Formally Verified Endgame Tables

Joe Leslie-Hurd

Intel Corp.
joe@gilith.com

Guest Lecture, Combinatorial Games
Portland State University
Thursday 25 April 2013

Talk Plan

Endgame Tables

- Hardy (1940) estimated the number of possible games of chess to be $\approx 10^{10^{50}}$.
- Shannon (1950) estimated the number of possible chess positions to be $\approx 10^{43}$.
- But the number of possible chess positions with n fixed pieces is $< 2 \times 16 \times 64^n$.
- Endgame tables (EGTs) solve chess for small values of n.

Categorize and Conquer

- Divide all possible chess positions into classes (e.g., KQKR).
  - **Warning:** It should never be possible for a chess game to leave a class and enter it again later.
- For each class C of positions define an enumeration $f : C \to [0..N)$.
  - Can often reduce N by using symmetry and eliminating illegal positions (e.g., touching kings).
- Compute an array DTM[N] of depth-to-mate values.
  - $\mathrm{DTM}[f(p)] = n$ means that starting from position p White can checkmate Black within n moves.
  - Use symmetry to find Black's depth-to-mate and draws.

Computing DTM Endgame Tables

Code (Initialize DTM)

```
initialize() {
  for each (p in C) {
    if Black to move and checkmated then
      DTM[f(p)] := 0
    else
      DTM[f(p)] := +∞
  }
}
```

Computing DTM Endgame Tables (II)

Code (Propagate DTM values)

```
iterate() {
  for each (p in C) {
    Q := the set of possible next positions from p
    if White to move then
      DTM[f(p)] := 1 + minimum DTM of positions in Q
    else if not in checkmate then
      DTM[f(p)] := maximum DTM of positions in Q
  }
}
```

Note: Q might include positions outside C

Computing DTM Endgame Tables (III)

Code (Converge to a fixed point)

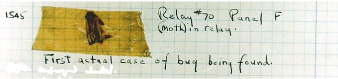
```
compute() {
  DTM := new Integer[N]
  initialize()
  while (DTM changes) {
    iterate()
  }
}
```

What could possibly go wrong?

The First Actual Computer Bug

- On 9 September 1945 the Harvard Mark II Machine broke down because a moth got caught between the points of Relay #70 in Panel F.
- At 3:45pm Grace Murray Hopper extracted it and taped it into the log book.
- In fact the term *bug* to mean a snag or defect was used by Edison as early as 1878.

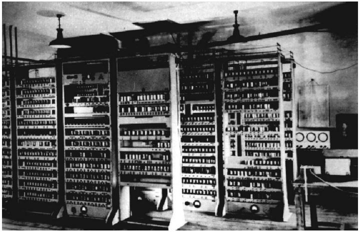
The Harvard Mark II Machine, an early computer boasting magnetic drum storage.



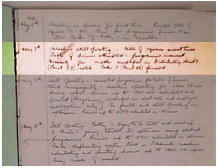
“First actual case of bug being found”

The First Software Bug

- The EDSAC I became operational on 6 May 1949, printing a table of square numbers.
- The very next day the log entry reports a software error.
- Maurice Wilkes recalls the experience of debugging a program in June 1949:
  "[T]he realization came over me with full force that a good part of the remainder of my life was going to be spent in finding errors in my own programs."



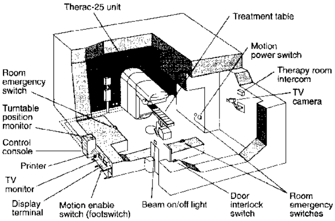
The EDSAC I, the first stored program computer.



"Machine still operating - table of squares several times. Table of primes attempted - programme incorrect"

Serious Software Bugs

- **1985–1987:** A particular combination of operator key presses on the Therac 25 radiation treatment machine blasted the patient with X-rays at 125 times the recommended dose, resulting in the death of 3 people.
- **4 June 1996:** The $2B Ariane 5 rocket exploded on its maiden flight because an assignment of a 64 bit number to a 16 bit buffer overflowed. The Inertial Reference System crashed and output a test pattern. The rocket controller interpreted this as real flight data, changed direction, disintegrated and self-destructed.



The Therac 25 radiation treatment machine.

The launch of the Ariane 5 rocket.

Endgame Table Software Bugs

Endgame tables have occasionally been found to contain errors:

- **1986:** Thompson's KQPKQ EGT was caveated as correct only in the absence of underpromotion.
- **1987:** Van Den Herik's KRP(a2)KbBP(a3) EGT replaced unavailable subgame EGTs with faulty chessic logic.
- **1999:** RetroEngine's EGTs assumed that the loser would never make a capture.
- **2002:** FEG's KNNK EGT assumed that White could never win, and in other EGTs sliding pieces could jump over pawns.

What About Testing?

