Industrial Use of ACL2: Applications, Achievements, Challenges, and Directions

J Strother Moore and Marijn J.H. Heule

Department of Computer Science
The University of Texas at Austin
{moore,marijn}@cs.utexas.edu

Abstract

Industrial applications of interactive theorem proving dates back to the eighties. Enabling and achieving industrial successes has been an important focus of the ACL2 community. The ARCADE call-for-papers appears to ignore these results and the potential of automated reasoning in industry in the future. We briefly describe the penetration of the ACL2 theorem proving system into the microprocessor industry, list some of milestones achieved, the obstacles standing in the way, and some future research directions.

1 Prehistory

Interactive theorem proving was on the doorstep of industrial application 30 years ago.

In the period 1987 through 1992, the Nqthm [2] prover was used to build a "verified stack," a hardware/software hierarchy whose base was a netlist description of a simple 32 bit microprocessor upon which were built an assembler, linker, loader, operating system, compilers, and applications: all were verified – and verified to "fit together" – with Nqthm. This was described in a special issue of JAR in 1989 [1] and later updated (upon fabrication of the chip) in the book [10].[1]

Concurrently with this project the same group also formalized about 80% of the machine code of the Motorola 68020 and then used Nqthm to verify 21 of the 22 programs in the Berkeley C String Library after compiling them with `gcc -o`. Three bugs were found. The same approach was used to verify other programs of practical interest including a C implementation of Hoare's *in situ* quick sort [3].

Shortly after arriving at that "doorstep" 30 years ago, interactive theorem proving stepped over it, into regular industrial application.

2 ACL2

A major attraction of Nqthm for industrial use was the fact that it supported an executable programming language as its logic. Prototypes or models of various computational artifacts could be built and tested on examples, but theorems could also be proved. The main problem holding back the industrial use of Nqthm was the fact that the programming language it supported was a homegrown Pure Lisp that did not execute fast enough.

In 1989, Boyer and Moore threw out their homegrown Lisp, adopted the first-order functional subset of ANSI standard Common Lisp as their logic, and implemented an Nqthm-like theorem prover for it. The resulting programming language, logic, and theorem prover is called ACL2:

[1]To keep the bibliography of this paper short we have chosen to cite only summary articles. Those articles contain citations to technical reports and articles giving full details.

A Computational Logic for Applicative Common Lisp. Eventually, Boyer left the project and Matt Kaufmann joined Moore as the co-author of ACL2.

ACL2 first found industrial use in 1993.

Proof of our contention that ACL2 was in industrial use over 20 years ago can be found in the paper "ACL2 Theorems about Commercial Microprocessors" [4]. The paper briefly described the (1) ACL2 formalization in collaboration with Motorola hardware designers, of a commercial digital signal processor (DSP), called CAP, then under development by Motorola, (2) a demonstration that the ACL2 model ran several times faster than Motorola's SPW engineering model on actual test suites, (3) the proof that the behavioral-level specification described every well-defined behavior of the CAP, (4) the definition of an ACL2 function that recognized pipeline hazards in microcode programs (part of the formalization of "well-defined"), (5) a proof that when the function approved a piece of microcode the code would run on the design as per the high-level semantics of the microcode, (6) the verification of the microcode for a finite impulse response (FIR) filter commonly used in DSPs and written by Motorola engineers, (6) the verification of the code for a statistical filtering and peak finding algorithm for scanning digital spectra, also written by Motorola engineers; and (7) the formalization and proof of correctness of the microcode for floating point division on the Advanced Micro Devices (AMD) AMD5K86 microprocessor (AMD's then-competitor with the Intel Pentium I), which was done in collaboration with the AMD floating-point design team and completed before the AMD5K86 was fabricated.

These projects, completed before the end of 1995, demonstrated that ACL2 was ready for prime time in industrial settings. It has been regularly used in such settings ever since. Among the noteworthy achievements are:

- verification of all elementary floating-point arithmetic on the AMD Athlon, after running 100M test vectors successfully comparing the ACL2 model with the AMD RTL simulator;
- verification of all elementary floating-point arithmetic on the AMD Opteron;
- verification of a silicon implementation of a JVM chip by Rockwell-Collins;
- verification of the Rockwell Collins AAMP7 crypto chip (the basis for obtaining NSA MILS certification);
- verification of the Greenhills operating system;
- verification of important invariants in the Sun JVM class loader and properties assured by the Sun byte-code verifier;
- verification of the Centaur Technology, Inc., Verilog design for the VIA Nano floating point adder which handles 32-bit, 64-bit, and 80-bit additions, is pipelined to deliver 4 results per cycle, has 1074 input signals including 26 clock signals and 374 output signals, consists of 33,700 lines of Verilog in 680 modules requiring 432,322 transistors;
- checking of a computationally surveyable proof of important properties of an Intel implementation of the elliptic curve key agreement including that $2^{255} - 19$ is prime and that the elliptic curve known as Curve25519 is an abelian group;
- verification of floating point designs at Oracle and ARM.

For more details see [7, 9]. In addition to technical matters, [7] describes "soft" aspects of the ACL2 user community that facilitate industrial penetration, including focusing on the needs of industry (as opposed to publication), excellent documentation, a liberal license, and very responsive maintenance.

