PGN/AN Verification for Legal Chess Gameplay

Neil Shah
`neilshah@cs.cmu.edu`

Guru Prashanth
`ggurugan@cs.cmu.edu`

May 10, 2015

Abstract

Chess has been widely regarded as one of the world's most popular games through the past several centuries. One of the modern ways in which chess games are recorded for analysis is through the PGN/AN (Portable Game Notation/Algebraic Notation) standard, which enforces a strict set of rules for denoting moves made by each player. In this work, we examine the use of PGN/AN to record and describe moves with the intent of building a system to verify PGN/AN in order to check for validity and legality of chess games. To do so, we formally outline the abstract syntax of PGN/AN movetext and subsequently define denotational state-transition and associated termination semantics of this notation.

1 Introduction

Chess is a two-player strategy board game which is played on an eight-by-eight checkered board with 64 squares. There are two players (playing with white and black pieces, respectively), which move in alternating fashion. Each player starts the game with a total of 16 pieces: one king, one queen, two rooks, two knights, two bishops and eight pawns. Each of the pieces moves in a different fashion. The game ends when one player *checkmates* the opponents' king by placing it under threat of capture, with no defensive moves left playable by the losing player. Games can also end with a player's resignation or mutual stalemate/draw. We examine the particulars of piece movements later in this paper.

Chess has a rich history, with modern variants of the game appearing around the year 1200 [4]. Though traditionally played over the board, other variants such as tournament, correspondence and computerized chess have recently gained popularity. In the 19th century, the world chess federation (Fédération Internationale des Échecs, known as FIDE) popularized and imposed the use of Algebraic Notation (AN) to record moves made in chess games. The standard was devised in order to be easy to interpret and read by humans and be shared universally. Given the advent of computerized chess, Portable Game Notation (PGN) – essentially a wrapper around standard AN – was devised as a plain-text computer-processible format for recording games.

Though chess is a game rich in "theory" (strategy) and studied for countless hours by competitive players, mistakes are inevitable in tense gameplay. In chess history, illegal moves made under mental/time pressure or otherwise inattentiveness have resulted in deterioration of chess careers as well as humiliating circumstances for players involved [6, 7]. Though mistakes and illegal moves are common (though without stakes) for beginners, even professional players have fallen victim to such blunders with unfortunate consequences. GM (Grandmaster) Sergey Zagrebelny, GM Ilja Zaragatski, GM Anatoly Karpov are all examples of players who have accidentally played illegal moves in tournament-stakes situations, each with their associated ordeals, ridicule and resulting tournament disqualifications. As chess is considered to be a competitive modern sport, ELO rating, rankings, player sponsorships and titles are on the line for strong players. It is thus of utmost importance that the associated games and results appropriately showcase player ability according to *legal* chess gameplay.

With this motivation, we consider how to formalize the semantics of PGN/AN in order to design a PGN/AN checker which can find illegal moves in chess games.

2 Algebraic Notation Background

We specifically consider AN in this work, as it is functionally interchangeable with PGN and can be considered as "PGN without the frills." To clarify the differences between the two formats, an example of the same game annotated using both AN and PGN is given below.

Algebraic Notation:

```
1. e4 e5 2. Nf3 Nc6 3. Bb5 a6 4. Ba4 Nf6 5. O-O Be7 6. Re1 b5 7. Bb3 d6 8. c3 O-O 9. h3 Nb8 10. d4 Nbd7
11. c4 c6 12. cxb5 axb5 13. Nc3 Bb7 14. Bg5 b4 15. Nb1 h6 16. Bh4 c5 17. dxe5 Nxe4 18. Bxe7 Qxe7 19.
exd6 Qf6 20. Nbd2 Nxd6 21. Nc4 Nxc4 22. Bxc4 Nb6 23. Ne5 Rae8 24. Bxf7 Rxf7 25. Nxf7 Rxe1 26. Qxe1 Kxf7
27. Qe3 Qg5 28. Qxg5 hxg5 29. b3 Ke6 30. a3 Kd6 31. axb4 cxb4 32. Ra5 Nd5 33. f3 Bc8 34. Kf2 Bf5 35.
Ra7 g6 36. Ra6 Kc5 37. Ke1 Nf4 38. g3 Nxh3 39. Kd2 Kb5 40. Rd6 Kc5 41. Ra6 Nf2 42. g4 Bd3 43. Re6
```

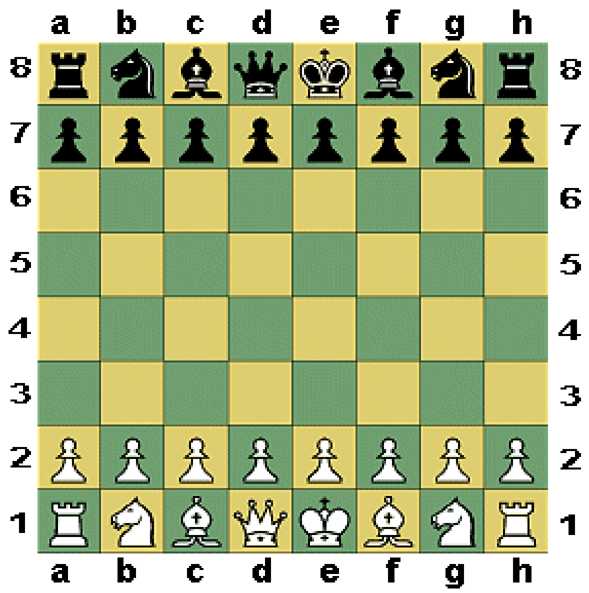

Figure 1: Initial chessboard configuration.

