SAT-based Analysis of the Legality of Chess Endgame Positions

Marko Maliković
Faculty of Humanities and Social Sciences
University of Rijeka
Sveučilišna avenija 4, 51000 Rijeka, Croatia
marko.malikovic@ffri.hr

Abstract. *Various analyses of chess endgames are made with different purposes. These analyzes are usually based on exhaustive analysis using previous generated corresponding databases. It is often not investigated whether the endgame positions are legal (or why are not legal). Legality of endgame positions can be proven in several ways, and in this paper we present one of them: high-level computer-assisted proof based on reduction to propositional logic, more precisely to SAT. As case study we focus on a King and Rook vs. King endgame, and reduction to SAT is performed by using a constraint solving system URSA. We are not aware of other computer-assisted high-level proof of a legality of some chess endgame. The presented methodology can be applied to other chess endgames. Therefore, the point of this paper is not only presenting a proof of legality of an endgame, but also presenting a new methodology for computer-assisted proving of legality of chess endgames in general*.

Keywords. Chess, Endgames, Legality, SAT, URSA

1 Introduction

A superficial definition of legal KRK (King and Rook vs. King) positions that naturally arise is the following: Legal KRK positions are those that meet the following conditions:
- Two kings are not on adjacent squares;
- The piece of the color to move does not attack the other color's king.

The above definition is straightforward and intuitive. But, there are important subtle issues concerning this notion. Let us consider the position shown in Fig. 1. According to the above definition, this position is legal if black is to move. However, if black is to move, what was the last move by white? It can be easily checked that there was no legal move by white that could have led to the current position, so the given position is impossible. We see that even in such a simple endgame as is KRK, there are situations in which is necessary to look into the problem more deeply. The problem becomes more complex by increasing the number of pieces on the chessboard and by changing the kinds of pieces.

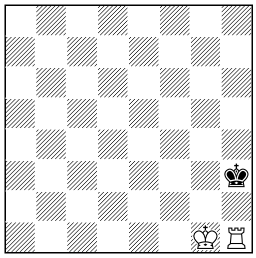

Figure 1. Illegal KRK position

Such problems are actually subject to retrograde chess analysis [7], [9]. Generally, the *ideal definition* of the legality of a position would be that a position is legal if it is reachable from the initial chess position by a sequence of legal moves. But such a definition is practically useless because the problem of checking the legality of a given position is then building a sequence of legal moves which leads from the initial chess position to the given position. It is clear that such a move sequence for endgames can be very long. In short, it is necessary to build a whole chess game that leads to a given position and it actually becomes a problem of *proof games* or *shortest proof games* [13]. Such an extension of the problem is generally unnecessary and is not suitable for chess endgames which contain a small number of pieces. In addition, it can be assumed that *strictly formal proving* of the legality of the whole chess games is impossible. If this is true, it would mean that even formal proof of the legality of the endgames with a small number of pieces would never have been possible, but should always be based on the searching, generating corresponding databases, etc.

Various analyses of chess endgames with different purposes are made with different methods. For example, using an arbitrary programming language, all positions and corresponding databases can be generated. Then, using a retrograde procedure [9],

[10], [12] some properties of these endgames can be verified (e.g. in which positions checkmate is possible). As another example, based on lookup tables with pre-calculated optimal moves for each legal position, it is possible to check the correctness of some strategy (which, for example, should always lead to checkmate or draw in the given endgame). This approach was, for example, used by Bramer [2] for testing the correctness of some endgame strategies, but also for refining strategies that turned out not to be correct. So, the mentioned approaches, instead of direct proofs, are based on a form of exhaustive analysis. The advantage of these approaches is that it is quite straightforward. Its drawback is that it does not provide a high-level, human-understandable and intuitive argument on *why* some position is illegal or *why* some strategy really works. What is most important for this paper is that it often happens that an analysis of endgames does not even investigate which of the positions in a given endgame is legal. In the case when illegal positions are correctly detected in the analysis, it often remains unknown which positions are illegal (and why are illegal). Finally, there may be errors in the results (due to errors in program code), but there are no formal proofs of illegality of positions.

