# Proving Correctness of a KRK Chess Endgame Strategy by using Isabelle/HOL and Z3

Filip Marić[1], Predrag Janičić[1], Marko Maliković[2]

[1] Faculty of Mathematics, University of Belgrade, Serbia
[2] Faculty of Humanities and Social Sciences, University of Rijeka, Croatia

**Abstract.** We describe an executable specification and a total correctness proof of a King and Rook vs King (KRK) chess endgame strategy within the proof assistant Isabelle/HOL. This work builds upon a previous computer-assisted correctness analysis performed using the constraint solver URSA. The distinctive feature of the present machine verifiable formalization is that all central properties have been automatically proved by the SMT solver Z3 integrated into Isabelle/HOL, after being suitably expressed in linear integer arithmetic. This demonstrates that the synergy between the state-of-the-art automated and interactive theorem proving is mature enough so that very complex conjectures from various AI domains can be proved almost in a ,,push-button" manner, yet in a rich logical framework offered by the modern ITP systems.

## 1 Introduction

Chess has always been a target for developing new techniques and approaches of artificial intelligence. One field of chess-related research is concerned with chess endgames where challenges are different from those in openings and midgames. In computer chess playing, endgames are often played based on or analyzed with respect to pre-calculated lookup tables (i.e., endgame databases), containing optimal moves for each legal position. In contrast, *chess endgame strategies* do not necessarily ensure optimal play, but should provide concise, understandable, and intuitive instructions usable both to human and computer players. One of the simplest chess endgames is the *King and Rook vs King (KRK)*. There are several strategies for white for this endgame, generated by humans, semi-automatically, or automatically [10], but only a few of them are really human-understandable. Correctness of a strategy should be ensured – if a player follows the strategy, he should always reach the best possible outcome. Proofs of strategy correctness are typically not given or even not mentioned, although informal proofs are sometimes provided [4]. Proving correctness of chess endgame strategies can be addressed using different approaches [10]. The first approach is a traditional, *"pen-and-paper"* with the drawback of often having missing parts or errors in the arguments. *Computer assisted* proofs can be classified according to two independent dimensions: proofs can be either *indirect* or *direct*, and can be either *informal* or *formal*. *Indirect proofs* are based on enumerations and case-analyses.

For example, the strategy can be applied to all legal positions and a corresponding endgame-database can be generated, which is then verified using a retrograde procedure (in the style of Thompson's work [13]). *Direct proofs* are high-level, mathematical proofs that explicitly formulate properties of the strategy (preconditions, postconditions, invariants, termination measures), prove them and show that they imply the strategy correctness. *Informal proofs* use unverified programs (either developed in a general-purpose programming language or in some specialized constraint programming system) to check many different positions or to discharge informally stated proof-obligations that somehow contribute to the overall informal correctness arguments. *Formal proofs* are machine-verifiable proofs, checked within a strict logical system of a proof-assistant.

A SAT-based constraint solver URSA [9] has been used for checking key correctness properties for a KRK endgame strategy [10] in a direct, but informal manner. The strategy considered was a slight modification of the one originally formulated by Bratko [4]. Bratko's original paper also contains a very informal correctness proof sketch. The strategy was described within the constraint solving system and several high-level lemmas were formulated and automatically checked by using the power of the constraint solver. The main feature of those proofs is that they required very little human effort and human reasoning. On the other hand, as the authors noted, although the main body of the proof is covered by the checked lemmas, some building blocks were missing to make the proof complete and glued together, mainly due to the lack of expressibility of constraint solving systems (e.g., one cannot express inductive definitions or inductive arguments in a system such as URSA). Also, the proof relies on the definitions specific for the KRK endgame, so there is no link with the rules for the original game of chess. The final conclusion was that, in order to have a full and completely reliable proof, a constraint solver must be replaced by some more expressible reasoning system such as proof-assistants. We believe that modern proof-assistants (e.g., Isabelle/HOL, Coq, HOL-Light), connected to powerful external automated theorem provers and solvers (e.g., Z3, Vampire, Spass, E-prover) are now capable of proving extremely complex combinatorial conjectures, such as those coming from chess. For instance, Isabelle/HOL has been connected to SMT solvers [3], enabling users to employ SMT solvers to discharge complex goals that arise in interactive theorem proving. SMT solvers provide object-level proofs for unsatisfiable formulas and these proofs are then reconstructed within Isabelle/HOL, yielding formal proofs in the above sense.

In this paper, we describe our successful experience with formalizing the KRK endgame correctness proof within Isabelle/HOL. The present formalization is complete and self-contained, and it provides an executable version of the strategy that is proved to be correct (winning for white) with respect to the rules of chess. One of our key goals was that all central lemmas must be proved automatically, if possible — in a ,,push-button" manner, as it was done within the URSA system. This turned out to be possible, due to the powerful integration of SMT solvers (in particular — Microsoft Research z3 solver) into Isabelle/HOL. In the paper we briefly describe some interesting parts of the formalization. Full formalization is available at `http://argo.matf.bg.ac.rs/formalizations/`.

## 2 Chess Rules and Endgame Strategies

In this section we describe the formalization of chess rules and the general theory of chess endgame strategies in Isabelle/HOL.

*Chess Rules.* The first cornerstone of our formalization are the rules of chess, as given in the FIDE handbook [6]. Hurd has already formalized these rules in HOL [7], and we closely follow his work. Like Hurd, we consider pawnless endgames, and do not consider castling (although our strategy is only for the KRK endgame, we want our basic definitions to be close to general chess rules and to allow later extensions to other endgames, so initial definitions cover other pieces as well). Basic types are defined as follows.

```
side = White | Black
piece = King | Queen | Rook | Bishop | Knight
square = "int × int"
```

We can define many relevant notions using only arithmetic operations and relations over squares coordinates. We show only some examples. The function `on_board (f,r)` $\longleftrightarrow$ `0` $\leq f \wedge f$ `<` `F` $\wedge$ `0` $\leq r \wedge r$ `<` `R` checks if the square is *on the board* (global constants `F = 8` and `R = 8`, for files and ranks, determine the size of the board). We can check if a square $sq$ is *between* two given squares $sq_1$ and $sq_2$ either horizontally, vertically, or diagonally (this is denoted by `sq_btw` $sq_1$ $sq$ $sq_2$). We can define the *scope* of each piece (i.e., whether a piece can reach one square from another).