Portable Game Notation:

```
[Event "F/S Return Match"]
[Site "Belgrade, Serbia Yugoslavia|JUG"]
[Date "1992.11.04"]
[Round "29"]
[White "Fischer, Robert J."]
[Black "Spassky, Boris V."]
[Result "1/2-1/2"]

1. e4 e5 2. Nf3 Nc6 3. Bb5 a6 {This opening is called the Ruy Lopez.} 4. Ba4 Nf6 5. O-O Be7 6. Re1 b5
7. Bb3 d6 8. c3 O-O 9. h3 Nb8  10. d4 Nbd7 11. c4 c6 12. cxb5 axb5 13. Nc3 Bb7 14. Bg5 b4 15. Nb1 h6
16. Bh4 c5 17. dxe5 Nxe4 18. Bxe7 Qxe7 19. exd6 Qf6 20. Nbd2 Nxd6 21. Nc4 Nxc4 22. Bxc4 Nb6 23. Ne5
Rae8 24. Bxf7+ Rxf7 25. Nxf7 Rxe1+ 26. Qxe1 Kxf7 27. Qe3 Qg5 28. Qxg5 hxg5 29. b3 Ke6 30. a3 Kd6 31.
axb4 cxb4 32. Ra5 Nd5 33. f3 Bc8 34. Kf2 Bf5 35. Ra7 g6 36. Ra6+ Kc5 37. Ke1 Nf4 38. g3 Nxh3 39. Kd2
Kb5 40. Rd6 Kc5 41. Ra6 Nf2 42. g4 Bd3 43. Re6 1/2-1/2
```

The main differences between the AN and PGN are that AN lacks any human annotations and metadata – specifically, AN contains only raw information of moves and move orders whereas PGN can also include information such as player names and titles, event name, date and human commentary. However, for the purposes of move verification, AN and PGN can be construed in the same fashion.

In the following subsections, we will briefly discuss several of the main facets of AN which broadly encompass the different types of chess moves.

2.1 General Movetext

Letters *a-h* denote *files* whereas numbers *1-8* denote *ranks.* A unique square is then indicated given the file and rank. For example, for all games of chess, the white king always starts on square *e1* (file *e* and rank 1).

to "h" and its horizontal row number (denoted rank) are labelled using numbers "1" to "8". Movetext describes the sequence as well as specifics of moves in the chess game. It is characterized by move number indicators (henceforth referred to as *steps*) suffixed by a period, followed by the corresponding white and black moves in the given step. Typically, each step number is followed by both white and black moves, though a game ending with the white player checkmating the black player will result in only a white move recorded for the last step, as the black player has no available moves to play by definition.

Most moves contain information which indicates a specific piece as well as a square on the board. The two components are simply concatenated for standard *traversal* moves, whereas an *x* is placed between the two for *capture* moves in which one piece captures another, removing it from the board (and play). A position on the board is considered *threatened* if another piece can capture any piece located in this position in one move. One additional notation that is commonly used is *check* which simply states that the King can be captured in the opponent's next move.

The piece indicators and movement descriptions are as follows:

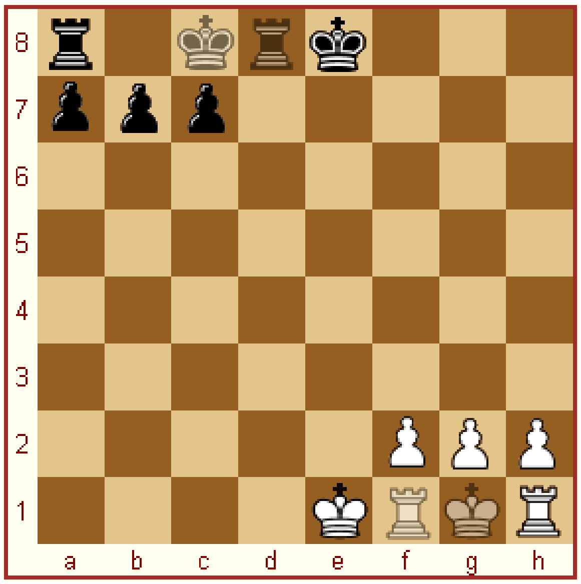

Figure 2: Positions induced by *kingside* castling from white and *queenside* castling from black. Opaque pieces are in pre-castling position, whereas transparent pieces are in post-castling position.

- King – K – The king is able to move one square in any direction on a given move. However, it may not move into a threatened position.
- Queen – Q – The queen can move any number of squares in any direction, though the direction must be fixed throughout the duration of traversal. For example, the queen can move along either diagonal, vertically or horizontally.
- Rook – R – The rook can move any number of squares vertically or horizontally, though the direction must be fixed throughout the duration of traversal.
- Bishop – B – The bishop can move any number of squares along either diagonal, though the direction must be fixed throughout the duration of traversal.
- Knight – N – The knight can only move in an *L*-shaped fashion – that is, it can move 2 squares horizontally and 1 square vertically, or 2 squares vertically and 1 square horizontally. The direction must be fixed along both the 2-square and 1-square traversals.
- Pawn – *empty*

For example, if a bishop moves to square *b5*, then the move is denoted *Bb5*. However, if the bishop captures a piece on *b5* (no matter which piece), the move is denoted *Bxb5*. In ambiguous cases (say, if two knights could move to the same square), then the file or rank of the piece is used to disambiguate (for example, *Ndc5* or *N7c5*).

Furthermore, note that if one player's move results in the other player's king being placed in check, the other player must respond to the check by ensuring that their own king is not in check at the end of his move. Pawns are given empty abbreviations for traversal moves as only one pawn can move to a square on a given white/black move (there is no need for disambiguation). For capture moves, however, piece indicators are used even for pawns as multiple pawns can capture on the same square on a given white/black move.

The board square indicators and general chessboard setup are shown in Figure 1 [1].

2.2 Exceptions

Though the piece movement rules presented in the previous subsection dictate the notation of most moves in typical chess games, there are several aspects of gameplay which they do not encompass. The three notable exceptions to general movetext are *castling*, *pawn promotion*, and *en passant*. These are described below.

2.2.1 Castling

Castling is the only type of move in chess in which *two* pieces move positions in a single move. It involves a player's king and either of his two rooks – castling with the *a*-file rook is called *queenside* castling, whereas castling with the *b*-file rook is called *kingside* castling. Castling involves the king shifting over two squares in the direction of either rook, with the corresponding rook moving to the square which the king crossed. Figure 2 shows the positions resulting from castling [2].

Castling cannot be done while the king is in check, if the king has moved already, or if any of the intermediary squares are occupied by pieces.

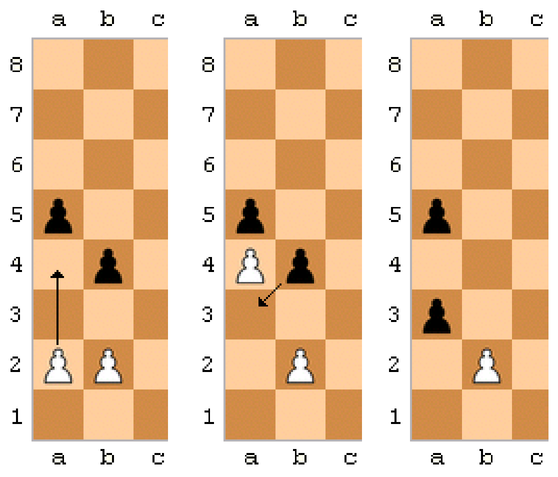

Figure 3: Black captures white's *a*-pawn en passant.

