PROVING CORRECTNESS OF A KRK CHESS ENDGAME STRATEGY BY SAT-BASED CONSTRAINT SOLVING

Marko Maliković*
Faculty of Humanities and Social Sciences, University of Rijeka
Slavka Krautzeka BB, 51000 Rijeka, Croatia

Predrag Janičić†
Faculty of Mathematics, University of Belgrade
Studentski trg 16, 11000 Belgrade, Serbia

September 9, 2013

Abstract

Chess endgame strategies in a concise and intuitive way describe the rules the player should follow to ensure win (or draw). Endgame strategies are useful for both computer and human players. Their correctness can be proved in several ways, and in this paper we present one of them: computer assisted proof based on reduction to propositional logic, more precisely to SAT. We focus on a strategy for the KRK endgame and reduction to SAT is performed by using a constraint solving system URSA. The relevant lemmas produced SAT instances with hundreds of thousands variables and clauses, but URSA still successfully handled them. As we are aware of, this is the first computer-assisted high-level proof of a correctness of a strategy for some chess endgame. The presented methodology can be applied to other endgames and other games as well. Therefore, the point of this paper is not only presenting a proof of correctness of an endgame strategy, but also presenting a new methodology for computer-assisted reasoning about chess problems.

1 Introduction

Playing endgames in games like chess poses serious challenges both to human and computer players. Namely, even chess endgames with only a very few pieces and with a winning position for white, can have a guaranteed mate only in tens of moves. As a result of this, techniques used by computer programs in midgames are not appropriate and other techniques must be used. One technique is based on lookup tables (i.e., endgame databases) with pre-calculated optimal moves for each legal position. However, such tables require a lot of memory and, in addition, they are completely useless for human players. One alternative to huge lookup tables, usable both to human and computer players, are *endgame strategies*. Endgame strategies provide concise, understandable, and intuitive instructions for the player. For automatically deriving knowledge or strategies for chess endgames, there have been attempts based on inductive logic programming [1, 28, 29], machine learning techniques [11, 14, 32, 34], genetic algorithms [18, 22], neural networks [15], etc. There are also approaches for automatically constructing evaluation functions for specific endgames [33]. However, fully automated approaches still cannot produce endgame strategies that are as understandable as human derived strategies (such as Bratko's strategy for KRK endgame [6, 26, 8]).

Endgame strategies do not need to ensure optimal moves (e.g., shortest path winning moves), but must ensure correctness – i.e., if player A follows the instructions of the strategy, he should always reach the best

*email: marko.malikovic@ffri.hr
†email: janicic@matf.bg.ac.rs

possible outcome. Thus, in the case of a winning strategy for player A, correctness of the strategy includes *termination* (i.e. the game ends if A follows the strategy) and *completeness* (i.e. if the game ends, then A wins). There are several approaches for proving correctness of chess endgame strategies:

1. **Computer-assisted informal indirect proof:** Within this approach, using an arbitrary programming language, the strategy is applied to all legal positions and the corresponding endgame database is generated. Then, using a retrograde procedure (in the style of Thompson's work [38]), it is verified that the endgame database is correct (e.g. that it always leads to checkmate). A variant of this approach was used by Bramer [4, 5] for testing correctness of some endgame strategies, but also for refining strategies that turned out not to be correct. So, in this approach, instead of a direct proof, correctness is based on a form of exhaustive analysis. The advantage of this approach is that it is quite straightforward. Its drawback is that it does not provide a high-level, understandable and intuitive, argument on *why* the strategy really works (in analogy: in finite-domain problems in mathematics, high-level, intuitive proofs and explanations are preferred to arguments based on exhaustive analysis). In addition, there may be errors in the implementation of the strategy or the verification procedure.

2. **Computer-assisted formal indirect proof:** Within this approach, using a proof-assistant[1] (instead of a general purpose programming language), the strategy can be applied to all legal positions and the corresponding endgame database can be generated. Then it can be proved that the strategy always leads to checkmate, but thanks to the proof-assistant, the argument is machine verifiable and trusted and can rely only on a small core describing the rules needed for the specific endgame. Hurd and Haworth constructed endgame databases that are formally (within the HOL system) verified to logically follow from the laws of chess [17]. Their framework could be used also to (formally) verify outputs of a certain endgame strategy. The advantage of these approaches is that they provide trusted arguments, but on the other hand, they also do not provide a high-level correctness proof of the strategy considered.

3. **Informal direct proof:** Within this approach, the strategy is analysed in a traditional mathematical manner, and the proof is given in a "pen-and-paper" manner. One example of such a proof is Bratko's proof of correctness of his KRK strategy [6]. The benefit of this approach is that the proof is high-level and understandable and provides real insights into why the strategy works. On the other hand, the drawback of such proofs is that there can be missing parts or errors in the arguments.

4. **Computer-assisted formal direct proof:** Within this approach, the strategy is analysed in a traditional mathematical manner, but this time in the rigorous environment of a proof assistant. The advantage of this approach is that it provides both formal, machine verifiable and high-level proof. The problem with this approach is that the automation available within proof assistants may still not be efficient enough to deal with very complex conjectures obtained from correctness conditions.

5. **Computer-assisted informal direct proof:** Within this approach, correctness of the strategy relies on several conjectures that are tested either by using a general-purpose programming language or by using a constraint programming system[2] (where the strategy and the conjectures are represented

[1]Interactive theorem proving systems or *proof assistants* are systems used to check (and guide) proofs constructed by the user, by verifying each proof step with respect to the given underlying logic. Proofs constructed within proof assistants are verbatim and detailed, and typically much longer than "traditional proofs" [2]. Actually, "traditional proofs" most often are not proofs at all, because of the many missing fragments, informal arguments, etc. Using proof assistants uncovered many flaws in many published mathematical proofs, published in both books and journals. Proof assistants are used for formalizing both classical and emerging mathematical knowledge. Some of the most popular modern proof assistants are Isabelle, Coq, HOL Light, PVS, Mizar, ACL2, etc [39].

[2]Constraint programming systems are used for specifying problems in their corresponding modelling languages and solving them with various techniques. Some of the dominant approaches are constraint logic programming over finite domains [19], answer set programming [13], and disjunctive logic programming [23]. There are hybrid systems that use custom specification languages and provide support for constraint programming (e.g., IBM ILOG OPL, COMET [25], G12 [36]). Also, there are libraries for constraint programming and combinatorial optimization for general purpose programming languages: Ilog for C++, Numberjack for Python, etc. Programs in specification languages describe a problem at a high-level, descriptive way and the specification

within a certain theory, e.g., propositional logic, linear arithmetic, etc). The latter variant is more beneficial and reliable, due to a more readable problem representation and due to the fact that most of the reasoning process is transferred to a dedicated solver for a certain theory (so risk of flaws is less). Still, the obtained arguments are not machine verifiable proofs and a theory that glues together all the conjectures into a single theorem cannot be built (in contrast to proofs produced with proof assistants). Also, the level of reliability is lower than in the approach with proof assistants, since there could be bugs in the mechanism for translating conjectures to the underlying theory. Errors in the specification are also possible, but this is also the case in the approach with proof assistants.

In this paper, we focus on proving correctness of a strategy for the KRK endgame (King and Rook vs. King). There are several strategies for white for this endgame, generated by humans, or semi-automatically, or automatically, using endgame databases or certain sets of human advices and using approaches such as inductive logic programming, genetic programming, neural networks, machine learning, etc, as listed above in the general context of chess endgames. However, only a few of them are really human-understandable. Although the consensus over the need of strategic play in computer programs for chess endgames existed before, one of the very first implementations of strategies for the KRK endgames was written in ALGOL 60 by Zuidema [41]. Zuidema didn't use guided search methods, but high-level advices. Correctness of the strategy was not proven. Seidel constructed a simple strategy that uses the ring structure (with four rings) of the board [35]. The strategy is based on the following: the black king can be forced to leave the ring $i + 1$ if and only if there exists a mate or stalemate pattern on the (inner) ring i. Morales used human assistance and inductive logic programming to produce short strategies for KRK [28, 29], but didn't prove their correctness. The strategy for KRK analysed in this paper is based on Bratko's advice-based strategy [6, 26, 8, 7], discussed in the next section.

For proving correctness of a strategy for the KRK chess endgame, we used the first and the fifth approach (from the five approaches listed above). In this paper we will focus on the fifth approach, since it is more demanding but it is also more beneficial and offers more. We proved correctness of the strategy by translating all the needed lemmas into propositional logic, i.e., to instances of the SAT problem.[3] To specify the required chess rules, the strategy and all relevant lemmas, we used a constraint programming system URSA [21]. URSA provides a high-level specification language, translates the constraints to SAT, invokes an underlying SAT solver, and returns the answer — *no*, if the constraint is not satisfiable, and otherwise *yes*, along with a model.

We are not aware of other specifications of chess strategies within a proof assistant or a constraint programming system. There is a work on retrograde chess analysis within Coq, but it does not consider chess strategies [24]. There is the aforementioned work on verifying chess endgame databases [17], but it does not deal with strategies understandable to humans.