- Testing is an effective technique for finding software bugs that appear frequently.
- **Example:** If you have a bug in your software that crashes the computer every 1,000,000 hours on average, then:
  - you need 1,000,000 hours of testing to spot the bug;
  - but every day it will crash one of your 50,000 users.
- **Problem:** How do you know when to stop testing?
  - *"Program testing can be used to show the presence of bugs, but never to show their absence!"* [Dijkstra]

Formal Verification

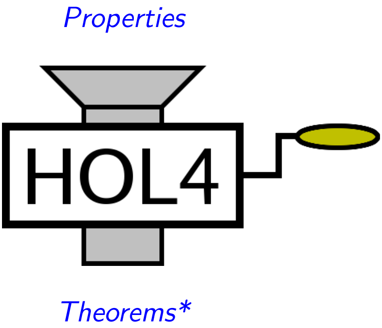
- Formal verification refers to a body of verification techniques that work by building a mathematical model of an artifact and proving properties about it.
- Formal verification is complementary to testing.
  - In general, testing techniques generate weak evidence about the real artifact. [Worry: Have I tested enough?]
  - In general, formal verification techniques generate strong evidence about a model of the artifact. [Worry: Is the model faithful enough?]
- The field of formal verification has been actively researched for over 60 years.[1]

[1] Alan M. Turing. Checking a large routine. In *Report of a Conference on High Speed Automatic Calculating Machines*, pages 67–69, Cambridge, England, June 1949. University Mathematical Laboratory.

Formal Verification Successes

- **Type checkers** prove data integrity properties of arbitrary programs during compilation.
- **Abstract interpretation tools** find memory safety errors such as buffer overflows or dangling pointers in open source codes.
- **The TERMINATOR tool** developed by Microsoft Research checks there are no infinite loops in Windows device drivers that would cause the OS to hang.
- **The CompCert project** used the Coq theorem prover to verify an optimizing compiler for a large subset of C.
- **The Isabelle/HOL theorem prover** was used to carry out a 20 man-year verification of the seL4 operating system kernel.

Interactive Theorem Proving



- Interactive theorem proving is a formal verification technique.
- The user makes logical definitions and guides the tool to prove formal properties of them.
- Automatic tactics generate pieces of proof as a by-product of breaking down properties.

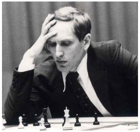
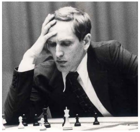
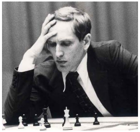
*Made with mechanically extracted proof.

Higher Order Logic

- Higher order logic is an expressive logic, allowing natural formalizations of most mathematical theories.
- **Example:** *Using \$3 and \$5 coins you can make every dollar amount greater than \$7:*

$$\forall x.\ x > 7 \implies \exists y, z.\ x = 3 * y + 5 * z$$

- This expressive power enables the construction of faithful mathematical models of systems in higher order logic.
- The main challenge in verifying properties of these systems using interactive theorem provers is proof automation:

 $\rightsquigarrow$  $\rightsquigarrow$  $\rightsquigarrow$ 

Theorem Provers in the LCF Design

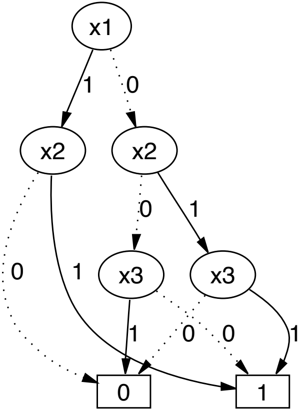
- A theorem $\Gamma \vdash \phi$ states *"if all of the hypotheses Γ are true, then so is the conclusion ϕ"*.
- The novelty of Milner's Edinburgh LCF theorem prover was to make `theorem` an abstract ML type.
- Values of type `theorem` can only be created by a small logical kernel which implements the primitive inference rules of the logic.
- Soundness of the whole ML theorem prover thus reduces to soundness of the logical kernel.

HOL4 theorem prover $\sim$ the elephant
logical kernel $\sim$ the ball

Binary Decision Diagrams

- Binary decision diagrams (BDDs) are a representation of propositional logic formulas.
- Every path from root to leaf respects a variable ordering, and there is maximal sharing of subterms.
- Gordon created a set of inference rules relating higher order logic formulas and BDDs:

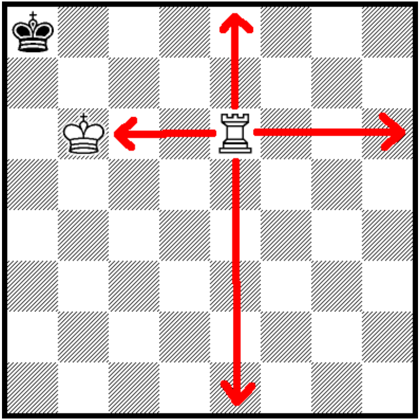
$$\frac{\Gamma \vdash t_1 = t_2 \quad \Delta \vdash t_1 \mapsto B}{\Gamma \cup \Delta \vdash t_2 \mapsto B}$$



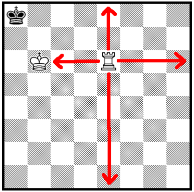
A binary decision diagram representation of $(x1 \wedge x2) \vee (\neg x1 \wedge (x2 \equiv x3))$.

