Formal Verification of Chess Endgame Databases

A case study in combining theorem proving and model checking

Joe Hurd

Computing Laboratory
Oxford University

ARG Lunch

Talk Plan

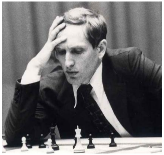
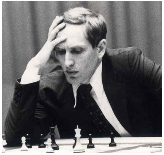
1. Combining Theorem Proving and Model Checking
    - Introduction to Theorem Proving and Model Checking
    - Combination Methods
    - The HolCheck Approach
2. Case Study: Chess Endgame Databases
    - Modelling the Two Player Game of Chess
    - Constructing Verified Chess Endgame Databases
    - Applications
3. Summary

Talk Plan

1. Combining Theorem Proving and Model Checking
   - Introduction to Theorem Proving and Model Checking
   - Combination Methods
   - The HolCheck Approach
2. Case Study: Chess Endgame Databases
   - Modelling the Two Player Game of Chess
   - Constructing Verified Chess Endgame Databases
   - Applications
3. Summary

Theorem Proving

- LCF-style theorem proving emphasizes high assurance.
  - Theorems can only be created by a logical kernel, which implements the inference rules of the logic.
- Higher order logic is expressive enough to naturally define many concepts of mathematics and formal language semantics:
  - probability via real analysis and measure theory;
  - the Property Specification Language for hardware.
- The main challenge is proof automation.

 $\rightsquigarrow$  $\rightsquigarrow$ 

Model Checking

- Model checking emphasizes automation.
  - Various efficient algorithms for deciding temporal logic formulas on finite state models.
- High level input languages support the modelling and checking of complex computer systems:
  - IEEE Futurebus+ cache coherence protocol.
- The main challenge is to reduce problems to a form in which they can be efficiently model checked.

Combination Methods (1)

- Approach 1: incorporate theorem proving techniques into existing model checkers:
  - disjunctive partitioning of transition relations;
  - assume-guarantee reasoning;
  - data abstraction.
- This approach extends the reach of state of the art model checkers:
  - enabling automatic verification of ever larger state spaces;
  - and even some infinite state systems.

Combination Methods (2)

- Approach 2: implement model checking algorithms using existing theorem provers as programming languages.
- Gordon created a set of inference rules relating higher order logic formulas and BDDs:

$$\frac{[a_1] \vdash t_1 = t_2 \qquad [a_2]\ t_1 \ \mapsto\ b}{[a_1 \cup a_2]\ t_2 \ \mapsto\ b}$$

- Amjad implemented a modal μ-calculus model checker called *HolCheck* as a derived inference rule in HOL4.
  - The resulting theorems depend only on the inference rules of HOL4 and the BuDDy BDD engine.
  - Used to verify several correctness properties of the AMBA bus architecture.

The HolCheck Approach

- Higher order logic is a common semantics in which to embed many logics.
- HOL4 can be used a scripting platform to implement verification tools.
  - Pro: No error-prone translation between tools.
  - Con: Performance penalty for implementing as a HOL4 derived rule (about 30% for *HolCheck*).
- Example: using a formalization of PSL semantics to translate hardware properties to Verilog monitors.
- This talk: using a formalization of the rules of chess to construct a verified chess endgame database.

Talk Plan

1. Combining Theorem Proving and Model Checking
   - Introduction to Theorem Proving and Model Checking
   - Combination Methods
   - The HolCheck Approach
2. Case Study: Chess Endgame Databases
   - Modelling the Two Player Game of Chess
   - Constructing Verified Chess Endgame Databases
   - Applications
3. Summary

Chess Endgame Databases

- Can solve certain classes of chess endgame by enumerating all positions in a database.
  - Compute depth to mate by working backwards from the checkmate positions.
  - Ken Thompson solved most five piece endgames, and the state of the art is now six piece endgames.
- Combine theorem proving and model checking to construct a verified endgame database:
  - model checking provides an automatic algorithm to construct the set of winning positions;
  - and implementing this algorithm in a theorem prover results in a theorem that the endgame database logically follows from the rules of chess.

Two Player Games

A two player game G is modelled in higher order logic with a four tuple

$$(L, M, \overline{M}, W)$$

- L is a predicate that holds on legal positions;
- M is the move relation for *Player I*;
- $\overline{M}$ is the move relation for *Player II*;
- and W is a predicate that holds on legal positions that are won for *Player I*.