`king_scope` $(f_1,r_1)$ $(f_2,r_2)$ $\longleftrightarrow$ $|f_1 - f_2| \leq$ `1` $\wedge$ $|r_1 - r_2| \leq$ `1` $\wedge$ $(f_1 \neq f_2 \vee r_1 \neq r_2)$

`rook_scope` $(f_1,r_1)$ $(f_2,r_2)$ $\longleftrightarrow$ $(f_1 =$ `=` $f_2 \vee r_1 =$ `=` $r_2) \wedge (f_1 \neq f_2 \vee r_1 \neq r_2)$

For two squares of the chessboard, the *Manhattan distance* (`mdist` $(f_1, r_1)$ $(f_2, r_2)$ `=` $|f_1 - f_2|$ `+` $|r_1 - r_2|$) is the sum of distances along both coordinates, and the *Chebyshev distance* (`cdist` $(f_1, r_1)$ $(f_2, r_2)$ `=` `max` $|f_1 - f_2|$ $|r_1 - r_2|$) is the minimal number of moves a king requires to move between them.

*Chess positions* can be represented in various ways (e.g., by an 8x8 matrix implicitly mapping positions to pieces, or by a list of piece positions, implicitly mapping pieces to positions). So, instead of fixing a concrete representation, we create an abstraction in a form of an Isabelle/HOL locale [2] and assume that chessboard positions will be represented by some type `'p` (usually a record type). Only some values of the type `'p` will correspond to valid positions, so we introduce a data-structure invariant `pos_inv` $p$ that is used to exclude values that are invalid. For example, a type `'p` might be a mapping that maps pieces to squares that they are on. In that case, the invariant should require that all pieces map to different squares, since if two pieces are mapped to the same square, the position would clearly be invalid. For each position, we must be able to check whether white or black is on turn (this is done using the function `turn` $p$), and for each square to determine if there is a piece on that square and – if yes, what piece it is (this is done using the function `on_sq` $p$ $sq$ that maps each square $sq$ to either `None`, or to `Some` piece and its side).

```
locale Position =
  fixes pos_inv  :: "'p ⇒ bool"
  fixes turn  :: "'p ⇒ side"
  fixes on_sq :: "'p ⇒ square ⇒ (side × piece) option"
```

All chess rules can be defined within this locale, they are parametric, and depend on the type `'p` and the above three functions. For example, in a position $p$, for a square $sq$ we can check if it is *empty* (`empty` $p$ $sq$ $\longleftrightarrow$ `on_sq` $p$ $sq$ `= None`), or *occupied* by a piece of a side $sd$ (`occupies` $p$ $sd$ $sq$ $\longleftrightarrow$ (`∃` $pc$`.` `on_sq` $p$ $sq$ `= Some (`$sd$`,` $pc$`)`)). In a position $p$, a square $sq_1$ attacks $sq_2$ if the *line between them is clear* (`clr_line` $p$ $sq_1$ $sq_2$ $\longleftrightarrow$ (`∀` $sq$`.` `sq_btw` $sq_1$ $sq$ $sq_2$ $\longrightarrow$ `empty` $p$ $sq$))[1], and if there is a piece on $sq_1$ such that $sq_2$ is in its scope.

```
attacks p sq₁ sq₂ ⟷ clr_line p sq₁ sq₂ ∧
  (case on_sq p sq₁ of
     None ⇒ False
     | Some (_, King) ⇒ king_scope sq₁ sq₂
     | Some (_, Rook) ⇒ rook_scope sq₁ sq₂
     ...)
```

A side $sd$ is *in check* in a position $p$ if its king is on a square $sq_1$, and there is an opponent's piece on some square $sq_2$ such that it attacks the king on $sq_1$.

```
in_chk sd p ⟷ (∃ sq₁ sq₂. on_sq p sq₁ = Some (sd, King) ∧
                          occupies p (opp sd) sq₂ ∧ attacks p sq₂ sq₁)
```

A *position is legal* if its satisfies the invariant, if all pieces are within the board bounds, and if the opponent of the player on turn is not in check[2].

```
all_on_board p ⟷ (∀ sq. ¬ empty p sq ⟶ on_board sq)
lgl_pos p ⟷ pos_inv p ∧ all_on_board p ∧ ¬ in_chk p (opp (turn p))
```

*Legal moves* are defined by the chess rules and from legal positions they lead to legal positions. The function `lgl_move` $p$ $p'$ checks if the position $p'$ is a result of a legal move from the position $p$. Finally, we define game outcomes (*checkmate*, *stalemate*, and *draw*).

```
game_over p ⟷ lgl_pos p ∧ ¬ (∃ p'. lgl_move p p')
checkmate p ⟷ game_over p ∧ in_chk p (turn p)
stalemate p ⟷ game_over p ∧ ¬ in_chk p (turn p)
```

A game is drawn if the position is such that neither player can possibly mate. To formalize this, we inductively define the set of positions *reachable* from a given position $p_0$ by applying only legal moves.

```
p₀ ∈ reachable p₀
⟦p ∈ reachable p₀; lgl_move p p'⟧ ⟹ p' ∈ reachable p₀
draw p ⟷ ¬ (∃ p'. p' ∈ reachable p ∧ checkmate p')
```

[1] Since squares that a knight attacks are not on the same line with the square that it is on, the clear line condition is always satisfied.

[2] This definition is weaker then the one given by FIDE, as it does not take into account reachability from the initial position. Still, this does not threaten the correctness of our results, as we do cover all legal positions in the strong FIDE sense.

