Formalization of a Strategy for the KRK Chess Endgame

Marko Maliković
Faculty of Humanities and Social Sciences
University of Rijeka
Slavka Krautzeka bb, 51000 Rijeka, Croatia
marko.malikovic@ffri.hr

Mirko Čubrilo
Faculty of Organization and Informatics
University of Zagreb
Pavlinska 2, 42000 Varaždin, Croatia
mirko.cubrilo@foi.hr

Predrag Janičić
Faculty of Mathematics
University of Belgrade
Studentski trg 16, 11000 Belgrade, Serbia
janicic@matf.bg.ac.rs

Abstract. *Chess has always been a challenging subject for various computer analyses and methodologies, and they often brought more general advances in the related computer science fields, such as search strategies, AI planning, data-mining, etc. However, interactive theorem proving has hardly been applied to chess. In this paper we present our formalization, within the Coq proof assistant, of one fragment of the chess game - KRK chess ending and several conjectures relevant for this endgame. We show that most of the considered notions and conjectures can be expressed in a simple theory such as linear arithmetic. In addition, in this paper we present a formalization of Bratko's strategy for the KRK endgame. The presented formalization will serve as a key step towards formal correctness proof for Bratko's strategy.*

Keywords. Interactive theorem proving, Coq, automated theorem proving, linear arithmetic, chess endgames, KRK chess endgame

1 Introduction

Over the last years interactive theorem proving has been successfully used for proving a number of complex theorems and for building a corpus of verified mathematical and computer science knowledge. These efforts lead to a „database of all important, established mathematical knowledge, strictly formalized and with all proofs having been checked automatically“ [1]. Interactive theorem proving is typically applied in pure mathematics and computer science, but other reasoning tasks can also be subject to formal analysis and interactive theorem proving. Chess, as a prototype of an intellectual game, is one example of such domain. Chess has always been a challenging subject of various computer analyses and methodologies, and they often brought more general advances in the related computer science fields, such as search strategies, AI planning, data-mining, etc. However, interactive theorem proving has hardly been applied to chess. In this paper we present our formalization of one fragment of the chess game (one chess ending) within the Coq proof assistant. Within the task of formalizing chess, we had two primary motivations:

- To show that the game of chess can be suitably described within a relatively simple theory such as linear arithmetic. Moreover, strategies for chess endings and their correctness can also be described, to a large extent, in terms of linear arithmetic. Such a description of chess and chess strategies can be used as a basis for formalization within a proof assistant such as Coq.
- To explore practical potentials of automation available within a proof assistant such as Coq, primarily automation related to decidable theories such as linear arithmetic.

We would like to point out that, as for many other problems treated by interactive theorem proving, proving correctness of chess endgames is not safety critical. Still, it is plausible to have machine verifiable proofs for such domains as well because:

- machine verifiable proofs often reveal flaws in known informal proofs;
- machine verifiable proofs provide building blocks that can be used for other conjectures in the same domain or even in some other domains;
- machine verifiable proofs are becoming a golden standard for all mathematical proofs;
- newly constructed proofs and the growing body of formally proved conjectures help further development of the technology of interactive theorem proving.

In formalizing and proving correct an endgame strategy, there are three stages, each with their own specifics and challenges:

- formalization of relevant chess rules;
- formalization of the strategy itself;

- formalization and proving of the correctness conjecture.

Within this paper, we will address the first two of the three stages above.

We are not aware of other formalizations of chess strategies within a proof assistant. There is a paper on retrograde chess analysis within Coq but it does not consider chess strategies [21].

The rest of the paper is organized as following: in Section 2 we give a brief background information on interactive theorem proving, Coq, linear arithmetic and chess endgame strategies. In Section 3, we outline the considered endgame strategy and present some analyses of its performance. In Section 4 we present our formalization of the relevant chess rules, in Section 5 we present our formalization of the considered chess endgame strategy and in Section 6 we draw final conclusions and discuss potential further work.

2 Background

In this section we give a background relevant to the work presented in this paper. First we briefly discuss interactive theorem proving and the system Coq, then linear arithmetic and how it can be automated in Coq, and finally the chess game and chess endgame strategies.