Two Player Games: Terminal Positions

The set of terminal (stuck) positions for a two player game G:

$$\begin{aligned} \text{terminal1}\ G &\equiv \{p \mid L_G(p)\ \wedge\ \forall p'.\ \neg M_G(p,p')\} \\ \text{terminal2}\ G &\equiv \{p \mid L_G(p)\ \wedge\ \forall p'.\ \neg \overline{M}_G(p,p')\} \end{aligned}$$

Two Player Games: Winning Positions

The set of legal positions won for *Player I* within a fixed number of moves:

$$\text{win2\_by}\ G\ 0 \equiv \{p \mid W_G(p)\}$$

$$\text{win1\_by}\ G\ n \equiv \{p \mid \exists p'.\ M_G(p,p')\ \wedge\ p' \in \text{win2\_by}\ G\ n\}$$

$$\begin{aligned}&\text{win2\_by}\ G\ (n+1) \equiv \\ &\quad \text{win2\_by}\ G\ n\ \cup \\ &\quad (\{p \mid L_G(p)\ \wedge\ \forall p'.\ \overline{M}_G(p,p')\ \implies\ p' \in \text{win1\_by}\ G\ n\} \\ &\quad\ - \text{terminal2}\ G)\end{aligned}$$

Two Player Games: Winning Positions

The set of all legal positions won for *Player I*:

$$\begin{aligned} \text{win1}\ G &\equiv \{p \mid \exists n.\ p \in \text{win1\_by}\ G\ n\} \\ \text{win2}\ G &\equiv \{p \mid \exists n.\ p \in \text{win2\_by}\ G\ n\} \end{aligned}$$

An endgame database is simply the winning set of the two player game of chess.

Two Player Games: Simulation

A two player game G_1 simulates another game G_2 with lifting function f if:

- f is a surjective function from L_{G_1} to L_{G_2};
- every move in G_1 lifts to a move in G_2;
- for every move from $f(p_1)$ to p'_2 in G_2, there can be found a position p'_1 such that p_1 to p'_1 is a move in G_1;
- $W_{G_1}(p_1) \iff L_{G_1}(p_1) \land W_{G_2}(f(p_1))$.

The boolean model of chess simulates the natural model, which allows the winning set of positions to be lifted from the boolean model to the natural model.

Two Player Games: Restriction

A two player game G_1 is a restriction of another game G_2 if:

- $L_{G_1} \subseteq L_{G_2}$;
- every move in G_1 also occurs in G_2;
- there are no moves in G_2 from a position in L_{G_1} to a position outside L_{G_1};
- $W_{G_1} = W_{G_2} \cap L_{G_1}$.

This allows the winning set of positions on a restricted category of chess endgames to be lifted to the unrestricted model of chess.

Modelling Chess

Three different models of chess (without pawns or castling):

1. a natural model that aims to be a self-evidently correct model of the laws of chess;
2. a concrete model that concisely describes positions with a (small) fixed set of pieces on the board;
3. a boolean model that is a straightforward translation of the concrete model but only using boolean variables.

Verification strategy: A manual proof that the concrete model is a restricted simulation of the natural model, plus automatic boolification tools to connect the concrete and boolean models. Construct the winning sets in the boolean model using BDDs.

Modelling Chess

Three different models of chess (without pawns or castling):

1. a natural model that aims to be a self-evidently correct model of the laws of chess;
2. a concrete model that concisely describes positions with a (small) fixed set of pieces on the board;
3. a boolean model that is a straightforward translation of the concrete model but only using boolean variables.

Verification strategy: A manual proof that the concrete model is a restricted simulation of the natural model, plus automatic boolification tools to connect the concrete and boolean models. Construct the winning sets in the boolean model using BDDs.

Chess: A Natural Model

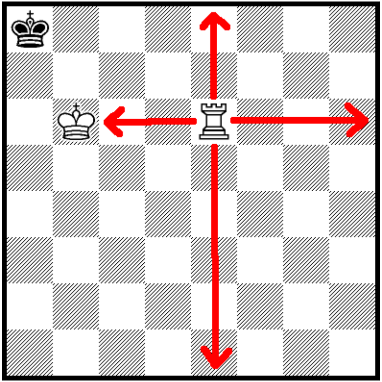
- Types:
  side $\equiv$ White | Black
  piece $\equiv$ King | Queen | Rook | Bishop | Knight
  square $\equiv \mathbb{N} \times \mathbb{N}$
  position $\equiv$ side $\times$ (square $\rightarrow$ (side $\times$ piece) option)
- Constants:
  file $(x, \_) \equiv x$ rank $(\_, y) \equiv y$
  board $\equiv \{sq \mid \text{file } sq < 8 \ \wedge\ \text{rank } sq < 8\}$
  on_square $(\_, f)\ sq \equiv f\ sq$
  empty $posn\ sq \equiv$ on_square $posn\ sq =$ NONE