Of course, we don't think that it is questionable whether the KRK endgame is a win for white (because there are so many supporting arguments), nor even that it is questionable (before a computer-assisted proof) whether the strategy for KRK discussed in this paper is really correct. Instead, the purpose of this work is to present a new approach for dealing with complex chess conjectures and the strategy for KRK is used primarily as a case study. This new approach is a general reliable methodology that can be applied to a range of problems not only in chess, but also in other games with perfect information.

does not say how the problem is to be solved. Some constraint systems are based on SAT, including SUGAR [37], FznTini [16], MXG [27, 31], URSA [21].

[3]SAT is the problem of deciding if a given propositional formula in CNF (conjunctive normal form) is satisfiable, i.e., if there is any assignment to variables such that all clauses are true. SAT was the first problem shown to be NP-complete [9], and it still holds a central position in the field of computational complexity. In recent years, tremendous advances, including both high-level and low-level algorithmic techniques, have been made in SAT solving technology [30, 10, 40, 3]. These advances in SAT solving make possible to decide the satisfiability of some industrial SAT problems with hundreds of thousands of variables and millions of clauses.

2 Strategy for White for the KRK Endgame

We assume standard chess notions such as legal moves, mate, stalemate, etc. By a *legal KRK position* we mean a legal position with three pieces: the white king, the white rook, and the black king. There are 399112 such positions, while the strategy is applied only to 175168 of them — those with white on turn (and the black king is not checked). Some of these positions might not be reachable from the starting chess position, but we don't consider this (very challenging) problem. Instead, we consider all 175168 positions with white on turn (and we will prove that the strategy leads to checkmate, starting from any of these positions — which is stronger than proving it only for positions reachable from the starting chess positions). The KRK endgame is drawn as soon the white rook is captured and this immediately ends the game (since neither player can checkmate the opponent's king with any series of legal moves, by Article E.I.01A.5.2 of FIDE Handbook[4]).

The strategy for the KRK endgame analysed in this paper is a winning strategy for white. It is based on Bratko's advice-based strategy that can be outlined as follows [6, 26, 8, 7]:[5]

Mate: Look for a way to mate black in two moves;

Squeeze: If the above is not possible, then look for a way to further constrain the *room* — the area on the chessboard to which the black king is confined by the white rook;

Approach: If the above is not possible, then look for a way to move the king closer to the black king (to help the rook in squeezing the black king);

KeepRoom: If none of the above pieces of advice works, then look for a way of maintaining the present achievements in the sense of **Squeeze** and **Approach** (i.e. make a waiting move);

Divide: If none of the above pieces of advice works, then look for a way of obtaining a position in which the rook divides the two kings either vertically or horizontally.

The strategy has a number of hidden details 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.

We made several significant modifications to Bratko's strategy (in the following text, we assume that columns and rows of the chessboard are associated with numbers $0, 1, \ldots, 7$ — see the discussion in Section 3.1):

- We eliminated *search* implicitly involved in the steps **Mate** and **Divide**. Indeed, these steps require exploring the possible moves of white and black several steps ahead. Search can be problematic for a human who tries to learn and follow the strategy. Also, search can be problematic for computer representation in logical terms, because search corresponds to the alternation of quantifiers ("find a move such that there is a move such that for each move ...") and requires more involved reasoning. Thus, in our strategy there is no search and the next move is computed looking ahead no further than one ply. Instead of Bratko's step **Mate**, in our strategy there are two steps: **ImmediateMate** and **ReadyToMate**. In order to avoid the step **Divide**, Jakulin introduced attributes **RookHome** and **RookSafe** [20]. We modified these two attributes, and on the basis of them, we introduced two new steps in the strategy (steps that replace the search-based **Divide** step).

- In our version of the step **Squeeze**, if there are more than one squeezing step, the one that reduces the *room* maximally is chosen (in Bratko's strategy this issue is not addressed).

- We changed the notion of *room* (the space of the chessboard where the black king is guarded by the white rook), extensively used in Bratko's strategy. This space is rectangular and its area equals $m \cdot n$, where m and n are lengths of its sides (Figure 1). In one of the strategy's steps, the rook has to move

[4] `http://www.fide.com/fide/handbook.html`

[5] Although white can theoretically castle in some KRK positions, castling is not considered as a legal move.

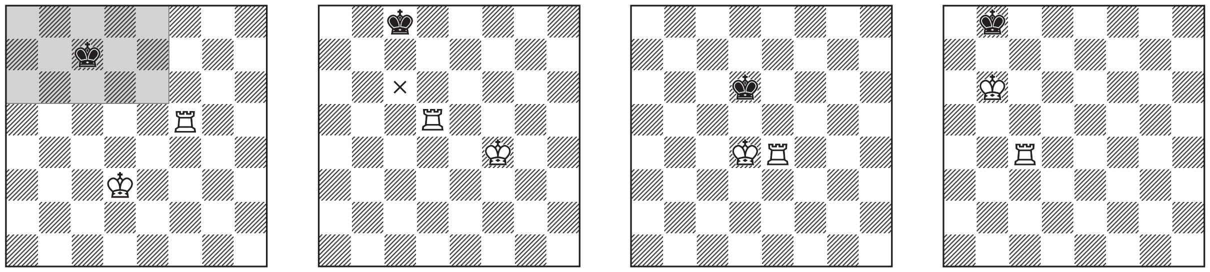

Figure 1: From left to right: illustration of the notion of *room*, of the notion of *critical square*, of the notion of *L-pattern*, and of the *ready to mate* position

in such a way that this area decreases. However, a simple insight enables simplifying this condition: if the rook moves, only one of m and n changes, and since the formula (implicitly universally quantified):

$$x = z \lor y = u \Rightarrow (x \cdot y < z \cdot u \Leftrightarrow x + y < z + u)$$

is valid (over natural numbers), it is sufficient, in this case, to represent this area by $m+n$, not by $m \cdot n$. This change in the notion of *room* simplifies relevant constraints and reasoning about the strategy.

- In the steps **Approach** and **KeepRoom**, a diagonal move of the white king is preferred to the non-diagonal move and this is necessary for correctness of the strategy. Because of this, these steps are divided into two steps each (in Bratko's strategy these steps are also present but as substeps of **Approach** and **KeepRoom**).

In our formulation of the strategy, the squares of the chessboard are represented as (x, y) pairs of natural numbers. For formulating the strategy, we use several notions (standard notions or notions introduced by Bratko):

Manhattan distance: For two squares of the chessboard (x_1, y_1) and (x_2, y_2), the Manhattan distance equals $|x_1 - x_2| + |y_1 - y_2|$.

Chebyshev distance: For two squares of the chessboard (x_1, y_1) and (x_2, y_2), the Chebyshev distance is the minimal number of moves a king requires to move between them, i.e., $\max(|x_1 - x_2|, |y_1 - y_2|)$.

Room: If the rook is on the square (x_R, y_R) and the black king on (x_k, y_k), then the *room* equals:

$$\begin{cases} 15, & \text{if } x_R = x_k \text{ or } y_R = y_k \\ x_R + y_R, & \text{if } x_R > x_k \text{ and } y_R > y_k \\ x_R + 7 - y_R, & \text{if } x_R > x_k \text{ and } y_R < y_k \\ 7 - x_R + y_R, & \text{if } x_R < x_k \text{ and } y_R > y_k \\ 7 - x_R + 7 - y_R, & \text{if } x_R < x_k \text{ and } y_R < y_k \end{cases}$$

Critical square: The *critical square* is the square adjacent to the square of the rook in the direction of the black king; if the rook and the black king are in the same column or the same row, then the critical square is between them, otherwise, it is diagonal to the square of the rook (one example of *critical square* is shown in Figure 1). More precisely, if the rook is on the square (x_R, y_R) and the black king on (x_k, y_k), then the coordinates (x, y) of the critical square are given by the following equalities:

$$x = \begin{cases} x_R, & \text{if } x_R = x_k \\ x_R - 1, & \text{if } x_R > x_k \\ x_R + 1, & \text{if } x_R < x_k \end{cases} \qquad y = \begin{cases} y_R, & \text{if } y_R = y_k \\ y_R - 1, & \text{if } y_R > y_k \\ y_R + 1, & \text{if } y_R < y_k \end{cases}$$

Rook exposed: The rook is *exposed* if white is on turn, and the Chebyshev distance between the rook and the white king is greater by at least 2 than the Chebyshev distance between the rook and the black king; also, the rook is exposed if black is on turn, and the Chebyshev distance between the rook and the white king is greater by at least 1 than the Chebyshev distance between the rook and the black king.

Rook divides: The white rook *divides* two kings if its x coordinate is (strictly) between x coordinates of the two kings, or if its y coordinate is (strictly) between y coordinates of the two kings (or both).

L-pattern: Three KRK pieces form an *L-pattern* if the kings are in the same row (column), at the Manhattan distance 2, and if the rook and the white king are in the same column (row) and at the Manhattan distance 1 (one example of a position in which the pieces form a *L-pattern* is shown in Figure 1).

White King on Edge: The white king is on edge if it is on any of the four edges of the chessboard (including corner squares).