2.1 Interactive Theorem Proving and Coq

Interactive theorem proving is a process of development of formal mathematical proofs by interaction between a computer and a human. In this interaction, the computer is equipped with a proof assistant tool (i.e., "an interactive theorem prover") that checks and guides steps performed by the human, by verifying each proof step with respect to the given underlying logic. The importance of interactive theorem proving comes from the fact that "traditional proofs" most often are not proofs at all, because of the many missing fragments, informal arguments, etc. Interactive theorem proving uncovered many flaws in many mathematical proofs. On the other hand, proofs constructed within proof assistants are verbatim and detailed, and typically much longer than "traditional proofs" [3]. When checking proofs, correctness of proof assistants themselves is also critical. Proof assistants often have a very small kernel that checks all derivations, according to de Bruijn criterion [2]. This small core can consist of just tens of lines of code and can be manually verified.

Interactive theorem proving gets more and more popular and the body of formalized both classical and modern mathematical and computer science knowledge is increasing. There are also significant theorems proved for the first time thanks to proof assistants. Some of the most popular modern proof assistants are Coq, Isabelle, HOL Light, PVS, Mizar, ACL2, etc [29].

The Coq system [5], [17], [25] is implemented in Objective Caml and works within the theory of the calculus of inductive constructions (CIC). This theory is a typed λ calculus with polymorphism, dependent types and a primitive notion of inductive types. Coq also provides a dependently typed functional programming language. However, since Coq follows the propositions-as-types, proofs-as-programs Curry-Howard interpretation, the distinction between programming and proving is blurred. In Coq, a tactic, described in the language Ltac [12], is a program which expresses the sequence of basic logical steps. Coq has over 150 tactics that assist the user in developing a formal proof. Proofs in Coq are mainly built in interactive fashion, but there are various decision procedures and tactics based on automatic theorem proving that provide automation. Coq has been used in a wide range of domains and for proving a number of complex conjectures - for instance, for the four colour theorem [14], the fundamental theorem of algebra [13], for implementing and proving correct methods for automatically proving theorems in geometry [19], for proving correctness of a compiler [20], etc. Coq has been used in formalizing reasoning tasks not in pure mathematics or computer science, like solving sudoku problems or solving 2×2 Rubik's cube.[1]

2.2 Linear Arithmetic

Linear arithmetic (or Presburger arithmetic) is a fragment of Peano arithmetic that uses only addition (and not multiplication). In linear arithmetic, multiplication by a constant number is allowed, and nx is just a shorthand for $x+x+\ldots+x$ where x occurs n times. For subtraction it holds $x \quad y = 0$ if $x < y$. In contrast to the whole of arithmetic, linear arithmetic is decidable [22]. This theory is rather simple, but still expressible enough for many applications in computer science [4].

There are several decision procedures for linear arithmetic [18]. Like decision procedures for other theories, there are several methods to add decision procedures for linear arithmetic to Coq [15], [6], [7]. This task usually requires a substantial engineering efforts and a profound understanding of the internals of the decision procedure. One of the approaches is to implement the decision procedure in Ltac or in Ocaml[2], so that for each input instance it produces a proof term that can be checked by Coq. This approach is used by the *romega* tactic, by Crégut [11], [25]. This tactics provides decision procedure for quantifier-free linear arithmetics over natural numbers. It generates Coq proof terms from traces obtained from the Omega test [23].

[1] http://www-sop.inria.fr/marelle/Laurent.Thery/me.html
[2] Ocaml is the implementation language of Coq.

2.3 Chess and Chess Endgame Strategies

We assume that the reader is familiar with the chess rules, so we do not describe them here. KRK denotes the chess ending with white player having a king and a rook, and black player having only a king.

Chess endgames (including KRK, as one of the simplest) are often studied by exhaustive analysis and in the context of tablebases. An endgame tablebase is an ordered list of all positions in the endgame with interesting values precalculated [10]. First endgame tablebases were constructed by Thompson [26], [27]. Building a chess tablebase is typically based on retrograde analysis: first, all mate positions are listed, then all positions that can lead to those positions, etc. Building a tablebase does not require deep insights or complex algorithms, but can be computationally or memory expensive for some endgames. Also, tablebases sometimes reveal chess knowledge that is not considered by or relevant for typical chess players. For instance, tablebases can detect positions with optimal strategy leading to mate only after 250 moves. Such revelations obtained by tablebases may be intriguing, but actually they don't provide much of typical chess knowledge and insights: "current state of the art of machine-learning programs is that many ad hoc recipes are produced. Moreover, they are hardly intelligible to human experts. In fact, the database itself is a long list of ad hoc recipes" [28]. This motivates work on constructing strategies for specific endings and automated analysis of tablebases based on machine learning and data mining and extraction of human understandable knowledge of winning strategies [16], [10]. Automatically extracted strategies still cannot compete with human constructed strategies.

