An ACL2 Tutorial

Matt Kaufmann and J Strother Moore

Department of Computer Sciences, University of Texas at Austin,
Taylor Hall 2.124, Austin, Texas 78712
{kaufmann,moore}@cs.utexas.edu

Abstract. We describe a tutorial that demonstrates the use of the ACL2 theorem prover. We have three goals: to enable a motivated reader to start on a path towards effective use of ACL2; to provide ideas for other interactive theorem prover projects; and to elicit feedback on how we might incorporate features of other proof tools into ACL2.

1 Introduction

The name "ACL2" [14] stands for "A Computational Logic for Applicative Common Lisp" and refers to a functional programming language, a formal logic, and a mechanized proof system. It was developed by the authors of this paper, with early contributions by Bob Boyer, and is the latest in the line of "Boyer-Moore" theorem provers [2, 3] starting with the 1973 Edinburgh Pure Lisp Prover' [1].

The ACL2 logic and programming language are first-order and admit total recursive function definitions, and are based on a non-trivial purely functional subset of the Common Lisp [20] programming language. Thus, ACL2 can be built on many Lisp platforms. We have extended this subset in some important ways, in no small part because ACL2 is written primarily in itself! Extensions include additional primitives; a *program mode* that avoids proof obligations; a *state* with applicative semantics supporting file I/O and global variables; and applicative property lists and arrays with efficient under-the-hood implementations.

This extended abstract describes a one-hour tutorial, not presented here, but accessible from the "demos" link on the ACL2 home page [14]. Our ambitious goal is to create effective ACL2 users. Of course, no such system can be absorbed deeply in just one hour, so we point to useful documentation and references. Our focus on demos and references suits a second and probably more important goal of this talk, given that most in our audience will already be committed to their favorite theorem prover: To provide a sense of ACL2 interaction that could provide ideas for other interactive theorem prover projects. Conversely, we hope that this introduction to ACL2 will stimulate suggestions for how to improve ACL2 by incorporating ideas from other proof tools.

2 About ACL2

We invite the reader to browse the ACL2 web pages starting with the home page [14], to find: papers on applications and on foundations; tutorials and demos; documentation; mailing list pages; and other useful information.

The remaining sections summarize a few ACL2 demos and ACL2 features that they illustrate. In those demos we will refer to sections of the online hypertext user's manual [16] with the notation "see documentation". This note skips the logical foundations [12, 13], focusing instead on the use of ACL2. We conclude this section with a few words about how the ACL2 proof engine attempts an individual proof and how ACL2 supports interactive proof development.

Figure 1 shows the ACL2 proof engine as an orchestrated collection of automated tactics, including a simplifier that incorporates conditional congruence-based rewriting as well as decision procedures.

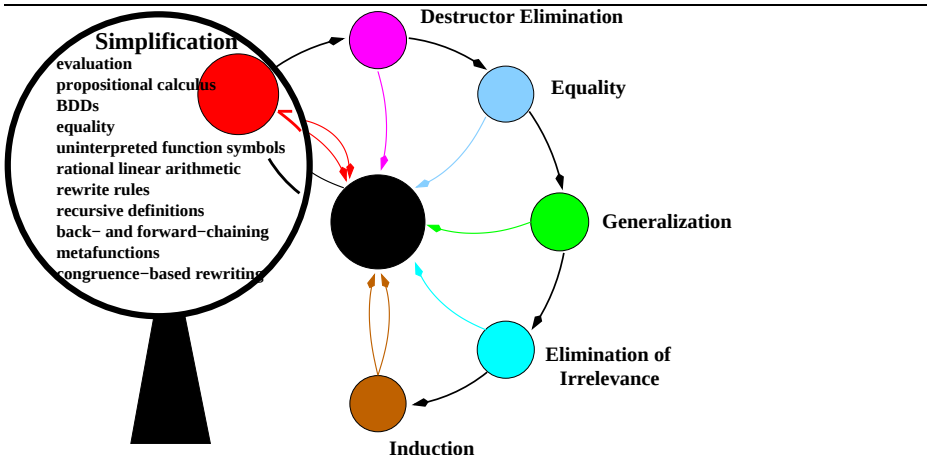

Fig. 1. The ACL2 Waterfall, highlighting the Simplifier

Proof attempts often fail at first! Figure 2 illustrates user interaction with ACL2. The user submits a definition or theorem, which ACL2 attempts to prove using definitions and rules stored in the *logical world*, a database that includes rules stored from definitions and theorems. If the attempt succeeds, then ACL2 makes a corresponding extension to its logical world. Otherwise, ACL2 provides output that can suggest lemmas to prove, and it also offers a variety of other proof debugging tools [15]. Ultimately, a completed proof may be checked by *certifying* the resulting books: input files of events, in particular definitions and proved theorems. Books can be developed independently and combined into libraries of rules that are valuable for automating future proof attempts.