Due to the above reasons, in this paper we show one possible method of proving the legality of endgames. This method is based on the use of SAT-based constraint solving.[1] We are not aware of other computer-assisted high-level proof of the legality of some chess endgame. A similar approach as the one in this paper is used in [8] but for other purposes (for proving the correctness of a chess endgame strategy).

The rest of the paper is organized as follows: Section 2 provides an overview of various types of the illegality of positions, in Section 3 we present the main features of the SAT-based constraint solving system URSA, in Section 4 we give URSA specification of the chess rules for KRK and in Section 5 we provide the main proof of legality of intuitively legal KRK positions.

2 Illegality of positions

Lippold [5], [6] distinguishes the following types of illegality of positions:

Initially illegal are positions whose illegality "can be seen immediately". This refers to situations in which the pawn is in the first or last row, kings are on adjacent squares, two pieces are in the same square, a piece of the color to move threatens the other color's king;

Derivedly illegal are positions which are initially legal but *simply (once)* or *n times derived illegal*. It is *simply* derived illegal if there is no initially legal position from which it can be reached with one legal move. For example, the position at Fig. 2 with black to move is simply derived illegal. Position is *n times derived illegal* if every position, from which it can be reached with one legal move, is initially or *m* times derived illegal with an arbitrary *m* smaller than *n* and furthermore at least one position is *n-1* times and no position is *n* times derived illegal. With black to move a three times derived illegal position is showed at Fig. 3. The position at Fig. 3 has only one previous position (with white pawn at *g2*) while this previous position has several possible previous positions. Two of them are shown in Fig. 4. But none of these positions has any previous position (because the black king cannot be simultaneously in check with the white queen and the other white bishop).

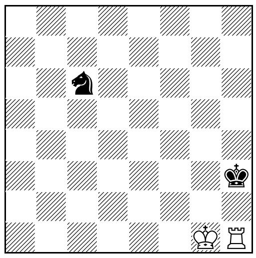

Figure 2. Simply derived illegal KRKN position

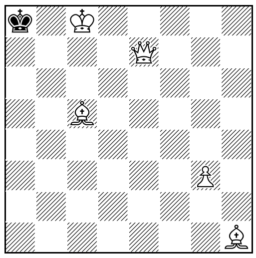

Figure 3. With black to move a three times derived illegal position

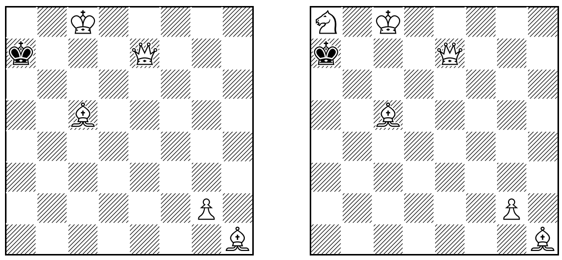

Figure 4. Two possible positions two moves before the position at Fig. 3

[1] 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 [3], 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 [1]. These advances in SAT solving make it possible to decide the satisfiability of some industrial SAT problems with hundreds of thousands of variables and millions of clauses.

Isolatedly illegal are positions that are neither initially illegal nor derivedly illegal but which cannot be reached from the initial chess position and are therefore illegal according to the ideal definition. Such is for example the position with the white bishop in the corner *a1* and with the white pawn at *b2*, as is shown at Fig. 5.

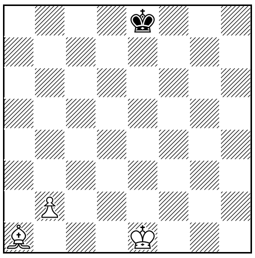

Figure 5. Isolatedly illegal position

There is one additional aspect that is important in an analysis of the legality of endgames. Sometimes it is not sufficient to look only at the positions of the analyzed endgame in checking whether it is derivedly illegal. The position at Fig. 6 is derivedly illegal if we consider only KRKN positions. But if we allow that the rook comes from *a1* and captured some black piece at *a4* then this position is (possibly) legal in some 5-pieces endgame. So, in order to check legality of some 4-pieces positions, it is sometimes necessary to check the legality of previous 5-pieces positions. Also, even if we restrict endgame to 4-pieces KRKN, then the position in Fig. 7 is not derivedly illegal because the rook could come from *a1* and captured black knight at *a4*. The position in Fig. 7 would be derivedly illegal only if we restrict the endgame to the 3-pieces KRK endgame.

