a specific pathway. A new problem that then occurs is that fortifications does not work with the current spiral border gadget as stopping its free movement defeats its entire purpose. To solve this problem we can create a new gadget that replaces the purpose of the spiral border gadget in the two unsolved checker classes. This gadget consists of a single king surrounded (in the diagonal directions) by 2 men of the opposite color (see Figure 27). By placing (at least) one of these gadgets in the attack zone in such a way that an attack ends by freeing up the king we can leave out the spiral border gadget in the 2 checker classes with flying kings. Ideally we would want to leave the spiral border in so that we have one general proof for all. For this to happen we would need to adapt the spiral border in such a way that it works as intended when there are no flying kings, but does not interfere with the rest of the proof when there are flying kings. However it will require some more research to determine if this is possible.

Finally, even after we change Robson's proof so that it works for all the broad classes of checkers, there are still different rules for individual games. Some of these rules are irrelevant (e.g. size differences in the board), some are easy to implement (e.g. must capture the maximum number of pieces), but some are really difficult to incorporate (e.g. men cannot jump over kings). There are even rules that are in conflict with each other (e.g. only allowed to capture horizontally versus "regular" diagonally capturing). Would it be possible to have one proof that works for all these different checker games? If not, how many interchangeable (sub-)gadgets or variants would there be necessary to make it work? All these questions and more are unanswered and are interesting research topics for future research.

References

[1] L. J. Stockmeyer and A. K. Chandra, Provably difficult combinatorial games, this Journal, 8 (1979), pp. 151-174.

[2] A. S. Fraenkel and D. Lichtenstein, Computing a perfect strategy for n by n chess requires time exponential in n, J. Combin. Theory, 31 (1981), pp. 199-214.

[3] J. M. Robson. N by N Checkers is Exptime complete. SIAM Journal on Computing, 13(2):252–267, 1984.

[4] Akihiro Uejima, Hiroaki Suzuki and Atsuki Okada. "The Complexity of Generalized Pipe Link Puzzles". In: Journal of Information Processing 25 (2017).

[5] Charles J. Colbourn. "The complexity of completing partial Latin squares". In: Discrete Applied Mathematics 8, 25-30 (1984)

[6] Richard Kaye. "Minesweeper is NP-complete". In: The Mathematical Intelligencer volume 22, 9–15 (2000)

[7] Mitchell Donkers. "The NP-completeness of pen and paper puzzles". (2021, September 19): https://studenttheses.uu.nl/bitstream/handle/20.500.12932/1383/Thesis.pdf?sequence=1&isAllowed=y [Online; accessed 16-01-2024]

[8] R. A. Hearn, Amazons is PSPACE-complete. (2005, February 2): https://www.semanticscholar.org/reader/a1e0b427629b9cdd6181b9fd37518f26f6ff182f

[9] S. Even and R. E. Tarjan. A combinatorial problem which is complete in polynomial space. Journal of the Association for Computing Machinery, 23(4):710–719, 1976.

[10] Stefan Reisch. Hex ist PSPACE-vollständig (Hex is PSPACE-complete). Acta Informatica, 15:167–191, 1981.

[11] S. Iwata and T. Kasai, The Othello game on an n*n board is PSPACE-complete, Theor. Comp. Sci. 123 (1994) 329-340.

[12] Brandon McPhail. Light Up is NP-complete. (2005, February 28): http://mountainvistasoft.com/docs/lightup-is-np-complete.pdf

[13] Jan N. van Rijn. The complexity of Klondike, Mahjong, Nonograms and Animal Chess. (2012, June): https://liacs.leidenuniv.nl/assets/2012-01JanvanRijn.pdf

[14] Holzer, M., Ruepp, O. (2007). The Troubles of Interior Design–A Complexity Analysis of the Game Heyawake. In: Crescenzi, P., Prencipe, G., Pucci, G. (eds) Fun with Algorithms. FUN 2007. Lecture Notes in Computer Science, vol 4475. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-72914-3_18

[15] Strategus: https://strategus.appspot.com/rules.html [Online; accessed 13-08-2024]

[16] WDCF Rules committee. World Checkers Draughts Federation (WCDF): https://wcdf.net/rules.htm [Online; accessed 13-08-2024]

[17] FIDE General Assembly. The International Chess Federation (FIDE): https://www.fide.com/FIDE/handbook/LawsOfChess.pdf [Online; accessed 13-08-2024]

[18] Sipser, M. (2012). *Introduction to the Theory of Computation* (3rd ed.).

[19] Wikimedia Foundation. Checkers. (August 12, 2024): https://en.wikipedia.org/wiki/Checkers [Online; accessed 14-08-2024]

[20] John H. Reif. (October, 1984). “The complexity of two-player games of incomplete information”. In: Journal of Computer and System Sciences Volume 29, Issue 2, October 1984, Pages 274-301. https://doi.org/10.1016/0022-0000(84)90034-5