Data Assurance in Opaque Computations

Joe Hurd[1] and Guy Haworth[2]
[1] Galois, Inc., 421 SW 6th Ave., Suite 300 Portland, OR 97204, USA
joe@galois.com
[2] School of Systems Engineering, University of Reading, UK
guy.haworth@bnc.oxon.org

Abstract. The chess endgame is increasingly being seen through the lens of, and therefore effectively defined by, a data 'model' of itself. It is vital that such models are clearly faithful to the reality they purport to represent. This paper examines that issue and systems engineering responses to it, using the chess endgame as the exemplar scenario. A structured survey has been carried out of the intrinsic challenges and complexity of creating endgame data by reviewing the past pattern of errors during work in progress, surfacing in publications and occurring after the data was generated. Specific measures are proposed to counter observed classes of error-risk, including a preliminary survey of techniques for using state-of-the-art verification tools to generate EGTs that are correct by construction. The approach may be applied generically beyond the game domain.

Keywords: chess, correctness, data assurance, endgame, HOL, verification

1 Introduction

The laws of chess in use today date back to the end of the 15th century [1], while the rules of play, which need not concern us here, are regularly revised by FIDE [2]. There have thus been 500 years of attempts to analyse the game, the last 50 years being increasingly assisted by the computer since world-class computer chess was declared to be an aim of Artificial Intelligence at the 1956 Dartmouth Conference [3].

One approach has been the complete enumeration and analysis of endgames which are small enough to be computable in practice. Heinz [4] cites Ströhlein's 1970 Ph.D. thesis [5] as the first computer construction of endgame tables (EGTs). Today, 6-man chess is essentially[1] solved [6] and the EGTs are being distributed by DVD, ftp transfer and p2p[2] collaboration [7]. With new workers, ideas and technology already active, the prospects for 7-man chess being similarly solved by 2016 are good.

The intrinsic problem of correctness of chess endgame data is summed up in the following quotation [8]:

> 'The question of data integrity always arises with results which are not self-evidently correct. Nalimov runs a separate self-consistency check on each EGT

[1] *Essentially*, because available 6-man EGTs do not include lone Kings or castling rights.
[2] The promulgation of the Nalimov EGTs is the second most intense use of eMule.

after it is generated. Both his EGTs 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.'

A verification check should not only be separate but independent. While Nalimov's verification test is separate and valuable in that it has faulted some generated EGTs, it is not independent as it shares 80% of its code with the generation code [9]. Nevertheless, it should be added that Nalimov's EGTs have not been faulted to date and are widely used without question.

The problem of correctness is of course not unique to chess endgames or computer software. Famous examples include the faulty computer hardware that caused the Intel Pentium processor to divide certain numbers incorrectly [10]. In the field of mathematics, the *Classification of Simple Groups* theorem has a proof thousands of pages long which it is not feasible to check manually [11]. The recent computer-generated proof of the Four Colour Theorem [12] will bring some comfort to those who find opaque proofs by exhaustion lacking in both aesthetics and auditability.

Modern society is increasingly dependent on the integrity of its digital infrastructure, especially in a real-time, safety-critical context; globalization has lead to greater homogeneity and standardised systems in all sectors, including that of Information Technology. The Internet, the Web and search engines leave their users ever more vulnerable to systemic failure, e.g., severings of vital FLAG cables [13-14], Internet root nameserver corruption [15] and a Google search-engine bug [16].

It is therefore appropriate to look for evidence of highly assured system integrity and for tools to help to provide that manifest integrity. Chess endgames are not safety-critical but serve as a case study to demonstrate the issues of assurance management.

The main contributions here are the creation of a framework for analysing data assurance at every stage of the data life cycle, the use of this framework to analyse EGT vulnerabilities and the proposal of countermeasures and remedies. Naturally, mathematical proofs cannot apply directly to the real world, and measures have to be taken to check against the fallibilities of man and machine. However, these measures and the HOL4/BDD-based approach [17] alluded to, demonstrate and deliver the highest levels of assurance to date.