3 Bratko's Strategy for KRK

Bratko's strategy for white for the KRK ending can be outlined as follows [8], [9]:

1. Look for a way to mate black in two moves;
2. If the above is not possible, then look for a way to further constrain the area on the chess board to which the black king is confined by white rook;
3. If the above is not possible, then look for a way to move the king closer to the black king so as to help the rook in squeezing the black King;
4. If none of the above pieces of advice 1, 2, 3 works, then look for a way of maintaining the present achievements in the sense of 2 and 3 (i.e. make a waiting move);
5. If none of 1, 2, 3 or 4 is attainable then look for a way of obtaining a position in which the rook divides the two kings either vertically or horizontally.

Actually, the strategy has a number of hidden details [8], [9] and this shows that it is very difficult to have a simple winning strategy (not to mention an optimal strategy) even for a simple ending such as KRK. Some details of the strategy will be given in Section 5.

Bratko's strategy is the subject of analysis in this paper. It is correct, i.e., from any position with white to move, following Bratko's strategy white wins. The correctness can be proved in different ways. Bratko gave one high-level proof of correctness of his strategy but that proof is informal [9].

Our goal was to formalize the strategy – make this first critical step towards formally proving that the strategy is correct. The formal proof can follow Bratko's informal proof, but it is expected that many missing steps will have to be filled or some flaws corrected. Note that Bratko's strategy is not optimal and we do not address the optimality issue within our formalization.

4 Formalization of Chess Rules

In order to have suitable machine verifiable correctness proof for a chess endgame strategy, it is critical to have a simple and intuitive core that defines the chess rules and the strategy itself. Namely, once the proofs are constructed, they are verified by the proof assistant and are not subject to any doubt. What could cause some doubt is the formulation of the central conjecture, which boils down to basic definitions. Thus, it is essential to have the chess rules (and also Bratko's strategy) defined in a simple, concise, intuitive, and convincing way. In this section we will discuss some design decisions and will focus only on the fragment of the game relevant for the KRK ending. Our design decisions were largely motivated (apart from the quest for simplicity) by our aim to automate as much reasoning as possible. We will show that the chess rules can be simply described in terms of linear arithmetic over natural numbers[3] which would be beneficial since a significant portion of the proving process can be automated by decision procedures for linear arithmetic. In the rest of the material, if not explicitly stated otherwise, it is assumed that the underlying theory is linear arithmetic over natural numbers. We will not give the full formalization in Coq, but only some of its fragments, for illustration.

Chessboard and positions. There are several options for describing the chessboard and chess positions, including the following two natural options:

[3] Of course, chess rules and endgame strategy can be described in terms of even simpler theories, for instance - propositional logic. However, we find that linear arithmetic better suits that purpose - within it all conditions can be represented in a compact and intuitive way.

1. represent the chessboard as an 8×8 array (or as a list of lists in Coq) with each element containing a distinguished value for empty field or for a specific chess piece of specific color (this approach was used in the earlier work in retrograde chess analysis [21]);
2. associate each piece (possibly) on the board with a pair of its coordinates.

The former approach could be more suitable for the full chess game but the latter turns to be much more suitable for a restricted variant of the game – with only three pieces, as in the KRK ending. Namely, instead of dealing with values of 64 squares of the chessboard, only six values are considered. In addition, if one aims at exploiting a decision procedure for linear arithmetic, in the first approach he/she would have to get rid of lists in conjectures before trying to prove them. For the latter approach, extensions from one chess endgame to another, or to a full game are possible, but not very elegant (since variables for each new chess piece have to be introduced).