Using the above notions, our strategy (for white) can be represented as follows (it is applied in legal KRK positions, while the reached position must not be a stalemate):

ImmediateMate: If there is such, play a mating move;

ReadyToMate: If the above is not possible, then play a move that leads to the *ready to mate* position shown in Figure 1 or a symmetric one (by rotations or reflections).[6]

Squeeze: If none of the above is possible, make a move (by the rook) that reduces the *room*; in the reached position, the rook is not exposed and divides the two kings; if there are more such moves, chose the one that reduces the *room* maximally;

ApproachDiag: If none of the above is possible, then move the king diagonally if that can decrease the Manhattan distance between the white king and the critical square; in the reached position, it has to hold: (*i*) the rook is not exposed, (*ii*) the rook divides the two kings or there is a *L-pattern*, and (*iii*) the *room* is greater than 3 or the white king is not on edge.

ApproachNonDiag: If none of the above is possible, then move the king non-diagonally if that can decrease the Manhattan distance between the white king and the critical square; in the reached position, it has to hold: (*i*) the rook is not exposed, (*ii*) the rook divides the two kings or there is a *L-pattern*, and (*iii*) the *room* is greater than 3 or the white king is not on edge.

KeepRoomDiag: If none of the above is possible, then move the king diagonally if that does not increase the Chebyshev distance from the rook; in the reached position, it has to hold: (*i*) the rook is not exposed and divides the two kings, and (*ii*) the *room* is greater than 3 or the white king is not on edge.

KeepRoomNonDiag: If none of the above is possible, then move the king non-diagonally if that does not increase the Chebyshev distance from the rook; in the reached position, it has to hold: (*i*) the rook is not exposed and divides the two kings, and (*ii*) the *room* is greater than 3 or the white king is not on edge.

RookHome: If none of the above is possible, then move the rook to be horizontally adjacent or vertically adjacent to the white king; if there are more such moves, the one with the smallest Manhattan distance between the rook and the black king is chosen; in the reached position, the rook can be adjacent to the black king only if it is guarded by the white king.

[6]Note that this step does not cover all "mate in 2 moves" positions. However, it eliminates the need for search and still ensures correctness of the strategy. (In contrast, the step **ImmediateMate** covers all immediate mates.)

RookSafe: If none of the above is possible, then move the rook to some edge (if not already on that edge); in the reached position, the rook can be adjacent to the black king only if it is guarded by the white king and the Chebyshev distance from the black king is at least 2.

In the considered set of 175168 positions, the above strategy steps are applied in: 1512, 648, 119236, 12192, 4124, 3472, 184, 33368, 432 positions, respectively. The working of the strategy is illustrated by one sequence of moves, shown in Figure 2. The final position shown leads to mate in several moves.

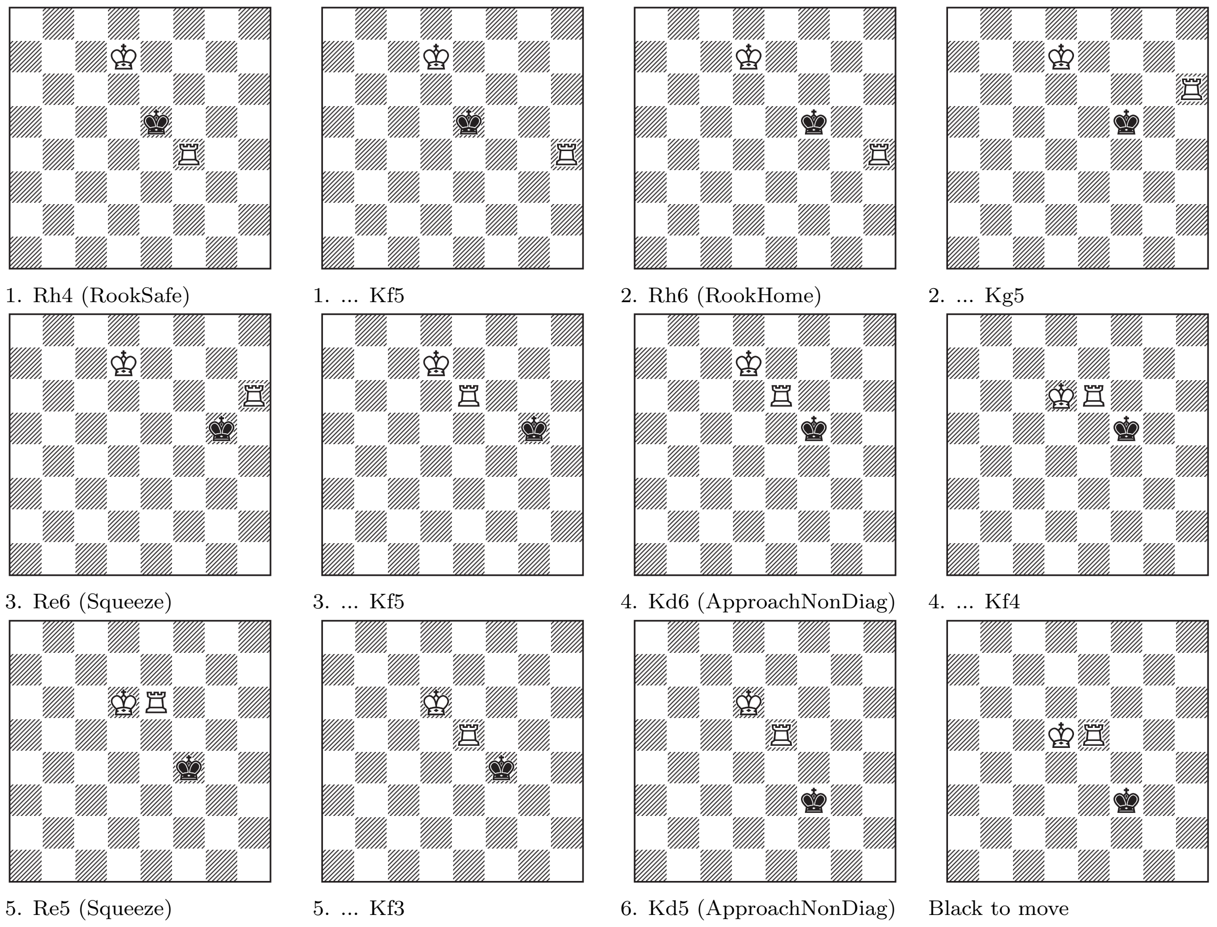

Figure 2: Illustration of the working of the strategy (below each chessboard in the sequence is the move that was made in that position; for black, the moves are chosen arbitrarily or in a way that shows more of the strategy)

The above strategy for white is correct (i.e., from any legal KRK position with white to move, following the strategy white wins) and proving this correctness is the subject of the paper. We implemented the strategy in the programming language C, using a suitable representation that enables efficient bit-wise operations. Then we applied the strategy to all legal KRK positions and generated the corresponding endgame database. Then, as discussed in Introduction, similarly as Bramer did [4, 5], by a retrograde procedure, we checked that the endgame database is correct (i.e., that it always leads to checkmate). Thanks to suitable data representation and efficient algorithms the strategy was proved correct in only a couple of seconds. However, instead of the correctness argument based on exhaustive analysis, we wanted a high-level, understandable and intuitive, argument on *why* the strategy really works. Due to this, we made a specification of the strategy and developed all needed lemmas within a SAT-based constraint solving system URSA. These specification and lemmas will be discussed in the next section.

Bratko gave one high-level proof of correctness of his strategy but that proof is informal, "pen-and-paper" and omits many important details [6]. Our strategy is closely related to Bratko's strategy, but our computer assisted proof significantly differs from his proof.

The strategy is not optimal and we will not address the optimality issue within our analysis. Still, it is interesting to consider how the strategy behaves comparing to the optimal strategy. While the optimal strategy leads to mate within 16 moves, with the above strategy mate is reached within 33 moves.[7] Figure 3 shows the number of legal KRK positions (out of a total of 175168) from which the mate is necessarily reached by the optimal play in 1, 2, ..., 16 moves, and also the number of legal KRK positions from which the mate is necessarily reached by following the strategy in 1, 2, ..., 33 moves (all presented data were generated by our C program that implements the strategy and other related functions). Hence, the strategy leads to win in accordance with the FIDE fifty-move rule. In the remaining text we will focus on proving that the strategy always leads to win (in a finite number of plies).

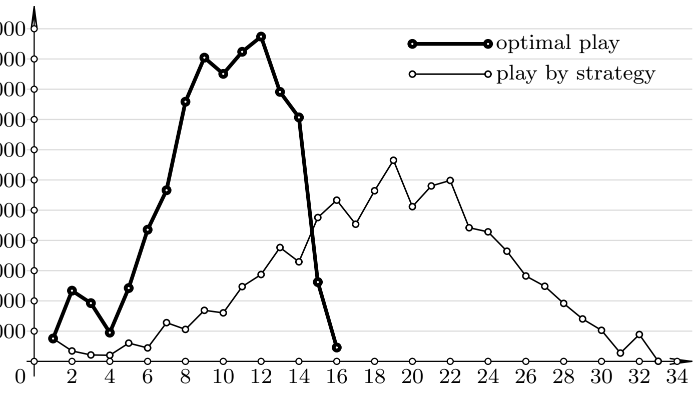

Figure 3: Number of positions with certain number of moves to win, following the optimal and the presented strategy

3 URSA Specification of the Chess Rules for KRK and the Strategy

In this section we briefly present our specification of the KRK endgame in the URSA specification language [21]. The complete URSA specification of the KRK endgame rules and the strategy have around 700 lines.[8] We will not present the full URSA specification, but only some of its fragments, for illustration.