The remainder of this paper is structured as follows: Section 2 reviews sources of error in the whole process of endgame data management, and Section 3 reviews the high-assurance approach taken to generating EGTs using higher order logic. The summary indicates the future prospect for generating and validating chess EGTs.

2 An Error Analysis of Endgame Data Management

It is convenient to refer here to the producer of data as the *author* and the user of that data as the *reader*. The reader may become the author of derived data by, e.g., a data-mining investigation. The lifecycle of the data, from first conception to use, is considered in three structured phases as follows:

1) Definition: the author
   – models the scenario which is to be the subject of a computation,

– analyses the requirements and designs/implements a computation

2) Computation:
   – the author runs the computation on a platform and generates the output,
3) Use:
   – the author manages the output: publishes, promulgates, comments,
   – the reader interprets and uses the results of the computation.

2.1 Phase 1: Definition, Design and Development

The design of the computation involves mapping the relevant characteristics of the chosen problem domain into a *representative model* on the computer. Just as a good choice of concepts and notation will facilitate a mathematical proof, the choice of language to describe the *real world* and *the model* will facilitate the faithful translation of the last, informal statement of requirements into their first formal statement. As with much of what follows, the human agent needed combines the qualities of a domain expert and a systems engineer and can carry the responsibility of ensuring that requirements are faithfully and auditably translated into systems.

Increasingly, people are working in teams – in academia, in industry, and internationally in the Open Source movement, across the web and towards the Semantic Web [18]. This further requires that the concepts in use are shared effectively and that the semantics of the language of any project are robust. In the late 1970s, a UK computer company defined a machine instruction for a new mainframe processor as 'loop while predicate is TRUE'. In the South of England, *while* means *as long as* but in the North where the processor was being manufactured, *while* commonly means *until* – the exact opposite. The second author was one of the few who realised the implications of this facet of regional language just in time.

The context in which a computer system works is important, as the system will interact with its context through the interfaces at its system boundary. Van den Herik ambitiously[3] computed a KRP(a2)KbBP(a3) EGT [19], substituting complex chessic logic for unavailable subgame EGTs. This ran the risk of *model infidelity* as chess is notoriously resistant to description by a small set of simple rules [20] and he rightly caveated the results. The logic and results were indeed faulted [21], leading to refined statistics [22] which are now being compared with results generated by Bleicher's FREEZER [23] and WILHELM [24].

More subtly, a particularly efficient algorithm for computing DTM[4] EGTs [25] exploits deferred adoption of subgame data, and therefore had to be forced to compute a minimum of *maxDTM* cycles even if an interim cycle did not discover any new decisive positions.[5]

Here are three examples of model infidelity in the categories of *one out* or *boundary* errors. Wirth [26] used the software RETROENGINE which assumed that captures ending a phase of play would always be made by the winner. As a result, his depths

[3] n.b., in 1987, computers were some 16,000 times less powerful than they are today.

[4] DTM ≡ Depth to Mate, the ultimate goal: the most common, Nalimov EGTs are to DTM.
DTZ ≡ Depth to (move-count) Zeroing (move): a target metric for computing Pawnful EGTs.

[5] As for endgames such as KQKR, KRKP, KRKR, KBPK and KBBKP.

are sometimes one ply too large. Secondly, De Koning's FEG EGT generator originally failed to note losses in 0 in endgames with maxDTM = 1 [27][6]. The so-called *KNNK Bug* infected 7 4-man[7], 35 5-man and then many of Bourzutschky's 6-man FEG EGTs [28] before the latter spotted the problem. Thirdly, Rasmussen's data-mining of Thompson's correct EGTs [29] resulted in a small number of errors stemming from various coding slips [30-33].

