Formally Verified Endgame Tables

Joe Hurd

Galois, Inc.
joe@gilith.com

Guest Lecture, Combinatorial Games
Portland State University
Tuesday 10 May 2011
Talk Plan

1. Endgame Tables
2. Software Errors
3. Formal Verification
4. Verified Endgame Tables
5. Summary
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 $f : C \rightarrow [0..N)$.
  - Can often reduce $N$ by using symmetry and eliminating illegal positions (e.g., touching kings).
- Compute an array $\text{DTM}[N]$ of depth-to-mate values.
  - $\text{DTM}[f(p)] = n$ 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)

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

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

What can go wrong?
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 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."

"Machine still operating - table of squares several times. Table of primes attempted - programme incorrect"
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.
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.
- **Question:** 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]
Static Analysis

- Static analysis is a program verification technique that is complementary to testing.
  - Testing works by **executing** the program and checking its run-time behavior.
  - Static analysis works by examining the **text** of the program.
- Driven by new techniques, static analysis tools have recently made great improvements in scope.
  - **Example 1:** Modern type systems can check **data integrity** properties of programs at compile time.
  - **Example 2:** Abstract interpretation techniques can find memory problems such as **buffer overflows** or **dangling pointers**.
  - **Example 3:** The TERMINATOR tool developed by Microsoft Research can find **infinite loops** in Windows device drivers that would cause the OS to hang.
Higher Order Logic Theorem Proving

- Interactive theorem proving is a static analysis method.
  - The user makes **logical definitions** and applies tactics to formally **prove properties** of them.

- Higher order logic is expressive enough to naturally formalize mathematics and verify software.
  - **Example 1:** Formalization of probability theory.
  - **Example 2:** Verification of the seL4 operating system kernel.

- The main challenge is **proof automation**:
Theorem Provers in the LCF Design

- A theorem $\Gamma \vdash \phi$ states “if all of the hypotheses $\Gamma$ are true, then so is the conclusion $\phi$”.

- 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.
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:

\[
\Gamma \vdash t_1 = t_2 \quad \Delta \vdash t_1 \leftrightarrow B \\
\Gamma \cup \Delta \vdash t_2 \leftrightarrow B
\]

A binary decision diagram representation of \((x_1 \land x_2) \lor (\neg x_1 \land (x_2 \equiv x_3))\).
Example: Define the set of squares that a rook attacks.
Define the required types:
- square ≡ \( \mathbb{N} \times \mathbb{N} \)
- position ≡ side \( \times \) (square \( \rightarrow \) (side \( \times \) piece) option)

Define the logical relation:
\[
\text{rookAttacks} : \text{position} \rightarrow \text{square} \rightarrow \text{square} \rightarrow \text{bool}
\]
\[
\text{rookAttacks} \ p \ a \ b \ \equiv \\
\quad a \neq b \land (\text{file } a = \text{file } b \lor \text{rank } a = \text{rank } b) \land \\
\quad \forall c. \ \text{betweenSquare} \ a \ c \ b \implies \text{emptySquare} \ p \ c
\]

Continue in this way to formalize a logical definition of
\[
\text{DTM} : \mathbb{N} \rightarrow \text{position set}
\]
Computing Verified Endgame Tables

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

\[ \vdash \text{decodePosition} \]

\[ (\text{Black}, [(\text{White, King}), (\text{White, Rook}),
(\text{Black, King}), (\text{Black, Bishop})]) \]

\[ [x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11},
x_{12}, x_{13}, x_{14}, x_{15}, x_{16}, x_{17}, x_{18}, x_{19}, x_{20}, x_{21}, x_{22}, x_{23}] \]

\[ \in \text{DTM 28} \]

\[ \rightarrow < 29, 907 > \]

Performance is sufficient to cover all 4 piece pawnless endgames.
Quiz: Find the only winning White move.
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:

\[
\frac{}{(\text{Black}, \lambda sq.}
\]

\[
\begin{align*}
\text{if } sq &= (3, 5) \text{ then Some (White, King)} \\
\text{else if } sq &= (5, 2) \text{ then Some (White, Rook)} \\
\text{else if } sq &= (1, 7) \text{ then Some (Black, King)} \\
\text{else if } sq &= (6, 7) \text{ then Some (Black, Bishop)} \\
\text{else None} &\in DTM 28
\end{align*}
\]
In fact, we can prove that checkmate in 29 is the longest possible win in the King and Rook versus King and Bishop endgame:

\[ \forall p, n. \]

\[ \text{toMove } p = \text{White} \land \]
\[ \text{hasPieces } p \text{ White [King, Rook]} \land \]
\[ \text{hasPieces } p \text{ Black [King, Bishop]} \land \]
\[ \text{allPiecesOnBoard } p \land \]
\[ p \in \text{DTM } n \implies \]
\[ p \in \text{DTM } 29 \]
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