2.2.2 Pawn Promotion

If the pawn of either player reaches its associated 8th rank (a white pawn reaches rank 8 or a black pawn reaches rank 1), the pawn is "promoted" to either a knight, bishop, rook or queen, depending on the player's choice. The pawn cannot remain a pawn once it has reached this rank. The promotion is denoted by appending either *=K*, *=B*, *=R* or *=Q* to the move depending on the piece that the pawn is promoted to. For example, if white promotes an *e*-file pawn to a queen, the move would be denoted in AN as *e8=Q*.

2.2.3 En Passant

En passant (French for *in passing*) is a special type of pawn capture which can occur only immediately after a pawn moves two ranks forward from its initial position, and an opponent pawn could have captured it had the pawn moved only one rank forward. The opponent pawn can capture the just-moved pawn as if it had only moved one rank forward with the associated resulting position.

The en passant capture option is only available the move immediately after the just-moved pawn moves two ranks forward. Figure 3 shows an example of white's pawn being captured en passant [3]. En passant capture is denoted in the same fashion as traditional pawn capture, with the assumption that the captured pawn had only moved one rank forward at the time of capture – black's en passant capture in Figure 3 would be denoted as *bxd3*.

3 Semantic Formalization

Having covered the appropriate background for AN movetext, we now pose the following question: Given the AN notation for a chess game, how can we determine if the play is indeed permitted within the realm of legal chess moves?

To this end, we take the following approach. First, we consider AN as a language with various types of commands encompassing player moves. We propose an abstract syntax for AN and describe the denotational state-transition and termination semantics of the various types of chess moves. In practice, a PGN/AN checker would start in a fixed initial state corresponding to the setup of a chessboard, and with each move transition to a new state. Termination semantics dictate non-terminating (in practice, nonzero return code) behavior for moves which are invalid according to chess rules. Summarily, our checker would function by iterating move-by-move over the PGN/AN and essentially "re-playing" the game. If the game is re-played successfully and the "AN program" terminates, the PGN/AN was valid. If the AN program does not terminate, the game must have contained an illegal move and the result is invalid. Note that here we focus on invalid *chess moves*, not on malformed or invalid PGN/AN. This can be easily checked for with a parser and is out-of-scope for this work.

3.1 Abstract Syntax

We begin by defining the abstract syntax of AN. AN is composed of *steps* (**S**) and various types of *moves* (**M**) including *traversals*, *captures*, *castles* and *promotions*. Steps are semantically defined as partial functions between natural numbers ($\mathbb{N}$) and moves, whereas moves are semantically defined as partial functions from states to states. *States* are defined as partial functions from *identifiers* (**Ide**) to natural numbers ($\mathbb{N}$). Identifiers are essentially program variables, drawn from fixed set of names (for example, x or y). That is,

$$\begin{aligned}
\mathbf{St} &: \mathbf{Ide} \rightharpoonup \mathbb{N} \\
\mathbf{M} &: \mathbf{St} \rightharpoonup \mathbf{St} \\
\mathbf{S} &: \mathbb{N} \rightharpoonup (\mathbf{St} \rightharpoonup \mathbf{St})
\end{aligned}$$

The abstract syntax is given below. We let S range over the set $\mathbf{S}$ of steps, M range over the set $\mathbf{M}$ of moves and n range over the set $\mathbb{N}^+$ of positive natural numbers. Furthermore, P ranges over the set of piece indicators $\mathcal{P}$ and Q ranges over the set of square indicators $\mathcal{Q}$. T ranges over the set of promotion candidate piece indicators $\mathcal{P}_{major}$ and W ranges over the set of pawn piece indicators $\mathcal{P}_{pawns}$. Note that $a :: b$ is the syntactic concatenation operator, which corresponds to appending b to a. $\mathcal{P}$, $\mathcal{Q}$ and their constituents are defined as follows.

$$\begin{aligned}
\mathcal{P} &= \mathcal{P}_{king} \cup \mathcal{P}_{major} \cup \mathcal{P}_{pawns} \\
\mathcal{P}_{king} &= \{K\} \\
\mathcal{P}_{major} &= \{Q, N, B, R\} \\
\mathcal{P}_{pawns} &= \{a, b, c, d, e, f, g, h\} \\
\mathcal{Q} &= \{f :: r | f \in \mathcal{F}, r \in \mathcal{R}\} \\
\mathcal{F} &= \{a, b, c, d, e, f, g, h\} \\
\mathcal{R} &= \{1, 2, 3, 4, 5, 6, 7, 8\}
\end{aligned}$$

$$\begin{aligned}
S &:= nil \mid n.\ M\ ::\ S \\
M &:= M_1\ M_2 \mid traversals \mid captures \mid castles \mid promotions \\
traversals &:= Q \mid P :: Q \\
captures &:= P :: x :: Q \\
castles &:= O - O \mid O - O - O \\
promotions &:= Q ::=:: T \mid W :: x :: Q ::=:: T
\end{aligned}$$

3.2 Denotational Semantics

Using these grammar rules, we can construct arbitrary AN notations corresponding to chess games. A multi-step AN notation can be constructed using the S rule, whereas steps can have two moves corresponding to both white and black player moves, or just a single move (indicating the black player's resignation or forced checkmate). Next, we outline the associated denotational semantics of the AN syntax.

3.2.1 Steps

We consider steps to map positive natural numbers to player moves, which are executed in the associated, ascending order.

The semantic function $[[S]] : \mathbb{N} \rightharpoonup (\mathbf{St} \rightharpoonup \mathbf{St})$ is given by:

$$\begin{aligned}
\textit{[[nil]]} &= \emptyset \\
\textit{[[n. M::S]]} &= \Big\{(n, [[M]])\Big\} \cup \Big\{(b, Y) \mid (b, Y) \in [[S]], n \neq b\Big\}
\end{aligned}$$

The semantic function allows for arbitrarily many steps corresponding to chess games of various lengths. Each step is composed of a move, which is defined next.

3.2.2 Moves

We must be careful in defining the semantics of moves. Each move changes the current state in some fashion. In order to capture *how* the state changes, we need to introduce some universal identifiers for each state which capture information about the pieces and their positions. On first thought, one considers that it may be sufficient to have an identifier for each piece and simply change the value when the piece moves. However, this strategy fails to deal with the notion of pawn promotion, as pawns can turn into other types of major pieces with different movement patterns. This will be important when considering termination semantics, later in the paper.