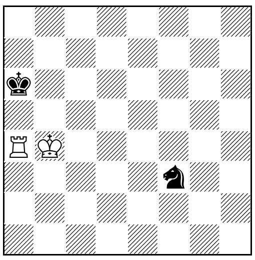

Figure 6. Derivedly illegal 4-pieces position

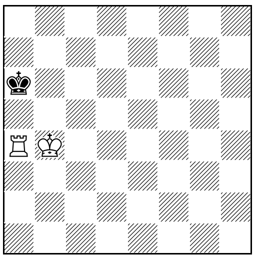

Figure 7. Derivedly illegal 3-pieces position

So, even if we study only the endgames, it is necessary to look at a chess game as a whole. Thus, we can conclude that the position in Fig. 7 is not actually an illegal position. It is legal because it can be expanded to some 4-pieces endgame.

3 Constraint solving system URSA

In the constraint solving system URSA [4], the problem is specified in a language which is imperative and similar to programming language C, there are control-flow structures and there is support for procedures. At the same time, this language is declarative, as the user does not have to provide a solving mechanism for the given problem.

Let us illustrate solving problems in URSA on one (artificial) chess-related toy problem. Let both columns and rows of the chessboard be denoted by the numbers 0, 1, ..., 7 and let the position of the white rook be given by (3,1). Suppose we want to find all the positions in which the black king is "left" and "lower" on the chessboard with respect to the white rook. For this we need to find all possible pairs of coordinates of the black king that satisfy a specified condition. There is a type in the URSA language for (unsigned) numerical with the names of variables starting with "n". If we introduce the coordinates of the black king as variables *nBKx* and *nBKy* then the specified condition can be expressed as $nBKx < 3 \land nBKy < 1$, and all possible positions can be obtained by the following URSA specification: *assert(nBKx<3 && nBKy<1);*, where *assert(b)* checks whether *b* is true.

In URSA, the representation of symbolic numerical variables corresponds to a binary representation of unsigned numbers.[2] Further, such variables are represented by the vectors of propositional formulae. In our example, chess coordinates are represented by vectors of propositional formulae of length 3 (because numerical values less than or equal to 7 can be represented by binary numbers of length 3). If *nBKx* and *nBKy* are represented by vectors $[a,b,c]$ and $[p,q,r]$, then the above assertion is translated by URSA to the following propositional formula: $(\neg a \land (\neg b \lor \neg c)) \land (\neg p \land \neg q \land \neg r)$. This formula is transformed into the following formula in CNF: $\neg a \land (\neg b \lor \neg c) \land \neg p \land \neg q \land \neg r$. 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).

[2] Variables *nBKx* and *nBKy* are symbolic variables, while operations over concrete values produce concrete values.

4 URSA Specification of the Chess Rules for KRK

In this section we briefly present our specification of the KRK endgame in the URSA specification language. The specification is largely in accordance with [8] with the changes that are needed due to the specific topic of this paper. In this paper some of the longer parts of the specification are not listed but most are still here. If something is skipped it is also noted.[3]

4.1 Underlying procedures

If both columns and rows of the chessboard are denoted by the numbers 0, 1, ..., 7 (which is more suitable than 1, 2, ..., 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 this information we use another type that exists in URSA: Boolean type with names of variables starting with "b" and of the length 1 with value 0 for *false* and 1 for *true*.