Formalizing the Laws of Chess

Example: Define the set of squares that a rook attacks.



Formalizing the Laws of Chess (II)



- Define the required types:
  - square $\equiv \mathbb{N} \times \mathbb{N}$
  - position $\equiv$
    side $\times$ (square $\to$ (side $\times$ piece) option)
- Define the logical relation:

$$\text{rookAttacks} : \text{position} \to \text{square} \to \text{square} \to \text{bool}$$

$$\text{rookAttacks}\ p\ a\ b \equiv$$

$$a \neq b\ \wedge\ (\text{file}\ a = \text{file}\ b\ \vee\ \text{rank}\ a = \text{rank}\ b)\ \wedge$$

$$\forall c.\ \text{betweenSquare}\ a\ c\ b \implies \text{emptySquare}\ p\ c$$

- Continue in this way to formalize a logical definition of
  DTM : $\mathbb{N} \to$ position set

Computing Verified Endgame Tables

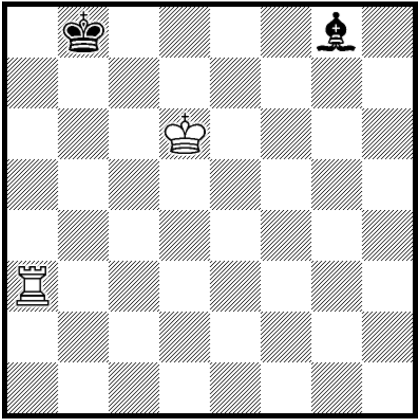
We build our verified endgame database in the usual way by working backwards from checkmates, but symbolically using BDDs.

$$
\begin{array}{ll}
\vdash & \text{decodePosition} \\
& \quad (\text{Black}, [(\text{White}, \text{King}), (\text{White}, \text{Rook}), \\
& \qquad\qquad (\text{Black}, \text{King}), (\text{Black}, \text{Bishop})])) \\
& \quad [x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, \\
& \quad\; x_{12}, x_{13}, x_{14}, x_{15}, x_{16}, x_{17}, x_{18}, x_{19}, x_{20}, x_{21}, x_{22}, x_{23}]) \\
& \in \text{DTM } 28 \\
\mapsto & < 29{,}907 >
\end{array}
$$

Performance is sufficient to cover all 4 piece pawnless endgames.

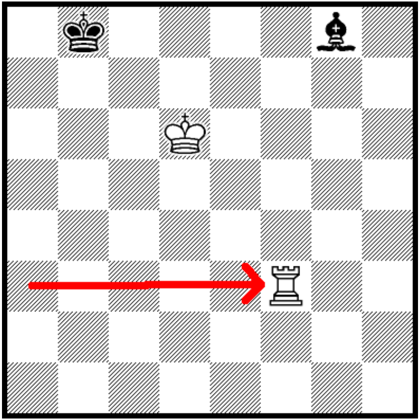
Querying the Endgame Tables

Quiz: Find the only winning White move.

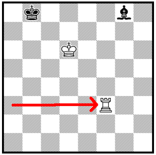


Querying the Endgame Tables (II)

Solution: Rf3 is checkmate in 29 (all other moves draw).



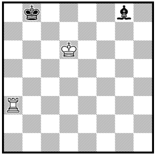
Querying the Endgame Tables (III)



Check the after-position by proving a theorem using our verified endgame table:

$$\begin{aligned}
\vdash \quad & (\text{Black}, \\
& \ \lambda sq. \\
& \quad \text{if } sq = (3,5) \text{ then Some } (\text{White}, \text{King}) \\
& \quad \text{else if } sq = (5,2) \text{ then Some } (\text{White}, \text{Rook}) \\
& \quad \text{else if } sq = (1,7) \text{ then Some } (\text{Black}, \text{King}) \\
& \quad \text{else if } sq = (6,7) \text{ then Some } (\text{Black}, \text{Bishop}) \\
& \quad \text{else None}) \ \in \ \text{DTM } 28
\end{aligned}$$

Querying the Endgame Tables (IV)



In fact, we can prove that checkmate in 29 is the longest possible win in the King and Rook versus King and Bishop endgame:

$$
\begin{aligned}
\vdash \quad & \forall p, n. \\
& \quad \mathrm{toMove}\ p = \mathrm{White}\ \land \\
& \quad \mathrm{hasPieces}\ p\ \mathrm{White}\ [\mathrm{King}, \mathrm{Rook}]\ \land \\
& \quad \mathrm{hasPieces}\ p\ \mathrm{Black}\ [\mathrm{King}, \mathrm{Bishop}]\ \land \\
& \quad \mathrm{allPiecesOnBoard}\ p\ \land \\
& \quad p \in \mathrm{DTM}\ n \implies \\
& \quad p \in \mathrm{DTM}\ 29
\end{aligned}
$$

Summary

- The world's first verified endgame table.
- Can prove that position classification logically follows from the laws of chess.
- Constructed as a fully automatic algorithm implemented in the HOL4 theorem prover.
- Please get in touch if you are interested in finding out more:

`joe@gilith.com`
`http://gilith.com/chess/endgames`