Chess: A Natural Model

- Types:
  side $\equiv$ White | Black
  piece $\equiv$ King | Queen | Rook | Bishop | Knight
  square $\equiv \mathbb{N} \times \mathbb{N}$
  position $\equiv$ side $\times$ (square $\rightarrow$ (side $\times$ piece) option)
- Constants:
  file $(x, \_) \equiv x$ rank $(\_, y) \equiv y$
  board $\equiv \{sq \mid \text{file } sq < 8 \ \wedge \ \text{rank } sq < 8\}$
  on_square $(\_, f)\ sq \equiv f\ sq$
  empty $posn\ sq \equiv$ on_square $posn\ sq =$ NONE

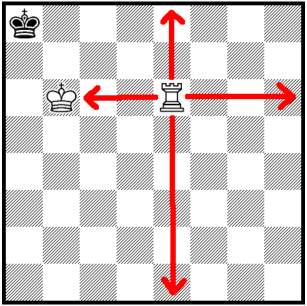
Chess: A Natural Model

Claim: it's easy to define the rules of chess in higher order logic.



Proof by example: define the set of squares that a rook attacks.

Chess: A Natural Model



$$\text{rook\_attacks}\ posn\ sq_1\ sq_2 \equiv$$
$$sq_1 \neq sq_2 \ \wedge\ (\text{file}\ sq_1 = \text{file}\ sq_2 \ \vee\ \text{rank}\ sq_1 = \text{rank}\ sq_2)$$
$$\wedge\ \forall sq.\ \text{square\_between}\ sq_1\ sq\ sq_2 \implies \text{empty}\ posn\ sq$$

Chess: A Concrete Model

- placement $\equiv$ (side $\times$ piece) $\times$ square
- posn $\equiv$ side $\times$ placement list
- Define a legal position predicate, move relations and winning position predicate as higher order logic functions.
- Due to the concrete nature of positions, these functions are just list manipulation and can be executed in the logic.
- Also define a lifting function abstract : posn $\rightarrow$ position.
- Hardest part of the verification: proving that this concrete model of chess is a simulation of the natural model ($\approx$ 2000 lines of tactic proof).

Chess: A Boolean Model

- Fix a *category* of chess positions: the side to move and a list of the pieces on the board.
- The only freedom left is the squares the pieces are on, and this is what needs to be translated to boolean variables.
- Note that every position in the same category translates to the same number of boolean variables.
- The user specifies the encoding, and then the automatic boolification in the HOL4 theorem prover takes over.
- 'Automatic' translations of the legal position predicate, move relations and winning position predicates happen by decoding and then rewriting with the definitions of the concrete model versions.

Verified Endgame Databases: Algorithm

- Build a verified endgame database by working backwards from checkmates, but symbolically using BDDs.
- When computing the set of positions won in $n + 1$ moves in a category C must consider the set of positions won in n moves in all the categories that can be reached from C in one move.
- Work up from the smaller categories to the bigger ones, iterating to a fixed point to compute the winning sets.
- Subtlety: Even though a fixed point is reached in 7 moves for King and two Rooks versus King, must still iterate 16 moves back because that was necessary for King and Rook versus King to converge!

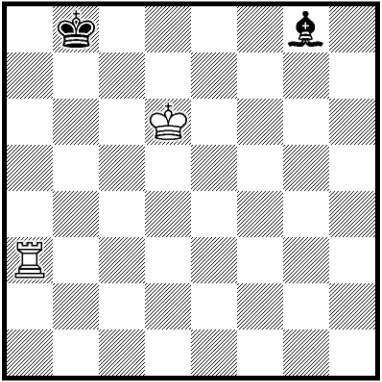
Verified Endgame Databases: BDDs

- Experimented with several variable orderings: the best interleaves the variables in each of the squares but not the variables for the file and rank in a square.
    - King and Rook versus King and Rook benchmark:
      No interleaving: 1512s
      Interleave squares: 543s
      Also interleave files and ranks within squares: 835s
- Created a calculus of BDD conversions of type term $\rightarrow$ term_bdd, which greatly clarified the code for the BDD computations.