*Endgame Strategies.* The *strategy for white* is given by `st_wht_move` $p\ p'$ — a relation describing all positions $p'$ that can be reached from $p$ by a strategy move. A strategy is deterministic if there is always at most one such position. For each legal position with white on turn, a strategy returns only legal moves. Additionally, a strategy can be characterized by an invariant maintained throughout a play (e.g., in KRK endgame, white rook must not be captured, otherwise the game would be drawn). We define a slot for such invariant (`st_inv` $p$) and require that each move of white, and each move of black following a move of white maintains it.

```
locale Strategy = Position +
  fixes st_wht_move  :: "'p ⇒ 'p ⇒ bool"
  fixes st_inv  :: "'p ⇒ bool"
  assumes
  ⟦lgl_pos p; turn p = White; st_inv p; st_wht_move p p'⟧ ⟹ lgl_move p p'
  ⟦lgl_pos p; turn p = White; st_inv p; st_wht_move p p'⟧ ⟹ st_inv p'
  ⟦lgl_pos p; turn p = White; st_inv p; st_wht_move p p'; lgl_move p' p''⟧
    ⟹ st_inv p''
```

A *strategy play* is a sequence of alternating moves: strategy moves by white, and arbitrary legal moves by black. The set of reachable positions in a play is defined as an inductive set.

```
st_move p p' ⟷ (turn p = White ∧ st_wht_move p p') ∨
                (turn p = Black ∧ lgl_move p p')

p₀ ∈ st_reachable p₀
⟦p ∈ st_reachable p₀; st_move p p'⟧ ⟹ p' ∈ st_reachable p₀
```

A strategy for white is *winning* if it is terminating and partially correct, i.e., if every strategy play starting from a legal position $p_0$ with white on turn that satisfies the strategy invariant, terminates in a position where black is mated. If there is no infinite strategy play, there is no set $\mathcal{P}$ containing $p_0$ such that for each position in $\mathcal{P}$ a strategy move can be made.

```
st_start p₀ ⟷ turn p₀ = White ∧ lgl_pos p₀ ∧ st_inv p₀
locale WinningStrategy = Strategy + assumes
   st_start p₀ ⟹ ¬ (∃ 𝒫. p₀ ∈ 𝒫 ∧ (∀ p ∈ 𝒫. ∃p' ∈ 𝒫. st_move p p'))
   ⟦st_start p₀; p ∈ st_reachable p₀; ¬ (∃ p'. st_move p p')⟧ ⟹
    turn p = Black ∧ checkmate p
```

It can be proved that a strategy is winning for white if there is a well-founded ordering of subsequent white-on-turn positions in each strategy play, if white can always make a strategy move, and it never leads to a stalemate. Therefore, a strategy is winning for white if it meets assumptions of `WiningStrategyOrdering` (since it a sublocale of the `WiningStrategy`, i.e., if the assumptions of the former are satisfied, the assumptions of the latter are satisfied too).

```
locale WinningStrategyOrdering = Strategy +
   fixes ordering :: "'p ⇒ ('p × 'p) set"
   assumes
```

$[\![$st_start $p_0]\!] \Longrightarrow$ wf (ordering $p_0$)

$[\![$st_start $p_0$; $p \in$ st_reachable $p_0$; turn $p$ = White;
st_wht_move $p\ p'$; lgl_move $p'\ p''[\![ \Longrightarrow (p'', p) \in$ ordering $p_0$

$[\![$st_start $p_0$; $p \in$ st_reachable $p_0$; turn $p$ = White$]\!] \Longrightarrow \exists\ p'.$ st_wht_move $p\ p'$

$[\![$st_start $p_0$; $p \in$ st_reachable $p_0$; turn $p$ = White; st_wht_move $p\ p'[\![ \Longrightarrow$
$\neg$ stalemate $p'$

## 3 KRK Chess Endgame and Bratko-style Strategy

In this section we describe our formalization of KRK chess endgame. We give a very brief description of the specialization of chess rules for this case and of Bratko-style strategy for the KRK endgame (we denote it by BTK) [10].

*KRK chess endgame.* Although the KRK endgame follows the general chess rules introduced in the previous section, due to the specific nature of the game with just three pieces on the board, most notions can be characterized by much simpler conditions. Therefore, all general chess definitions are adapted to the KRK case and are reformulated through alternative definitions. Each such definition is proved to be just a specific instance of its corresponding general chess definition, and later used to simplify the correctness proofs. Since all following definitions are based on KRK-specific definitions used in the URSA specification [10], our work shows that the URSA specification follows from general chess rules.

Since there are only three pieces on the board, each position can be represented by the following simple data-structure.

```
record KRKPosition =
  WK ::"square" (* position of white king *)
  BK ::"square" (* position of black king *)
  WRopt ::"square option" (* position of white rook (None if captured) *)
  WhiteTurn ::"bool" (* Is white on turn? *)
```

Note that the option type is used only for the rook position, as kings must always be present on the board[3]. The following abbreviations are introduced.

BlackTurn $p \longleftrightarrow \neg$ WhiteTurn $p$,
WR $p$ = the(WRopt $p$), WRcapt $p \longleftrightarrow$ WRopt $p$ = None

The KRKPosition record interprets the Position locale, as all required components can be easily implemented.

KRK.pos_inv $p \longleftrightarrow$ WK $p \neq$ BK $p \wedge$ WRopt $p \neq$ Some(WK $p$) $\wedge$ WRopt $p \neq$ Some(BK $p$)

KRK.to_move $p$ = (if WhiteTurn $p$ then White else Black)

KRK.on_sq $p\ sq$ = (if WK $p$ = $sq$ then Some (White, King)
else if BK $p$ = $sq$ then Some (Black, King)
else if WRopt $p$ = Some $sq$ then Some (White, Rook)
else None)"

[3] This is only implicitly stated in the FIDE chess rules, as positions are defined to be legal only if they are reachable from the starting state where both kings are present, and kings cannot be captured. In our KRK formalization, the condition that both kings are present is implicitly imposed by the position representation.