In the following text, the second approach for representing the chessboard will be assumed. A position is defined as a record type which is a common type for a bundle of objects: the coordinates of the white king (*WKx,WKy*), the coordinates of the white rook (*WRx,WRy*), the coordinates of the black king (*BKx,BKy*), and the value *OnTurn* on which player is on turn that can be *W=1* or *B=2*. The notion of position is represented in Coq as follows:

```
Record Position := position {WKx : nat; WKy : nat; WRx : nat; WRy :
nat; BKx : nat; BKy : nat; OnTurn : nat}.
```

Although the white can theoretically castle in some positions, this information is not maintained (and castling is not considered as a legal move). Also, information relevant for the „fifty-moves rule“ is not maintained.[4]

It has to be ensured that in each position, the value for each coordinate is between 1 and 8. In linear arithmetic over natural numbers, however, it is more suitable (and is less computationally demanding) to use zero-based representation of rows and columns, so constraints for all coordinates in one position should be only that they are less than or equal to 7, which can be represented in Coq as follows:

```
Definition ChessboardDimension (P : Position) := WKx P <= 7 /\
WKy P <= 7 /\ BKx P <= 7 /\ BKy P <= 7 /\ WRx P <= 7 /\ WRy P <= 7.
```

Coordinates (*WKx,WKy*) are always associated to some position *P* (e.g, *WKx P*) but, for simplicity, in the following informal text (not in Coq code, of course), we will often skip writing the relevant position *P*.

[4] Still, as said, within the correctness proof for Bratko's strategy it is shown that the "fifty-moves rule" is obeyed.

Legal positions. The constraints on legal positions can be represented in terms of linear arithmetic. For instance, the white king and the white rook cannot be on the same square:

```
Definition NotOnSameSquare (P : Position) :=
WKx P <> WRx P \/ WKy P <> WRy P.
```

Also, the rule that the two kings cannot be on the same or adjacent squares can be expressed as:

```
Definition NotKingNextKing (P : Position) := WKx P > BKx P + 1 \/
BKx P > WKx P + 1 \/ WKy P > BKy P + 1 \/ BKy P > WKy P + 1.
```

For defining legal positions, a condition that the black king is attacked by the white rook:

```
Definition BlackKingAttacked (P : Position) :=
WRx P = BKx P /\ (WKx P <> WRx P \/ WKx P = WRx P /\ (WKy P <=
BKy P /\ WKy P <= WRy P \/ BKy P <= WKy P /\ WRy P <= WKy P))
\/
WRy P = BKy P /\ (WKy P <> WRy P \/ WKy P = WRy P /\ (WKx P <=
BKx P /\ WKx P <= WRx P \/ BKx P <= WKx P /\ WRx P <= WKx P)).
```

A position *P* is legal if all coordinates of all pieces are less than or equal to 7, if the white king and the white rook are not on the same square, if the two kings are not on the same or adjacent squares and if the black king is not attacked when the white is on turn:

```
Definition LegalPosition (P : Position) :=
ChessboardDimension P /\ NotOnSameSquare P /\
NotKingNextKing P /\ ~(BlackKingAttacked P /\ OnTurn P = W).
```

However, there are still some subtle issues concerning legal positions. Let us consider the position on Fig. 1.

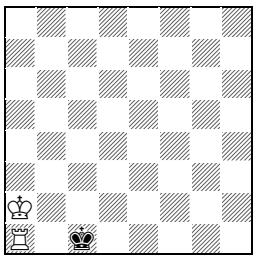

Figure 1. An example of illegal position

According the above definition, this position is legal only if the white is not on turn. But, if the black is on turn, what was the last move of the white? It can be easily checked that there was no legal move of the white that could have led to the current position, so the given position is impossible. Because of such situations (subject to retrograde chess analysis [21]), it is difficult to concisely define legal positions. The correct definition would be that a position is legal if it is reachable from the initial chess position, but such definition is practically useless.

Nonetheless, the above definition of the legal position is sufficient for the purposes of presented work because we consider a strategy of white, therefore only the initial positions in which the white is on turn.

