ACL2: Implementation of a Computational Logic

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

June 10, 2015

HELLO!

HELLO!

I'm so happy to be visiting here! I plan to be in Gothenburg until August 15.

HELLO!

I'm so happy to be visiting here! I plan to be in Gothenburg until August 15.

Today I'll discuss a logic and software tool, ACL2, which has been my focus off and on since the early 1990s.

HELLO!

I'm so happy to be visiting here! I plan to be in Gothenburg until August 15.

Today I'll discuss a logic and software tool, ACL2, which has been my focus off and on since the early 1990s.

But my intention in Gothenburg is to return to my roots in model theory, especially models of set theory and arithmetic.

HELLO!

I'm so happy to be visiting here! I plan to be in Gothenburg until August 15.

Today I'll discuss a logic and software tool, ACL2, which has been my focus off and on since the early 1990s.

But my intention in Gothenburg is to return to my roots in model theory, especially models of set theory and arithmetic. *Thanks, Ali!*

Outline

Outline

Overview

Overview

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** **C***omputational* **L***ogic for* **A***pplicative* **C***ommon* **L***isp".*

OVERVIEW

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*** ***C****omputational* **L***ogic for* **A***pplicative* **C***ommon* **L***isp".*

Goal for this talk:
Say something about ACL2 of interest to logicians.

Overview

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*** ***C****omputational* ***L****ogic for* ***A****pplicative* ***C****ommon* ***L****isp".*

Goal for this talk:
Say something about ACL2 of interest to logicians.

- The focus will be on mechanizing logic for a practical proof assistant.

Overview

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*** ***C****omputational* ***L****ogic for* ***A****pplicative* ***C****ommon* ***L****isp".*

Goal for this talk:
Say something about ACL2 of interest to logicians.

- The focus will be on mechanizing logic for a practical proof assistant.
- Boring or not, logical challenges must be addressed!

Overview

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* **L***ogic for* **A***pplicative* **C***ommon* **L***isp".*

Goal for this talk:
Say something about ACL2 of interest to logicians.

- The focus will be on mechanizing logic for a practical proof assistant.
- Boring or not, logical challenges must be addressed! (Note: ACL2 does not generate formal proofs.)

OVERVIEW: THE PLAN FOR TODAY

I'll start by following these slides . . .

OVERVIEW: THE PLAN FOR TODAY

I'll start by following these slides . . .
. . . but I'd be happy to take us in whatever direction you'd prefer (if I can!).

Overview: The plan for today

I'll start by following these slides . . .
. . . but I'd be happy to take us in whatever direction you'd prefer (if I can!).

I've prepared about an hour's worth of material, so there should be plenty of time to explore . . .

Overview: The plan for today

I'll start by following these slides . . .
. . . but I'd be happy to take us in whatever direction you'd prefer (if I can!).

I've prepared about an hour's worth of material, so there should be plenty of time to explore . . .
. . . and of course, I can skip slides.

Overview: The plan for today

I'll start by following these slides . . .
. . . but I'd be happy to take us in whatever direction you'd prefer (if I can!).

I've prepared about an hour's worth of material, so there should be plenty of time to explore . . .
. . . and of course, I can skip slides.

Please feel free to ask questions!

OVERVIEW: THE PLAN FOR TODAY

I'll start by following these slides . . .
. . . but I'd be happy to take us in whatever direction you'd prefer (if I can!).

I've prepared about an hour's worth of material, so there should be plenty of time to explore . . .
. . . and of course, I can skip slides.

Please feel free to ask questions!

Let's start with some context.

Overview: Formal Verification

Many organizations now use tools to *formally verify* hardware and software systems, augmenting traditional **testing** by using tools based on some notion of **proof**.

Overview: Formal Verification

Many organizations now use tools to *formally verify* hardware and software systems, augmenting traditional **testing** by using tools based on some notion of **proof**.

Such tools are typically *equivalence checkers*, *model checkers*, or *static checkers*.

Overview: Formal Verification

Many organizations now use tools to *formally verify* hardware and software systems, augmenting traditional **testing** by using tools based on some notion of **proof**.

Such tools are typically *equivalence checkers*, *model checkers*, or *static checkers*.

But occasionally, *interactive theorem provers* (ITPs) are used, e.g. Coq, Isabelle, HOL4, PVS, Agda — or ACL2.

Overview: Formal Verification

Many organizations now use tools to *formally verify* hardware and software systems, augmenting traditional **testing** by using tools based on some notion of **proof**.