Verified Endgame Databases: Result

$$
\begin{array}{l}
[\,]\ \ \text{abstract} \\
\quad (\text{decoder} \\
\quad\quad (\text{posn\_coder} \\
\quad\quad\quad (\text{Black}, [(\text{White}, \text{King}); (\text{White}, \text{Rook}); \\
\quad\quad\quad\quad\quad (\text{Black}, \text{King}); (\text{Black}, \text{Bishop})])) \\
\quad\quad [b_0; b_1; b_2; b_3; b_4; b_5; b_6; b_7; b_8; b_9; b_{10}; b_{11}; \\
\quad\quad\ b_{12}; b_{13}; b_{14}; b_{15}; b_{16}; b_{17}; b_{18}; b_{19}; b_{20}; b_{21}; b_{22}; b_{23}]) \\
\in \text{win2\_by chess } 28 \\
\mapsto \\
\text{<29,907>}
\end{array}
$$

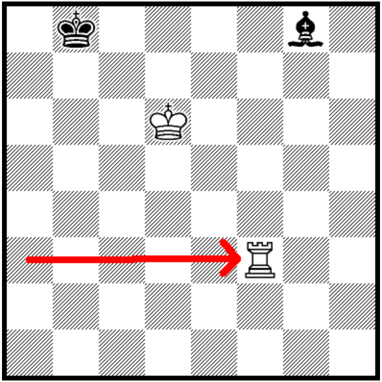
Verified Endgame Databases



One White move is checkmate in 29, all other moves draw.

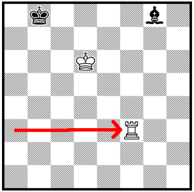
What is the winning move?

Verified Endgame Databases



Rf3!!

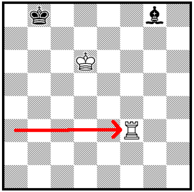
Verified Endgame Databases



The result of querying our verified endgame database on this position:

$$\vdash \ (\text{Black}, \\ \lambda sq. \\ \text{if } sq = (3,5) \text{ then SOME (White, King)} \\ \text{else if } sq = (5,2) \text{ then SOME (White, Rook)} \\ \text{else if } sq = (1,7) \text{ then SOME (Black, King)} \\ \text{else if } sq = (6,7) \text{ then SOME (Black, Bishop)} \\ \text{else NONE}) \ \in \ \text{win2\_by chess } 28 \ \wedge \ \cdots$$

Verified Endgame Databases



In fact, checkmate in 29 is the longest possible win in the King and Rook versus King and Bishop endgame.

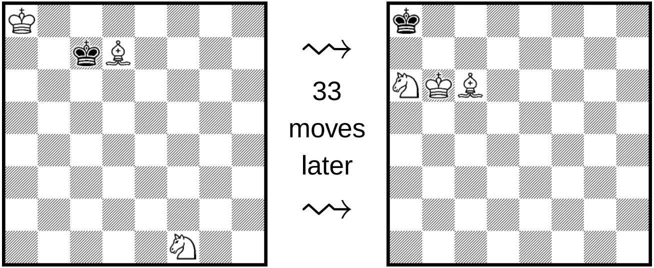
$$\vdash \forall p.\ \text{all\_on\_board}\ p \ \wedge\ \text{to\_move}\ p = \text{White}\ \wedge\ \text{has\_pieces}\ p\ \text{White}\ [\text{King}; \text{Rook}]\ \wedge\ \text{has\_pieces}\ p\ \text{Black}\ [\text{King}; \text{Bishop}] \implies p \in \text{win1 chess} \iff p \in \text{win1\_by chess } 28$$

Application 1: Golden Reference Endgame Database

- The state of the art in endgame database correctness is summed up in the following quotation:
  "Both [Nalimov's endgame databases] and those of Wirth yield exactly the same number of mutual zugzwangs [...] for all 2-to-5 man endgames and no errors have yet been discovered."
- Improvement: our verified endgame database logically follows from the rules of chess.
- Can use as a golden reference to test other endgame databases:
  - randomly sample positions to check evaluation;
  - and also compute global properties such as the number of positions of a certain type (BDD computation).

Application 2: Teaching Aid for Chess Beginners

- Have used the verified endgame database to create some educational web pages showing the best lines of defence.
- Example: Checkmating a bare King with King, Bishop and Knight is something that beginners struggle to learn.



Talk Plan

1. Combining Theorem Proving and Model Checking
   - Introduction to Theorem Proving and Model Checking
   - Combination Methods
   - The HolCheck Approach
2. Case Study: Chess Endgame Databases
   - Modelling the Two Player Game of Chess
   - Constructing Verified Chess Endgame Databases
   - Applications
3. Summary

Summary

- This case study illustrates the HolCheck approach to combining model checking and theorem proving.
  - Demonstrates how to prove sophisticated properties of a highly abstract model by reducing to a boolean model.
- The first verified chess endgame database:
  - constructed by a fully automatic model checking algorithm;
  - and implemented as a HOL4 derived rule (with BDDs);
  - so query results logically follow from the rules of chess.
- Can solve all four piece pawnless endgames without any performance tuning.
  - Scope for improvement in boolification of the move relation and in choice of BDD engine.