3 Demo: Basics of Interaction with ACL2

This demo develops a proof that for a recursively-defined notion of permutation, the reverse of a list is a permutation of the list. We illustrate how to use ACL2

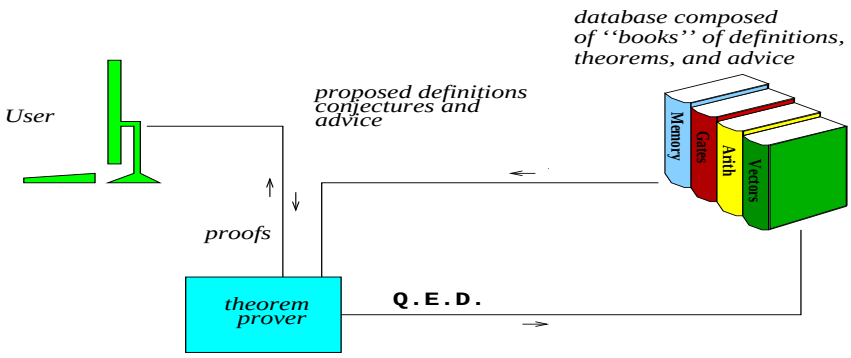

Fig. 2. Basic ACL2 Architecture

interactively, specifically highlighting several aspects of using ACL2. We use a shell running under Emacs, but an Eclipse-based interface is available [7].

- Top-down proof development (see the-method [11]), using simplification checkpoints to debug proof failures
- Helpful proof automation, in particular conditional rewriting with respect to equivalence and congruence relations and a loop-stopper heuristic
- Library development and local scopes
- The proof-checker goal manager

4 Demo: A JVM Model

Next, we demonstrate the so-called M5 model of the Java Virtual Machine (JVM). In the process we illustrate several aspects of ACL2:

- Library re-use via include-book
- Namespace support through Lisp *packages*
- Efficient evaluation supporting simulation of models
- Lisp *macros*, providing user-defined syntax

5 Demo: Proof Debugging and Theory Management

This short demo shows how ACL2 can help to identify rules that slow it down.

- Management of theories (disabling and enabling rules)
- Helpful suggestions from the tool
- Accumulated-persistence for statistics on rule use
- DMR (Dynamic Monitoring of the Rewrite stack)

6 Concluding Remarks

Below is a partial list of useful ACL2 features not mentioned above, many of them requested by users. For more about ACL2, see the home page [14].

- a primitive for user-installed executable counterparts [9]
- proof control [10], including user-installed metatheoretic simplifiers, user-supplied syntactic conditions for rewrite rules, and dynamically computed hints
- traditional tactics (macro-commands) for the proof-checker
- partially-defined functions (see encapsulate) and, mimicking second-order logic, *functional instantiation* [5, 13]
- a defpun utility [18] for defining non-terminating tail-recursive functions, built on top of ACL2 with macros
- capabilities for system-level control, such as user-defined tables, state with applicative semantics, and an extended macro capability (make-event) useful in defining templates for creating events
- guards, which may be viewed as a general, semantic replacement for types
- many modes and switches (see switches-parameters-and-modes)
- hooks to external tools [17], built on a *trust tag* mechanism (defttag) [6]
- experimental extensions others have initiated, to support:
  - Real numbers (through non-standard analysis) [8]
  - Hash cons, function memoization, and applicative hash tables [4]
  - Parallel evaluation [19]
- more debugging tools, e.g. to trace or to inspect the rewrite stack
- diverse tools for querying the logical database (see history)
- quantification via Skolemization (see defun-sk)

Acknowledgements. This material is based upon work supported by DARPA and the National Science Foundation (NSF) under Grant No. CNS-0429591 and also NSF grant EIA-0303609. We also thank Sandip Ray for helpful feedback on a preliminary version of this paper.