A proposal here is that elements of a system should be designed to be self-identifying. There is a requirement in the next two phases, *Computation* and *Use*, that agents should confirm at the time that they are working with appropriate inputs.[8] The deepest endgames require a 2-byte cell per position in their Nalimov EGTs. However the access-code cannot determine cell-size at runtime and has to be appropriately configured for the cell-size chosen. This creates the potential for a mismatch between access-code and EGT and this inevitably occurred on occasion [35]. Most recently, Bourzutschky provided the last 16 Nalimov-format EGTs to KPPKPP by converting his FEG EGTs, and picked a 2-byte format for the two endgames, KQPK(B/R)P, [36] where Nalimov chose 1-byte.[9] Clearly, runtime checks on the actual parameters of Nalimov-style[10] EGT files would obviate several problems.

Finally, Tamplin had to manage many coding issues in porting Nalimov-originated code from one environment to another,[11] amongst which was the synchronization of parallelism, a technical issue which is becoming commonplace on multi-core platforms. It is expected but not guaranteed that this issue would be detected by independent verification testing.

2.2 Phase 2: Computation

This phase reviews hardware and software errors, and the human errors of incorrect input and inadequate verification. First, a caution against assuming the correctness of the infrastructure used for a computation. In his Turing Award lecture, Thompson [37] noted that anyone could insert their own nuances at any level of the computing infrastructure – application, collector, compiler or even commodity hardware. However, one has the reassurance that others are using the same infrastructure in a different way and there is perhaps some 'safety in numbers'.

A correct computation requires the correct input. The computation of a DT*x* EGT[12] can require compatible EGTs for subgames. Because of the lack of self-identification noted above, Tamplin [35], [38] had to manage his file directories with great care to ensure correctness when computing first DTZ then DTZ_{50} EGTs: the one slip was picked up by a verification run.

In the spirit of this paper, Schaeffer [39] detailed the various difficulties his team had encountered in computing 10-man EGTs. Like Tamplin, they had to manage a

[6] FEG also suffered temporarily from the *Transparent Pawn Bug*: 'model infidelity' again.
[7] The seven 4-man endgames affected were KBK(B/N/P), KNK(B/N/R) and KNNK.
[8] This is just one example of the benefits of *run-time binding* [34].
[9] Another EGT conversion from 2-byte to 1-byte produced full EN/MB compatibility.
[10] e.g., game, endgame, metric, side-to-move, block/cell-size, date, version, comments …
[11] Nalimov write compilers for Microsoft, and used their non-standard features in his programs.
[12] e.g., a DTZ_k EGT computing DTZ in the context of a hypothetical *k*-move drawing-rule.

large set of files with great care. With regard to platform integrity, he noted not only application coding errors in scaling up from 32-bit to 64-bit working but also compiler errors in compromising 64-bit results by using 32-bit working internally. For a while, Nalimov dropped multiples of 2^{32} positions in counts of draws in his EGT statistics because of 32-bit limitations.[13,14]

Schaeffer [39] compared his checkers EGTs with those of a completely independent computation and found errors on both sides. Despite the fact that modern microchips devote a greater proportion of their real estate to self-checking, Schaeffer also noted hardware errors in CPU and RAM. He also noted errors in disks, which should give pause to think about the physics and material science of today's storage products. Checksums at disc-block level were added to prevent storage and copying errors promulgating. Nalimov EGTs were integrity-checked by the DATACOMP software, and those investing in them soon learned to check MD5SUM file-signatures as well.

With regard to software testing practice, Schaeffer [39] notes than an EGT-verification which operates only within an endgame, i.e., without regard to the positions of successor endgames, will not pick up *boundary errors* caused by misinheriting subgame information.

2.3 Phase 3: Use

The scope of the *use* phase includes errors of user cognition, data persistency and data access. Clearly, the mindsets of author and reader in, respectively, phases 1 and 3 have to be aligned if the data is not to be misinterpreted. For example, in relating to some early EGTs [40], it is necessary to remember that they are not of the now-prevalent Nalimov type, and that Thompson caveated his original KQPKQ EGT as correct only in the absence of underpromotions. Stiller [41] found an 'error' in the EGT traced to this cause. Thompson's data is for Black to move (btm) positions only, and the depth of White to move (wtm) positions is reported as the depth of the succeeding btm position in the line, i.e. not including the next wtm move and one less than what is now commonly understood to be the depth. This interface quirk nearly produced a systematic *one out* error in Nunn's 2nd edition of [42]. Further, the values reported by Thompson are either *White wins in n moves* or *White does not win*. Thus 0-1/= zugzwang positions are invisible, and 0-1/1-0 zugzwangs are not distinguishable from =/1-0 ones. As with all extant EGTs, castling is assumed not to be an option, currently reasonable as castling rights have never survived to move 49 [43], but this does mean that EGTs will not help solve some Chess Studies.[15]