Some academic research further supports the industrial relevance of ACL2. At the University of Texas at Austin ACL2 was used to model the x86 instruction set architecture and supports formal analysis of both user-level and system-level code as well as efficient execution of the formal model [6]. The same group has recently verified an efficient checker for SAT proofs [5], which is already in use at Centaur. The verified checker increases the runtime of the tool chain by only 10% on large proofs. In the coming weeks a proof of 3 petabytes will be checked to show that any proof can be validated with a verified checker.

3 Reasons ACL2 is Used in Industry

Perhaps the main reason ACL2 has been adopted by industry is that it provides fast and efficient execution, well-supported (standard Common Lisp) programming and debugging environments across many hardware/software platforms, and a powerful theorem prover. Because ACL2's behavior can be heavily influenced by previously proved lemmas it is possible to develop libraries of theorems that automate many proofs in some domains. This allows, for example at Centaur, nightly verification runs of all previously verified modules that were changed during the day. Centaur devotes about 150 CPUs to this nightly run.

Because it is a general purpose programming language, there are a variety of other tools written in ACL2, including linters, an RTL design browser, a reverse engineering tool used to extract clock trees, and a tool to produce understandable reports about synthesized circuits. All Centaur analysis tools are driven off the same source: an ACL2 object representing the parsed Verilog design of the entire chip.

In addition, because ACL2 can be extended with new proof techniques coded in ACL2 and verified by ACL2, industry frequently uses ACL2 to implement special-purpose tools. Among the most commonly used extensions is a verified mechanism for "bit blasting" using BDDs or AIGs so that many finite arithmetic problems can be solved without user interaction [11]. The proof of correctness of this tool establishes that if the bit-blasting algorithm succeeds then the formula is actually a theorem of ACL2, which means that theorems proved with the tool can be mixed freely and soundly with other ACL2 theorems. Furthermore, the tool is just an ACL2 function and can be invoked within the ACL2 environment. By so integrating verified bit-blasting into ACL2, users are able to employ the rest of ACL2's powerful symbolic techniques to glue together the results of decision procedures, allowing more sophisticated specifications to be confidently verified.

4 Industrial Complaints about ACL2

The primary complaints by industry regarding ACL2 are probably not what most researchers in theorem proving would predict. The most often heard complaint is that ACL2 does not execute fast enough. The second most heard complaint is that it is inconvenient as a scripting language. The third is that it does not support visualization and graphics.

Note that these complaints rarely concern more academic matters like expressive limitations of first-order logic, absence of strong typing, and absence of explicit quantifiers.

5 Future Directions

In [8], ACL2 developers Kaufmann and Moore published a list of theorem proving research topics they regarded as important. The six areas discussed were analogy, learning and data mining; open architecture; parallel and collaborative theorem provers (including integration of decision procedures like SAT); user interface and interactive steering; education (meaning teaching engineers more about how to specify and verify their digital designs), and building a verified theorem prover. While this list is 13 years old, we think it still includes valuable research directions.

However, in light of the fact that it was assembled with ACL2 in mind, we omitted to say that we believe a useful theorem prover should support an executable programming language with good debugging tools, compilers, etc. In light of the complaints listed above by the industrial users of ACL2 we would add to our list: making the logic execute faster, integrating features from a scripting language, and supporting graphics.

Perhaps the best summary of our experience is: if you want to build a theorem prover used by industry, listen to what industrial users want.

References

[1] Bevier, W.R., Hunt, Jr., W.A., Moore, J S., Young, W.D.: Special issue on system verification. Journal of Automated Reasoning 5(4), 409–530 (1989)

[2] Boyer, R.S., Moore, J S.: A Computational Logic Handbook, Second Edition. Academic Press, New York (1997)

[3] Boyer, R.S., Yu, Y.: Automated proofs of object code for a widely used microprocessor. Journal of the ACM 43(1), 166–192 (January 1996)

[4] Brock, B., Kaufmann, M., Moore, J S.: ACL2 theorems about commercial microprocessors. In: Srivas, M., Camilleri, A. (eds.) Formal Methods in Computer-Aided Design (FMCAD'96). pp. 275–293. Springer-Verlag, LNCS 1166, Heidelberg (November 1996), http://www.cs.utexas.edu/users/moore/publications/bkm96.ps.Z

[5] Cruz-Filipe, L., Heule, M.J.H., Hunt Jr., W.A., Kaufmann, M., Schneider-Kamp, P.: Efficient certified RAT verification. In: Accepted for CADE (2017)

[6] Goel, S.: Formal Verification of Application and System Programs Based on a Validated x86 ISA Model. Ph.D. thesis, University of Texas at Austin (2016)

[7] Hunt, Jr., W.A., Kaufmann, M., Moore, J S., Slobodova, A.: Industrial hardware and software verification with ACL2. In: Verified Trustworthy Software Systems. vol. 375. The Royal Society (2017 (to appear)), (Article Number 20150399)

[8] Kaufmann, M., Moore, J S.: Some key research problems in automated theorem proving for hardware and software verification. Revista de la Real Academia de Ciencias (RACSAM) 98(1), 181–196 (2004)

[9] Kaufmann, M., Rager, D. (eds.): Thirteenth International Workshop on the ACL2 Theorem Prover and Its Applications, vol. 192. EPTCS (October 2015)

[10] Moore, J S.: Piton: A Mechanically Verified Assembly-Level Language. Automated Reasoning Series, Kluwer Academic Publishers (1996)

[11] Swords, S.: A verified framework for symbolic execution in the ACL2 theorem prover. http://hdl.handle.net/2152/ETD-UT-2010-12-2210 (2010)