

Machines Reasoning About Machines: 2015

J Strother Moore^(✉)

Department of Computer Science, University of Texas, Austin, USA
moore@cs.utexas.edu

Abstract. Computer hardware and software can be modeled precisely in mathematical logic. If expressed appropriately, these models can be executable, i.e., run on concrete data. This allows them to be used as simulation engines or rapid prototypes. But because they are formal they can be manipulated by symbolic means: theorems can be proved about them, directly, with mechanical theorem provers. But how practical is this vision of machines reasoning about machines? In this highly personal talk, I will describe the 45 year history of the “Boyer-Moore theorem prover,” starting with its use in Edinburgh, Scotland, to prove simple list processing theorems by mathematical induction (e.g., the reverse of the reverse of x is x) to its routine commercial use in the microprocessor industry (e.g., the floating point operations of the Via Nano 64-bit X86 microprocessor are compliant with the IEEE standard). Along the way we will see applications in program verification, models of instruction set architectures including the JVM, and security and information flow. I then list some reasons this project has been successful. The paper also serves as an annotated bibliography of the key stepping stones in the applications of the prover.

1 Introduction

If we had some exact language ... or at least a kind of truly philosophic writing, in which the ideas were reduced to a kind of alphabet of human thought, then all that follows rationally from what is given could be found by a kind of calculus, just as arithmetical or geometrical problems are solved.

— *Leibniz (1646–1716)*

Instead of debugging a program, one should prove that it meets its specifications, and this proof should be checked by a computer program.

— *John McCarthy, “A Basis for a Mathematical Theory of Computation,” 1961*

This paper is a highly personal recounting of 45 years spent in pursuit of the dream of building machines that can reason about machines. In my case, the machine doing the reasoning is the “Boyer-Moore theorem prover” and the machines about which it reasons are various algorithms, hardware designs, and software.

The first theorems proved by the “Boyer-Moore theorem prover,” in Edinburgh, Scotland, in 1972, were simple list processing theorems such as

```
(equal (append (append a b) c)
      (append a (append b c)))
```

and

```
(implies (true-listp x)
      (equal (rev (rev x)) x))
```

where `append` and `rev` were defined recursively. The theorem prover automatically chose often-appropriate induction schemes and used axioms as rewrite rules. It could also generalize the conjecture being proved in order to set up successful inductions. By 2015, the “Boyer-Moore theorem prover” was being used nightly at a microprocessor company to verify functional correctness and other properties of various modules checked in the day before.

So how practical is the dream of machines reasoning about machines? Today, it is a reality.¹ But what were the stepping stones from the associativity of `append` to microprocessors?

2 A Quick History and Acknowledgments

There is no such thing as the “Boyer-Moore theorem prover.” There have been a series of provers starting with the “Edinburgh Pure Lisp Theorem Prover” developed by Bob Boyer and me starting in 1971 in Edinburgh, Scotland. That prover was inspired by the ideas of John McCarthy (Lisp as a specification language and computational logic), Woody Bledsoe (how to build a heuristic theorem prover), Rod Burstall (induction as a routine proof step), A.P. Morse, and J.R. Shoenfield (logical foundations). It supported a home-grown untyped, first order, “pure Lisp” as the logic and was fully automatic; there was no way the user could change its behavior.

It evolved through QTHM (an unreleased version that supported quantification) to Nqthm (“New Quantified Theorem Prover”) and then ACL2 (“A Computational Logic for Applicative Common Lisp”). All are known as the “Boyer-Moore theorem prover.” But since about 1992, ACL2 has been developed and maintained by Matt Kaufmann and me, a fact recognized by the 2005 ACM Software System Award to Boyer, Kaufmann, and Moore for “the Boyer-Moore theorem prover.”

ACL2 is available in open source form, without cost, under the terms of a 3-clause BSD license. See the ACL2 home page [31]. Many megabytes of documentation are available online and I refer to it sometimes in this paper by writing “see:DOC *x*,” which means: go to the ACL2 home page, click on The User’s Manual link, then click on the ACL2+Books Manual link and type *x* into the **Jump to** box.

While only three people, Bob Boyer, Matt Kaufmann, and me, are responsible for the code of those provers, many people have contributed to their success.

¹ For a discussion of why I consider this “reasoning” see [41].

Most often these contributors are users of the tools who pushed them beyond their intended limits, suggested ideas for overcoming those limits, tested prototypes, re-worked large proof scripts, etc., and demonstrated the flexibility and capacity of the provers. A quick scan of the bibliography will reveal some of their names.