Once the basic functions are interpreted, instances of all general definitions (e.g., legal positions, legal moves, stalemate, checkmate) for the KRK case are available. However, as we said, most of them are significantly simplified and reformulated, this time without quantifiers, so simpler reasoning methods can be used to reason about their properties. For example, requirement that all pieces are within the board bounds is defined in the following way (compare this with the original definition that uses the universal quantifier).

```
KRK.all_on_board p ⟷
  on_board (WK p) ∧ on_board (BK p) ∧ (¬ WRcapt p ⟶ on_board (WR p))
```

It is proved that this simplified `KRK.all_on_board` $p$ definition is equivalent to the original `all_on_board` $p$ definition instantiated by the `KRKPosition` type and its corresponding basic function definitions (**lemma** "`all_on_board` $p$ ⟷ `KRK.all_on_board` $p$"). Such proofs were not too hard, but Isabelle/HOL could not do them automatically (due to the rich language and the need of reasoning about arbitrarily quantified statements, the record type, tuples, etc).

The legality of positions can be reduced to requiring that all pieces are on different squares, that kings are not next to each other, and that if white is on turn, then the rook does not attack the black king (in KRK endgames, no diagonal lines but only horizontal and vertical lines need to be considered).

```
sq_btw_hv (f₁,r₁) (f, r) (f₂, r₂) ⟷
  (f₁ = f ∧ f = f₂ ∧ btw r₁ r r₂) ∨ (r₁ = r ∧ r = r₂ ∧ btw f₁ f f₂)
KRK.WR_attacks_BK p ⟷
  ¬ WRcapt p ∧ rook_scope (WR p) (BK p) ∧ ¬ sq_btw_hv (WR p) (WK p) (BK p)
KRK.kings_separated p ⟷ ¬ king_scope (WK p) (BK p)
KRK.lgl_pos p ⟷ KRK.pos_inv p ∧ KRK.all_on_board p ∧
  KRK.kings_separated p ∧ (WhiteTurn p ⟶ ¬ KRK.WR_attacks_BK p)"
```

Again, it is formally shown that this simplified definition of `KRK.lgl_pos` $p$ is equivalent to the original `lgl_pos` $p$ definition instantiated to the KRK case (**lemma** "`lgl_pos` $p$ ⟷ `KRK.lgl_pos` $p$").

Moves are defined as functions that modify the record representing the position. Move of the black king is the most complicated (as it can capture a rook).

```
KRK.moveBK p sq = (let p' = p (| BK := sq, WhiteTurn := True |)
                   in if WR p = sq then p' (| WRopt := None |) else p')
```

With these available, legal moves can be easily characterized. For example, a legal move of the black king can be characterized as follows.

```
KRK.BK_attacks_sq p sq ⟷ king_scope (BK p) sq
KRK.lgl_move_BK p₁ p₂ ⟷ KRK.lgl_pos p₁ ∧ BlackTurn p₁ ∧ KRK.lgl_pos p₂ ∧
  KRK.BK_attacks_sq p₁ (BK p₂) ∧ p₂ = KRK.moveBK p₁ (WK p₂)
```

Legal moves of two other pieces are characterized similarly. It is easily proved that all legal moves of black pieces are legal moves of the black king and all legal moves of white pieces are legal moves of either the white king or the white rook.

*Bratko-style KRK Strategy Definition.* Bratko's strategy can be outlined as follows. Try to *mate* in two moves. If that is not possible, then try to *squeeze* the room — the area to which the black king is confined by the white rook. Otherwise, try to *approach* the black king, to help the rook in squeezing (the approach is towards the *critical square* — a square adjacent to the rook in the direction of the black king). Otherwise, try to maintain the present achievements in the sense of squeeze and approach (i.e. make a waiting move). Otherwise, try to obtain a position such that the rook divides the two kings either vertically or horizontally. The strategy has a number of hidden details (its detailed description consumes more than a full page [10]) and that shows that it is very difficult to have a concise winning strategy (not to mention optimal strategy) even for a simple endgame such as KRK.

One of the central notions in the strategy is *room* (Figure 1). Following the strategy, white iteratively squeezes the black king and reduces the room, until black can be mated. The room space is always rectangular (e.g., of dimension $f \times r$). Originally, room was measured by its area. However, we noticed that instead of the area, half-perimeter $(f + r)$ can be used, which has equivalent key properties but it does not use multiplication and the arithmetic constraints remain linear. Critical square and room are formalized as follows.

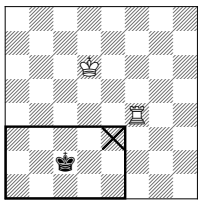

**Fig. 1.** Room and critical square

```
towards a b = (if a = b then a else if a > b then a - 1 else a + 1)
BTK.critical_sq p = (let (R^f, R^r) = WR p; (k^f, k^r) = BK p
                     in (towards R^f k^f, towards R^r k^r)
BTK.room p = (let (R^f,R^r) = WR p; (k^f,k^r) = BK p
              in (if R^f = k^f ∨ R^r = k^r then F + R - 1
                  else let f = if R^f > k^f then R^f else F - 1 - R^f;
                           r = if R^r > k^r then R^r else R - 1 - R^r
                       in f + r))
```

Note that when the black king and the white rook are in line, the black king is not confined, so the room takes the maximal value (`F + R - 1`).

After some initial moves, an invariant is established that the white rook is not exposed (its king can always approach and protect it, without having to move it) and that it divides two kings (either horizontally or vertically) or in some special cases that they form an L-shaped pattern (kings are in the same row (column), at distance 2, and the rook and the white king are in the same column (row) at distance 1). These notions are formalized as follows.

```
BTK.WR_exposed p ⟷
   (WhiteTurn p ∧ cdist (WK p) (WR p) > cdist (BK p) (WR p) + 1) ∨
   (BlackTurn p ∧ cdist (WK p) (WR p) > cdist (BK p) (WR p))
BTK.WR_divides p ⟷ (let (R^f,R^r) = WR p;(k^f,k^r) = BK p;(K^f,K^r) = WK p
   in (btw k^f R^f K^f ∨ btw k^r R^r K^r))
BTK.Lpattern p ⟷ (let (R^f,R^r) = WR p; (k^f,k^r) = BK p; (K^f,K^r) = WK p
   in (K^r = k^r ∧ |K^f − k^f| = 2 ∧ R^f = K^f ∧ |R^r − K^r| = 1) ∨
      (K^f = k^f ∧ |K^r − k^r| = 2 ∧ R^r = K^r ∧ |R^f − K^f| = 1)
```