Legal moves. The rules for moving pieces can also be simply described in terms of linear arithmetic. They are divided into: (*i*) parts specifying movements rules themselves; (*ii*) a constraint stating that all other pieces remained on their original positions if not captured by the moving piece; (*iii*) a condition stating 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:

```
Definition MoveWhiteKing (P1 P2 : Position) :=
WKx P2 - WKx P1 <= 1 /\ WKx P1 - WKx P2 <= 1 /\
WKy P2 - WKy P1 <= 1 /\ WKy P1 - WKy P2 <= 1 /\
(WKx P1 <> WKx P2 \/ WKy P1 <> WKy P2).
```

and the part (*i*) specifying that after a move of the white king all other pieces remained on their original positions:

```
Definition OtherAfterMoveWhiteKing (P1 P2 : Position) :=
BKx P2 = BKx P1 /\ BKy P2 = BKy P1 /\
WRx P2 = WRx P1 /\ WRy P2 = WRy P1.
```

A definition of a legal move of the white king (involving notions outlined above) is:

```
Definition LegalMoveWhiteKing (P1 P2 : Position) :=
MoveWhiteKing P1 P2 /\
OtherAfterMoveWhiteKing P1 P2 /\
OnTurn P1 = W /\ OnTurn P2 = B /\
LegalPosition P2.
```

Note that, following the representation of the chessboard, movement rules for both kings have to be specified - the related definition *LegalMoveBlackKing* is analogous to the one given above.

Mate, stalemate and draw. Positions that are mate, stalemate or draw are defined simply by using the introduced definitions. For instance, a position *P* is mate (black is mated) if black is checked and black has no legal moves:

```
Definition Mate (P : Position) := BlackKingAttacked P /\
OnTurn P = B /\ forall P' : Position, ~ LegalMoveBlack P P'.
```

Stalemate is defined similarly:

```
Definition Stalemate (P : Position) := ~BlackKingAttacked P /\
OnTurn P = B /\ forall P' : Position, ~LegalMoveBlack P P'.
```

Draw (that occurs if the white rook has been captured) and the terminating position are defined as follows:

```
Definition Draw (P : Position) := OnTurn P = W /\ BKx P = WRx P /\
BKy P = WRy P.

Definition GameEnd (P : Position) :=
Mate P \/ Stalemate P \/ Draw P.
```

The above definition of mate is simple and intuitive, but there is one drawback. Within the first step of Bratko's strategy, it is required to check if the position is mate-in-two-moves. This check can be represented by a definition simulating minimax search, i.e., a definition that involves alternation of quantifiers. Of course, that is legitimate, but would disable automation in proving conjectures involving this definition i.e. using a procedure for quantifier-free fragment of linear arithmetic. That is why we derived an explicit definition of mate-in-two-moves – step by step, firstly by explicitly defining mate, mate-in-one-ply, mate-in-two-plies, and mate-in-three-plies positions. In order to simplify this task, we used symmetries, so mating situations were explicitly described one for one edge or one corner. As an example, we give definitions of concrete mating situations and the explicit definition of mate (Symmetric defines eight sorts of symmetries between two chess positions):

```
Definition MateEdgeOneCase (P : Position) :=
BKx P = 0 /\ WKx P = 2 /\ BKy P = WKy P /\ WRx P = 0 /\ WRy P >=
WKy P + 2 /\ ChessboardDimension P /\ OnTurn P = B.

Definition MateCornerOneCase (P : Position) :=
BKx P = 0 /\ BKy P = 0 /\ WKx P = 2 /\ WKy P = 1 /\ WRx P = 0 /\
WRy P = 2 /\ ChessboardDimension P /\ OnTurn P = B.

Definition MateConcrete (P : Position) := exists PS : Position,
(MateEdgeOneCase PS \/ MateCornerOneCase PS) /\
Symmetric P PS.
```

In the definition stated above, instead of the condition *LegalPosition P* the weaker condition *ChessboardDimension P* is used. It is sufficient for our purposes, discussed below.

The last definition of mate (*MateConcrete*) is non-trivial and involves concrete positions and symmetries. Hence, because of quest for simplicity we don't want to use it as an alternative for the first, implicit definition. Instead, we proved that the explicit mate implies the implicit mate (it can be proved that the opposite also holds, but that is not required for proving the strategy correct):

```
Lemma MateConcreteIsValid :
forall P : Position, MateConcrete P -> Mate P.
```