Such tools are typically *equivalence checkers*, *model checkers*, or *static checkers*.

But occasionally, *interactive theorem provers* (ITPs) are used, e.g. Coq, Isabelle, HOL4, PVS, Agda — or ACL2.

As far as I know, ACL2 is the only ITP used with some regularity at several companies.

Overview: Formal Verification

Many organizations now use tools to *formally verify* hardware and software systems, augmenting traditional **testing** by using tools based on some notion of **proof**.

Such tools are typically *equivalence checkers*, *model checkers*, or *static checkers*.

But occasionally, *interactive theorem provers* (ITPs) are used, e.g. Coq, Isabelle, HOL4, PVS, Agda — or ACL2.

As far as I know, ACL2 is the only ITP used with some regularity at several companies.

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

Overview: Formal Verification

Many organizations now use tools to *formally verify* hardware and software systems, augmenting traditional **testing** by using tools based on some notion of **proof**.

Such tools are typically *equivalence checkers*, *model checkers*, or *static checkers*.

But occasionally, *interactive theorem provers* (ITPs) are used, e.g. Coq, Isabelle, HOL4, PVS, Agda — or ACL2.

As far as I know, ACL2 is the only 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**.

Overview: Formal Verification

Many organizations now use tools to *formally verify* hardware and software systems, augmenting traditional **testing** by using tools based on some notion of **proof**.

Such tools are typically *equivalence checkers*, *model checkers*, or *static checkers*.

But occasionally, *interactive theorem provers* (ITPs) are used, e.g. Coq, Isabelle, HOL4, PVS, Agda — or ACL2.

As far as I know, ACL2 is the only 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**.

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

Overview: ITP Systems

Just a few words about interactive theorem proving...

Overview: ITP Systems

Just a few words about interactive theorem proving. . .

- Yearly ITP conference

OVERVIEW: ITP SYSTEMS

Just a few words about interactive theorem proving...

- Yearly ITP conference
- Many ITP systems (*e.g.*, ACL2) can send sub-problems to automatic proof tools, e.g., SAT solvers for Boolean problems.

Overview: ITP Systems

Just a few words about interactive theorem proving. . .

- Yearly ITP conference
- Many ITP systems (*e.g.*, ACL2) can send sub-problems to automatic proof tools, e.g., SAT solvers for Boolean problems.
- ITP is typically more scalable than automatic theorem proving, but requires some human assistance.

OVERVIEW: ITP SYSTEMS

Just a few words about interactive theorem proving. . .

- Yearly ITP conference
- Many ITP systems (*e.g.*, ACL2) can send sub-problems to automatic proof tools, e.g., SAT solvers for Boolean problems.
- ITP is typically more scalable than automatic theorem proving, but requires some human assistance. For ACL2: prove lemmas used to simplify terms in later proofs.

Overview: ITP Systems

Just a few words about interactive theorem proving. . .

- Yearly ITP conference
- Many ITP systems (*e.g.*, ACL2) can send sub-problems to automatic proof tools, e.g., SAT solvers for Boolean problems.
- ITP is typically more scalable than automatic theorem proving, but requires some human assistance. For ACL2: prove lemmas used to simplify terms in later proofs.

REMARK (thanks to J Moore for this):

> *All industrial-scale deduction tools are, in a deep sense, interactive, even the ones that claim to be automatic. The issue is HOW MUCH interaction is required to do interesting things.*

Overview: On Using ACL2

This talk will focus on logical aspects of ACL2, so will say rather little about *using* ACL2.

OVERVIEW: ON USING ACL2

This talk will focus on logical aspects of ACL2, so will say rather little about *using* ACL2.

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

OVERVIEW: ON USING ACL2

This talk will focus on logical aspects of ACL2, so will say rather little about *using* ACL2.

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

http://www.cs.utexas.edu/users/kaufmann/talks/acl2-intro-2015-04/acl2-intro.pdf

OVERVIEW: ON USING ACL2

This talk will focus on logical aspects of ACL2, so will say rather little about *using* ACL2.

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

http://www.cs.utexas.edu/users/kaufmann/talks/acl2-intro-2015-04/acl2-intro.pdf

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

http://www.cs.utexas.edu/users/kaufmann/talks/acl2-intro-2015-04/demos.tgz

OUTLINE

OUTLINE

ACL2 Introduction

- Freely available, including libraries of *certifiable books*

ACL2 Introduction

- Freely available, including libraries of *certifiable books*
- Let's explore the ACL2 home page.