Rather, a better strategy is to use the board squares as identifiers themselves. Thus, at each state, we keep track of the 64 squares and the pieces which reside on them. For example, since the start of each chess game is the same, our initial state will always have pawns residing on the 2nd and 7th ranks. Thus, the identifiers $a2 - h2$ and $a7 - h7$ will be set to some $n \in \mathbb{N}$ corresponding to a pawn. Furthermore, since the movement patterns of pieces will differ based on which player they are controlled by (i.e: white pawns will march towards increasing ranks, whereas black pawns will march towards decreasing ranks), we will assign white and black pieces different identifiers. Again, this will prove useful when considering termination semantics, later in the paper. Without this rule, there would be no way to discern whether black captured a white piece (legal) or his own piece (illegal). We can use the following numbering scheme.

- 0 – empty square
- 1 – white queenside rook
- 2 – white queenside knight
- 3 – white queenside bishop
- 4 – white queen
- 5 – white king
- 6 – white kingside bishop
- 7 – white kingside knight
- 8 – white kingside rook
- 9-16 – white $a - h$ pawns
- 17 – black queenside rook
- 18 – black queenside knight
- 19 – black queenside bishop
- 20 – black queen
- 21 – black king
- 22 – black kingside bishop
- 23 – black kingside knight
- 24 – black kingside rook
- 25-32 – black $a - h$ pawns

We also introduce the notation σ_{id}^{val} to refer to a state exactly like σ, but with the identifier id assigned to the value val. For example, σ_{d1}^{5} would change the value of the $d1$ identifier to 5, meaning that square $d1$ is now host to the white king. When used in semantic definitions, id corresponds to the value of id in state σ (source state), unless otherwise specified.

We will further introduce a *turn* identifier so as to indicate which player is making a certain move. This is needed to disambiguate between white playing a move (say, $e5$) versus black making the very same move. We can initialize *turn* to 0 for the initial state, and increment it upon each state transition (player move). Thus, if turn is even, white makes the next move – else, if turn is odd, then black makes the next move.

As we will see, we will require more identifiers and some auxiliary functions to capture and utilize necessary information to re-play the AN for checking. These will be introduced as-needed.

The semantic function $[[M]] : \mathbf{St} \rightharpoonup \mathbf{St}$ is given by:

$$[[M_1\ M_2]] = \Big\{(\sigma, \sigma'') \mid \exists \sigma'.(\sigma, \sigma') \in [[M_1]] \ \&\ (\sigma', \sigma'') \in [[M_2]]\Big\}$$

For traversal moves, pertaining to movement of only a single piece, it is apparent that to account for the change in state, we need to know which player is making the move, and the source and destination squares which need to be updated. For example, if the white player plays $e4$, then the new state must change the status of the $e4$ square to host a pawn as well as change the status of the square which the pawn was originally placed at to empty. Lastly, *turn* must be updated to reflect that the next move should be played by the black player. The very same situation arises for capture moves. However, AN does not include information about source, but only destination squares. For this predicament, we will find it very useful to introduce some auxiliary functions: *GetTraversalSourceSquare* and *GetCaptureSourceSquare*. Though we do not provide technically exact versions of these functions in this paper, we give pseudocode in Algorithms 1 and 2. This is mainly to ensure clarity of text as making each of them rigorous would only involve unravelling "if" statements (something we already did formally in class).

Algorithm 1: GTSS – GetTraversalSourceSquare

```
Data: target_square that the piece moves to, piece_id ($K$, $N7$, $Ra$), state $\sigma$
Result: source_square of the piece, piece_number as per the numbering scheme
if turn is even then
    set color to white
else
    set color to black
if piece_id is king then
    check if there is a color king 1 square away from target_square;
    return the king's source_square and number;
if piece_id is queen then
    check if there is an unblocked color queen on the same diagonal/horizontal/vertical as target_square;
    return the queen's source_square and number;
if piece_id is rook then
    check if there is an unblocked color rook on the same horizontal/vertical as target_square;
    return the rook's source_square and number;
if piece_id is bishop then
    check if there is an unblocked color bishop on the same diagonal as target_square;
    return the bishop's source_square and number;
if piece_id is knight then
    check if there is a color knight 2 squares horizontal and 1 square vertical or 1 square horizontal and 2 squares
    vertical from target_square and number;
    return the knight's source_square;
if piece_id is pawn (piece_id $\in \mathcal{P}_{pawns}$) then
    check if there is an unblocked color pawn 1 square (down if white, up if black) from target_square or 2 squares
    (down if white, up if black) from target_square if target_square is on rank 4 or 5;
    return the pawn's source_square and number;
```

Algorithm 2: GCSS – GetCaptureSourceSquare

```
Data: target_square that the piece moves to, piece_id ($K$, $N7$, $Ra$), state $\sigma$
Result: source_square of the piece, piece_number as per the numbering scheme
if turn is even then
    set color to white
else
    set color to black
if piece_id is king then
    check if there is a color king 1 square away from target_square;
    return the king's source_square and number;
if piece_id is queen then
    check if there is an unblocked color queen on the same diagonal/horizontal/vertical as target_square;
    return the queen's source_square and number;
if piece_id is rook then
    check if there is an unblocked color rook on the same horizontal/vertical as target_square;
    return the rook's source_square and number;
if piece_id is bishop then
    check if there is an unblocked color bishop on the same diagonal as target_square;
    return the bishop's source_square and number;
if piece_id is knight then
    check if there is a color knight 2 squares horizontal and 1 square vertical or 1 square horizontal and 2 squares
    vertical from target_square;
    return the knight's source_square and number;
if piece_id is pawn (piece_id $\in \mathcal{P}_{pawns}$) then
    check if there is an unblocked color pawn 1 square (down-left or down-right if white, up-left or up-right if black)
    from target_square (normal capture), or if there is a pawn 1 square horizontally away from target_square on
    appropriate rank (5 if white, 4 if black) which moved in the previous turn (en passant);
    return the pawn's source_square and number;
```

The two functions are very similar. Given the target square, piece id (both available from the movetext) and current state, *GetTraversalSourceSquare* returns the source square of the specified piece which supposedly moves to occupy the target square, as well as the piece number (according to the numbering scheme posed above). Using the same information, *GetCaptureSourceSquare* returns the source square and piece number of the specified piece which supposedly captures another piece which currently occupies the target square. Implementing these methods is painstaking, but relatively straightforward given the clearly defined movement and capture patterns of chess pieces. In case of more specific piece ids which include rank number or file letter, the number of squares which must be checked for the associated piece is simply reduced to those which lie on the corresponding rank or file.