There are arguments for computing EGTs to a variety of metrics [45] and therefore they need to identify their particular metric to any chess-engine using them. It can be demonstrated that when a chess-engine mistakes a DTZ EGT for a DTM EGT, it will prefer the position-depth in the current phase of play before a capture to that in the

[13] The second author sympathises: he made a similar error in a 1970s statistics package.

[14] Nalimov also once provided an incorrect and as yet unidentified statistics file for KBPKN.

[15] e.g. r1b5/8/8/7n/8/p7/6P1/RB2K2k w Q [44]. White draws: 1. Be4 Ra4 2. Bc6 Ra6 3. g4+ Rxc6 4. gxh5 Ra6 5. h6 Bf5 6. h7 Bxf7 7. O-O-O+ K~ 8. Rd6 Ra4 (8. … Rxd6 =) 9. Rd4 etc.

next phase, resulting in the bizarre refusal of a piece *en prise*.[16] Thus, cell-size, metric and presumed *k*-move-rule in force must be part of the self-description of an EGT.

Disc drives built 'down to cost' are perhaps the weakest part of PCs and laptops and subject to crash: this is a strong selling point for the diskless notebook. CDs/DVDs, particularly rewritable, are prone to environmental wear and handling damage [46], and it is somewhat ironic that the ancient materials of stone, parchment and shellac are more long-lived. To check against data decay, and for data persistency, it is necessary to check that data files have not subsequently been corrupted, e.g., by file-transfer (upload, download, CD burn or reorganisation) or even deterioration during long-term storage. File use should therefore be preceded and followed by file-signature checks on input files, and it seems surprising that this is not an inbuilt facility in computers' operating systems. RAID systems are excellent but not immune to the failure-warning system being accidentally turned off, e.g., by software update.

Incorrect file-access code can turn an uncorrupted file into a *virtually corrupted* file: the Nalimov 1-byte/2-byte syndrome is an example here. This phenomenon afflicted KINGSROW in a World Computer-Checkers Championship [47], causing it to lose an otherwise drawn game and putting it out of contention for the title.

Finally, there are some errors where the source has not been defined [39], [48]. Thompson [40] also cites errors in the KQP(g7)KQ EGT by Kommissarchik [49] but these errors[17] did not prevent this EGT from assisting Bronstein during an adjournment to a win in 1975.[18]

Following this review of the lifecycle of data, it is clear that if readers are to be assured of data integrity, authors must provide self-identifying files with file-signatures and a certificate of provenance describing the production process and the measures that have been and should be taken to ensure integrity.

3 Correct-by-Construction Endgame Tables

As the preceding section demonstrates, errors can creep in at any point in the lifecycle of data, and there is no single solution that will eliminate all errors. A pragmatic approach is to analyse the errors that have occurred, and introduce a remedy that will reduce or eliminate a common cause of errors. For example, RAIDs and file signatures are both remedies designed to tackle errors caused by faulty hard disc drives.

A common cause of errors observed in practice is the misinterpretation of EGT data. Is the context in which the reader is using the data compatible with the context in which the author computed it? Stated this way, it is clear that this problem affects a broad class of data, not just EGTs, and there is a general approach to solving the problem based on assigning meaning to the data. If data carries along with it a description of what it means, then it is possible to check that the author and reader use it in com-

[16] e.g., given 8/3Q4/8/k7/6r1/8/8/K7 w and a DTZ EGT interpreted as a DTM EGT, a chess-engine will play 1. Qf5+? Kb6 2. Qe6+? Kc5 3. Qf5+? Kc4 4. Qe6+? Kc5 (pos 3w) 5. Qc8+? Kd5 6. Qf5+? Kc4 7. Qe6+? (pos. 4b)

[17] There are btm KQQ(g8)KQ draws and wins which Komissarchik did not anticipate.