Moreover, we prove that mate, mate-in-one-ply, mate-in-two-plies, and mate-in-three-plies positions are related in the expected way:

```
Lemma
MateInOnePlyConcreteLeadsToMateConcrete :
forall P1 : Position, MateInOnePlyConcrete P1 -> exists P2 :
Position, LegalMoveWhiteRook P1 P2 /\ MateConcrete P2.
```

```
Lemma
MateInTwoPliesConcreteLeadsToMateInOnePlyConcrete :
forall P1 : Position, MateInTwoPliesConcrete P1 -> forall P2 :
Position, LegalMoveBlack P1 P2 -> MateInOnePlyConcrete P2.

Lemma
MateInThreePliesConcreteLeadsToMateInTwoPliesConcrete :
forall P1 : Position, MateInThreePliesConcrete P1 -> exists P2 :
Position, LegalMoveWhite P1 P2 /\ MateInTwoPliesConcrete P2.
```

A simple consequence of the lemmas above is that in mate-in-one-ply or mate-in-three-plies position the white can mate.

5 Formalization of Bratko's KRK Strategy

Formalization of Bratko's strategy for KRK poses new challenges for formalization within linear arithmetic. For instance, the strategy extensively uses the notion of "room", i.e., the area of the chessboard in which the black king is and that is guarded by the white rook (see Fig. 2).

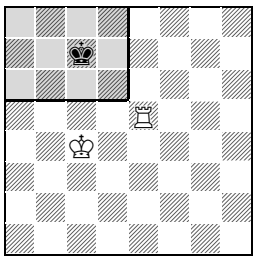

Figure 2. Room area

The area is rectangular and its area equals $m \cdot n$, where m and n are lengths of its sides. In some steps of the strategy, the rook has to move in such a way that this area decreases but this cannot be expressed in terms of linear arithmetic (because of multiplication). However, a simple insight still enables using linear arithmetic: if the rook moves, only one of m and n changes, and since the formula (implicitly universally quantified):

$$x = z \vee y = u \Rightarrow \left(x \cdot y < z \cdot u \Leftrightarrow x + y < z + u\right)$$

is valid in linear arithmetic, it is sufficient, in this case, to consider the area as $m + n$, not as $m \cdot n$.

The formalization of Bratko's strategy includes details hidden in the strategy overview given in Section 3. For example, Room and the condition that white has to reduce this Room is defined as follows:

```
Definition Room (P : Position) :=
match (WRx P - BKx P) + (BKx P - WRx P) with
| 0 => 15
| _ => match (WRy P - BKy P) + (BKy P - WRy P) with
          | 0 => 15
          | _ => match BKx P - WRx P with
                    | 0 => match BKy P - WRy P with
                              | 0 => WRx P + WRy P
                              | _ => WRx P + (7 - WRy P)
                              end
                    | _ => match BKy P - WRy P with
                              | 0 => (7 - WRx P) + WRy P
                              | _ => (7 - WRx P) + (7 - WRy P)
                              end
                    end
          end
end.

Definition NewRoomSmaller (P1 P2 : Position) :=
Room P1 > Room P2.
```