Note that one important difference between the two functions is that in the case that the piece id is a pawn, the squares that are examined are different. The pawn is the only piece in the game of chess which has distinct traversal and capture patterns – the pawn traverses vertically, but captures diagonally.

Furthermore, *GetCaptureSourceSquare* also accounts for en passant captures. One subtlety in this capability is that en passant captures can only happen on the immediately subsequent move after the to-be-captured pawn traverses 2 squares. However, given the 65 current state identifiers (one for each square and one for turn), this is unenforcible. Fortunately, the solution is relatively straightforward: we can introduce an extra identifer for each of the 16 pawns which takes on the value of the current state's *turn* identifier when the pawn traverses 2 squares in a single move. For example, if white plays $e4$ from the pawn's starting position $e2$ on turn 20, we can set the value of an identifier $e2\_2step$ to 20. Prefixing these identifier names with the pawn's starting positions is sufficient to distinguish white and black pawns on all files, as each pawn has a unique starting position fixed across all chess games. When checking for en passant conditions, one only need to check whether the current state's turn is 1 higher than the associated pawn's 2-square turn counter. To enable easier checking of the distance between 2 squares used for pawn turn counter checking, let us introduce auxiliary functions *rank* and *file* which, given a square, return the rank and file respectively.

Of course, if *GetTraversalSourceSquare* or *GetCaptureSourceSquare* returns no pieces, then the game is invalid, as it contains an impossible move. Conversely, if the return set has more than one piece, the AN is malformed as a well-formed AN would have rank/file specifications in case of ambiguity (malformed PGN/AN is not addressed in this work). These will be broached to a greater extent when discussing termination/non-termination.

We henceforth abbreviate *GetCaptureSourceSquare* as *GCSS* and *GetTraversalSourceSquare* as *GTSS*.

Equipped with these new identifiers and auxiliary functions, we can write the semantic function for $[[traversals]] : \mathbf{St} \rightharpoonup \mathbf{St}$ as below.

$$[[Q]] = \begin{cases} \left\{(\sigma, \sigma') \mid \sigma' = \sigma^{v,0,turn,turn+1}_{Q,\mathcal{S},\mathcal{S}\_2step,turn} \ , \ (\mathcal{S}, v) = GTSS(Q, file(Q), \sigma)\right\} & |rank(\mathcal{S}) - rank(Q)| = 2 \\ \left\{(\sigma, \sigma') \mid \sigma' = \sigma^{v,0,turn+1}_{Q,\mathcal{S},turn} \ , \ (\mathcal{S}, v) = GTSS(Q, file(Q), \sigma)\right\} & otherwise \end{cases}$$

$$[[P::Q]] = \left\{(\sigma, \sigma') \mid \sigma' = \sigma^{v,0,turn+1}_{Q,\mathcal{S},turn} \ , \ \ (\mathcal{S}, v) = GTSS(Q, P, \sigma)\right\}$$

In the case that we have a simple pawn move, we modify only the appropriate square identifiers and turn count. In case of a 2-square pawn move, we additionally set the turn counter for the relevant pawn to the value of *turn* in σ.

We can also write the semantic function for $[[captures]] : \mathbf{St} \rightharpoonup \mathbf{St}$ as below.

$$[[P::x::Q]] = \left\{(\sigma, \sigma') \mid \sigma' = \sigma^{v,0,turn+1}_{Q,\mathcal{S},turn} \ , \ \ (\mathcal{S}, v) = GCSS(Q, P, \sigma)\right\}$$

Semantically, capture moves are very similar to traversal moves. The major difference is that in traversal moves, the destination square is empty, whereas in capture moves the destination square contains an opponent's piece. However, these are construed in the same in that the destination square is occupied by the piece which traversed/captured onto that square, whereas the source square becomes empty.

Next, we consider castling moves. Castling is only allowed if intermediary squares between the king and corresponding rook are empty, and if the king has not moved before. This necessitates the use of a few more identifiers. Specifically, if a player's king moves (traverses or captures), there should be an identifier to serve as a flag. For the initial state, we can say that the identifiers w_king and b_king are set to 0, for the white king and black king respectively. Upon moving the king, the associated flag should be set to 1. Thus, before we move onto castles semantics, let us revise the affected previous semantic evaluations for traversals

$$[[P::Q]] = \begin{cases} \left\{(\sigma, \sigma') \mid \sigma' = \sigma^{v,0,1turn+1}_{Q,\mathcal{S},w\_king,turn} \ , \ \ (\mathcal{S}, v) = GTSS(Q, P, \sigma)\right\} & P \in \mathcal{P}_{king}, mod(turn, 2) = 0 \\ \left\{(\sigma, \sigma') \mid \sigma' = \sigma^{v,0,1turn+1}_{Q,\mathcal{S},b\_king,turn} \ , \ \ (\mathcal{S}, v) = GTSS(Q, P, \sigma)\right\} & P \in \mathcal{P}_{king}, mod(turn, 2) = 1 \\ \left\{(\sigma, \sigma') \mid \sigma' = \sigma^{v,0,turn+1}_{Q,\mathcal{S},turn} \ , \ \ (\mathcal{S}, v) = GTSS(Q, P, \sigma)\right\} & otherwise \end{cases}$$

and for captures

$$\textit{[[P::x::Q]]} = \begin{cases} \left\{(\sigma,\sigma') \mid \sigma' = \sigma^{v,0,1turn+1}_{Q,\mathcal{S},w\_king,turn} \ , \ \ (\mathcal{S},v) = GCSS(Q,P,\sigma)\right\} & P \in \mathcal{P}_{king}, mod(turn,2) = 0 \\ \left\{(\sigma,\sigma') \mid \sigma' = \sigma^{v,0,1turn+1}_{Q,\mathcal{S},b\_king,turn} \ , \ \ (\mathcal{S},v) = GCSS(Q,P,\sigma)\right\} & P \in \mathcal{P}_{king}, mod(turn,2) = 1 \\ \left\{(\sigma,\sigma') \mid \sigma' = \sigma^{v,0,turn+1}_{Q,\mathcal{S},turn} \ , \ \ (\mathcal{S},v) = GCSS(Q,P,\sigma)\right\} & otherwise \end{cases}$$