The strategy uses several kinds of moves that are applied in a fixed order (if one kind of move is not applicable, then the next one is tried, and so on). For example, one kind of move is the `ImmediateMate` and it is applicable if white can mate in a single move. The `Squeeze` is applicable if white can reduce the room, while keeping the rook not exposed, dividing the two kings, and avoiding a stalemate position for black. These relations are formalized as follows.

```
BTK.immediate_mate_cond p ⟷ KRK.BK_cannot_move p ∧ KRK.WR_attacks_BK p

BTK.squeeze_cond p p' ⟷
   BTK.room p' < BTK.room p ∧ BTK.WR_divides p' ∧
   ¬ BTK.WR_exposed p' ∧ (KRK.BK_cannot_move p' ⟶ KRK.WR_attacks_BK p')
```

In order to apply some rule, its condition must hold but, in addition, no previous moves can be applicable, so their conditions must not hold for any legal move of white. This requires to universally quantify over all possible moves of white pieces. We introduce the function `kings_square (f, r) k` that for values $k$ between 1 and 8, gives coordinates of 8 squares that surround the given central square `(f, r)`. Similarly, the function `rooks_square (f, r) k` for values $k$ between 1 and `F + R` gives all squares that are in line with the rook (first horizontally, and then vertically). Then we introduce bounded quantification (that is unfolded in the proofs to stay within the quantifier-free fragment) and predicates that encode that a certain kind of move cannot be applied. We show this only for the `ImmediateMate`, as other moves follow a similar pattern.

```
all_n P n ⟷ ∀ i. 1 ≤ i ∧ i ≤ n ⟶ P i

no_mate_WK p ⟷ all_n 8 (λ k. let sq = kings_square (WK p) k in
   KRK.WK_can_move_to p sq ⟶ ¬ BTK.immediate_mate_cond (KRK.moveWK p sq))

no_mate_WR p ⟷ all_n (F+R) (λ k. let sq = rooks_square (WR p) k in
   KRK.WR_can_move_to p sq ⟶ ¬ BTK.immediate_mate_cond (KRK.moveWR p sq))

no_immediate_mate p ⟷ no_mate_WK p ∧ no_mate_WR p
```

Note that a mate cannot occur as a consequence of a white king's move.

Finally, we introduce the relation `BTK.st_wht_move` $p\ p'\ m$, encoding that a position $p'$ is reached from a position $p$ after a strategy move of a kind $m$.

```
MoveKind = ImmediateMate | ReadyToMate | Squeeze | ApproachDiag |
   ApproachNonDiag | KeepRoomDiag | KeepRoomNonDiag | RookHome | RookSafe

BTK.st_wht_move p p' m ⟷
(if m = ImmediateMate then
   KRK.lgl_move_WR p p' ∧ BTK.immediate_mate_cond p'
else
   no_immediate_mate p ∧
   if m = ReadyToMate then
      KRK.legal_move_white p p' ∧ BTK.ready_to_mate_cond p'
   else
      no_ready_to_mate p ∧
      ...
                        if m = RookSafe then
                           KRK.lgl_move_WR p p' ∧ BTK.rook_safe_cond p p'
                        else False)
```

*Executable Specification.* The relational specification `BTK.st_wht_move` $p$ $p'$ $m$ is not executable. In many cases, for a given position $p$ there are several possible values of $p'$ and $m$ that satisfy the previous relation. We defined a deterministic, executable function `(`$p'$`,` $m$`) = BTK.st_wht_move_fun` $p$ that returns a new position and a move type corresponding to the selected strategy move. In most cases this function iterates through all legal moves of white pieces (in some fixed order) until it finds a first move that satisfies the relational specification. Since the iteration order is fixed, the function will be deterministic, but in positions that allow several applicable moves, the choice is made rather arbitrarily (as the iteration order is chosen rather arbitrarily). An interesting exception is the squeeze move. To make the strategy more efficient, the optimal squeeze (the one that confines the black king the most) is always played (if there are several such moves, the first one found in the iterating process is used).

## 4 Correctness Proofs for Bratko-style Strategy

In this section we describe central correctness arguments for the strategy. All major proof steps were done automatically, by formulating the goals in LIA and applying the SMT solver.

*Linear Arithmetic Formulation.* The quantifier-free fragment of the theory of linear integer arithmetic (LIA) is very convenient for expressing our goals, so we formulated all our definitions in the language of LIA. This can be seen as an illustration how to prepare a problem (not only chess-related) for solving by automated solvers. Our definitions on this layer usually closely follow previously given definitions for the KRK case and Bratko's (BTK) strategy. However, in our LIA definitions, we never use quantifiers and don't use the record, product, nor the option type that were present on the KRK layer, but only the pure language of LIA. All KRK positions are represented in an unpacked form and functions receive six integers (usually denoted as $K^f$, $K^r$ for white king file and rank coordinates, $k^f$, $k^r$ for black king, and $R^f$, $R^r$ for white rook) instead of a record that collects them. Note that all the following definitions assume that the rook is present on the board, since they are applied only in such situations. Here are some examples.