We don't explain other components of the strategy in detail, but most of them should be easily understandable from the explanations given in the previous text:

```
Definition MateIn2 (P1 : Position) :=
  MateInOnePlyConcrete P1 \/ MateInThreePliesConcrete P1.

Definition SqueezeCond (P1 : Position) :=
  ~MateIn2 P1 /\ (exists P2 : Position, LegalMoveWhiteRook P1 P2
  = SqueezeMove P1 P2 /\ (forall P3 : Position, LegalMoveBlack P2
  P3 /\ NewRoomSmaller P1 P3 /\ NotWhiteRookExposed P2 /\
  WhiteRookDivides P2 /\ ~Stalemate P2)).

Definition ApproachCond (P1 : Position) :=
  ~MateIn2 P1 /\ ~SqueezeCond P1 /\ (exists P2 : Position,
  LegalMoveWhiteKing P1 P2 = ApproachMove P1 P2 /\ (KingDiag
  P1 P2 \/ ~KingDiag P1 P2 /\ KingNotDiag P1 P2) /\
  ApproachCriticalSquare P1 P2 /\ NotWhiteRookExposed P2 /\
  (WhiteRookDivides P2 \/ LPattern P2) /\ (RoomGt3 P2 \/
  ~WhiteKingEdge P2)).

Definition KeepRoomCond (P1 : Position) :=
  ~MateIn2 P1 /\ ~SqueezeCond P1 /\ ~ApproachCond P1 /\
  (exists P2 : Position, LegalMoveWhiteKing P1 P2 =
  KeepRoomMove P1 P2 /\ (KingDiag P1 P2 \/ ~KingDiag P1 P2 /\
  KingNotDiag P1 P2) /\ NotWhiteRookExposed P2 /\
  WhiteRookDivides P2 /\ WhiteKingAndRookNotDiverging P1 P2
  /\ (RoomGt3 P2 \/ ~ WhiteKingEdge P2)).

Definition DivideIn2Cond (P1 : Position) :=
  ~MateIn2 P1 /\ ~SqueezeCond P1 /\ ~ApproachCond P1 /\
  ~KeepRoomCond P1 /\ (exists P2 : Position, LegalMoveWhite P1
  P2 = DivideIn2Move P1 P2 /\ (forall P3 : Position,
  LegalMoveBlack P2 P3 /\ (exists P4 : Position, LegalMoveWhite
  P3 P4 /\ WhiteRookDivides P4 /\ NotWhiteRookExposed P4))).

Definition DivideIn3Cond (P1 : Position) :=
  ~MateIn2 P1 /\ ~SqueezeCond P1 /\ ~ApproachCond P1 /\
  ~KeepRoomCond P1 /\ ~DivideIn2Cond P1 /\ (exists P2 :
  Position, LegalMoveWhite P1 P2 = DivideIn2Move P1 P2 /\
  (forall P3 : Position, LegalMoveBlack P2 P3 /\ (exists P4 :
  Position, LegalMoveWhite P3 P4 /\ (forall P5 : Position,
  LegalMoveBlack P4 P5 /\ (exists P6 : Position, LegalMoveWhite
  P5 P6 /\ WhiteRookDivides P6 /\ NotWhiteRookExposed P6))))).

Definition Strategy (P1 P2 : Position) :=
  MateIn2 P1 \/
  SqueezeMove P1 P2 \/
  ApproachMove P1 P2 \/
  KeepRoomMove P1 P2 \/
  DivideIn2Move P1 P2 \/
  DivideIn3Move P1 P2.
```

6 Conclusions and Future Work

In this paper we presented our formalization in Coq of the KRK chess endgame and Bratko's strategy for the white player for this endgame. We showed that, with some observations, the most of the considered notions and conjectures can be expressed in a simple theory of linear arithmetic and it appears that the whole of the chess game can also be suitably represented in this theory. Concerning the strategy itself, our formalization led to some simplifications and revealed some important details neglected or omitted in the original presentation. For instance, we proved that the notion of "room" can be expressed with addition instead of multiplication and we detected that Bratko's PROLOG implementation is incorrect for some positions. Our formalization is, to our knowledge, the first non-trivial formalized chess knowledge.

For our future work, in order to prove correctness of Bratko's strategy, we plan to use various sorts of automation and to explore the limits of automation for linear arithmetic within Coq. We are also planning to formally (within a proof assistant) analyze other chess endgames, but also other sorts of chess problems.

References

[1] Anonymous: The qed manifesto, In Proceedings of the 12th International Conference on Automated Deduction – CADE-12, volume 814 of Lecture Notes in Computer Science, Springer, 1994, pp. 238-251.

[2] Barendregt, H., Barendsen, E.: Autarkic computations in formal proofs, Journal of Automated Reasoning, Vol. 28, No. 3, 2002, pp. 321-336.

[3] Barendregt, H., Wiedijk, F.: The challenge of computer mathematics, Philosophical Transactions of the Royal Society, Vol. 363, No. 1835, 2005, pp. 2351-2375.

[4] Barrett, C., Sebastiani, R., Seshia, S. A., Tinelli, C.: Satisfiability Modulo Theories, volume 185 of Frontiers in Artificial Intelligence and Applications, chapter 26, IOS Press, 2009, pp. 825-885.

[5] Bertot, Y. Castéran, P.: Interactive Theorem Proving and Program Development, Springer-Verlag, 2004.

[6] Besson, F.: Fast reflexive arithmetic tactics the linear case and beyond, In Types for Proofs and Programs, International Workshop, TYPES 2006, volume 4502 of Lecture Notes in Computer Science, Springer, 2006, pp. 48-62.

[7] Boutin, S.: Using reflection to build efficient and certified decision procedures, In Abadi, M., Ito, T. (editors), Proceedings of TACS'97, volume 1281 of Lecture Notes in Computer Science. Springer-Verlag, 1997.