In URSA, the problem is specified in a language which is imperative and similar to C. At the same time, this language is declarative, as the user does not have to provide a solving mechanism for the given problem. There are two types in the URSA language: (unsigned) numerical (with names of variables starting with '`n`') and Boolean (with names of variables starting with '`b`'). Variables can have concrete (ground) or symbolic values (which can further be dependent or independent). In the latter case, variables are represented by vectors of propositional formulae — of the length 1 for Boolean variables, or of the length n for numerical values, where n is chosen by the user. Representation of symbolic numerical variables by

[7]A strategy for KRK endgame developed using inductive logic programming [28] leads to mate within 57 moves, while another variant leads to mate in less than 50 moves.

[8]The URSA specification and the matching C program are available online from: `http://argo.matf.bg.ac.rs/downloads/software/krk.zip`.

propositional formulae corresponds to binary representation of unsigned numbers. Operations over concrete values produce concrete values. Operations over symbolic values boil down to bit-wise logical operations that, combined together, produce new symbolic values i.e., vectors of propositional formulae. All arithmetic operations (both over concrete and symbolic values) over numerical values are performed with respect to n (i.e., modulo 2^n). In URSA, there are control-flow structures (in the style of C) and there is support for procedures. An URSA specification is symbolically executed and the given constraint corresponds to one propositional formula. It is transformed into CNF and passed to one of the underlying SAT solvers. If this formula is satisfiable, the system returns one of its models, or lists them all if required. For instance, in the specification `nv = nu+1; assert(nv==2);`, the value of `nu` is accessed before `nu` was defined, so it will be independent and will have a symbolic value (i.e., a vector of propositional variable), the variable `nv` will be dependent and symbolic and will be assigned the value of `nu` increased by `1` (giving another symbolic value), and the conjecture is that this value equals `2`. This conjecture is transformed to a propositional formula, and then this formula is transformed to CNF, the SAT solver is invoked and URSA confirms that the conjecture can be true and provides `1` as a value for `nu`. Let us also illustrate an operation of URSA on one (artificial) chess-related toy problem. Let both columns and rows of the chessboard be denoted by the numbers $0, 1, \ldots, 7$, let the position of the white rook be given by (3,1) and let the position of the black king be given by (`nBKx`,`bBKy`). All positions in which the black king is left and lower with respect to the white rook can be obtained by the following URSA specification: `assert(nBKx<3 && nBKy<1);`. If the numerical values are, for instance, represented by vectors of propositional formulae of the length 3, and if `nBKx` and `nBKy` are represented by $[a, b, c]$ and $[p, q, r]$, then the above assertion is translated by URSA to the following propositional formula: $(\neg a \wedge (\neg b \vee \neg c)) \wedge (\neg p \wedge \neg q \wedge \neg r)$. This formula is transformed into the following formula in CNF: $\neg a \wedge (\neg b \vee \neg c) \wedge \neg p \wedge \neg q \wedge \neg r$. Over the set of variables a, b, c, p, q, r, there are three models for this formula: a model $0, 0, 0, 0, 0, 0$, a model $0, 0, 1, 0, 0, 0$ and a model $0, 1, 0, 0, 0, 0$. The underlying SAT solver can find them, and on the basis of these models, URSA returns three solutions for (`nBKx`,`bBKy`): $(0, 0)$, $(1, 0)$, $(2, 0)$.

Within the URSA system, the solving process is transferred to an underlying SAT solver, so the problem specification itself is the most critical part (assuming that the URSA translation and the used SAT solver are reliable). As a result, in developing the specification, we tried to have a simple and intuitive core that defines the chess rules and the strategy itself.

3.1 Chessboard and Positions

If both columns and rows of the chessboard are denoted by the numbers $0, 1, \ldots, 7$ (which is more suitable than $1, 2, \ldots, 8$, since the former numbers can be represented by 3 bits), then each square can be represented by a pair of two such numbers, and hence, by 6 bits. In the case of KRK endgame, instead of dealing with values of 64 squares of the chessboard, it is more convenient to use only the positions of all three pieces (represented by 6 bits each). However, instead of passing six numbers as arguments to specification procedures, they can be packed together into one 18-tuple (i.e., into a bit-vector of the length 18). In addition, the information on which player is on turn (one bit) is needed, so each KRK position can be stored in 19 bits. For the rest of this paper, we will assume this representation and that URSA is used with 19-bits numbers.

With the chosen representation, the first miscellaneous procedures that are needed are those that pack individual coordinates (`(nWKx, nWKy)` of the white king, `(nBKx, nBKy)` of the black king, `(nWRx, nWRy)` of the white rook, along with `bWhiteOnTurn` which is true if white is on turn) into a 19-tuple `nPos` and vice versa (as in C, `&` denotes bit-wise conjunction, `|` denotes bit-wise disjunction, `<<` and `>>` denote left and right shift,[9] etc):

```
procedure Cartesian2Pos(nWKx, nWKy, nBKx, nBKy, nWRx, nWRy, bWhiteOnTurn, nPos) {
  nPos = ite(bWhiteOnTurn,1,0);
  nPos = (nPos << 3) | (nWRy & 7);
  nPos = (nPos << 3) | (nWRx & 7);
  nPos = (nPos << 3) | (nBKy & 7);
  nPos = (nPos << 3) | (nBKx & 7);
```

[9]Note that `&` and `|` are bit-wise operators applied on numerical values, while `&&` and `||` are logical operators applied on Boolean values.

```
  nPos = (nPos << 3) | (nWKy & 7);
  nPos = (nPos << 3) | (nWKx & 7);
}

procedure Pos2Cartesian(nPos, nWKx, nWKy, nBKx, nBKy, nWRx, nWRy, bWhiteOnTurn) {
  nWKx = nPos & 7;
  nWKy = nPos >>  3 & 7;
  nBKx = nPos >>  6 & 7;
  nBKy = nPos >>  9 & 7;
  nWRx = nPos >> 12 & 7;
  nWRy = nPos >> 15 & 7;
  bWhiteOnTurn = num2bool(nPos >> 18);
}

procedure IsWhiteOnTurn(nPos, bWhiteOnTurn) {
  bWhiteOnTurn = num2bool(nPos >> 18);
}
```

The procedure `Cartesian2Pos` assumes that the value `nPos` is "output argument", while `Pos2Cartesian` assumes that the value `nPos` is "input argument" (however, generally there are no input and output arguments in URSA procedures — each argument can have both roles, as in Prolog, for instance). In the procedure `IsWhiteOnTurn`, the "input argument" is `nPos` and it "outputs" `bWhiteOnTurn` which is true if white is on turn.

3.2 Legal Positions

The conditions that, in a certain position (represented by numerical value `nPos`), the white king and the white rook cannot be on the same square, that the two kings cannot be on the same or adjacent squares, and that the black king is attacked by the white rook, can be represented by the following procedures (with Boolean arguments as "output arguments"):

```
procedure NotOnSameSquare(nPos, bNotOnSameSquare) {
  call Pos2Cartesian(nPos, nWKx, nWKy, nBKx, nBKy, nWRx, nWRy, bWhiteOnTurn);
  bNotOnSameSquare = (nWKx != nWRx) || (nWKy != nWRy);
}

procedure NotKingNextKing(nPos, bNotKingNextKing) {
  call Pos2Cartesian(nPos, nWKx, nWKy, nBKx, nBKy, nWRx, nWRy, bWhiteOnTurn);
  bNotKingNextKing = nWKx>nBKx+1 || nBKx>nWKx+1 || nWKy>nBKy+1 || nBKy>nWKy+1;
}

procedure BlackKingAttacked(nPos, bBlackKingAttacked) {
  call Pos2Cartesian(nPos, nWKx, nWKy, nBKx, nBKy, nWRx, nWRy, bWhiteOnTurn);
  call Between(nWRx, nWRy, nWKx, nWKy, nBKx, nBKy, bBetween);
  bBlackKingAttacked = (nWRx==nBKx ^^ nWRy==nBKy) && !bBetween;
}
```

In the procedure `BlackKingAttacked` above, the procedure `Between` (not shown here) is used. After invoking the procedure `Between`, the variable `bBetween` equals `true` if and only if the white king is between the black king and the white rook.

