ACL2 Support for Automated and Interactive Proof

Matt Kaufmann
The University of Texas at Austin
Dept. of Computer Science

KeY Workshop, July 27, 2015
OUTLINE

Introduction

Background

Demos (I)

Rewriting in ACL2

Demos (II)

Very Brief Survey of ACL2 Features

Conclusion
OUTLINE

Introduction

Background

Demos (I)

Rewriting in ACL2

Demos (II)

Very Brief Survey of ACL2 Features

Conclusion
Quoting the **ACL2 home page**: 

*ACL2 is a logic and programming language in which you can model computer systems, together with a tool to help you prove properties of those models. “ACL2” denotes "A Computational Logic for Applicative Common Lisp".*

**Goal for this talk:**
Give a sense of the ACL2 system, especially how it supports a combination of *automated proof* and *user interaction*.

**BUT HOW CAN I ACHIEVE THIS GOAL?**

My answer ....
I DON’T KNOW!

But here is my plan.

▶ **BACKGROUND**: Present basics, history, usage, etc.
▶ **DEMOS**: Give a sense of how ACL2 can be used.
▶ **REWRITING**: Summarize the primary way to control the ACL2 prover.
▶ **OTHER STUFF**: Survey *briefly* other ways that ACL2 supports user interaction.

*And the most important thing:*

▶ Please ask questions during the talk!
OUTLINE

Introduction

Background

Demos (I)

Rewriting in ACL2

Demos (II)

Very Brief Survey of ACL2 Features

Conclusion
OUTLINE

Introduction

Background

Demos (I)

Rewriting in ACL2

Demos (II)

Very Brief Survey of ACL2 Features

Conclusion
BACKGROUND: OVERVIEW

- ACL2 is freely available, including libraries of certifiable books
- Let’s explore the ACL2 home page.
- ACL2 is written mostly in itself (!).
  - About 10 MB of source code (Version 7.1).
- Bleeding edge for libraries (community books) and the ACL2 system are available from Github.
  - Well over 400,000 events (theorems, definitions, other) are evaluated in the community books.
Background: Overview (continued)

Development history:

- Boyer-Moore Theorem Provers go back to the start of their collaboration in 1971.

Industrial usage: As far as I know, ACL2 is the only interactive theorem prover (ITP) used with some regularity at several companies:

- AMD, Centaur, IBM, Intel, Oracle, Rockwell Collins

There are also users in the U.S. Government and universities, including —

- UT Austin: x86 interpreter defined in ACL2, validation by co-simulation, proofs about x86 machine code
PARTIAL TIMELINE

- Boyer and Moore meet
- insertion sort
- binary adder
- expression compiler
- prime factorization
- BDX930 abandoned
- RSA
- unsolvability of halting problem
- Logic formalization (Spain), ongoing
- IBM floating point algorithms
- x86 ring model/proof
- fast consensus analysis
- Galois/Rockwell SHADE
- sixth ACL2 workshop
- Buyer/seller
- Rockwell Greenhills OS
- initial ACL2 workshop
- Rockwell JEM1
- clock sync
- biphase mark
- Motorola 68020
- AMD floating-point rtl, ongoing
- Logic formalization (Spain), ongoing
- AMD K5 floating-point division µcode
- real-time model
- Byzantine Generals
- micro Gypsy compiler
- Piton
- KIT OS kernel
- FM8502
- Gauss
- Unity
- FM9001
- FM9801
- Paris-Harrington Ramsey
- Motorola CAP
- DEC alpha
- Nqthm compiler
- Galois/Rockwell SHADE
- Kalman filters
- x86 ISA
- Y86 with STOBJ
- UCLID integration prototype
- AAMP7G MIL cert.
- Dijkstra shortest path
- ACM Software System Award
- Y86 with STOBJ
- X86 ISA
- X86 with STOBJ
**Background: Logic**

The ACL2 logic is a first-order logic with induction up to $\varepsilon_0$.

But all ACL2 theories extend a given *ground-zero* theory, which axiomizes data types for:

- numbers (complex rationals), characters, strings, symbols;
- trees and lists, using a pairing operation (**cons**).

ACL2 extensions are *conservative* (a demo will discuss this).

BACKGROUND: STRENGTHS

- Proof automation
- Support for user interaction
- Fast execution
- Documentation (about 100,000 lines for just the system)
- Interfaces include Emacs
  (Is that really an interface? A strength?) and the Eclipse-based ACL2 Sedan.

A potential weakness: first-order logic with only basic quantifier support (but recursion helps).
Background: More information

**NOTE:** A longer variant of this talk, but targeted to CS grad students and with more focus on *using* ACL2, is here:


That talk mentions this link to several demos and their logs:

OUTLINE

Introduction

Background

Demos (I)

Rewriting in ACL2

Demos (II)

Very Brief Survey of ACL2 Features

Conclusion
OUTLINE

Introduction

Background

Demos (I)

Rewriting in ACL2

Demos (II)

Very Brief Survey of ACL2 Features

Conclusion
Demos (I)

- All demos today, with logs, are in the gzipped tar file `demos.tgz` in this directory.

