# Project Summary

This project analyzes a chessboard using standard chess rules to determine whether a single King, controlled by player Black, is in checkmate, stalemate, or neither during player Black’s turn. Player White starts the game at the top of the board (so pawns capture by moving in a downwards diagonal) and controls any number of pieces, of any rank, in any configuration. The ONLY piece the player Black has on the board is their King.

This problem is translated into logic by encoding each possible move the king can make (8 moves, 1 in each direction), then encoding all the possible squares that a white piece is capable of capturing on their turn. If the black king is on a square a white piece is capable of capturing, it is in check. If all of the possible moves the black king can take will bring it to a space that white is able to capture, the black king has no moves and is thus either checkmate if currently in check, else stalemate.

# Propositions

Checkmate – Evaluates whether the Black King is in Checkmate, meaning it has no legal moves, and is currently in check. True if the Black King is checkmated.

Stalemate – Evaluates if the Black King is in Stalemate, meaning that it has no legal moves, however it is not currently in check. Stalemate additionally stipulates that the Black player has no other legal moves; in the model the only piece player Black controls is the King, meaning it only analyzes the King’s moves. True if the Black King is stalemated.

Check – Evaluates if the Black King is in Check, meaning that a White-controlled piece can move to its current location and capture it. True if the Black King is in check.

BK\_No\_Moves – Evaluates whether the Black King can make any legal moves. True if the Black King has no legal moves.

XX\_Space\_Occupied – Evaluates the presence of a given piece on a given space on the chessboard. “XX” is replaced by a code relating to each piece; “BK” represents the Black King, “WK” represents the White King, “WQ” represents the White Queen, “WR” the Rook, “WB” the Bishop, “WH” the Knight (WH representing “White Horse”, an alias of the Knight), and “WP” the White Pawn. These codes carry to other propositions as well. True if the space is occupied by piece XX.

XX\_Total\_Count – Uses propositional logic to evaluate the number of a given piece on the board. “XX” code carries from XX\_Space\_Occupied.

# 

# Constraints

¬(Checkmate)∨ ¬(Stalemate)

The Black King cannot both be in Checkmate and Stalemate; the game-ending conditions are mutually exclusive.

BK\_No\_Moves (Checkmate ∨ Stalemate)

If and only if the Black King has no legal moves, the game has come to an end, either in Checkmate or Stalemate.

BK\_No\_Moves ∧ Check <=> Checkmate

The Black King is in checkmate if and only if the Black King has no possible legal moves and the Black King is in Check.

BK\_No\_Moves ∧ ¬Check <=> Stalemate

The Black King is in stalemate if and only if the Black King has no possible legal moves and the

# Model Exploration

In an intermediate version of the model, we tried entering a real-life chessboard that was in checkmate into the model only to discover that the model believed this board-state to still be playable. After analyzing the real-life board state ourselves, we realized that the stalemate was a case of discovered mate: one of the moves the Black King could take allowed it to capture a threatening piece, only to be threatened itself by another of White’s pieces, making the move illegal. We therefore had to reconsider the code of our model to allow White’s pieces to project threat even to locations currently occupied by other allied pieces.

Once we had a working version of the model complete, one of the explorations we tried was setting the model to either be in Checkmate, Stalemate, or neither and then minimizing or maximizing the number of a singular given piece on the chessboard, in order to find out how many of a piece were the minimum to create a checkmate or stalemate condition, and the maximum of a piece that could exist on the board while avoiding both conditions. There were our results:

Queen – Minimum of 1 for Stalemate, 2 for Checkmate. 43 White Queens and 1 Black King can exist on the board without ending the game.

Rook – Minimum of 2 for Stalemate, 2 for Checkmate. 50 White Rooks can coexist with a Black King without ending the game.

Bishop – Minimum of 3 for Stalemate, 3 for Checkmate. 57 White Bishops can coexist with a Black King without ending the game.

Knight – Minimum of 2 for Stalemate, 3 for Checkmate. 61 White Knights can coexist with a Black King without ending the game.

Pawn – Minimum of 3 for Stalemate, 4 for Checkmate. 63 White Pawns (an entire board full!) can coexist with a Black King without ending the game.

King – Minimum of 2 for Stalemate, and no number of White Kings can checkmate a Black King. 58 White Kings can coexist with a Black King without ending the game, possibly for either side!

Another exploration we tried was to input an example board and have the program find out if the board was in check, checkmate or stalemate. This is done by using the potential moves of the white pieces to check the potential moves of the black king and to check if the king is in check. If the king has at least 1 legal potential move and is in check, then the king is simply in check. If there are no potential moves for the black king and the king is in check, then the king is in checkmate. If there are no potential moves for the black king and the king is not in check, then the king is in stalemate. Otherwise, there is no solution

Ex.

board = [

["BK",0,"WP",0,0,0,0,0],

[0,"WQ",0,0,0,0,0,0],

[0,0,0,0,0,0,0,0],

[0,0,0,0,0,0,0,0],

[0,0,0,0,0,0,0,0],

[0,0,0,0,0,0,0,0],

[0,0,0,0,0,0,0,0],

[0,0,0,0,0,0,0,0]]

Output:

This board has the king in check

Furthermore, the king is also in checkmate!

Finally, we’ve also used our model to find all the total solutions for a given board size. This will go through every possible board for the given board size and check if the board is in checkmate or stalemate. If you are going to try this out, it is highly recommended to start with a 3x3 board and go from there to make sure that your computer doesn’t crash considering the number of possible models. A good computer can maybe do a 5x5 board in less than an hour. It is interesting to know how many ways certain results can be accomplished with specific numbers of pieces on a chess board.

One example of exploring the model by counting solutions we performed was to see how many ways 2 rooks can put a king in stalemate on a 4x4 board, as well as a 5x5 board. On a 4x4 board we found there are 40 ways for this to happen, while on a 5x5 board there were 72.

# First-Order Extension

Using predicate logic would eliminate the need for the huge number of constraints implemented in our model; using first-order logic we could use a domain that constrains piece positions to the board (rather than manually checking to ensure pieces cannot leave the board, or be placed outside of it), n-ary function symbols to represent and verify the locations and types of pieces, and even logically quantify the threat locations of pieces rather than computing the threatened locations and then logically operating on that information.

For example, the function QueenThreat(x, y) is a binary function that maps a single x, y location (presumably that of a Queen on the coordinate plane that is the chessboard) onto a set of x, y coordinates, specifically all of those that are on the same x, y, x=y or x=-y axes that the Queen inhabits. A similar function, RookThreat(x,y) is a binary function that maps a single x,y location onto a set of x,y coordinates where a rook would be able to move. We would use such a function for all pieces. These logical functions would replace the enormously complex board-parsing system used in our model to where white is able to move, and thus where the black king cannot move.

Another example is BK\_No\_Moves. If we were to use a formula such as “∃x. BK\_Move(x) → ¬BK\_No\_Moves” where BK\_Move(x) denotes legal moves for the Black King, this would greatly simplify the constraint used to determine whether or not the Black King has legal moves from the unwieldy check against eight possible moves that the King could theoretically take and checking whether or not these squares are within White piece threat or whether or not they’re on the board.