Warren Hunt deserves special mention here. He formalized various hardware description languages in Nqthm and then ACL2 [23–25, 43], demonstrated the potential of our tools to model and verify commercial microprocessor designs [25, 28], trained many students in those techniques, helped start and build the verification teams at AMD, Centaur, and Oracle, and procured funding to maintain and develop ACL2.

Given this brief history it should be clear that neither “the Boyer-Moore theorem prover” nor most of the applications cited in this paper are my work alone. Indeed, I have had little or nothing to do with most of the recent applications other than co-authoring ACL2 with Bob Boyer and Matt Kaufmann. But I am in a unique position to report 45 years of formal methods history.

This paper will sketch the maturation of formal methods as seen from the perspective of the Boyer-Moore community by citing major verification projects carried out in our community with the Edinburgh Pure Lisp Theorem Prover, Nqthm, and ACL2.²

3 1970s – Simple List Processing

The most impressive theorem proved by the Edinburgh Pure Lisp Theorem prover [3, 37] was probably the correctness of an insertion sort algorithm: the result is ordered and the number of times any object is listed in the output is equal to the number of times it is listed in the input. This was proved fully automatically from the recursive definitions of insertion and sort and involved interesting generalizations. But two things made the Edinburgh Pure Lisp Theorem prover stand out at a time when most other theorem provers were resolution based: it routinely proved theorems by induction and it proved a wide variety of theorems automatically. For a complete list of the theorems proved automatically by 1973 see [3].

By 1979, we had added the use of previously proved theorems as rewrite (and other kinds of) rules, allowing the user to steer the prover by suggesting lemmas [4]. For a description of the prover as it stood in 1979 and a summary of the important theorems proved see [5]. The important theorems included the existence and uniqueness of prime factorizations [5], the correctness of a ripple carry adder [38], the termination of the Takeuchi function [39], the correctness of a McCarthy-Painter-like expression compiler with respect to an assembly language formalized with an operational semantics [5], the correctness of the Boyer-Moore fast string searching algorithm in FORTRAN 77 [5], the correctness of

² For a wonderful narrative of one person’s journey through formal methods applications, mainly with Nqthm and ACL2, see Russinoff’s page <http://www.russinoff.com/papers/>.

a linear-time majority vote algorithm in FORTRAN 77 [11], the soundness and completeness of a propositional tautology checker [5], and the correctness of a simple 1-dimensional real-time control algorithm (“cruise control”) [13]. It was truly possible to reason mechanically about a wide variety of simple computing “machines.”

The most important change to the prover in the 1970s, after the addition of lemmas, was the adoption of quote notation to represent constants in the logic. No longer was 3 represented by (ADD1 (ADD1 (ADD1 (ZERO)))); it was represented by (QUOTE 3). This change of term representation (and underlying logical changes) was described in [6] (the publication date of which belies when the work was actually done). The opened the door to much bigger models; for example, the opcodes for machine code instructions formalized with operational semantics were no longer huge terms.

4 1980s – Academic Math and Computer Science

In the 1980s Boyer and I moved to the University of Texas at Austin and started teaching and working with students. The use of QUOTE for constants allowed the efficient use of verified metafunctions [6]. We also integrated a linear arithmetic decision procedure (a process that took 4 years and, in our minds, did not warrant journal publication because it was “just engineering” since we had not invented a new decision procedure) [10].

Matt Kaufmann started calling this prover “Nqthm” which was originally just the name of the directory on which it was stored; the name stood for “New Quantified Theorem Prover.” The name “Nqthm” stuck.

Boyer and I proved the invertibility of the RSA encryption algorithm [9], the unsolvability of the halting problem [8], and the Turing completeness of Pure Lisp [7].

By the mid-1980s, using Nqthm, Russinoff had proved Wilson’s theorem [45], Shankar had proved Gödel’s incompleteness theorem [47, 48], and Hunt had proved the correctness of a microprocessor described at the gate level [23, 24].

In 1987, Computational Logic, Inc. (CLI or more fondly “Clerc”), was founded and we left UT to continue our work there.