Now, equipped with the newly introduced *w_king* and *b_king* identifiers for castling, we can write the semantic function for $[[castles]] : \mathbf{St} \rightharpoonup \mathbf{St}$ as below:

$$\textit{[[O-O]]} = \begin{cases} \left\{(\sigma,\sigma') \mid \sigma' = \sigma^{5,8,0,0,turn+1}_{g1,f1,e1,h1,turn}\right\} & mod(turn,2) = 0, w\_king = 0, f1 = 0, g1 = 0, h1 = 8 \\ \left\{(\sigma,\sigma') \mid \sigma' = \sigma^{21,24,0,0,turn+1}_{g8,f8,e8,h8,turn}\right\} & mod(turn,2) = 1, b\_king = 0, f8 = 0, g8 = 0, h1 = 24 \end{cases}$$

$$\textit{[[O-O-O]]} = \begin{cases} \left\{(\sigma,\sigma') \mid \sigma' = \sigma^{5,1,0,0,turn+1}_{c1,d1,e1,a1,turn}\right\} & mod(turn,2) = 0, w\_king = 0, b1 = 0, c1 = 0, d1 = 0, a1 = 1 \\ \left\{(\sigma,\sigma') \mid \sigma' = \sigma^{21,17,0,0,turn+1}_{c8,d8,e8,a8,turn}\right\} & mod(turn,2) = 1, b\_king = 0, b8 = 0, c8 = 0, f8 = 0, a1 = 17 \end{cases}$$

In the case of kingside castling $(O-O)$, depending on the player whose move it is, we check that the king has not moved and the appropriate rook is in place, and the pieces between the king and the kingside rook are empty before updating the state such that the king and kingside rook are on new squares. We additionally empty the squares that the king and rook used to occupy. Queenside castling $(O-O-O)$ happens similarly, but for the queenside rook.

Lastly, we consider pawn promotion. Promotion semantics are rather similar to traversal and capture semantics for pawns, with the exception that the destination square now hosts a difference piece number, corresponding to a knight, bishop, rook or queen. The semantic function for $[[promotions]] : \mathbf{St} \rightharpoonup \mathbf{St}$ is as below:

$$\textit{[[Q::=::T]]} = \begin{cases} \left\{(\sigma,\sigma') \mid \sigma' = \sigma^{2,0,turn+1}_{Q,\mathcal{S},turn} \ , \ (\mathcal{S},v) = GTSS(Q,file(Q),\sigma)\right\} & mod(turn,2) = 0, T = N \\ \left\{(\sigma,\sigma') \mid \sigma' = \sigma^{6,0,turn+1}_{Q,\mathcal{S},turn} \ , \ (\mathcal{S},v) = GTSS(Q,file(Q),\sigma)\right\} & mod(turn,2) = 0, T = B, file(Q) \in \{a,c,e,g\} \\ \left\{(\sigma,\sigma') \mid \sigma' = \sigma^{3,0,turn+1}_{Q,\mathcal{S},turn} \ , \ (\mathcal{S},v) = GTSS(Q,file(Q),\sigma)\right\} & mod(turn,2) = 0, T = B, file(Q) \in \{b,d,f,h\} \\ \left\{(\sigma,\sigma') \mid \sigma' = \sigma^{1,0,turn+1}_{Q,\mathcal{S},turn} \ , \ (\mathcal{S},v) = GTSS(Q,file(Q),\sigma)\right\} & mod(turn,2) = 0, T = R \\ \left\{(\sigma,\sigma') \mid \sigma' = \sigma^{4,0,turn+1}_{Q,\mathcal{S},turn} \ , \ (\mathcal{S},v) = GTSS(Q,file(Q),\sigma)\right\} & mod(turn,2) = 0, T = Q \\ \left\{(\sigma,\sigma') \mid \sigma' = \sigma^{18,0,turn+1}_{Q,\mathcal{S},turn} \ , \ (\mathcal{S},v) = GTSS(Q,file(Q),\sigma)\right\} & mod(turn,2) = 1, T = N \\ \left\{(\sigma,\sigma') \mid \sigma' = \sigma^{22,0,turn+1}_{Q,\mathcal{S},turn} \ , \ (\mathcal{S},v) = GTSS(Q,file(Q),\sigma)\right\} & mod(turn,2) = 1, T = B, file(Q) \in \{a,c,e,g\} \\ \left\{(\sigma,\sigma') \mid \sigma' = \sigma^{19,0,turn+1}_{Q,\mathcal{S},turn} \ , \ (\mathcal{S},v) = GTSS(Q,file(Q),\sigma)\right\} & mod(turn,2) = 1, T = B, file(Q) \in \{b,d,f,h\} \\ \left\{(\sigma,\sigma') \mid \sigma' = \sigma^{17,0,turn+1}_{Q,\mathcal{S},turn} \ , \ (\mathcal{S},v) = GTSS(Q,file(Q),\sigma)\right\} & mod(turn,2) = 1, T = R \\ \left\{(\sigma,\sigma') \mid \sigma' = \sigma^{20,0,turn+1}_{Q,\mathcal{S},turn} \ , \ (\mathcal{S},v) = GTSS(Q,file(Q),\sigma)\right\} & mod(turn,2) = 1, T = Q \end{cases}$$