[18] Grigorian-Bronstein, Vilnius: after 60m, 8/8/8/K2q2p1/8/2Q5/6k1/8 w {=} ... 76. Qd2?? Qc6+ {77. K~ Kh1} 0-1.

patible contexts. A prominent example of this approach is the Semantic Web project [18], which is working towards a world in which web pages include a standardized description of their contents.

It is possible to apply this approach to EGTs by creating a standardized description of their contents, which unambiguously answers the question: what exactly does each entry in the EGT mean with respect to the laws of chess? With such a description in place, the problem of verifying the correctness of the EGT reduces to providing sufficient evidence that each entry in the EGT satisfies its description.

This EGT verification process was carried out in a proof-of-concept experiment that generated *correct-by-construction* four piece pawnless EGTs [17], by using the following steps:

1. The laws of chess, less Pawns and castling, were defined in higher order logic, and these definitions were entered into the HOL4 theorem prover.
2. The format of EGTs represented as ordered binary decision diagrams was also defined in HOL4.
3. For each EGT, a formal proof was constructed in the HOL4 theorem prover that all of the entries follow from the laws of chess.

The remainder of this section will briefly examine the steps of this experiment.

3.1 Formalizing the Laws of Chess

The first step of the EGT verification process involves making a precise definition of the laws of chess, by translating them from the FIDE handbook [2] into a formal logic. The Semantic Web uses description logic to describe the content of web pages, but this is not expressive enough to naturally formalise the laws of chess, and so higher order logic is chosen instead.

The precise set of definitions formalising the laws of chess are presented in a technical report [17]; to illustrate the approach it suffices to give one example. Here is an excerpt from the FIDE handbook:

> **Article 3.3.** The rook may move to any square along the file or the rank on which it stands.

And here is the corresponding definition in higher order logic:

$$\mathsf{rookMaybeMoves}\ sq_1\ sq_2 \equiv (\mathsf{sameFile}\ sq_1\ sq_2 \vee \mathsf{sameRank}\ sq_1\ sq_2) \wedge (sq_1 \neq sq_2)$$

The definition of rook moves is completed by combining the definition of **rookMaybeMoves** with a formalisation of Article 3.5, which states that the Rook, Bishop and Queen may not move over any pieces.

In this way the whole of the laws of chess (with no pawns or castling) are formalised into about 60 definitions in higher order logic, culminating in the important

$$\mathsf{depthToMate}\ p\ n \equiv \ldots$$

EGT relation, which formalises the familiar metric from Nalimov's EGTs that position p has DTM n.

3.2 Formal Verification of EGTs

In itself the formalisation of the laws of chess into higher order logic is nothing more than a mathematical curiosity. However, matters get more interesting when the definitions are entered into an interactive theorem prover, such as the HOL4 system [50]. These theorem provers are designed with a simple logical kernel that is equipped to execute the rules of inference of the logic, and it is up to the user to break apart large proofs into a long sequence of inferences that are checked by the theorem prover. The theorem prover provides proof tools called tactics to help with the breaking apart of large proofs, but since everything must ultimately be checked by the logical kernel, the trusted part of the system is very small.[19] As a consequence of this design, a proof that can be checked by an interactive theorem prover is highly reliable. This is why it is a significant milestone that the Four Colour Theorem is now underpinned by a formal proof that has been completely checked by an interactive theorem prover [12].

How can this capability of reliable proof checking be harnessed to check EGTs? In principle, for each *(p,n)* DTM entry in an EGT, a proof of the relation **depthToMate** ***p n*** could be constructed and checked, thus establishing that the EGT entry did indeed logically follow from the laws of chess as formalised in HOL4. However, in practice the size of any EGT would make this strategy completely infeasible.