`LIA.on_board` $sq^f$ $sq^r$ $\longleftrightarrow$ `0` $\leq$ $sq^f$ $\wedge$ $sq^f$ `<` `F` $\wedge$ `0` $\leq$ $sq^r$ $\wedge$ $sq^r$ `<` `R`

`LIA.all_on_board` $K^f$ $K^r$ $k^f$ $k^r$ $R^f$ $R^r$ $\longleftrightarrow$
  `LIA.on_board` $K^f$ $K^r$ $\wedge$ `LIA.on_board` $k^f$ $k^r$ $\wedge$ `LIA.on_board` $R^f$ $R^r$

`LIA.king_scope` $s_1^f$ $s_1^r$ $s_2^f$ $s_2^r$ $\longleftrightarrow$ $|s_1^f - s_2^f| \leq 1 \wedge |s_1^r - s_2^r| \leq 1 \wedge (s_1^f \neq s_2^f \vee s_1^r \neq s_2^r)$

`LIA.pos_inv` $K^f$ $K^r$ $k^f$ $k^r$ $R^f$ $R^r$ $\longleftrightarrow$
  `(`$K^f \neq k^f$ $\vee$ $K^r \neq k^r$`)` $\wedge$ `(`$R^f \neq K^f$ $\vee$ $R^r \neq K^r$`)` $\wedge$ `(`$R^f \neq k^f$ $\vee$ $R^r \neq k^r$`)`

It is shown that these definitions are equivalent to the KRK ones (under the assumption that the rook is not captured, and that coordinates of pieces are unpacked from the record). For example:

**lemma**
**assumes** `"WK` $p$ `= (`$K^f$`,`$K^r$`)" "BK` $p$ `= (`$k^f$`,`$k^r$`)" "WR` $p$ `= (`$R^f$`,`$R^r$`)" "¬ WRcapt` $p$`"`
**shows** `"KRK.pos_inv` $p$ $\longleftrightarrow$ `LIA.pos_inv` $K^f$ $K^r$ $k^f$ $k^r$ $R^f$ $R^r$`"`

All KRK definitions and all BTK strategy definitions have their LIA counterparts. The connection between them is quite obvious, so the proofs of equivalence are rather trivial and are proved by Isabelle's native automated tactics. In the following proofs, translation from HOL terms to LIA terms is done manually, but could be automated by implementing a suitable tactic.

*Central Theorems.* In this section we present our proof that BTK strategy is winning for white i.e., that it interprets the `WinningStrategyOrdering` locale. First, the strategy relation and the invariant are defined.

$\texttt{BTK.st\_wht\_move}\ p\ p' \longleftrightarrow (\exists\ m.\ \texttt{BTK.st\_wht\_move}\ p\ p'\ m)$

$\texttt{BTK.st\_inv}\ p \longleftrightarrow \neg\ \texttt{WRcapt}\ p$

Before giving proofs, we introduce some auxiliary notions. We are often going to consider full moves (a move of white that follows the strategy, followed by any legal move of the black king)[4].

$\texttt{BTK.st\_full\_move}\ p_1\ p_2 \longleftrightarrow \exists\ p_1'.\ \texttt{BTK.st\_wht\_move}\ p_1\ p_1' \wedge \texttt{KRK.lgl\_move\_BK}\ p_1'\ p_2$

For positions $p_1$ and $p_2$ and a set of strategy move types $M$, we define the following relations (white is allowed to make a move only if its kind is in $M$).

$\texttt{BTK.st\_wht\_move}\ M\ p\ p' \longleftrightarrow (\exists\ m \in M.\ \texttt{BTK.st\_wht\_move}\ p\ p'\ m)$

$\texttt{BTK.st\_full\_move}\ M\ p_1\ p_2 \longleftrightarrow \exists\ p_1'.\ \texttt{BTK.st\_wht\_move}\ M\ p_1\ p_1'\ \wedge\ \texttt{KRK.lgl\_move\_BK}\ p_1'\ p_2$

Next, we show that the $\texttt{BTK.st\_wht\_move}\ p\ p'$ defines a correct strategy, i.e., that it interprets the `Strategy` locale (that all moves are legal, and that the invariant is maintained throughout each strategy play). The following theorem guarantees that every move made by the strategy is legal.

**theorem assumes** "$\texttt{BTK.st\_wht\_move}\ p_1\ p_2$"
**shows** "$\texttt{KRK.lgl\_move\_WK}\ p_1\ p_2\ \vee\ \texttt{KRK.lgl\_move\_WR}\ p_1\ p_2$"

This is trivial to prove, since we explicitly require that a legal move is made in all cases of the definition of $\texttt{BTK.st\_wht\_move}\ p_1\ p_2\ m$.

It is obvious that a move of white preserves the invariant (white rook remains not captured). The following theorem guarantees that the invariant also remains preserved after any subsequent legal move of black.

**theorem assumes** "$\neg\ \texttt{WRcapt}\ p_1$" "$\texttt{BTK.st\_full\_move}\ p_1\ p_2$"
**shows** "$\neg\ \texttt{WRcapt}\ p_2$"

The proof goes as follows. White could not have played the `ImmediateMate` move, since black has made the move. In all other case, except the `ReadyToMate` move, the condition $\neg$ `BTK.WR_exposed p` is imposed, and it guarantees that the rook cannot be captured. The `ReadyToMate` case is the only non-trivial case and we encode the problem in LIA and employ SMT solvers to discharge the goal.

Next, we prove that the strategy is winning i.e., that it interprets the `Winning-StrategyOrdering` locale (that the strategy is always applicable, that it never leads into stalemate, and that there is a well founded ordering consistent with full strategy moves). The next theorem shows that play can always be continued i.e., that white can always make a strategy move.

[4] In the chess literature, half-move is sometimes called *ply*, and full-move *move*.

**theorem assumes** "WhiteTurn $p_1$" "¬ WRcapt $p_1$" "KRK.lgl_pos $p_1$"
**shows** "∃ $p_2$. BTK.st_wht_move $p_1$ $p_2$"

The proof is based on the following lemma, that guarantees that either Squeeze, RookHome, or RookSafe move are always applicable. The lemma is again proved automatically, by rewriting it into LIA and employing SMT solver.

**lemma assumes** "WhiteTurn $p$" "¬ WRcapt $p$" "KRK.lgl_pos $p$"
**shows** "¬ BTK.no_squeeze $p$ ∨ ¬ BTK.no_rook_safe $p$ ∨ ¬ BTK.no_rook_home $p$"

The following theorem shows that black is never in a stalemate.

**theorem assumes** "¬ WRcapt $p_1$" "BTK.st_wht_move $p_1$ $p_2$"
**shows** "¬ KRK.stalemate $p_2$"

This is also proved by analyzing different moves. After the ImmediateMate, black is mated and that is not a stalemate. All other moves, except ReadyToMate, by their definition require that stalemate did not occur, so they are trivial. The only complicated case is ReadyToMate, so we again use SMT solver to discharge it.