By convention, appropriate for the representation used in this paper, if the rook is captured, then the black king and the white rook are on the same square (and the game is considered drawn, as defined in Section 3.4). Whether the rook is captured is represented by the following procedure:

```
procedure RookCaptured(nPos, bRookCaptured) {
  call Pos2Cartesian(nPos, nWKx, nWKy, nBKx, nBKy, nWRx, nWRy, bWhiteOnTurn);
  bRookCaptured = nWRx==nBKx && nWRy==nBKy;
}
```

Finally, the procedure that checks whether a position is a legal KRK position can be represented as follows:

```
procedure LegalKRKPosition(nPos, bLegalKRKPosition) {
  call Pos2Cartesian(nPos, nWKx, nWKy, nBKx, nBKy, nWRx, nWRy, bWhiteOnTurn);
  call NotOnSameSquare(nPos, bNotOnSameSquare);
  call NotKingNextKing(nPos, bNotKingNextKing);
  call BlackKingAttacked(nPos, bBlackKingAttacked);
  call RookCaptured(nPos, bRookCaptured);
```

```
    bLegalKRKPosition = bNotOnSameSquare && bNotKingNextKing && !bRookCaptured &&
                        !(bBlackKingAttacked && bWhiteOnTurn);
  }
```

Recall that each coordinate of each piece is stored in three bits, so there is no need to ensure that they are between 0 and 7.

Note that URSA is a declarative constraint-solving system and there can be procedure with no "input arguments" and "output arguments". For example, when invoking the procedure `LegalKRKPosition` one can use a concrete value for position `nPos` and `bLegalKRKPosition` will be a ground Boolean value — true, if and only if the position is legal. However, one can also use a symbolic value for `nPos` and `bLegalKRKPosition` will be set to the condition that `nPos` is legal in terms of propositional variables forming the representation of `nPos`. In this case, one can assert `bLegalKRKPosition` and URSA will respond that there are 399112 values of `nPos` that lead to `bLegalKRKPosition` equal true.

The above definition of legal positions is simple and intuitive, but, there are still some subtle issues concerning this notion. Let us consider the position shown in Figure 4 (left). According to the above definition, this position is legal only if black is on turn. However, if black is on turn, what was the last move of white? It can be checked that there was no legal move of white that could have led to the current position, so the given position is impossible. Due to such situations (subject to retrograde chess analysis [24]), it is difficult to effectively define legal positions: the ideal definition would be that a position is legal if it is reachable from the initial chess position, but such a definition is practically useless. As we have already said, we don't consider this issue in our proofs: we will prove that the presented strategy leads to checkmate from each of 175168 positions with white on turn and legal in the weaker sense (as defined above). Hence, the strategy will be proven correct with this weaker notion of legal position used.

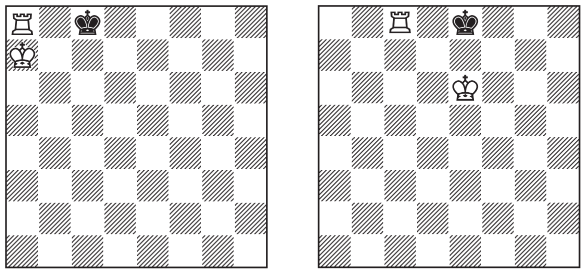

Figure 4: Example of an impossible position (left) and one mate position (right)

3.3 Legal Moves

The rules for moving pieces are divided into: (*i*) parts specifying movements rules themselves; (*ii*) a constraint that all other pieces remained on their original positions if not captured by the moving piece; (*iii*) a condition that the current player is indeed on turn and that another player is on turn after the move; (*iv*) the achieved position is legal. As an illustration, we give the part (*i*) specifying movement rules for the white king:

```
  procedure MoveWhiteKing(nPos1, nPos2, bMoveWhiteKing) {
    call Pos2Cartesian(nPos1, nWKx1, nWKy1, nBKx1, nBKy1, nWRx1, nWRy1, bWhiteOnTurn1);
    call Pos2Cartesian(nPos2, nWKx2, nWKy2, nBKx2, nBKy2, nWRx2, nWRy2, bWhiteOnTurn2);
    call ChebyshevDistance(nWKx1, nWKy1, nWKx2, nWKy2, nCD);
    bMoveWhiteKing = nCD==1;
  }
```

and the procedure that integrates all the constraints:

```
  procedure LegalMoveWhiteKing(nPos1, nPos2, bLegalMoveWhiteKing) {
    call Pos2Cartesian(nPos1, nWKx1, nWKy1, nBKx1, nBKy1, nWRx1, nWRy1, bWhiteOnTurn1);
    call Pos2Cartesian(nPos2, nWKx2, nWKy2, nBKx2, nBKy2, nWRx2, nWRy2, bWhiteOnTurn2);
    call MoveWhiteKing(nPos1, nPos2, bMoveWhiteKing);
    call OtherAfterMoveWhiteKing(nPos1, nPos2, bOtherAfterMoveWhiteKing);
    call LegalKRKPosition(nPos2, bLegalKRKPosition2);
    bLegalMoveWhiteKing = bMoveWhiteKing && bOtherAfterMoveWhiteKing &&
                          bWhiteOnTurn1 && !bWhiteOnTurn2 && bLegalKRKPosition2;
}
```

The procedure defining legal moves for the black king is defined by analogy. The procedure for the white rook is different, but defined in the same spirit and we don't show it here.

3.4 Mate, Stalemate, Draw

In defining positions that are mate or stalemate, the following procedure that checks whether a square is attacked by white is used:

```
procedure SquareAttackedByWhite(nPos, nX, nY, bSquareAttacked) {
  call Pos2Cartesian(nPos, nWKx, nWKy, nBKx, nBKy, nWRx, nWRy, bWhiteOnTurn);
  call Between(nWRx, nWRy, nWKx, nWKy, nX, nY, bBetween);
  call ChebyshevDistance(nWKx, nWKy, nX, nY, nCD);
  bSquareAttacked = (((nWRx==nX ^^ nWRy==nY) && !bBetween) || nCD==1);
}
```

The procedure `BlackKingCannotMove` (not listed here) uses the above procedure and checks whether black is on turn and all of the squares reachable by the black king are attacked by white. It does not use search, but explicitly checks all potentially reachable squares. With this procedure, defining mate and stalemate is simple:

```
procedure Mate(nPos, bMate) {
  call BlackKingCannotMove(nPos, bBlackKingCannotMove);
  call BlackKingAttacked(nPos, bBlackKingAttacked);
  bMate = !bWhiteOnTurn && bBlackKingCannotMove && bBlackKingAttacked;
}

procedure Stalemate(nPos, bStalemate) {
  call BlackKingCannotMove(nPos, bBlackKingCannotMove);
  call BlackKingAttacked(nPos, bBlackKingAttacked);
  bStalemate = !bWhiteOnTurn && bBlackKingCannotMove && !bBlackKingAttacked;
}
```

As already said, the position is drawn if the rook is captured:

```
procedure Draw(nPos, bDraw) {
  call IsWhiteOnTurn(nPos, bWhiteOnTurn);
  call RookCaptured(nPos,bRookCaptured);
  bDraw = bWhiteOnTurn && bRookCaptured;
}
```

In proving correctness of the strategy presented in this paper, the above definitions are used for sets of positions, but they can be also used for single, concrete positions. For example, the above definition of mate can be used for checking if a given position (illustrated in Figure 4, right) is mate:

```
nWKx=4; nWKy=5;
nBKx=4; nBKy=7;
nWRx=2; nWRy=7;
bWhiteOnTurn=false;
call Cartesian2Pos(nWKx,nWKy,nBKx,nBKy,nWRx,nWRy,bWhiteOnTurn,nPos);
call Mate(nPos,bMate);
assert(bMate);
```

URSA computes `bMate` and returns the answer `yes`. Since all calculations in this example were ground, the SAT solver was not invoked in this case. However, assertions can be also simbolic and then URSA works as a constraint solver. For example, the following URSA code can be used for listing/counting all stalemate positions:

```
call LegalKRKPosition(nPos,bLegalKRKPosition);
call Stalemate(nPos,bStalemate);
assert_all(bLegalKRKPosition && bStalemate);
```

URSA translates `bLegalKRKPosition && bStalemate` to a propositional formula and invokes the SAT solver to check if it is satisfiable. It turns (in less than 1s) that it is satisfiable and, moreover, it has 68 models, which URSA transforms back in terms of input values `nPos`, yielding all 68 stalemate KRK positions.