A more promising approach is to formalise the program that generates the EGT, and construct a proof that it could only generate EGT entries that followed from the laws of chess. This idea of using logic to formally verify computer programs is very old[20], but it is only recently that theorem proving technology has made it practical for significant programs, such as the Verified C Compiler [52]. This is the most promising approach for generating verified EGTs of a realistic size, but it is still currently too labour-intensive for anything less than critical infrastructure.[21]

An alternative approach works sufficiently well to construct a prototype verified EGT [17]. The whole EGT is formalised in the logic, as a sequence of sets of positions of increasing DTM. Each set of positions is encoded as a bit-vector, and stored as an *ordered binary decision diagram* or BDD [53]. Using a combination of deduction and BDD operations [54], it is possible to use the previous DTM set to construct a proof that the next set is correct. The verification is bootstrapped with the set of checkmate positions at DTM 0, which is easily checked by expanding the definition of a checkmate position.

Due to the difficulty of compressing sets of positions using BDDs [55-56], it is only possible to generate verified EGTs for four piece pawnless endgames using this technique, but as a result there are now many win/draw/loss positions that come with a proof that they logically follow from the laws of chess [17], [57].[22]

---

[19] Typically, just a few hundred lines of code.

[20] The earliest reference known to the authors is a paper of Turing published in 1949 [51].

[21] It is left as a challenge to the research community to come up with a safety-critical application of EGTs.

[22] Naturally, the numbers generated by the verification agree with Nalimov's results.

4 Summary

This paper has examined the correctness of endgame data from multiple perspectives. A structured survey was presented on the past pattern of errors in EGTs managed during work in progress, surfacing in publications, and occurring after the data was generated. Specific remedies were proposed to counter observed classes of error from creeping in to endgame data. A particular challenge is to pin down the precise meaning of the data in an EGT, so that the reader uses the data in a context that is compatible with the context in which the author computed it. A possible solution to pinning down this meaning was described that used higher order logic, and the presence of a machine-readable specification opened the door to a discussion of techniques for using interactive theorem provers to generate EGTs that are correct by construction.

Although endgame data has been the focus of the paper, the methodology of examining data assurance carries over to many opaque computations. Specifically, the learning points are that:

- it is vital to collect data on errors that have occurred in practice, to ground any discussion of data assurance,
- there is no magic solution, but rather individual remedies must be introduced that counter observed classes of error, and
- the precise meaning of the data, the exact context in which it was computed, must be encoded in some form and made available with the data, to counter misinterpretations on the part of the reader.

As society becomes increasingly dependent on computers and data generated by opaque computations, we cannot afford to overlook techniques for safeguarding data assurance.

Acknowledgments. Our review illustrates the intrinsic challenges and complexity of the computations addressed here. The fact that there are so few, largely temporary, errors is a testimony to the awareness, skill and rigour of those who have contributed significant EGT results to date. Great achievements lead to others, e.g., the solving of Checkers in 2007 [58]. Throughout, the ICCA and ICGA encouraged and championed work on endgames. We thank all those involved for their contributions in the endgame field and for the example they have set in the field of Systems Engineering.

References

1. Hooper, D., Whyld, K.: The Oxford Companion to Chess. OUP, 2nd Edition (1992)
2. FIDE: The Laws of Chess. FIDE Handbook E.1.01A. http://www.fide.com/component/-handbook/?id=124&view=article (2009)
3. McCorduck, P.: Machines Who Think: A Personal Inquiry into the History and Prospects of Artificial Intelligence. A.K.Peters (2004)
4. Heinz, E.A.: Endgame databases and efficient index schemes. ICCA J. 22-1, 22--32 (1999)
5. Ströhlein, T.: Untersuchungen über kombinatorische Spiele. Ph.D. thesis, Technical University of Munich (1970)
6. Haworth, G.M^{c}C.: 6-man Chess Solved. ICGA J. 28-3, 153 (2005)