$$
\textit{[[W::x::Q::=::T]]} = \begin{cases}
\left\{(\sigma,\sigma') \mid \sigma' = \sigma^{2,0,turn+1}_{Q,\mathcal{S},turn}\ ,\ (\mathcal{S},v) = GCSS(Q, file(Q), \sigma)\right\} & mod(turn,2)=0, T=N \\
\left\{(\sigma,\sigma') \mid \sigma' = \sigma^{6,0,turn+1}_{Q,\mathcal{S},turn}\ ,\ (\mathcal{S},v) = GCSS(Q, file(Q), \sigma)\right\} & mod(turn,2)=0, T=B, file(Q) \in \{a,c,e,g\} \\
\left\{(\sigma,\sigma') \mid \sigma' = \sigma^{3,0,turn+1}_{Q,\mathcal{S},turn}\ ,\ (\mathcal{S},v) = GCSS(Q, file(Q), \sigma)\right\} & mod(turn,2)=0, T=B, file(Q) \in \{b,d,f,h\} \\
\left\{(\sigma,\sigma') \mid \sigma' = \sigma^{1,0,turn+1}_{Q,\mathcal{S},turn}\ ,\ (\mathcal{S},v) = GCSS(Q, file(Q), \sigma)\right\} & mod(turn,2)=0, T=R \\
\left\{(\sigma,\sigma') \mid \sigma' = \sigma^{4,0,turn+1}_{Q,\mathcal{S},turn}\ ,\ (\mathcal{S},v) = GCSS(Q, file(Q), \sigma)\right\} & mod(turn,2)=0, T=Q \\
\left\{(\sigma,\sigma') \mid \sigma' = \sigma^{18,0,turn+1}_{Q,\mathcal{S},turn}\ ,\ (\mathcal{S},v) = GCSS(Q, file(Q), \sigma)\right\} & mod(turn,2)=1, T=N \\
\left\{(\sigma,\sigma') \mid \sigma' = \sigma^{22,0,turn+1}_{Q,\mathcal{S},turn}\ ,\ (\mathcal{S},v) = GCSS(Q, file(Q), \sigma)\right\} & mod(turn,2)=1, T=B, file(Q) \in \{a,c,e,g\} \\
\left\{(\sigma,\sigma') \mid \sigma' = \sigma^{19,0,turn+1}_{Q,\mathcal{S},turn}\ ,\ (\mathcal{S},v) = GCSS(Q, file(Q), \sigma)\right\} & mod(turn,2)=1, T=B, file(Q) \in \{b,d,f,h\} \\
\left\{(\sigma,\sigma') \mid \sigma' = \sigma^{17,0,turn+1}_{Q,\mathcal{S},turn}\ ,\ (\mathcal{S},v) = GCSS(Q, file(Q), \sigma)\right\} & mod(turn,2)=1, T=R \\
\left\{(\sigma,\sigma') \mid \sigma' = \sigma^{20,0,turn+1}_{Q,\mathcal{S},turn}\ ,\ (\mathcal{S},v) = GCSS(Q, file(Q), \sigma)\right\} & mod(turn,2)=1, T=Q
\end{cases}
$$

Note that the semantic evaluations for both promotions by traversal and promotions by capture are very similar. This is mostly a result of the innate similarity between traversal moves and capture moves in general. Of course, the auxiliary function we use to look for source squares differs between the two mechanics.

In both cases, we consider 5 possibilities of state transition depending on player turn. 4 of these possibilities arise as a result of the 4 types of promotions available to players (knight, bishop, rook, queen). The 5th possibility denotes the more subtle difference between the promotion mechanics of pawns to bishops. Namely, when a white pawn is promoted to a bishop on the $\{a, c, e, g\}$ files, it lands on a white square, which designates its movement patterns to be the same as those of the white *kingside* bishop. However, if a white pawn is promoted to a bishop on the $\{b, d, f, h\}$ files, it lands on a black square, which designates its movement patterns to be the same as those of the white *queenside* bishop. A similar argument follows for black pawns. We do not need to be peculiar about knight, rook and queen promotion mechanics, as both (white/black) knights, rooks and queen can reach all the squares on the board. The queen and rook cases are obvious – a rook can travel some number of files horizontally and ranks vertically to reach any square on the board, and the queen's movement capacity outclasses that of the rook by including diagonal movement. For knights, Schwenk proved the existence of a closed *knight's tour* on modern chessboards [5]. A closed knight's tour is a sequence of moves of a knight on a chessboard in which the knight travels to each square only once – it is an instance of the Hamiltonian path problem. For this reason, we change the destination square Q to host the queenside variants of each (though the kingside variants would be equally reasonable).

3.3 Termination Semantics

Now that we have defined all the required state-transition semantics for moves, let us consider the termination behavior of moves. Specifically, we define termination behavior of moves as

$$\mathcal{T} : \mathbf{M} \to (\mathbf{St} \to \{\bot, \top\})$$

Traditionally, $\bot$ corresponds to non-termination whereas $\top$ corresponds to termination. However, in practice, we will consider $\bot$ to be nonzero exit-code type termination (error). This is simply because non-termination is not useful in practice, whereas error-based termination can easily signify an invalid PGN/AN program. However, we will henceforth refer to $\bot$ as non-termination for brevity.

To consider non-termination inducing behavior, we must take a closer look at the cases in which the state transitions we have defined cannot occur in legal chess gameplay. One important facet we have thus far not mentioned for all move types concerns the rule that moves which place the king in check are invalid. This will be a requirement for all move types, be they traversals, captures, castling or pawn promotion. To examine whether a king is in check, we can introduce another auxiliary function *IsInCheck*. The pseudocode is given in Algorithm 3.

Algorithm 3: IIC – IsInCheck

```
Data: state $\sigma'$, color of king to check (0 for white, 1 for black)
Result: 1 if the color has a king in check, 0 otherwise
set G to be the square that color king is on;
if unblocked ¬color's queen on same diagonal, file or rank then
    return 1;
if ¬color's king is 1 square away then
    return 1;
if unblocked ¬color's rook on same rank or file then
    return 1;
if unblocked ¬color's bishop on same diagonal then
    return 1;
if ¬color's knight 2-squares vertical and 1-square horizontal or 2-squares horizontal and 1-square vertical away then
    return 1;
if ¬color's pawn 1-square up-left or up-right if white, down-left or down-right if black) then
    return 1;
return 0;
```

This function simply takes a would-be state σ' which is the resulting state of some transition from the previous state σ, along with the necessary color king to check and checks whether the king is in check or not. The function could be written more cleverly to utilize the $GCSS$ function introduced earlier, but the full pseudocode is written for interpretability – the idea is that if a king can be captured by any of the opponent's pieces in the state σ', then it is in check and the move just played was invalid. *IsInCheck* will henceforth be abbreviated as IIC.

With this newly introduced function, we can now describe the termination behavior for each of the move types.

The termination behavior for traversals can be written as follows:

$$\mathcal{T}(Q) = \{(\sigma, \top) \mid \exists\sigma'.(\sigma,\sigma') \in [[Q]], \neg IIC(\sigma', mod(turn-1,2))\; Q \neq 0\} \cup \{(\sigma,\bot) \mid otherwise\}$$
$$\mathcal{T}(P{::}Q) = \{(\sigma, \top) \mid \exists\sigma'.(\sigma,\sigma') \in [[P{::}Q]], \neg IIC(\sigma', mod(turn-1,2)), Q \neq 0\} \cup \{(\sigma,\bot) \mid otherwise\}$$

Thus, a player's traversal move is legal and will terminate if the move played is playable according to the state-transition semantics (which govern whether movement constraints are followed for the given piece), the player's king is not in check after the move, and the destination square Q is not already occupied in the source state σ. If any of these conditions are not meant ("otherwise"), the traversal move is invalid and results in non-termination.