Finally, we prove termination. We show that the relation $R$ = {($p_2$, $p_1$). $p_1$ ∈ BTK.st_reachable $p_0$ ∧ BTK.st_full_move $p_1$ $p_2$} is well-founded, for a legal initial position $p_0$. If it would not be well-founded, then there would be a non-empty set with no minimal element i.e., there would be a non-empty set $Q$ such that a strategy play can always be extended by a strategy move of white, followed by a move of black:

∀ $p$ ∈ $Q$. $p$ ∈ BTK.st_reachable $p_0$ ∧ (∃ $p'$ ∈ $Q$. BTK.st_full_move $p$ $p'$)

Since in such infinite play, white must not make ImmediateMate and ReadyToMate move, as otherwise, the play would finish in a checkmate position, the following is implied ($\mathcal{M}$ denotes a set of two mate moves, and $\overline{\mathcal{M}}$ denotes its complement).

∀ $p$ ∈ $Q$. $p$ ∈ BTK.st_reachable $p_0$ ∧ (∃ $p'$ ∈ $Q$. BTK.st_full_move $\overline{\mathcal{M}}$ $p$ $p'$)

We show that this is a contradiction. The first observation is that the RookHome and RookSafe moves can be played only within the first three moves of a strategy play. This is proved by induction, using the following theorem that we proved automatically using LIA and SMT solvers.

**theorem assumes** "¬ WRcapt $p_1$"
"BTK.st_full_move $p_1$ $p_2$" "BTK.st_full_move $p_2$ $p_3$"
"BTK.st_full_move $p_3$ $p_4$" "BTK.st_wht_move $p_4$ $p'_4$ $m$"
**shows** "$m$ ≠ RookHome" "$m$ ≠ RookSafe"

Therefore, starting from some position $p'_0$ ∈ $Q$ (a position reached after three moves), all moves of white in our infinite strategy play are basic moves ($\mathcal{B}$ denotes the set of basic moves: Squeeze, Approach, or KeepRoom).

∀ $p$ ∈ BTK.st_reachable $p'_0$ ∩ $Q$. (∃ $p'$ ∈ $Q$. BTK.st_full_move $\mathcal{B}$ $p$ $p'$)

Next we proved that there is a position $p_m$ that satisfies the following condition.

$p_m$ ∈ BTK.st_reachable $p'_0$ ∩ $Q$ ∧ BTK.room $p_m$ ≤ 3 ∧ ¬ BTK.WR_exposed $p_m$

To prove this, we use the following lemma, stating that when started from a situation where the rook is exposed or the room is grater than 3, after three strategy basic moves, the rook is not exposed anymore and either the room decreased, or it stayed the same, but the Manhattan distance to the critical square decreased. Again, this lemma is proved automatically, by expressing it in language of LIA and employing SMT solver.

**theorem assumes** "¬ WRcapt $p_1$" "BTK.WR_exposed $p_1$ ∨ BTK.room $p_1$ > 3"
"BTK.st_full_move $\mathcal{B}$ $p_1$ $p_2$" "BTK.st_full_move $\mathcal{B}$ $p_2$ $p_3$"
"BTK.st_full_move $\mathcal{B}$ $p_3$ $p_4$"
**shows** "(BTK.WR_exposed $p_4$, BTK.room $p_4$, BTK.mdist_cs $p_4$) <
(BTK.WR_exposed $p_1$, BTK.room $p_1$, BTK.mdist_cs $p_1$)"

Note that the last conclusion is expressed as lexicographic comparison between the ordered triples that contain a bool (BTK.WR_exposed $p$) and two integers (BTK.room $p$ and BTK.mdist_cs $p$). The Boolean value False is considered less than True, and integers are ordered in the standard way. Since these integers are non-negative for all legal positions, this lexicographic ordering is well-founded. Then we consider the following relation.

R' = {($p_2$, $p_1$). ¬ WRcapt $p_1$ ∧ (BTK.WR_exposed $p_1$ ∨ BTK.room $p_1$ > 3) ∧
BTK.st_full_move $\mathcal{B}$ $p_1$ $p_2$}

By the previous theorem, the third power of this relation is a subset of the lexicographic ordering of triples that was well-founded, so the relation $R'$ itself is well-founded. Then, every non-empty set has a minimal element in this relation, so there is an element $p_m$ such that the following holds.

$p_m$ ∈ BTK.st_reachable $p'_0$ ∩ $Q$
∀ $p$. ($p$, $p_m$) ∈ $R'$ ⟶ $p$ ∉ BTK.st_reachable $p'_0$ ∩ $Q$

As we already noted, the play from BTK.st_reachable $p'_0$ ∩ $Q$ can always be extended by a strategy basic move, followed by a move of the black king, i.e., there is a position $p'_m \in Q$ such that BTK.st_full_move $\mathcal{B}$ $p_m$ $p'_m$. Then, $p'_m$ would also be in BTK.st_reachable $p'_0$ ∩ $Q$, so $(p'_m, p_m) \notin R'$. Since it holds that ¬ WRcapt $p_m$ and BTK.st_full_move $\mathcal{B}$ $p_m$ $p'_m$, it must not hold that BTK.WR_exposed $p_1$ ∨ BTK.room $p_1$ > 3, so $p_m$ is the required position, satisfying ¬ BTK.WR_exposed $p_m$ ∧ BTK.room $p_m$ ≤ 3. From such position, the fifth move by white will be either a ImmediateMate or a ReadyToMate. This holds by the following theorem (again, proved automatically, using LIA and SMT solver).

**theorem assumes** "¬ WRcapt $p_0$" "BTK.room $p_0$ ≤ 3" "¬ BTK.WR_exposed $p_0$"
"BTK.st_full_move $\mathcal{B}$ $p_0$ $p_1$" "BTK.st_full_move $\mathcal{B}$ $p_1$ $p_2$"
"BTK.st_full_move $\mathcal{B}$ $p_2$ $p_3$" "BTK.st_full_move $\mathcal{B}$ $p_3$ $p_4$"
"BTK.st_wht_move $p_4$ $p'_4$ $m$"
**shows** "$m \in \mathcal{M}$"

Since $p_m$ ∈ BTK.st_reachable $p'_0$ ∩ $Q$, white is on turn and the play can be infinitely extended by basic moves. However, by the previous theorem, the fifth move of white must be a mating move, giving the final contradiction.