7. Kryukov, K.: 'EGTs Online'. http://kirill-kryukov.com/chess/tablebases-online/ (2007)
8. Nalimov, E.V., Haworth, G.M^{c}C., Heinz, E.A.: Space-efficient indexing of chess endgame tables. ICGA J. 23-3, 148-162 (2000)
9. Nalimov, E.V.: Private Communications (2000)
10. Coe, T.: Inside the Pentium FDIV bug. Dr. Dobb's Journal 229, 129--135 & 148 (1995)
11. Gorenstein, D., Lyons, R. Solomon R.: The Classification of Finite Simple Groups. AMS (1994).
12. Devlin, K.: Last doubts removed about the proof of the Four Color Theorem. MAA Online http://www.maa.org/devlin/devlin_01_05.html (2005)
13. RIPE: Mediterranean Cable Cut – A RIPE NCC Analysis. http://www.ripe.net/projects/-reports/2008cable-cut/index.html (2008)
14. BBC: Repairs begin on undersea cable. http://news.bbc.co.uk/1/hi/technology/7795320.stm (2008)
15. Pouzzner, D.: Partial failure of Internet root nameservers. The Risks Digest 19-25 (1997)
16. BBC: 'Human error' hits Google search. http://news.bbc.co.uk/1/hi/technology/7862840-.stm (2009)
17. Hurd, J.: Formal verification of chess endgame databases. In Joe Hurd, Edward Smith, and Ashish Darbari. Theorem proving in higher order logics: Emerging trends proceedings. Technical Report PRG-RR-05-02, 85-100. Oxford University Computing Laboratory (2005)
18. Shadbolt, N., Hall, W., Berners-Lee, T.: The Semantic Web Revisited. IEEE Intelligent Systems 21-3 96--101 (2006)
19. Herik, H.J. van den, Herschberg, I.S., Nakad, N.: A Six-Men-Endgame Database: KRP(a2)KbBP(a3). ICCA J. 10-4 163--180 (1987)
20. Michalski, R.S., Negri, P.G.: An Experiment on Inductive Learning in Chess End Games. Machine Intelligence 8, pp. 175-192. Ellis Horwood (1977)
21. Sattler, R.: Further to the KRP(a2)KbBP(a3) Database. ICCA J. 11-2/3, 82--87 (1988)
22. Herik, H.J. van den, Herschberg, I.S., Nakad, N.: A Reply to R. Sattler's Remarks on the KRP(a2)-KbBP(a3) Database. ICCA J. 11-2/3, 88--91 (1988)
23. Bleicher, E.: FREEZER. http://www.freezerchess.com/ (2009)
24. Andrist, R.B.: WILHELM. http://www.geocities.com/rba_schach2000/index_english.htm (2009)
25. Wu, R., Beal, D.F.: Solving Chinese Chess Endgames by Database Construction. Information Sciences 135-3/4, 207--228 (2001)
26. Wirth, C., Nievergelt, J.: Exhaustive and Heuristic Retrograde Analysis of the KPPKP Endgame. ICCA J. 22-2, 67--80 (1999)
27. Tay, A.: A Guide to Endgame Tablebases. http://www.horizonchess.com/FAQ/Winboard/-egtb.html (2009)
28. Merlino, J.: Regarding FEG 3.03b – List Found. http://www.horizonchess.com/FAQ/-Winboard/egdbbug.html (2002)
29. Chessbase: FRITZ ENDGAME T3. http://www.chessbase.com/workshop2.asp?id=3179 (2006)
30. Roycroft, A.J.: *C* Correction. EG 7-119, 771 (1996)
31. Roycroft, A.J.: The Computer Section: Correction. EG 8-123, 47--48 (1997)
32. Roycroft, A.J.: *C*. EG 8-130 Supplement, 428 (1998)
33. Roycroft, A.J.: Snippets. EG 8-131, 476 (1999)
34. Jones, N.D., Muchnick, S.S.: TEMPO: A Unified Treatment of Binding Time and Parameter Passing Concepts in Programming Languages. LNCS 66 (1978)
35. Bourzutschky, M.S., Tamplin, J.A., Haworth, G.M^{c}C.: Chess endgames: 6-man data and strategy. Theoretical Computer Science, Vol. 349-2, 140--157 (2005)
36. Bourzutschky, M.S.: Tablebase version comparison. http://preview.tinyurl.com/d3wny4 (2006-08-10)
37. Thompson, K.: Reflections on Trusting Trust. CACM 27-8, 761--3 (1984)