References

1. R. S. Boyer and J S. Moore. Proving theorems about pure lisp functions. *JACM*, 22(1):129–144, 1975.
2. R. S. Boyer and J S. Moore. *A Computational Logic*. Academic Press, NY, 1979.
3. R. S. Boyer and J S. Moore. *A Computational Logic Handbook*. Academic Press, London, UK, 1997.
4. R. S. Boyer and Jr. W. A. Hunt. Function memoization and unique object representation for ACL2 functions. In *Proceedings of the Sixth International Workshop on the ACL2 Theorem Prover and Its Applications*, pages 81–89, NY, 2006. ACM.
5. R.S. Boyer, D.M. Goldschlag, M. Kaufmann, and J S. Moore. Functional instantiation in first-order logic. In V. Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pages 7–26. Academic Press, 1991.

6. P. Dillinger, M. Kaufmann, and P. Manolios. Hacking and extending ACL2. In *ACL2 Workshop 2007*, Austin, Texas, November 2007. http://www.cs.uwyo.edu/∼ruben/acl2-07/.
7. P. Dillinger, P. Manolios, J S. Moore, and D. Vroon. ACL2s: "The ACL2 Sedan". *Theoretical Computer Science*, 174(2):3–18, 2006. See http://acl2s.peterd.org/acl2s/doc/.
8. R. Gamboa and M. Kaufmann. Non-Standard Analysis in ACL2. *Journal of Automated Reasoning*, 27(4):323–351, 2001.
9. D. A. Greve, M. Kaufmann, P. Manolios, J S. Moore, S. Ray, J. L. Ruiz-Reina, R. Sumners, D. Vroon, and M. Wilding. Efficient execution in an automated reasoning environment. *Journal of Functional Programming*, 18(1):3–18, January 2008. See also Tech. Rpt. TR-06-59, Dept. of Computer Sciences, Univ. of Texas at Austin, http://www.cs.utexas.edu/ftp/pub/techreports/tr06-59.pdf.
10. W. A. Hunt, Jr., M. Kaufmann, R. Krug, J S. Moore, and E. Smith. Meta reasoning in ACL2. In J. Hurd and T. Melham, editors, *Proceedings of the 18th International Conference on Theorem Proving in Higher Order Logics (TPHOLs 2005)*, volume 3603 of *LNCS*, pages 163–178, Oxford, England, 2005. Springer-Verlag.
11. M. Kaufmann. Modular proof: The fundamental theorem of calculus. In M. Kaufmann, P. Manolios, and J S. Moore, editors, *Computer-Aided Reasoning: ACL2 Case Studies*, pages 75–92, Boston, MA., 2000. Kluwer Academic Press.
12. M. Kaufmann and J S. Moore. A precise description of the ACL2 logic. In *http://www.cs.utexas.edu/users/moore/publications/km97a.ps.gz*. Dept. of Computer Sciences, University of Texas at Austin, 1997.
13. M. Kaufmann and J S. Moore. Structured Theory Development for a Mechanized Logic. *Journal of Automated Reasoning*, 26(2):161–203, 2001.
14. M. Kaufmann and J S. Moore. The ACL2 home page. In *http://www.cs.utexas.edu/users/moore/acl2/*. Dept. of Computer Sciences, University of Texas at Austin, 2008.
15. M. Kaufmann and J S. Moore. Proof Search Debugging Tools in ACL2. In R. Boulton, J. Hurd, and K. Slind, editors, *Tools and Techniques for Verification of System Infrastructure, A Festschrift in honour of Prof. Michael J. C. Gordon FRS*. Royal Society, London, http://www.ttvsi.org/, March 2008.
16. M. Kaufmann and J S. Moore. The ACL2 User's Manual. In *http://www.cs.utexas.edu/users/moore/acl2/#User's-Manual*. Dept. of Computer Sciences, University of Texas at Austin, 2008.
17. M Kaufmann, J S. Moore, S. Ray, and E. Reeber. Integrating external deduction tools with ACL2. In C. Benzmueller, B. Fischer, and G. Sutcliffe, editors, *Proceedings of the 6th International Workshop on Implementation of Logics (IWIL 2006)*, volume 212 of *CEUR Workshop Proceedings*, pages 7–26, 2006. to appear in the Journal of Applied Logic.
18. P. Manolios and J S. Moore. Partial functions in ACL2. *Journal of Automated Reasoning*, 31(2):107–127, 2003.
19. D. L. Rager. Adding parallelism capabilities to ACL2. In *Proceedings of the Sixth International Workshop on the ACL2 Theorem Prover and its applications*, pages 90–94, New York, NY, USA, 2006. ACM.
20. G. L. Steele, Jr. *Common Lisp The Language*. Digital Press, second edition, 1990.