- ACL2 programming and evaluation
  [DEMO]: file `demo-1.lsp` (log `demo-1-log.txt`)

- ACL2 as an automatic theorem prover
  [DEMO]: file `demo-2.lsp` (log `demo-2-log.txt`)
  - ACL2 provides *automation* for induction, linear arithmetic, Boolean reasoning, rule application, ...
  - ... but lemmas are usually needed (sometimes from libraries).
ACL2 supports formally verified extensions.

In particular, GL is a verified clause processor defined and verified by an ACL2 user, Sol Swords. GL does proofs about finite domains by bit-blasting.

The next demo illustrates GL. It also shows the use of LOCAL, for “private” events (using conservativity).

[DEMO]: book demo-gl.lisp
(log demo-gl-log.txt)
OUTLINE

Introduction

Background

Demos (I)

Rewriting in ACL2

Demos (II)

Very Brief Survey of ACL2 Features

Conclusion
OUTLINE

Introduction

Background

Demos (I)

Rewriting in ACL2

Demos (II)

Very Brief Survey of ACL2 Features

Conclusion
ACL2 is typically controlled with conditional rewrite rules, although there are other rule-classes too.

The basic idea: the ACL2 rewriter automatically applies the rule

\[ H \rightarrow L = R \]

by replacing an instance \( L/s \) of \( L \) by \( R/s \), when the rewriter can verify \( H/s \).

The documentation topic for rewrite shows many ways to control the rewriter (needed only occasionally). I’ll mention only a few:
Rewriting in ACL2 (2)

- **backchain-limit**: limit effort to relieve hypotheses
- **force**: defer proving $H$ if necessary
- **hide**: hide a term from the rewriter
- **syntaxp**: attach a heuristic filter on a rule

Example of **syntaxp**: consider $3 + (4 + x)$.
It’s already in normal form: right-associated.
Our wish: $3 + (4 + x) = (3 + 4) + x = 7 + x$.

```lisp
ACL2 !>:pe associativity-of-+
-997 (DEFAXIOM ASSOCIATIVITY-OF-+
  (EQUAL (+ (+ X Y) Z) (+ X (+ Y Z))))

ACL2 !>:pe fold-consts-in-+
-158 (DEFTHM FOLD-CONSTS-IN+-
  (IMPLIES (AND (SYNTAXP (QUOTE X))
    (SYNTAXP (QUOTE Y)))
  (EQUAL (+ X (+ Y Z)) (+ (+ X Y) Z))))
```

ACL2 !>
Rewriting in ACL2 (3)

\[ H \rightarrow L = R \quad \text{(ordinary rewrite rule)} \]

\[ H \rightarrow L \sim R \quad \text{(congruence-based rewrite rule)} \]

where \( \sim \) is an equivalence relation.

Users prove equivalence, congruence, and refinement rules, to tell ACL2 when such rewrites are valid.


OUTLINE

Introduction

Background

Demos (I)

Rewriting in ACL2

Demos (II)

Very Brief Survey of ACL2 Features

Conclusion
OUTLINE

Introduction

Background

Demos (I)

Rewriting in ACL2

Demos (II)

Very Brief Survey of ACL2 Features

Conclusion
Time permitting . . .

These demos are excerpted from my TPHOLs 2008 talk.

**NOTE:** Here I’ll just show highlights, to give an impression about ACL2 usage.

- **JVM demo**

- **Our final demo** illustrates “the method” recommended for dealing with proof failures.
OUTLINE

Introduction

Background

Demos (I)

Rewriting in ACL2

Demos (II)

Very Brief Survey of ACL2 Features

Conclusion
Very Brief Survey of ACL2 Features

Let’s see a bit more about how ACL2 supports proof development . . .

▶ . . . by exploring briefly the ACL2 documentation.

NOTE:
I would be very happy to elaborate on any of these topics!
In particular, we might explore a few debugging features, as time and interest permit.

- accumulated-persistence
- break-rewrite
- cgen
- cw-gstack
- disassemble$
- dmr
- failed-forcing
- failure
- find-lemmas
- forward-chaining-reports
- guard-debug
- measure-debug
- nil-goal
- print-gv
- profile-acl2
- profile-all
- proof-checker
- proof-tree
- pstack
- quick-and-dirty-subsumption-replacement-step
- redo-flat
- remove-hyps
- set-debugger-enable
- set-guard-msg
- sneaky
- spacewalk
- splitter
- time-tracker
- trace
- walkabout
- watch
OUTLINE

Introduction

Background

Demos (I)

Rewriting in ACL2

Demos (II)

Very Brief Survey of ACL2 Features

Conclusion
ACL2 has a 25 (or 44) year history and is used in industry.

“Microprocessor design goes daily through numerous optimizations that affect thousands of lines of code. These optimizations must be proved correct.”
— Anna Slobodova, verification manager at Centaur Technology

ACL2 provides automation but scales to large problems . . . . . . with libraries and with user guidance.

For more information, see the ACL2 home page.

THANK YOU!