ACL2 Introduction

- Freely available, including libraries of *certifiable books*
- Let's explore the ACL2 home page.
- ACL2 is written mostly in itself (!).

ACL2 Introduction

- 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).

ACL2 Introduction

- 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.

ACL2 Introduction

- 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.
- Workshop series: #13 is at UT, Oct. 1-2, 2015.

ACL2 Introduction

- 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.
- Workshop series: #13 is at UT, Oct. 1-2, 2015.
- History

ACL2 Introduction

- 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.
- Workshop series: #13 is at UT, Oct. 1-2, 2015.
- History
  - Bob Boyer and J Moore started ACL2 in 1989. I joined and Bob dropped out in 1993. J and I continue its development.

ACL2 Introduction

- 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.
- Workshop series: #13 is at UT, Oct. 1-2, 2015.
- History
  - Bob Boyer and J Moore started ACL2 in 1989. I joined and Bob dropped out in 1993. J and I continue its development.
  - *Boyer-Moore Theorem Provers* go back to the start of their collaboration in 1971.

ACL2 Demos

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

ACL2 Demos

- 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 DEMOS

- 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, . . .

ACL2 DEMOS

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

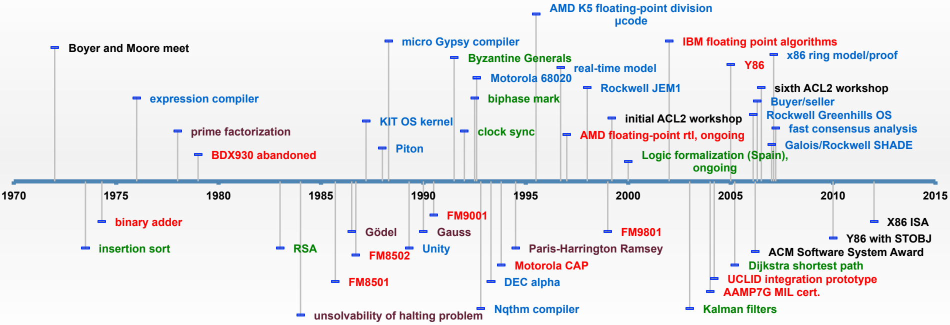
- ACL2 as an automatic theorem prover
  [DEMO]: file `demo-2.lsp`
  (log `demo-2-log.txt`)
  - ACL2 provides automation for induction, linear arithmetic, Boolean reasoning, rule application, ...
- The demos above, with logs, are in the gzipped tar file `demos-1-and-2.tgz` in this directory.

ACL2 Demos

- 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, . . .
- The demos above, with logs, are in the gzipped tar file `demos-1-and-2.tgz` in this directory.
- Interfaces include Emacs, ACL2 Sedan (Eclipse-based), none.

Partial timeline



Some ACL2 features *not* discussed further today:

- Prover algorithms
  - Waterfall, linear arithmetic, Boolean reasoning, . . .
  - Rewriting: Conditional, congruence-based, rewrite cache, syntaxp, bind-free, . . .
- Using the system effectively
  - The-method and introduction-to-the-theorem-prover
  - Theories, hints, rule-classes, . . .
  - Accumulated-persistence, brr, proof-checker, dmr, . . .
- Programming support, including (just a few):
  - Guards
  - Hash-cons and function memoization
  - Packages
  - Mutable State, stobjs, arrays, applicative hash tables, . . .
- System-level: Emacs support, books and certification, abbreviated printing, parallelism (ACL2(p)), . . .

OUTLINE

Outline

Overview

ACL2 Introduction

Foundations

Conclusion

FOUNDATIONS (1)

The ACL2 logic is a first-order logic with induction up to ε_0.

FOUNDATIONS (1)

The ACL2 logic is a first-order logic with induction up to ε_0.

But all ACL2 theories extend a given *ground-zero* theory, which is essentially Peano Arithmetic with ε_0-induction, extended with data types for:

Foundations (1)

The ACL2 logic is a first-order logic with induction up to ε_0.

But all ACL2 theories extend a given *ground-zero* theory, which is essentially Peano Arithmetic with ε_0-induction, extended with data types for:

- characters,
- strings,
- symbols,
- complex numbers with rational coefficients, and
- closure under a pairing operation (`cons`).

FOUNDATIONS (2)

Evolving theories: conservative extensions

FOUNDATIONS (2)

Evolving theories: conservative extensions

- Theory T_1 is a *conservative extension* of theory T_0 if every theorem of T_1 in the language of T_0 is a theorem of T_0.