38. Tamplin, J.: EGT-query service extending to 6-man pawnless endgame EGTs in DTC, DTM, DTZ and DTZ_{50} metrics. http://chess.jaet.org/endings/ (2006)
39. Schaeffer, J., Björnsson, Y., Burch, N., Lake, R., Lu, P., Sutphen, S.: Building the Checkers 10-piece Endgame Databases. In: Advances in Computer Games 10, pp. 193-210 (2003)
40. Thompson, K.: Retrograde Analysis of Certain Endgames. ICCA J., 9-3, 131-139 (1986)
41. Stiller, L.B.: Parallel Analysis of Certain Endgames. ICCA J,, 12-2, 55-64 (1989)
42. Nunn, J.: Secrets of Pawnless Endings. Expanded Edition 2, Gambit (2002)
43. Krabbé, T.: Private Communication (2008-09-05)
44. Herbstman, A.O.: Draw Study 172. EG 5, 195 (1967)
45. Haworth, G.McC.: Strategies for Constrained Optimisation. ICGA J. 23-1, 9-20 (2000)
46. Byers, F.R.: Care and Handling of CDs and DVDs: A Guide for Librarians and Archivists. CLIR/NIST. See also http://www.clir.org/pubs/reports/pub121/contents.html (2003)
47. Fierz, M. Cash, M., Gilbert, E.: The 2002 World Computer-Checkers Championship. ICGA J. 25-3, 196--198 (2002)
48. Schaeffer, J.: One Jump Ahead: Challenging Human Supremacy in Checkers. Springer-Verlag, New York, N.Y. (1997)
49. Komissarchik, E.A., Futer, A.L.: Ob Analize Ferzevogo Endshpilia pri Pomoshchi EVM. Problemy Kybernet, 29, 211--220 (1974). Reissued in translation by Chr. Posthoff and I.S. Herschberg under the title 'Computer Analysis of a Queen Endgame'. ICCA J. 9-4, 189-198 (1986)
50. Gordon, M.J.C., Melham, T.F.: Introduction to HOL: A theorem-proving environment for higher order logic. Cambridge University Press (1993)
51. Turing, A.M.: Checking a large routine. In: Report of a Conference on High Speed Automatic Calculating Machines, pp. 67-69. Cambridge University Mathematical Laboratory (1949)
52. Leroy, X.: Formal certification of a compiler back-end or: programming a compiler with a proof assistant. In: Proceedings of the 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2006), pp 42-54. ACM (2006).
53. Bryant, R.E.: Symbolic Boolean manipulation with ordered binary-decision diagrams. ACM Computing Surveys 24-3, 293-318 (1992)
54. Gordon, M.J.C.: Programming combinations of deduction and BDD-based symbolic calculation. LMS J. of Computation and Mathematics, 5, 56-76 (2002)
55. Edelkamp, S: Symbolic exploration in two-player games: Preliminary results. In The International Conference on AI Planning & Scheduling (AIPS), Workshop on Model Checking, 40-48, Toulouse, France (2002)
56. Kristensen, J.T.: Generation and compression of endgame tables in chess with fast random access using OBDDs. Master's thesis, U. of Aarhus, Dept. of Computer Science (2005)
57. Hurd, J.: Chess Endgames. http://www.gilith.com/chess/endgames (2005)
58. Schaeffer, J., Burch, N., Björnsson, Y., Kishimoto, A., Müller, M., Lake, R., Lu, P., Sutphen, S.: Checkers is Solved. Science 317-5844, 1518-1522 (2007)