## 5 Related Work

The presented formalization is, as said, closely related to the one based on constraint solving [10]. Still, the present work is a step forward, since it includes a formal development of relevant chess rules within the proof assistant and all proofs are trustworthy in a stronger sense. Not only that this work glues together conjectures checked earlier by the constraint solver, it also revealed some minor deficiencies (e.g., imprecise definition of legal moves) in the earlier formalization.

Although properties of two-player board games are typically explored using brute-force analyses, other approaches exist, similar to the constraint solving based one. For instance, binary decision diagrams were applied for game-theoretical analysis of a number of games [5]. Again, this approach cannot provide results that can be considered trustworthy in the sense of proof assistants. There is only a limited literature on using interactive theorem proving for analyzing two-player board games. A retrograde chess analysis has been done within Coq, but it does not consider chess strategies [11]. Hurd and Haworth constructed large, formally verified endgame databases, within the HOL system [8]. Their work is focused on endgame tables and it is extremely difficult (if not impossible) to extract some concise strategy descriptions and high-level insights from these tables, so we addressed a different problem and in a different way.

Before using Isabelle/SMT, we formalized the  strategy within Coq [12]. However, neither Omega, a built-in solver for quantifier-free formulae in LIA, nor a far more efficient Micromega and corresponding tactics for solving goals over ordered rings (including LIA), were efficient enough. SMTCoq [1] is an external Coq tool that checks proof witnesses coming from external SAT and SMT solvers (zChaff and veriT). Coq implements constructive logic, while veriT reasons classically. SMTCoq was designed to work with type `bool` (which is decidable in Coq) but not with type `Prop` which is natural type for propositions in Coq. Construction of complex theories over type `bool` in Coq can be quite inconvenient and has many pitfalls. There are plans for further improvement of SMTCoq.

## 6 Conclusions and Further Work

In the presented work, chess — a typical AI domain — has been used as an illustration for showing that the state-of-the-art theorem proving technology has reached the stage when very complex combinatorial conjectures can be proved in a trusted way, with only a small human effort. Our key point is that this is possible thanks to synergy of very expressible interactive provers and very powerful automated provers (SMT solvers in our case). The considered conjectures push the provers up to the limits[5] and while Isabelle/SMT interface can be further improved (e.g., proof checking time could be reduced), our experience with it

[5] The largest SMT formula in the proof has more than 67,000 atoms. Proofs were checked in around 8 CPU minutes on a multiprocessor 1.9GHz machine with 2GB RAM per CPU when SMT solvers are used in the oracle mode and when SMT proof reconstruction was not performed. SMT proof reconstruction is the slowest part of

can be seen as a success story. Our second point is that the presented work can be seen as an exercise not only in automation, but also in suitable formalization of non-trivial combinatorial problems. Namely, computer theorem provers are powerful tools in constructing and checking proofs, but they only work modulo the given definitions. The only way to check definitions is by human inspection, and one must be extremely careful when doing this step. Reducing everything to a small set of basic definitions (as we reduced specific KRK definitions to the basic chess rules) is an important step in ensuring soundness.

For future work, we are planning to analyse different generalizations of the presented central theorem. For example, unlike approaches based on SAT or endgame tables, our approach is not enumerative in its nature and can be used for arbitrary board sizes. We will also use a similar approach for proving other related conjectures in chess and other two-player intellectual games.

*Acknowledgments.* The authors are grateful to Sascha Böhme and Jasmin Christian Blanchette for their assistance in using SMT solvers from Isabelle/HOL and to Chantal Keller for her assistance in using SMT solvers from Coq.

## References

1. M. Armand, G. Faure, B. Grégoire, C. Keller, L. Théry, and B. Werner. A Modular Integration of SAT/SMT Solvers to Coq through Proof Witnesses. In *Certified Programs and Proofs*, volume 7086 of *LNCS*. Springer, 2011.
2. C. Ballarin. Interpretation of locales in isabelle: Theories and proof contexts. In *MKM*, 2006.
3. S. Böhme and T. Weber. Fast LCF-style proof reconstruction for Z3. In *Interactive Theorem Proving*, volume 6172 of *LNCS*. Springer, 2010.
4. I. Bratko. Proving correctness of strategies in the AL1 assertional language. *Information Processing Letters*, 7(5), 1978.
5. S. Edelkamp. Symbolic Exploration in Two-Player Games: Preliminary Results. In *AIPS-02 Workshop on Planning via Model-Checking*, 2002.
6. FIDE. The FIDE Handbook, chapter E.I. The Laws of Chess, 2004. Available for download from the FIDE website.
7. J. Hurd. Formal verification of chess endgame databases. In *Theorem Proving in Higher Order Logics: Emerging Trends*, Oxford University CLR Report, 2005.
8. J. Hurd and G. Haworth. Data assurance in opaque computations. In *Advances in Computer Games*, volume 6048 of *LNCS*. Springer, 2010.
9. P. Janičić. URSA: A System for Uniform Reduction to SAT. *Logical Methods in Computer Science*, 8(3:30), 2012.
10. M. Maliković and P. Janičić. Proving Correctness of a KRK Chess Endgame Strategy by SAT-Based Constraint Solving. *ICGA Journal*, 36(2), 2013.
11. M. Maliković and M. Čubrilo. What were the last moves? *International Review on Computers and Software*, 5(1), 2010.
12. M. Maliković, M. Čubrilo, and P. Janičić. Formalization of a Strategy for the KRK Chess Endgame. In *Conference on Information and Intelligent Systems*, 2012.
13. K. Thompson. Retrograde analysis of certain endgames. *ICCA Journal*, 9(3), 1986.

---

proof-checking, but it can be done in a quite reasonable time of 29 CPU minutes. The whole formalization has around 12,000 lines of Isabelle/Isar code.