Hunt’s microprocessor work positioned us to do “system verification,” i.e., the verification of a hardware/software stack. The transition from hardware to software was facilitated by the implementation of a stack based assembly language, called Piton, which could be translated into Hunt’s machine code for a register-based machine with a linear memory, but which was a convenient target for higher-level language compilers [2, 40]. I designed, implemented, and verified the correctness of the assembly/link/load process. Bevier verified an operating system [1], Flatau and Young verified compilers [18, 53], and Wilding verified some applications [52]. Furthermore, all the pieces were proved to “fit together:” the assumptions made at one level were verified at the next level so that one theorem ensured that if the preconditions of an application were satisfied then compiling, assembling, linking, and loading it on the gate machine and running

it produced the results predicted by the high-level language. The entire “verified stack” was first published in a special issue of the **Journal of Automated Reasoning** in 1989 [2].

In the early 1990s the stack was ported to a machine design by Hunt and Brock that was fabricated by LSI Logic from an NDL netlist verified with Nqthm [25, 40]. Also in that time frame, Yu and Boyer proved the functional correctness of 21 of the 22 routines in the Berkeley Unix C String Library – performed by compiling the library with `gcc -o` to obtain binary machine code for the Motorola 68020 and then verifying that with respect to a formal operational semantics capturing 80 % of the user-level 68020 instructions [14], and, by the same technique, they proved a variety of other C programs, including the C code for binary search and Hoare’s *in situ* Quick Sort from [34].

Boyer’s website <http://www.cs.utexas.edu/users/boyer/ftp/nqthm/> contains the sources for the 1992 version of Nqthm and all the proof scripts mentioned above and many others. See also [12].

However, the size of the models being formalized with Nqthm, the acceptance of the Common Lisp standard [50], and the inefficiency of our homegrown pure Lisp motivated us to adopt an applicative subset of Common Lisp as our logic. Boyer and I started the development of ACL2 in August, 1989.

5 1990s – Commercial Breakthrough

Two key projects with ACL2 convinced us and others that it was suitable for proving properties of commercially interesting models. One was the modeling and proofs about a Motorola digital signal processor and the other was the verification of the pseudocode for floating point division on the AMD K5 microprocessor before it was fabricated. Both projects are described in the 1996 paper [17]. The first project lasted 31 months starting in late 1993, the second project lasted about 2 months starting in June, 1995. For technical details see [16, 42].

All of the elementary floating point arithmetic for the AMD Athlon was verified to be IEEE compliant by Russinoff using ACL2 [44]; this proof was done before the Athlon was fabricated and was based on a mechanical translation of the RTL (a variant of Verilog) description of the design. At the same time at Rockwell Collins, an ACL2 microarchitectural simulator for the first silicon-implemented JVM (the design became the JEM1 of aJile Systems, Inc.) was produced [22].

6 2000s – Gradual Acceptance

By this time ACL2 was in fairly routine if *ad hoc* use at a variety of companies manufacturing microprocessors and low level software. Scholarly papers were not always written because the work was not regarded as research so much as “just” verification that designs were correct. For example, at AMD properties of the Opteron and other desktop microprocessors were verified, and a verified ACL2

BDD package was built [51] that achieved about 60 % of the speed of the CUDD package (however, it did not support dynamic variable reordering).

At IBM, the algorithms used for floating point division and square root on the IBM Power 4 were verified [46]. At Rockwell Collins, the instruction equivalence of different implementations of a commercial microprocessor [20] was proved; in addition they proved that the microcode for the Rockwell Collins AAMP7 implements a given security policy having to do with process separation [21] allowing the AAMP7 to be certified for Multiple Independent Levels of Security (MILS).

At UT Austin, the Sun Java Virtual Machine was formalized and certain properties of Sun bytecode verifier as described in JSR-139 for J2ME JVMs [35] and the class loader [36] were verified.

Largely in response to the needs of users, ACL2 continued to evolve with the addition of such proof techniques as equivalence relations (see :DOC equivalence), congruence-based rewriting (see :DOC congruence), support for non-linear arithmetic [27], extended metafunctions and clause processors (allowing the use of external proof tools like SAT solvers) [26,33], proof debugging tools [30], and many system programming facilities to support faster execution, faster loading of files, and the use of ACL2 both as a prototyping language and a language in which efficient, verified verification tools can be produced [19,29,32]. In addition, Bob Boyer, Warren Hunt, Jared Davis, Sol Swords and Matt Kaufmann prototyped the use of hash cons (unique representation of Lisp conses) and memoization which Matt Kaufmann and I eventually added to the standard release of ACL2 [15].

7 2010 – Integrated with the Design Process

ACL2 continues to be used on an *ad hoc* basis at many companies (including now Intel) to help gain confidence in designs. But there is one company where ACL2 is deeply integrated into the microprocessor design workflow: Centaur Technology.