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, etc.)[4]:

```
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);
  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);}
```

The procedure *Cartesian2Pos* assumes that the value *nPos* is an "output argument", while *Pos2Cartesian* assumes that the value *nPos* is an "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 chess, the term *distance* refers to the minimal number of moves a certain piece needs to reach a target square from the starting square. For kings two kinds of distances are the most important: *Chebyshev distance* as minimal number of any king moves, and *Manhattan distance* which is restricted to orthogonal king moves. Procedures which give specified distances are as follows (to compute the Chebyshev distance we also need a procedure that gives a maximum of two numbers):

```
procedure Max(nx,ny,nMax) {
  nMax = ite(nx>=ny,nx,ny);}

procedure ChebyshevDistance(nx1,ny1,nx2,ny2,nCD) {
call Max(nx2-nx1,ny2-ny1,nMax1);
call Max(nx2-nx1,ny1-ny2,nMax2);
call Max(nx1-nx2,ny2-ny1,nMax3);
call Max(nx1-nx2,ny1-ny2,nMax4);
nCD = ite(nx2>=nx1,
          ite(ny2>=ny1,nMax1,nMax2),
          ite(ny2>=ny1,nMax3,nMax4));}

procedure ManhattanDistance(nx1,ny1,nx2,ny2,nMD) {
  nMD = ite(nx2>=nx1,
            ite(ny2>=ny1,(nx2-nx1)+(ny2-ny1),(nx2-nx1)+(ny1-ny2)),
            ite(ny2>=ny1,(nx1-nx2)+(ny2-ny1),(nx1-nx2)+(ny1-ny2)));}
```

As we will see, when specifying the problem that we deal in this paper we'll need both of the above distances.

The following procedure *Between* gives an answer to whether some square is between two other squares:

```
procedure Between(nx1,ny1,nx2,ny2,nx3,ny3,bBetween) {
bBetween = (nx1==nx2 && nx2==nx3 && ((ny1<ny2 && ny2<ny3) || (ny1>ny2 && ny2>ny3))) ||
(ny1==ny2 && ny2==ny3 && ((nx1<nx2 && nx2<nx3) || (nx1>nx2 && nx2>nx3)));}
```

After invoking the procedure *Between*, the variable *bBetween* equals *true* if and only if the square (*nx2*,*ny2*) is between squares (*nx1*,*ny1*) and (*nx3*,*ny3*).

4.2 Initially legal KRK 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 case where the white rook is captured, only two kings remain on the chessboard and it makes no

[3] The URSA specification is available online from: http://www.ffri.hr/~marko/sat_endgames/sat_krk.zip.

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

sense to consider such positions. By convention, and appropriate for the representation used in this paper, in such situations the black king and the white rook are on the same square. 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 initially 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);}
```

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*.

4.3 Moves of pieces

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) the condition that the current player is indeed on turn and that another player is on turn after the move. As an illustration, we give the part (i) specifying movement rules for the white king (*nPosS* is starting position and *nPosE* is ending position):

```
procedure MoveWhiteKing(nPosS,nPosE,bMoveWhiteKing) {
  call Pos2Cartesian(nPosS,nWKxS,nWKyS,nBKxS,nBKyS,nWRxS,nWRyS,bWhiteOnTurnS);
  call Pos2Cartesian(nPosE,nWKxE,nWKyE,nBKxE,nBKyE,nWRxE,nWRyE,bWhiteOnTurnE);
  call ChebyshevDistance(nWKxS,nWKyS,nWKxE,nWKyE,nCD);
  bMoveWhiteKing = (nCD==1);}
```

and the procedure that integrates all the constraints:

```
procedure LegalMoveWhiteKing(nPosS,nPosE,bLegalMoveWhiteKing) {
  call Pos2Cartesian(nPosS,nWKxS,nWKyS,nBKxS,nBKyS,nWRxS,nWRyS,bWhiteOnTurnS);
  call Pos2Cartesian(nPosE,nWKxE,nWKyE,nBKxE,nBKyE,nWRxE,nWRyE,bWhiteOnTurnE);
  call MoveWhiteKing(nPosS,nPosE,bMoveWhiteKing);
  call OtherAfterMoveWhiteKing(nPosS,nPosE,bOtherAfterMoveWhiteKing);
  bLegalMoveWhiteKing = bMoveWhiteKing && bOtherAfterMoveWhiteKing && bWhiteOnTurnS &&
!bWhiteOnTurnE;}
```

The procedure defining 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.

Finally, in one procedure we unite all possible moves of one player. For example, for the white player the procedure is as follows:

```
procedure LegalMoveWhite(nPosS,nPosE,bLegalMoveWhite) {
  call LegalMoveWhiteKing(nPosS,nPosE,bLegalMoveWhiteKing);
  call LegalMoveWhiteRook(nPosS,nPosE,bLegalMoveWhiteRook);
  bLegalMoveWhite = bLegalMoveWhiteKing || bLegalMoveWhiteRook;}
```

5 Legality of KRK positions

5.1 KRK positions with at least one previous position

We want to show that the KRK positions with at least one previous position are also legal according to the ideal definition. So, first we have to limit the set of KRK positions on a set with at least one previous position. Such positions *nPos* may be obtained by the following constraint (*nPos0* are eventual previous positions of *nPos*):

```
call LegalKRKPosition(nPos,bLegalKRKPosition);
call LegalKRKPosition(nPos0,bLegalKRKPosition0);
call LegalMoveWhite(nPos0,nPos,bW);
call LegalMoveBlack(nPos0,nPos,bB);
bPosHasPrevPos = bLegalKRKPosition && bLegalKRKPosition0 && (bW || bB);
```

In the upper constraint, *bLegalKRKPosition* and *bLegalKRKPosition0* are equal *true* if *nPos* and *nPos0* are initially legal positions, and *bW* or *bB* are *true* for all white's or black's move which leads from *nPos0* to *nPos*. Finally, *bPosHasPrevPos* is equal *true* if all the *bLegalKRKPosition*, *bLegalKRKPosition0* and (*bW* || *bB*) are equal *true*.

5.2 Proof of the main theorem

As we have already mentioned, we want to prove that all the initially legal positions with at least one previous position are legal also according to the ideal definition. To prove this, we will prove the following theorem:

Theorem: For each initially legal position P (which has some previous initially legal position P_0) there is a sequence of four previous initially legal positions P_4, P_3, P_2, P_1 (with legal moves from P_4 to P_3, from P_3 to P_2, from P_2 to P_1 and from P_1 to P) such that:

(1) P_4 is *closer* to one selected position *PFix* than P;[5]
(2) P_4 has some previous position;
(3) *PFix* is legal according to the ideal definition.

Proof:

(1) First we choose a concrete position *PFix* showed at Fig. 8.

[5] The meaning of the term *closer* is explained in the proof that follows.

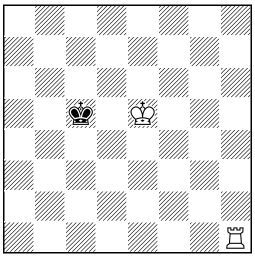

Figure 8. Concrete position *PFix*

Note that we did not specify which color is to move into position *PFix* (*PFix* with white to move and *PFix* with black to move are two different positions). We did that because it greatly simplifies the proofs that follow. However, this means that we have to prove point (3) of the theorem for both cases (which is a much easier problem).

Next, we found a mapping (measure) *M* from the set of legal KRK positions to the set of natural numbers such that for all positions *P* exists a sequence of four previous legal positions P_4, P_3, P_2, P_1 such that $M(P_4) < M(P)$. The measure of position *P* is relative with respect to position *PFix*, and is equal to:

$$M(P)=2\cdot CDWK+MDWK+2\cdot CDBK+MDBK+DWR$$

If the squares in which the pieces are in a position *PFix* is called *target squares*, then in the above formula is:

CDWK = Chebyshev distance of white king at *P* and his target square
CDBK = Chebyshev distance of black king at *P* and his target square
MDWK = Manhattan distance of white king at *P* and his target square
MDBK = Manhattan distance of black king at *P* and his target square
DWR = minimal number of moves in which white rook can reach his target square (it can be 0, 1 or 2)

The mapping *M* is specified by the following URSA procedure:

```
procedure Measure(nPos,nMeasure) {
  call Pos2Cartesian(nPos,nWKx,nWKy,nBKx,nBKy,nWRx,nWRy,bWhiteOnTurn);
  call ChebyshevDistance(nWKx,nWKy,4,4,nCDWK);
  call ChebyshevDistance(nBKx,nBKy,2,4,nCDBK);
  call ManhattanDistance(nWKx,nWKy,4,4,nMDWK);
  call ManhattanDistance(nBKx,nBKy,2,4,nMDBK);
  nDWR = ite(nWRx==7 && nWRy==0,0,ite(nWRx==7 ^^ nWRy==0,1,2));
  nMeasure=2*nCDWK+nMDWK+2*nCDBK+nMDBK+nDWR;}
```

The key feature of the measure *M* is that for any legal KRK position *P*, there exists a sequence of four previous legal positions P_4, P_3, P_2, P_1 such that the measure is less in P_4 than in *P*, and that is exactly what assertion (1) claims. This assertion can be encoded in URSA as follows:

```
call LegalKRKPosition(nPos4,bLegalKRKPosition4);
call LegalKRKPosition(nPos3,bLegalKRKPosition3);
call LegalKRKPosition(nPos2,bLegalKRKPosition2);
call LegalKRKPosition(nPos1,bLegalKRKPosition1);
bSequenceOfLegalPositions = bLegalKRKPosition4 && bLegalKRKPosition3 &&
bLegalKRKPosition2 && bLegalKRKPosition1;

call Exists4PliesThatMeasureDecrease(nPos,nPos1,nPos2,nPos3,nPos4,bMD4);

assert_all(bPosHasPrevPos && bSequenceOfLegalPositions && !bMD4);
```

The condition *bSequenceOfLegalPositions* states that all of P_4, P_3, P_2, P_1 are initially legal KRK positions. Further, in the procedure *Exists4PliesThatMeasureDecrease(nPos,nPos1,nPos 2,nPos3,nPos4,bMD4)* (not listed here because it has over 50 lines and two auxiliary procedures) the value of *bMD4* is equal to *false* only if $M(P_4) \geq M(P)$ for all sequences P_4, P_3, P_2, P_1. Finally, the main assertion *bPosHasPrevPos && bSequenceOfLegalPositions && !bMD4* is in fact a negation of assertion (1). URSA proves that this is unsatisfiable (i.e., there is no such position *P*), which proves the assertion (1). URSA gives a result in this form:

```
No solutions found
[Formula generation: 799.34s; conversion to CNF: 860.352s; total: 1659.69s]
[Solving time: 909.02s]
[Formula size: 1041262 variables, 6391975 clauses]
```

(2) This assertion actually ensures that position P_4 is not four times derived illegal. To prove this, we can prove a stronger assertion (but which is easier to set up and which requires less computer resources). This stronger assertion is: *For each legal position P_1 which is a previous position of some legal position P there is at least one previous position P_2.*

To prove this, we need an additional procedure:

```
procedure ExistsPreviousPositionOfPreviousPosition(nPos1,nPos2,bPP)
{
  bPP = false;
  for (nMoveWi=1;nMoveWi<=24;nMoveWi++) {
    call NextMoveWhite(nPos1,nMoveWi,nPos2,bNMW);
    call LegalMoveWhite(nPos2,nPos1,bLMW);
    bPP ||= bNMW && bLMW;
  }
  for (nMoveBm=1;nMoveBm<=8;nMoveBm++) {
    call NextMoveBlack(nPos1,nMoveBm,nPos2,bNMB);
    call LegalMoveBlack(nPos2,nPos1,bLMB);
    bPP ||= bNMB && bLMB;
  }
}
```

The value of *bPP* will be equal to *false* only if for some position P_1 there is no previous position P_2.

Now, assertion (2) can be simply encoded in URSA in the following way and proved (also using proof by refutation):