Next, we consider the termination behavior for captures. Given the innate similarity between traversal and capture moves semantically, we might expect their termination behavior to be similar. This is *mostly* true. The termination behavior for captures can be written as follows:

$$\begin{aligned}\mathcal{T}(P{::}x{::}Q) =&\Big\{(\sigma,\top) \mid \exists\sigma'.(\sigma,\sigma') \in [[P{::}x{::}Q]], \neg IIC(\sigma', mod(turn-1,2)), turn = 0, Q \in \{17\cdots 32\}\Big\}\\ &\cup \Big\{(\sigma,\top) \mid \exists\sigma'.(\sigma,\sigma') \in [[P{::}x{::}Q]], \neg IIC(\sigma', mod(turn-1,2)), turn = 1, Q \in \{1\cdots 16\}\Big\}\\ &\cup \Big\{(\sigma,\bot) \mid otherwise\Big\}\end{aligned}$$

Unlike for traversal moves, in capture moves, the destination square Q should *not* be empty (0). If the capture move is played by white, then the destination square should have a black piece, whereas if the capture move is played by black, then the destination square should have a white piece. Also providing that the movement constraints are followed and the king is not in check after the move is played, the capture is legal. If any of these conditions are not met, the capture move is invalid and results in non-termination.

Castling is a special case, as castling rules dictate that the king should not be in check when the move is played. Of course, the king should not be in check after the move is played either. We can write the termination behavior as follows.

$$\begin{aligned}\mathcal{T}(O\text{-}O) =&\{(\sigma,\top) \mid \exists\sigma'.(\sigma,\sigma') \in [[O\text{-}O]], \neg IIC(\sigma, mod(turn,2)), \neg IIC(\sigma', mod(turn-1,2))\}\\ &\cup \{(\sigma,\bot) \mid otherwise\}\end{aligned}$$

$$\begin{aligned}\mathcal{T}(\textit{O-O-O}) =&\{(\sigma,\top) \mid \exists\sigma'.(\sigma,\sigma') \in [[\textit{O-O-O}]], \neg IIC(\sigma, mod(turn,2)), \neg IIC(\sigma', mod(turn-1,2))\}\\ &\cup \{(\sigma,\bot) \mid otherwise\}\end{aligned}$$

When checking for legality of a castling move, we must check that the movement is allowable by the state transition semantics (are the intermediary squares empty, are the king and rook on appropriate squares, has the king stayed in place) and that the move does not start or end with the player's king in check. Thus, if *turn* indicates white's move in σ, we check that white's king is not in check in both σ as well as the would be σ'. The same can be said for black.

Lastly, we consider the termination behavior of pawn promotion. Since promotion state-transition semantics are mostly similar to traversal and capture state-transition semantics with the exception of the destination square retaining a pawn number, we might expect that the termination behavior is similar. In fact, it is almost the same – it is given below.

$$\mathcal{T}(\textit{Q::=::T}) = \{(\sigma,\top) \mid \exists\sigma'.(\sigma,\sigma') \in [[\textit{Q::=::T}]], \neg IIC(\sigma', mod(turn-1,2)), Q \neq 0\} \cup \{(\sigma,\bot) \mid otherwise\}$$

$$\begin{aligned}\mathcal{T}(\textit{W::x::Q::=::T}) =&\Big\{(\sigma,\top) \mid \exists\sigma'.(\sigma,\sigma') \in [[\textit{W::x::Q::=::T}]], \neg IIC(\sigma', mod(turn-1,2)), turn = 0, Q \in \{17\cdots 32\}\Big\}\\ &\cup \Big\{(\sigma,\top) \mid \exists\sigma'.(\sigma,\sigma') \in [[\textit{W::x::Q::=::T}]], \neg IIC(\sigma', mod(turn-1,2)), turn = 1, Q \in \{1\cdots 16\}\Big\}\\ &\cup \Big\{(\sigma,\bot) \mid otherwise\Big\}\end{aligned}$$

The major differences between normal traversals/captures and promotion traversals/captures are given by the state-transition semantics, which indicate a new piece number for the destination square. However, the termination semantics have equivalent conditions with respect to the emptiness of the destination square for traversal promotions and presence of an opponent's piece in the destination square for capture promotions. Of course, both are valid only if the player's king is not in check after the move is played.

4 Conclusion

In this work, we study the Portable Game Notation/Algebraic Notation (PGN/AN) specification used to record moves made in chess games. We are motivated by the desire to prohibit illegal/invalid chess gameplay, particularly at the higher levels of competition which influence tournament results, player rankings and sponsorships. We begin by introducing some basic tenets of gameplay (pieces, movement, positioning) and the special rules pertaining to these. Next, with the goal of establishing the precise rules needed to enable PGN/AN checking given a game recorded using this notation, we formalize its abstract syntax. Next, we present denotational semantics associated with this syntax and describe how we can go about checking the validity of a game by "re-playing" it move-by-move, with each move inducing a transition between states. Finally, we discuss termination behavior – specifically, we distinguish between cases of valid moves (which result in termination) and invalid moves (which result in non-termination, or error-induced program exit). Given these, one could write code to automatically verify games recorded in PGN/AN. Such a tool could be utilized in both offline (batch) or online (streaming) fashion to preserve integrity of chess games.

References

[1] `http://www.chessstrategiesblog.com/wp-content/uploads/algebraic_notation.gif`.

[2] `http://media.mahalo.com/images/d/d1/BeforeandAfter_ak_011708.jpg`.

[3] `http://upload.wikimedia.org/wikipedia/commons/a/ae/Ajedrez_captura_al_paso_del_peon.png`.

[4] D. Hooper and K. Whyld. *The Oxford companion to chess.* Oxford University Press Oxford, 1984.

[5] A. J. Schwenk. Which rectangular chessboards have a knight's tour? *Mathematics Magazine*, pages 325–332, 1991.

[6] G. Serper. The illegal move that wrecked a chess career. `http://www.chess.com/article/view/the-illegal-move-that-wrecked-a-chess-career`, Mar. 2015.

[7] I. Zaragatski. The 7 most illegal chess moves of all time. `https://chess24.com/en/read/news/the-7-most-illegal-chess-moves-of-all-time`, Aug. 2014.