Centaur produces x86 microprocessors (and is the third largest such manufacturer after Intel and AMD). In 2009, Hunt and Swords used ACL2 to verify the media unit of the Centaur (VIA) Nano 64-bit x86 CPU [28], which is part of Centaur’s QuadCore processor. More significantly, the ACL2 team at Centaur has built an infrastructure that re-verifies designs nightly, running on hundreds of machines. In addition, ACL2-verified tools check wire-level engineering changes and clock trees. Errors introduced one day are addressed the next. This infrastructure is described in [49].

The cost of their ACL2 staff is less than the license fees previously paid for regression testing.

8 Lessons

Why has ACL2 succeeded to the extent that it has? Among the reasons are

- Everything is done in a single logic and that logic is an efficient, executable, ANSI standard programming language; models have dual use as emulators and formal semantics, and properties can be composed.
- We put a “human in the loop” doing things humans do well – inventing concepts and lemmas – allowing new proofs of modified modules to be found (often) automatically.
- We have spent decades engineering the system to be rugged and to handle large models.
- We encourage modelers to focus on problems that are appropriate for modeling with a programming language and to be bit- and cycle-accurate; conventional mathematical techniques are used to hide or expose the resulting complexity and to glue results together.
- We have an extremely talented and dedicated user community.
- Industry has no other option: modern machines are too complicated to be designed accurately without mechanized reasoning.

9 Conclusion

Formal methods saves times and costs less than informal ones – once sufficient investment is made in formalizing past designs and the terminology of the design group.

Complexity is not an argument against formal methods; it is an argument for formal methods.

References

1. Bevier, W.R.: A verified operating system kernel. Ph.D. dissertation, University of Texas at Austin (1987)
2. Bevier, W., Hunt Jr., W.A., Moore, J.S., Young, W.: Special issue on system verification. *J. Autom. Reasoning* **5**(4), 409–530 (1989)
3. Boyer, R.S., Moore, J.S.: Proving theorems about pure lisp functions. *JACM* **22**(1), 129–144 (1975)
4. Boyer, R.S., Moore, J.S.: A lemma driven automatic theorem prover for recursive function theory. In: 5th International Joint Conference on Artificial Intelligence, pp. 511–519 (1977)
5. Boyer, R.S., Moore, J.S.: *A Computational Logic*. Academic Press, New York (1979)
6. Boyer, R.S., Moore, J.S.: Metafunctions: Proving them correct and using them efficiently as new proof procedures. In: Boyer, R.S., Moore, J.S. (eds.) *The Correctness Problem in Computer Science*. Academic Press, London (1981)
7. Boyer, R.S., Moore, J.S.: A mechanical proof of the turing completeness of pure lisp. In: Bledsoe, W.W., Loveland, D.W. (eds.) *Contemporary Mathematics: Automated Theorem Proving: After 25 Years*, vol. 29, pp. 133–168. American Mathematical Society, Providence (1984)
8. Boyer, R.S., Moore, J.S.: A mechanical proof of the unsolvability of the halting problem. *J. Assoc. Comput. Mach.* **31**(3), 441–458 (1984)