3.5 URSA Specification of the KRK Strategy

In this section we briefly present our specification of the strategy for white for the KRK endgame in the URSA specification language. The presented strategy for white is specified in URSA in the style of the procedures given above. As an example of miscellaneous procedures, we give the procedure that defines the notion of *room* (extensively used in the strategy and in correctness proofs):

```
procedure Room(nPos, nRoom) {
  call Pos2Cartesian(nPos, nWKx, nWKy, nBKx, nBKy, nWRx, nWRy, bWhiteOnTurn);
  nRoom = ite (nWRx==nBKx, 15,
           ite (nWRy==nBKy, 15,
             ite (nWRx>nBKx,
               ite (nWRy>nBKy, nWRx+nWRy, nWRx+(7-nWRy)),
                 ite (nWRy>nBKy, (7-nWRx)+nWRy, (7-nWRx)+(7-nWRy)))));
}
```

Also, we give procedures corresponding to two steps of the strategy:

```
procedure ImmediateMateCond(nPos1, nPos2, bImmediateMateCond) {
  call LegalMoveWhiteRook(nPos1, nPos2, bLegalMoveWhiteRook);
  call Mate(nPos2, bMate);
  bImmediateMateCond = bLegalMoveWhiteRook && bMate;
}

procedure ApproachDiagCond(nPos1, nPos2, bApproachDiagCond) {
  call LegalMoveWhiteKing(nPos1, nPos2, bLegalMoveWhiteKing);
  call KingDiag(nPos1, nPos2, bKingDiag);
  call ApproachCriticalSquare(nPos1, nPos2, bApproachCriticalSquare);
  call NotWhiteRookExposed(nPos2, bNotWhiteRookExposed);
  call WhiteRookDivides(nPos2, bWhiteRookDivides);
  call LPattern(nPos2, bLPattern);
  call RoomGt3(nPos2, bRoomGt3);
  call WhiteKingEdge(nPos2, bWhiteKingEdge);
  call Stalemate(nPos2, bStalemate);
  bApproachDiagCond = bLegalMoveWhiteKing && bKingDiag && bApproachCriticalSquare &&
                      bNotWhiteRookExposed && (bWhiteRookDivides || bLPattern) &&
                      (bRoomGt3 || !bWhiteKingEdge) && !bStalemate;
}
```

The above two procedures define constraints between two positions `nPos1` and `nPos2` that have to be satisfied in order for a specific strategy step to be applicable.

The procedure that defines the whole of the strategy uses procedures for individual steps. For a given `nPos1`, instead of search over all positions `nPos2`, only potential 22 legal moves are considered (8 for the king and 14 for the rook). This is done in turn for each strategy step (ordered as in Section 2). So, in the procedure `StrategyStep(nPos1,nPos2,b,nStep)` (not listed here because it has almost 60 lines), the value of `b` is equal to the condition that `nPos2` can be reached from `nPos1` by the strategy step `nStep`. The procedure is designed in such a way that its arguments can be either concrete or symbolic. For instance, if the procedure is used with symbolic `nPos1` and concrete `nStep`, it can be used for counting the number of positions in which a concrete strategy step is applicable.

4 Proving Correctness of KRK Strategy

Proving correctness of the presented KRK strategy for white is decomposed into several conjectures: on the top level—into termination (*if white follows the strategy, any KRK game ends*) and partial correctness (*if white follows the strategy and if a KRK endgame ends, then black is mated*). These conjectures lead to the central theorem, stated as follows:

Theorem 1 (Total correctness) *Starting from any legal KRK position, if white plays according to the strategy, the game will end with black mated in a finite number of plies.*

For presenting conjectures that lead to the above theorem, we will first introduce some notions and one basic lemma. In all of the following lemmas and theorems, we will assume that white follows the given strategy (even if not stated explicitly).

Definition 1 (Relations *Strategy*, *LegalMoveBlack* and *Move*)
For two positions p_1 and p_2, it holds $LegalMoveBlack(p_1, p_2)$ iff p_2 is reached from p_1 by a legal move of black (and whose URSA specification is discussed in Section 3.3).

For two positions p_1 and p_2, it holds $Strategy(p_1, p_2)$ iff p_2 is reached from p_1 following the strategy for white described in Section 2 (and whose URSA specification is discussed in Section 3.5).

For two positions p_1 and p_2, it holds $Move(p_1, p_2)$ iff there is a position p such that $Strategy(p_1, p)$ and $LegalMoveBlack(p, p_2)$. The relation $Move$ is also written in infix form as $\rightarrow$, i.e., $Move(p_1, p_2) \equiv p_1 \rightarrow p_2$.

Since both players follow the chess rules (the strategy for white incorporates the general chess rules), all positions obtained in a KRK game in which white follows the strategy are legal positions. Moreover, each position is a legal KRK position, i.e., the white rook cannot be captured by black, as stated by the following lemma.

Lemma 1 *Starting from any legal KRK position, after a legal step by black, the obtained position is again a legal KRK position.*

This lemma can be simply encoded in URSA (following the presented KRK specification) in the following way and proved (using proof by refutation):

```
call LegalKRKPosition(nPos1,bLegalKRKPosition1);
call StrategyStep(nPos1,nPos2,b1,nStep1);
call LegalMoveBlack(nPos2,nPos3,b2);
call LegalKRKPosition(nPos3,bLegalKRKPosition3);
bKRKLegalityNotPreserved = (bLegalKRKPosition1 && b1 && b2 && !bLegalKRKPosition3);
assert(bKRKLegalityNotPreserved);
```

In the above URSA specification, the condition `bKRKLegalityNotPreserved` states that there is a sequence of positions, such that `nPos1` is a legal KRK position, `nPos2` is obtained from `nPos1` by a strategy step for white, `nPos3` is obtained from `nPos2` by a move of black, and `nPos3` *is not* a legal KRK position. URSA solves the assertion and claims that `bKRKLegalityNotPreserved` is not satisfiable, i.e., there is no such sequence, so Lemma 1 holds. By a simple inductive argument, if white follows the strategy, after each move of white the reached position is again a legal KRK position (i.e., the rook will never be captured by black).

In the above and the following URSA specifications, whenever the procedure `StrategyStep` is invoked, this call incorporates the condition that white is on turn (so this condition is not given explicitly).

4.1 Termination

In order to prove that the KRK game always ends if white follows the given strategy, it is sufficient to prove that the relation $\rightarrow$ is well-founded (i.e., there is no infinite sequence of moves of white, playing following the strategy, and of black), as stated by the following theorem.

Theorem 2 (Termination) *The relation $\rightarrow$ is well-founded.*

A straightforward way to prove that a relation ρ over $D \times D$ is well-founded is to find a mapping (that serves as a termination measure) $m : D \mapsto \mathbf{N}$, such that $m(x) > m(y)$ whenever $x \rho y$. However, we failed to find such a measure (we suspect that it does not exist) and we had to make a more elaborated proof of Theorem 2, involving several complex lemmas.

In our proof, we use a mapping m from the set of legal KRK positions to the set of natural numbers, such that $m(P)$ equals:

- 100, if white is on turn, and the rook is exposed or the black king does not have squares to move to (i.e., white is threatened by stalemate),
- $6r + d$, otherwise,

where r is the *room* for the position P, and d is the Manhattan distance between the white king and the critical square for the position P. The mapping is specified by the following URSA procedure:

```
procedure Measure(nPos, nMeasure) {
  call Pos2Cartesian(nPos, nWKx, nWKy, nBKx, nBKy, nWRx, nWRy, bWhiteOnTurn);
  call Room(nPos, nRoom);
  call CriticalSquare(nPos, nCSx, nCSy);
  call ManhattanDistance(nWKx, nWKy, nCSx, nCSy, nMD);
  call NotWhiteRookExposed(nPos, bNotWhiteRookExposed);
  call ThreatenedByStalemate(nPos, bThreatenedByStalemate);
  nMeasure=ite(bNotWhiteRookExposed && !bThreatenedByStalemate,6*nRoom+nMD,100);
}
```

The mapping m is not a terminating measure, but it still has similar features, i.e., it decreases in one move for many positions. There are other mappings that could be used instead of m, but we find m suitable (and it is constructed along and in accordance with constructing our termination lemmas). The mapping m combines quantities r and d used in the strategy itself.[10] It is beneficial to combine r and d, since there are some positions in which, when the strategy is used, only one of r and d decreases. In addition, m penalize (by the measure 100) positions in which the white rook is exposed or the black king does not have available squares (so white has to avoid stalemate). These situations can occur in starting positions only, and the strategy cannot lead to such situations. The measure assigned to these positions (100), distinguishes them from all other positions reachable by the strategy (as the maximal measure for them is 96). This is important as the value $6r + d$ can be very low for positions in which the white rook is exposed or white is threatened by stalemate, but still for these positions mate can be reached only in a large number of moves. The key feature of the measure m is given by the following lemma.

Lemma 2 *Starting from any legal KRK position, the measure m decreases each three moves if:*

- *within these three moves white didn't use the strategy step* **ReadyToMate***;*
- *within these three moves white didn't use the steps* **RookHome** *and* **RookSafe***;*
- *the measure m is greater than 24 in the first position in this sequence.*

This lemma can be encoded in URSA as follows and proved:

```
call LegalKRKPosition(nPos1,bLegalKRKPosition1);
call StrategyStep(nPos1,nPos2,b1,nStep1);
call LegalMoveBlack(nPos2,nPos3,b2);
call StrategyStep(nPos3,nPos4,b3,nStep3);
call LegalMoveBlack(nPos4,nPos5,b4);
call StrategyStep(nPos5,nPos6,b5,nStep5);
call LegalMoveBlack(nPos6,nPos7,b6);
call Measure(nPos1,nMeasure1);
call Measure(nPos7,nMeasure7);
bTerminationLemma1 = (bLegalKRKPosition1 && b1 && b2 && b3 && b4 && b5 && b6 &&
                      nStep1!=nsRookSafe && nStep3!=nsRookSafe && nStep5!=nsRookSafe &&
                      nStep1!=nsRookHome && nStep3!=nsRookHome && nStep5!=nsRookHome &&
                      nStep1!=nsReadyToMate && nStep3!=nsReadyToMate && nStep5!=nsReadyToMate &&
                      nMeasure1>24 && nMeasure1<=nMeasure7);
assert(bTerminationLemma1);
```

The condition `bTerminationLemma1` states that there is a sequence of moves, described in Lemma 2, linking positions `nPos1` and `nPos7` such that m(`nPos1`) is less than or equal to m(`nPos7`). URSA proves that this is unsatisfiable (i.e., there is no such sequence), which proves the lemma. The above lemma states that the measure decreases in each three steps, except in three specific situations. We will prove that these three situations do not compromise termination.