[8] Bratko, I.: PROLOG Programming for Artificial Intelligence, Addison-Wesley, 1990.

[9] Bratko, I.: Proving correctness of strategies in the AL1 assertional language, Information Processing Letters, Vol. 7, No. 5, 1978, pp. 223-230.

[10] Breda, G.: Krk chess endgame database knowledge extraction and compression, Master's thesis, Technische Universität Darmstadt, 2006.

[11] Crégut, P.: Une procédure de décision réflexive pour un fragment de l'arithmétique de presburger, In Informal proceedings of the 15th Journées Francophones des Langages Applicatifs, Charente-Maritime, 2004.

[12] Delahaye, D.: A Tactic Language for the System Coq, In Parigot, M., Voronkov, A. (editors), Logic for Programming and Automated Reasoning, volume 1955, Springer, 2000, pp. 85-95.

[13] Geuvers, H. et. al.: The "Fundamental Theorem of Algebra" Project, available at `http://www.cs.ru.nl/~freek/fta/`, Accessed: 27th April 2008.

[14] Gonthier, G.: Formal Proof–The Four-Color Theorem, Notices of the American Mathematical Society, Vol. 55, No. 11, 2008, pp. 1382–1393.

[15] Grégoire, B., Mahboubi, A.: Proving equalities in a commutative ring done right in coq, In Hurd, J., Melham, T. F. (editors), Theorem Proving in Higher Order Logics, TPHOLs 2005, volume 3603 of Lecture Notes in Computer Science, Springer, 2005, pp. 98-113.

[16] Guid, M., Mozina, M., Sadikov, A., Bratko, I.: Deriving concepts and strategies from chess tablebases, In Advances in Computer Games, ACG 2009, volume 6048 of Lecture Notes in Computer Science, Springer, 2010, pp. 195-207.

[17] Huet, G., Kahn, G., Paulin-Mohring, C.: The Coq Proof Assistant - A Tutorial, available at `http://coq.inria.fr/distrib/current/f`

iles/Tutorial.pdf, Accessed: 26th December 2011.

[18] Janičić, P., Green, I., Bundy, A.: A comparison of decision procedures in Presburger arithmetic, In Tošić, R., Budimac, Z. (editors), Proceedings of the VIII Conference on Logic and Computer Science (LIRA '97), Novi Sad, Yugoslavia, September 1-4, University of Novi Sad, 1997. Also available from Edinburgh as DAI Research Paper No. 872, pp. 91-101.

[19] Janičić, P., Narboux, J., Quaresma, P.: The area method: a recapitulation, Journal of Automated Reasoning, 2012. To appear.

[20] Leroy, X.: Formal certification of a compiler back-end, or: programming a compiler with a proof assistant, In 33rd symposium Principles of Programming Languages, ACM Press, 2006, pp. 42-54.

[21] Maliković, M., Čubrilo, M.: What were the last moves?, International Review on Computers and Software, Vol. 5, No. 1, 2010, pp. 59-70.

[22] Presburger, M.: Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt, In Sprawozdanie z I Kongresu metematyków slowiańskich, Warszawa, 1929, pp. 92-101.

[23] Pugh, W.: The omega test: a fast and practical integer programming algorithm for dependence analysis, In ACM/IEEE conference on Supercomputing, 1991, pp. 4-13.

[24] Stansifer, R.: Presburger's Article on Integer Arithmetic: Remarks and Translation, Technical Report TR 84-639, Department of Computer Science, Cornell University, September 1984.

[25] The Coq development team: The Coq proof assistant reference manual, Version 8.3, TypiCal Project, 2012.

[26] Thompson, K.: Retrograde analysis of certain endgames, International Computer Chess Association Journal, Vol. 9, No. 3, 1986, pp. 131-139.

[27] Thompson, K.: 6-piece endgames, International Computer Chess Association Journal, Vol. 19, No. 4, 1996, pp. 215-226.

[28] van den Herik, H. J., Uiterwijk, J. W. H. M., van Rijswijck, J.: Games solved: Now and in the future, Artificial Intelligence, Vol. 134, No. 1-2, 2002, pp. 277-311.

[29] Wiedijk, F. (editor): The Seventeen Provers of the World, volume 3600 of Lecture Notes in Computer Science, Springer, 2006.