9. Boyer, R.S., Moore, J.S.: Proof checking the rsa public key encryption algorithm. *Am. Math. Monthly* **91**(3), 181–189 (1984)
10. Boyer, R.S., Moore, J.S.: Integrating decision procedures into heuristic theorem provers: a case study of linear arithmetic. In: Hayes, J.E., Richards, J., Michie, D. (eds.) *Machine Intelligence 11*, pp. 83–124. Oxford University Press, Oxford (1988)
11. Boyer, R.S., Moore, J.S.: Mjrtjy - a fast majority vote algorithm. In: Boyer, R.S. (ed.) *Automated Reasoning: Essays in Honor of Woody Bledsoe*. Automated Reasoning Series, pp. 105–117. Kluwer Academic Publishers, Dordrecht (1991)
12. Boyer, R.S., Moore, J.S.: *A Computational Logic Handbook*, 2nd edn. Academic Press, New York (1997)
13. Boyer, R.S., Moore, J.S., Green, M.W.: The use of a formal simulator to verify a simple real time control program. In: *Beauty is Our Business: A Birthday Salute to Edsger W. Dijkstra*. pp. 54–66. Springer-Verlag Texts and Monographs in Computer Science (1990)
14. Boyer, R.S., Yu, Y.: Automated proofs of object code for a widely used microprocessor. *J. ACM* **43**(1), 166–192 (1996)
15. Boyer, R.S., Warren A., Hunt, J.: Function memoization and unique object representation for acl2 functions. In: *ACL2 2006: Proceedings of the Sixth International Workshop on the ACL2 Theorem Prover and Its Applications*, pp. 81–89. ACM, New York (2006)
16. Brock, B., Hunt Jr., W.A.: Formal analysis of the motorola CAP DSP. In: Hinchey, M.G., Bowen, J.P. (eds.) *Industrial-Strength Formal Methods*, pp. 81–115. Springer-Verlag, London (1999)
17. Brock, B., Kaufmann, M., Moore, J.S.: ACL2 theorems about commercial microprocessors. In: Srivas, M., Camilleri, A. (eds.) *FMCAD 1996*. LNCS, vol. 1166, pp. 275–293. Springer, Heidelberg (1996). <http://www.cs.utexas.edu/users/moore/publications/bkm96.ps.Z>
18. Flatau, A.D.: A verified implementation of an applicative language with dynamic storage allocation. Ph.D. thesis, University of Texas at Austin (1992)
19. Goel, S., Hunt, W., Kaufmann, M.: Simulation and formal verification of x86 machine-code programs that make system calls. In: Claessen, K., Kuncak, V. (eds.) *FMCAD 2014: Proceedings of the 14th Conference on Formal Methods in Computer-Aided Design*, pp. 91–98. EPFL, Switzerland (2014). <http://www.cs.utexas.edu/users/hunt/FMCAD/FMCAD14/proceedings/final.pdf>
20. Greve, D., Wilding, M.: Evaluatable, high-assurance microprocessors. In: *NSA High-Confidence Systems and Software Conference (HCSS)*, Linthicum, MD, March 2002. <http://hokiepokie.org/docs/hcss02/proceedings.pdf>
21. Greve, D., Wilding, M.: A separation kernel formal security policy (2003)
22. Greve, D.A.: Symbolic simulation of the JEM1 microprocessor. In: Gopalakrishnan, G.C., Windley, P. (eds.) *FMCAD 1998*. LNCS, vol. 1522, pp. 321–333. Springer, Heidelberg (1998)
23. Hunt, Jr., W.A.: FM8501: a verified microprocessor. Ph.D. thesis, University of Texas at Austin (1985). (Published as a book by the same title, Cambridge University Press 1994)
24. Hunt Jr., W.A. (ed.): *FM8501: A Verified Microprocessor*. LNCS, vol. 795. Springer, Heidelberg (1994)
25. Hunt, Jr., W.A., Brock, B.: A formal HDL and its use in the FM9001 verification. In: *Proceedings of the Royal Society*, April 1992
26. Hunt Jr., W.A., Kaufmann, M., Krug, R.B., Moore, J.S., Smith, E.W.: Meta reasoning in ACL2. In: Hurd, J., Melham, T. (eds.) *TPHOLs 2005*. LNCS, vol. 3603, pp. 163–178. Springer, Heidelberg (2005)