```
call LegalKRKPosition(nPos,bLegalKRKPosition);
call LegalKRKPosition(nPos1,bLegalKRKPosition1);
call LegalMoveWhite(nPos1,nPos,bW);
call LegalMoveBlack(nPos1,nPos,bB);
bPosHasPrevPos = bLegalKRKPosition && bLegalKRKPosition1 && (bW || bB);
call LegalKRKPosition(nPos2,bLegalKRKPosition2);
call ExistsPreviousPositionOfPreviousPosition(nPos1,nPos2,bPP);

assert_all(bPosHasPrevPos && bLegalKRKPosition2 && !bPP);
```

URSA solves the assertion and claims that it is not satisfiable, so assertion (2) holds.

(3) As we explained in the Introduction of this paper, our intention is to avoid *formal proof* that the position *PFix* is legal according to the ideal definition. It is sufficient to find a legal chess game that leads to this position. As we already mentioned, actually there are

two positions *PFix* (with white to move and with black to move). If position *PFix* is legal (more precisely, both its variants) then it is trivial to find such games (not taking into account the quality of these games). Such games can be derived using any of the computer programs for solving *proof games* (for example, see [11]). So, we give one game which leads to a variant of *PFix* in which is black to move: *1. e4 e5 2. d4 exd4 3. Qxd4 d5 4. Qxd5 Qxd5 5. exd5 c6 6. dxc6 bxc6 7. b4 c5 8. bxc5 Bxc5 9. Be3 Bxe3 10. fxe3 f5 11. e4 fxe4 12. Nf3 exf3 13. gxf3 g5 14. f4 gxf4 15. Bg2 f3 16. Bxf3 h5 17. Bxh5+ Kd7 18. c4 Nf6 19. Nc3 Nxh5 20. Nd5 Nf4 21. Nb6+ axb6 22. c5 bxc5 23. Rc1 Rxh2 24. Rxh2 Nd5 25. Rxc5 Nb4 26. Rxc8 Nxa2 27. Rxb8 Rxb8 28. Rxa2 Rb2 29. Rxb2 Kd6 30. Rh2 Kc6 31. Kd2 Kd6 32. Kd3 Kc6 33. Kd4 Kd6 34. Ke4 Kc6 35. Ke5 Kc5 36. Rh1*.

With this the whole theorem is proved.

Now we can give the following conclusion: To conclude whether some KRK position is legal according to the ideal definition, it is sufficient to conclude whether it is initially legal and whether it has at least one previous position.

Note that in a set of legal KRK positions (which are not covered by this conclusion), there are (possibly) still those which are legal because they can be extended to some 4-pieces position (see Fig. 7). To prove their (possible) legality, it is necessary to extend the system to 4-pieces endgames and conduct proofs analogous to the proof presented in this paper.

References

[1] Biere, A; Heule, M. J. H; van Maaren, H; Walsh, T. (editors). *Handbook of Satisfiability, volume 185 of Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

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

[3] Cook, S. A. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151-158, ACM Press, 1971.

[4] Janičić, P. URSA: A System for Uniform Reduction to SAT. *Logical Methods in Computer Science*, 8(3:30):1-39, 2012.

[5] Lippold, D. Legality of Positions of Simple Chess Endgames, Endgame Tablebases Web Page, http://archive.is/hXsdC, downloaded: July 15th 2012.

[6] Lippold, D. The legitimacy of positions in endgame databases. *ICCA Journal*, 20(1):20-28, 1997.

[7] Maliković, M; Čubrilo, M. What Were the Last Moves?. *International Review on Computers and Software*, 5(1):59-70, 2010.

[8] Maliković, M; Janičić, P. Proving Correctness of a KRK Chess Endgame Strategy by SAT-based Constraint Solving. *ICGA Journal*, 36(2):81-99, 2013.

[9] Thompson, K. Retrograde analysis of certain endgames. *ICCA Journal*, 9(3):131-139, 1986.

[10] Thompson, K. 6-piece Endgames. *ICCA Journal*, 19(4): 215-226, 1996.

[11] Wassong, P. The Natch home page, http://natch.free.fr, downloaded: December 4th 2013.

[12] Wendroff, B; Warnock, T; Stiller, L; Mayer, D; Brickner, R. Bits and pieces: constructing chess endgame databases on parallel and vector architectures. *Applied Numerical Mathematics*, 12(1-3): 285-295, 1993.

[13] Wilts, G; Frolkin, A. *Shortest Proof Games.* Privately published in Karlsruhe, 1991.