The condition on the step **ReadyToMate** is covered by the the following lemma.

Lemma 3 *If white uses the strategy step* **ReadyToMate***, then mate follows after the next move.*

This lemma can be encoded in URSA as follows:

[10]In his pen-and-paper correctness proof, Bratko also used these two quantities [6] in showing that each game ends.

```
call LegalKRKPosition(nPos1,bLegalKRKPosition1);
call StrategyStep(nPos1,nPos2,b1,nStep1);
call LegalMoveBlack(nPos2,nPos3,b2);
call StrategyStep(nPos3,nPos4,b3,nStep3);
call Mate(nPos4,bMate);
bMateAfterReadyToMate = (bLegalKRKPosition1 && b1 && b2 && b3 && nStep1==nsReadyToMate && !bMate);
assert(bMateAfterReadyToMate);
```

The condition on the steps **RookHome** and **RookSafe** is covered by the following lemma.

Lemma 4 *Starting from any legal KRK position, the steps* **RookHome** *and* **RookSafe** *can be used by white only within the first three moves.*

This lemma can be encoded in URSA in the following way:

```
call LegalKRKPosition(nPos1,bLegalKRKPosition1);
call StrategyStep(nPos1,nPos2,b1,nStep1);
call LegalMoveBlack(nPos2,nPos3,b2);
call StrategyStep(nPos3,nPos4,b3,nStep3);
call LegalMoveBlack(nPos4,nPos5,b4);
call StrategyStep(nPos5,nPos6,b5,nStep5);
call LegalMoveBlack(nPos6,nPos7,b6);
call StrategyStep(nPos7,nPos8,b7,nStep7);
bTerminationLemma2 = bLegalKRKPosition1 && b1 && b2 && b3 && b4 && b5 && b6 && b7 &&
                     (nStep7==nsRookHome || nStep7==nsRookSafe);
assert_all(bTerminationLemma2);
```

The condition `bTerminationLemma2` states that there is a sequence of moves, such that **RookHome** or **RookSafe** is used in the fourth move. URSA proves that this condition is unsatisfiable.[11] From this lemma (and Lemma 1), it follows that using **RookHome** or **RookSafe** is impossible in n-th move, for $n > 3$, so Lemma 4 holds.

The condition on the measure less or equal to 24 is covered by the following lemma.

Lemma 5 *If the measure is less or equal to 24 in a position with white to move, then the game ends in at most* 7 *moves.*

This lemma can be encoded in URSA in the following way:

```
call LegalKRKPosition(nPos1,bLegalKRKPosition1);
call StrategyStep(nPos1,nPos2,b1,nStep1);
call LegalMoveBlack(nPos2,nPos3,b2);
call StrategyStep(nPos3,nPos4,b3,nStep3);
call LegalMoveBlack(nPos4,nPos5,b4);
call StrategyStep(nPos5,nPos6,b5,nStep5);
call LegalMoveBlack(nPos6,nPos7,b6);
call StrategyStep(nPos7,nPos8,b7,nStep7);
call LegalMoveBlack(nPos8,nPos9,b8);
call StrategyStep(nPos9,nPos10,b9,nStep9);
call LegalMoveBlack(nPos10,nPos11,b10);
call StrategyStep(nPos11,nPos12,b11,nStep11);
call LegalMoveBlack(nPos12,nPos13,b12);
call StrategyStep(nPos13,nPos14,b13,nStep13);
call LegalMoveBlack(nPos14,nPos15,b14);
call Measure(nPos1,nMeasure1);
bTerminationLemma3 = (bLegalKRKPosition1 && nMeasure1<=24 && b1 && b2 && b3 && b4 &&
                      b5 && b6 && b7 && b8 && b9 && b10 && b11 && b12 && b13 && b14);
assert_all(bTerminationLemma3);
```

In summary, if the strategy step **ReadyToMate** is used, or if the measure of a position is less or equal to 24, then the game ends in at most 7 moves. The steps **RookHome** and **RookSafe** can be used only within the first three moves of the play, and if they are not used, the measure decreases each three steps. Since the measure is always positive, the relation $\rightarrow$ is well-founded (as stated by Theorem 2), i.e., the game eventually ends.

[11]Actually, even a stronger conjecture can be proved for **RookSafe** – it can be used by the white only in the first move. However, for our purposes, this weaker variant is sufficient.

4.2 Partial correctness

Generally, if a chess game ends, the last move was made either by white or by black, and the possible outcomes are mate, stalemate and draw. Partial correctness of the presented strategy states that if the game ends, then it must be that the last move was made by white and black is mated as stated by the following theorem.

Theorem 3 (Partial correctness) *If a KRK game ends, then the last move was made by white and black is mated.*

The above theorem follows from the following (proved) lemmas (associated with corresponding URSA specifications).

Lemma 6 *Black cannot reach draw.*

```
call LegalKRKPosition(nPos1,bLegalKRKPosition1);
call StrategyStep(nPos1,nPos2,b1,nStep1);
call LegalMoveBlack(nPos2,nPos3,b2);
call Draw(nPos3,bDraw);
bLemmaPartialCorrectness1 = (bLegalKRKPosition1 && b1 && b2 && bDraw);
assert(bLemmaPartialCorrectness1);
```

Lemma 7 *If a KRK game ends, then the last move was made by white (i.e., in any legal KRK position with white on turn, there is a move that white can play according to the strategy).*

```
call LegalKRKPosition(nPos1,bLegalKRKPosition1);
call StrategyStep(nPos1,nPos2,b1,nStep1);
bLemmaPartialCorrectness2 = (bLegalKRKPosition1 && b1 && !(nStep1>=1 && nStep1<=9));
assert(bLemmaPartialCorrectness2);
```

Lemma 8 *If white is on turn in a legal KRK position, the position reached after white's move is not stalemate.*

```
call LegalKRKPosition(nPos1,bLegalKRKPosition1);
call StrategyStep(nPos1,nPos2,bStrategyUsed,nStep1);
call Stalemate(nPos2,bStalemate);
bLemmaPartialCorrectness3 = (bLegalKRKPosition1 && bStrategyUsed && bStalemate);
assert(bLemmaPartialCorrectness3);
```

Lemma 9 *If, after the move of white, black cannot move, then black is mated.*

```
call LegalKRKPosition(nPos1,bLegalKRKPosition1);
call StrategyStep(nPos1,nPos2,bStrategyUsed,nStep1);
call BlackKingCannotMove(nPos2,bBlackKingCannotMove);
call Mate(nPos2,bMate);
bLemmaPartialCorrectness4 = (bLegalKRKPosition1 && bStrategyUsed && bBlackKingCannotMove && !bMate);
assert(bLemmaPartialCorrectness4);
```

By Lemma 6 (closely related to Lemma 1), the game cannot be drawn (i.e., the white rook is never captured) and by Lemma 7, white always has a move to play, so the game cannot end by a move of black. By Lemma 8, the game cannot end as stalemate, and by Lemma 9 if the game ends, it must be that black is mated (somewhat redundant, as all other options are eliminated).

4.3 Efficiency

Within the URSA system, all the conjectures given above are translated to instances of the SAT problem and solved by the underlying SAT solver. Most of the generated formulae were huge — some of the conjectures had almost half a million propositional variables and more than one and a half million clauses. Still, the system solved them all, taking between 11 and 5651 seconds (see Table 1). It is common knowledge that bigger SAT instances are not necessarily harder (and this is also supported by the presented results). Instead, hardness is related to *constrainedness* [12], which is often related to the ratio of the number of clauses and the number of variables.

Lema	1	2	3	4	5	6	7	8	9
Number of variables	66351	205443	136676	264420	463770	66350	65890	70489	70489
Number of clauses	234756	726730	482753	935930	1641425	234757	233293	248496	248505
Time (in seconds)	11	5651	25	4457	129	11	11	25	24

Table 1: The number of variables and clauses in SAT instances generated from conjectures of the lemmas and CPU time used for solving these instances (on a computer PC T5870 2.00GHz, 1.8 GB RAM)

The presented results show that the presented approach is effective and practically applicable, even for very complex conjectures. Still, developing an appropriate specification of the problem is of crucial importance and flaws in this modelling could lead to conjectures too hard for solving. For instance, it is beneficial to use less expensive operations whenever possible. For example, in our specification we used addition instead of multiplication in the notion of *room* (see Section 2).

The size of the formulae also show that it was beneficial to use a system like URSA for generating them. Generating them by an ad-hoc tool would be much more error-prone.

5 Conclusions and Future Work

In this paper we presented our computer-assisted high-level proof of a correctness of a strategy for white for the KRK chess endgame (based on one Bratko's strategy). As far we are aware, this is the first proof of this kind.