27. Hunt Jr., W.A., Krug, R.B., Moore, J.: Linear and nonlinear arithmetic in ACL2. In: Geist, D., Tronci, E. (eds.) CHARME 2003. LNCS, vol. 2860, pp. 319–333. Springer, Heidelberg (2003)
28. Hunt Jr., W.A., Swords, S.: Centaur technology media unit verification. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 353–367. Springer, Heidelberg (2009)
29. Kaufmann, M.: – Invited Talk – ACL2 support for verification projects. In: Kirchner, C., Kirchner, H. (eds.) CADE 1998. LNCS (LNAI), vol. 1421, pp. 220–238. Springer, Heidelberg (1998)
30. Kaufmann, M., Moore, J.S.: Proof search debugging tools in ACL2. In: A Festschrift in Honour of Prof. Michael J. C. Gordon FRS. Royal Society, London, March 2008
31. Kaufmann, M., Moore, J.S.: The ACL2 home page. In: Dept. of Computer Sciences, University of Texas at Austin (2014). <http://www.cs.utexas.edu/users/moore/acl2/>
32. Kaufmann, M.: Abbreviated output for input in ACL2: an implementation case study. In: Proceedings of ACL2 Workshop 2009, May 2009. <http://www.cs.utexas.edu/users/sandip/acl2-09>
33. Kaufmann, M., Moore, J.S., Ray, S., Reeber, E.: Integrating external deduction tools with ACL2. *J. Appl. Logic* **7**(1), 3–25 (2009)
34. Kernighan, B.W., Ritchie, D.M.: The C Programming Language, 2nd edn. Prentice Hall, Englewood Cliff (1988)
35. Liu, H., Moore, J.S.: Executable JVM model for analytical reasoning: a study. In: Workshop on Interpreters, Virtual Machines and Emulators 2003 (IVME 2003). ACM SIGPLAN, San Diego, June 2003
36. Liu, H.: Formal Specification and Verification of a JVM and its Bytecode Verifier. Ph.D. thesis, University of Texas at Austin (2006)
37. Moore, J.S.: Computational logic: Structure sharing and proof of program properties. Ph.D. dissertation, University of Edinburgh (1973). <http://www.era.lib.ed.ac.uk/handle/1842/2245>
38. Moore, J.S.: Automatic proof of the correctness of a binary addition algorithm. *ACM SIGARG Newslett.* **52**, 13–14 (1975)
39. Moore, J.S.: A mechanical proof of the termination of takeuchi’s function. *Inf. Process. Lett.* **9**(4), 176–181 (1979)
40. Moore, J.S.: Piton: A Mechanically Verified Assembly-Level Language. Automated Reasoning Series. Kluwer Academic Publishers, Dordrecht (1996)
41. Moore, J.S.: A mechanized program verifier. In: Meyer, B., Woodcock, J. (eds.) VSTTE 2005. LNCS, vol. 4171, pp. 268–276. Springer, Heidelberg (2008)
42. Moore, J.S., Lynch, T., Kaufmann, M.: A mechanically checked proof of the correctness of the kernel of the AMD5K86 floating point division algorithm. *IEEE Trans. Comput.* **47**(9), 913–926 (1998)
43. Hunt Jr., W.A., Reeber, E.: Formalization of the DE2 language. In: Borriore, D., Paul, W. (eds.) CHARME 2005. LNCS, vol. 3725, pp. 20–34. Springer, Heidelberg (2005)
44. Russinoff, D.: A mechanically checked proof of IEEE compliance of a register-transfer-level specification of the AMD-K7 floating-point multiplication, division, and square root instructions. *London Math. Soc. J. Comput. Math.* **1**, 148–200 (1998). <http://www.onr.com/user/russ/david/k7-div-sqrt.html>
45. Russinoff, D.M.: An experiment with the boyer-moore theorem prover: a proof of wilson’s theorem. *J. Autom. Reasoning* **1**(2), 121–139 (1985)

46. Sawada, J.: Formal verification of divide and square root algorithms using series calculation. In: Proceedings of the ACL2 Workshop, 2002, Grenoble, April 2002. <http://www.cs.utexas.edu/users/moore/acl2/workshop-2002>
47. Shankar, N.: Proof-checking metamathematics. Ph.D. thesis, University of Texas at Austin (1986). (Published as the book *Metamathematics, Machines, and Gödel's Proof*, Cambridge University Press, 1994)
48. Shankar, N.: *Metamathematics, Machines, and Godel's Proof*. Cambridge University Press, Cambridge (1994)
49. Slobodova, A., Davis, J., Swords, S., Warren Hunt, J.: A flexible formal verification framework for industrial scale validation. In: Singh, S. (ed.) 9th IEEE/ACM International Conference on Formal Methods and Models for Codesign (MEMOCODE), pp. 89–97. IEEE (2011)
50. Steele Jr., G.L.: *Common Lisp The Language*, 2nd edn. Digital Press, Burlington (1990)
51. Sumners, R.: Correctness proof of a BDD manager in the context of satisfiability checking. In: Proceedings of ACL2 Workshop 2000, Department of Computer Sciences, Technical report TR-00-29, November 2000. <http://www.cs.utexas.edu/users/moore/acl2/workshop-2000/final/sumners2/paper.ps>
52. Wilding, M.: A mechanically verified application for a mechanically verified environment. In: Courcoubetis, C. (ed.) CAV 1993. LNCS, vol. 697, pp. 268–279. Springer, Heidelberg (1993)
53. Young, W.D.: A verified code generator for a subset of Gypsy. Technical report 33, Computational Logic. Inc., Austin, Texas (1988)

Automated Technology for Verification and Analysis
13th International Symposium, ATVA 2015, Shanghai,
China, October 12-15, 2015, Proceedings
Finkbeiner, B.; Pu, G.; Zhang, L. (Eds.)
2015, XIII, 520 p. 96 illus., Softcover
ISBN: 978-3-319-24952-0