The presented specification of the KRK strategy and our correctness proofs show:

- Even intuitively valid statements about chess strategies or other chess problems can involve many details and, hence, pen-and-paper proofs are very error prone.
- The game of chess, strategies for chess endgames such as KRK, and their correctness arguments can be described within a simple theory such as propositional logic.
- Translation of chess specifications can be suitably given in a high-level constraint programming system such as URSA, and then translated to SAT as the underlying theory.
- Although involving hundreds of thousands of propositional variables and clauses, all conjectures relevant for correctness of the KRK strategy can still be efficiently handled by modern SAT solving systems.

In our future work, we plan to use the same approach for proving properties of strategies for other chess endgames (e.g., KRKN, KBBK, KBNK). Also, we are planning to prove correctness of the KRK strategy (by proving the above lemmas and theorems) within a proof assistant such as Coq (using the paradigm of *computer-assisted formal direct proof*, discussed in Section 1). Such proofs would bring additional value. Namely, we proved, using the system URSA, the presented lemmas (leading to the correctness theorems), but in constraint programming systems one cannot build a theory that glues together all the conjectures into a single theorem. This is possible in proof assistants such as Coq, but the main challenge is to prove (even using all available automation) lemmas that are extremely combinatorially complex. The conjectures proved using a constraint solver would be vital in constructing a proof within a proof assistant. Apart from working on chess endgames, we plan to use the approach presented in this paper also for dealing with other sorts of problems in chess and in other games with perfect information.

References

[1] Michael Bain and Ashwin Srinivasan. Inductive logic programming with large-scale unstructured data. In *Machine Intelligence 14*, pages 235–250. Oxford University Press, 1995.

[2] Henk Barendregt and Freek Wiedijk. The challenge of computer mathematics. *Philosophical Transactions of the Royal Society*, 363(1835):2351–2375, 2005.

[3] Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications.* IOS Press, 2009.

[4] M. A. Bramer. Correct and optimal strategies in game playing programs. *The Computer Journal*, 23(4):347–352, 1980.

[5] Max Bramer. Machine-aided refinement of correct strategies for the endgame in chess. *Advances in Computer Chess*, 3:93–112, 1982.

[6] Ivan Bratko. Proving correctness of strategies in the al1 assertional language. *Information Processing Letters*, 7(5):223–230, 1978.

[7] Ivan Bratko. *PROLOG Programming for Artificial Intelligence (Second Edition).* Addison-Wesley, 1990.

[8] Ivan Bratko, Danny Kopec, and Donald Michie. Pattern-based representation of chess end-game knowledge. *Comput. J.*, 21(2):149–153, 1978.

[9] Stephen A. Cook. The complexity of theorem-proving procedures. In *STOC '71: Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158. ACM Press, 1971.

[10] Niklas Eén and Niklas Sörensson. An Extensible SAT-solver. In *Theory and Applications of Satisfiability Testing (SAT 2003)*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2004.

[11] J. Fürnkranz and M. Kubat, editors. *Machines that Learn to Play Games.* Nova Science Publishers, 2001.

[12] I.P. Gent, E. Macintyre., P. Prosser, and T. Walsh. The constrainedness of search. In *Proceedings of AAAI-96*, pages 246–252, Menlo Park, AAAI Press/MIT Press., 1996.

[13] Enrico Giunchiglia, Yuliya Lierler, and Marco Maratea. SAT-Based Answer Set Programming. In *The Nineteenth National Conference on Artificial Intelligence (AAAI 2004)*, pages 61–66. AAAI Press / The MIT Press, 2004.

[14] Matej Guid, Martin Mozina, Aleksander Sadikov, and Ivan Bratko. Deriving concepts and strategies from chess tablebases. In *Advances in Computer Games, ACG 2009*, volume 6048 of *Lecture Notes in Computer Science*, pages 195–207. Springer, 2010.

[15] G. M. Haworth and M. Velliste. Chess endgames and neural networks. *ICGA Journal (Journal of the International Computer Games Association)*, 21(4):211–227, 1998.

[16] Jinbo Huang. Universal booleanization of constraint models. In *Principles and Practice of Constraint Programming – CP 2008*, volume 5202 of *Lecture Notes in Computer Science*, pages 144–158. Springer, 2008.

[17] Joe Hurd and Guy Haworth. Data assurance in opaque computations. In *Advances in Computer Games – ACG 2009 (Revised Papers)*, volume 6048 of *Lecture Notes in Computer Science*, pages 221–231. Springer, 2010.

[18] Wayne Iba. Searching for better performance on the king-rook-king chess endgame problem. In *Proceedings of the Twenty-Fifth International Florida Artificial Intelligence Research Society Conference.* AAAI Press, 2012.

[19] Joxan Jaffar and Michael J. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19/20:503–581, 1994.

[20] Aleks Jakulin. Poskus indukcije človeku razumljivih pravil z atributnim učenjem pri šahovskih končcnicah. 2001. Experimetns with deriving human-understandable rules for chess endgames by atribute learning), unpublished paper, in Slovene.

[21] Predrag Janičić. URSA: A System for Uniform Reduction to SAT. *Logical Methods in Computer Science*, 8(3:30), 2012.

[22] Nicolas Lassabe, Stéphane Sanchez, Hervé Luga, and Yves Duthen. Genetically programmed strategies for chess endgame. In *Genetic and Evolutionary Computation Conference, GECCO 2006*, pages 831–838. ACM, 2006.

[23] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic*, 7(3):499–562, 2006.

[24] Marko Maliković and Mirko Čubrilo. What were the last moves? *International Review on Computers and Software*, 5(1):59–70, 2010.

[25] Laurent Michel and Pascal Van Hentenryck. The comet programming language and system. In *Principles and Practice of Constraint Programming - CP 2005*, volume 3709 of *Lecture Notes in Computer Science*, pages 881–881. Springer, 2005.

[26] Donald Michie and Ivan Bratko. Advice table representations of chess end-game knowledge. In *AISB/GI (ECAI)*, pages 194–200, 1978.

[27] David G. Mitchell and Eugenia Ternovska. A framework for representing and solving np search problems. In *National Conference on Artificial Intelligence – AAAI 2005*, pages 430–435. AAAI Press / The MIT Press, 2005.

[28] Eduardo F. Morales. Learning patterns for playing strategies. *ICCA Journal*, 17(1):15–26, 1994.

[29] Eduardo F. Morales. Learning playing strategies in chess. *Computational Intelligence*, 12(1):65–87, 1996.

[30] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: engineering an efficient sat solver. In *DAC '01: Proceedings of the 38th conference on Design automation*, pages 530–535. ACM Press, 2001.

[31] Nikolay Pelov and Eugenia Ternovska. Reducing inductive definitions to propositional satisfiability. In *International Conference on Logic Programming – ICLP 2005*, volume 3668 of *Lecture Notes in Computer Science*, pages 221–234. Springer, 2005.

[32] Aleksander Sadikov and Ivan Bratko. Learning long-term chess strategies from databases. *Machine Learning*, 63(3):329–340, 2006.

[33] Mehdi Samadi, Zohreh Azimifar, and Mansoor Zolghadri Jahromi. Learning: An effective approach in endgame chess board evaluation. In *The Sixth International Conference on Machine Learning and Applications, ICMLA 2007*, pages 464–469. IEEE Computer Society, 2008.

[34] Michael Schlosser. Knowledge discovery in endgame databases. In *Advances in Intelligent Data Analysis, IDA-97*, volume 1280 of *Lecture Notes in Computer Science*, pages 423–435. Springer, 1997.

[35] R. Seidel. Deriving correct pattern descriptions and rules for the krk endgame by deductive methods. In *Advances in Computer Chess 4*, pages 19–36. Pergamon Press, 1986.

[36] Peter J. Stuckey, Maria J. García de la Banda, Michael J. Maher, Kim Marriott, John K. Slaney, Zoltan Somogyi, Mark Wallace, and Toby Walsh. The g12 project: Mapping solver independent models to efficient solutions. In *Principles and Practice of Constraint Programming - CP 2005*, volume 3709 of *Lecture Notes in Computer Science*, pages 13–16. Springer, 2005.

[37] Naoyuki Tamura, Akiko Taga, Satoshi Kitagawa, and Mutsunori Banbara. Compiling finite linear csp into sat. *Constraints*, 14(2):254–272, 2009.

[38] Ken Thompson. Retrograde analysis of certain endgames. *International Computer Chess Association Journal*, 9(3):131–139, 1986.

[39] Freek Wiedijk, editor. *The Seventeen Provers of the World*, volume 3600 of *Lecture Notes in Computer Science*. Springer, 2006.

[40] Lintao Zhang and Sharad Malik. The Quest for Efficient Boolean Satisfiability Solvers. In *Automated Deduction (CADE 2002)*, volume 2392 of *Lecture Notes in Computer Science*, pages 295–313. Springer, 2002.

[41] C. Zuidema. Chess: How to Program the Exceptions? Technical Report IW21/74, Department of Informatics, Mathematical Center Amdsterdam., 1974.