FOUNDATIONS (2)

Evolving theories: conservative extensions

- Theory T_1 is a *conservative extension* of theory T_0 if every theorem of T_1 in the language of T_0 is a theorem of T_0.
- ACL2 extensions are conservative . . .

FOUNDATIONS (2)

Evolving theories: conservative extensions

- Theory T_1 is a *conservative extension* of theory T_0 if every theorem of T_1 in the language of T_0 is a theorem of T_0.
- ACL2 extensions are conservative ...
  - ... even with recursive definitions, since “termination” must be provable.

FOUNDATIONS (2)

Evolving theories: conservative extensions

- Theory T_1 is a *conservative extension* of theory T_0 if every theorem of T_1 in the language of T_0 is a theorem of T_0.
- ACL2 extensions are conservative . . .
  - . . . even with recursive definitions, since "termination" must be provable.
  - M. Kaufmann and J Moore, "Structured Theory Development for a Mechanized Logic." *Journal of Automated Reasoning* 26, no. 2 (2001) 161-203.

FOUNDATIONS (2)

Evolving theories: conservative extensions

- Theory T_1 is a *conservative extension* of theory T_0 if every theorem of T_1 in the language of T_0 is a theorem of T_0.
- ACL2 extensions are conservative . . .
  - . . . even with recursive definitions, since “termination” must be provable.
  - M. Kaufmann and J Moore, “Structured Theory Development for a Mechanized Logic.” *Journal of Automated Reasoning* 26, no. 2 (2001) 161-203.
- Importance: One may want to introduce new concepts to carry out some proofs, but this must be done conservatively in order to believe the results.

FOUNDATIONS (3)

Fun example in ACL2(r), an extension of ACL2 that supports the real numbers due to Ruben Gamboa:

Foundations (3)

Fun example in ACL2(r), an extension of ACL2 that supports the real numbers due to Ruben Gamboa:
The Overspill Principle of non-standard analysis.

FOUNDATIONS (3)

Fun example in ACL2(r), an extension of ACL2 that supports the real numbers due to Ruben Gamboa:
The Overspill Principle of non-standard analysis.

- `overspill.lisp`: Nice result

FOUNDATIONS (3)

Fun example in ACL2(r), an extension of ACL2 that supports the real numbers due to Ruben Gamboa:
The Overspill Principle of non-standard analysis.

- `overspill.lisp`: Nice result
- `overspill-proof.lisp`: Ugly proof, but `local` to the main proof, by conservativity

FOUNDATIONS (4)

Many "simple" logical issues require care in the implementation. While `LOCAL` is a great example, there are others.

FOUNDATIONS (4)

Many "simple" logical issues require care in the implementation. While `LOCAL` is a great example, there are others.

We'll look at just a few on the next slides.

DEFATTACH (1)

`Defattach` allows non-conservative extensions. Example:

DEFATTACH (1)

`Defattach` allows non-conservative extensions. Example:

- **Constraint for** "specification" function `spec`:
  $x \in \mathbb{Z} \implies \texttt{spec}(z) \in \mathbb{Z}$

DEFATTACH (1)

`Defattach` allows non-conservative extensions. Example:

- **Constraint for** "specification" function `spec`:
  $x \in \mathbb{Z} \implies \texttt{spec}(z) \in \mathbb{Z}$
- **Define** function `f`: $\texttt{f}(x, y) = \texttt{spec}(x + y)$

DEFATTACH (1)

`Defattach` allows non-conservative extensions. Example:

- **Constraint for** "specification" function `spec`:
  $x \in \mathbb{Z} \implies \texttt{spec}(z) \in \mathbb{Z}$
- **Define** function `f`: $\texttt{f}(x, y) = \texttt{spec}(x + y)$
- **Define** "implementation function" `impl`: $\texttt{impl}(x) = 10 * x$

DEFATTACH (1)

`Defattach` allows non-conservative extensions. Example:

- **Constraint for** “specification” function `spec`:
  $x \in \mathbb{Z} \implies \texttt{spec}(z) \in \mathbb{Z}$
- **Define** function `f`: $\texttt{f}(x, y) = \texttt{spec}(x + y)$
- **Define** “implementation function” `impl`: $\texttt{impl}(x) = 10 * x$
- Attach `impl` to `spec`:
  `(defattach spec impl)`

DEFATTACH (1)

`Defattach` allows non-conservative extensions. Example:

- **Constraint for** "specification" function `spec`:
  $x \in \mathbb{Z} \implies \texttt{spec}(z) \in \mathbb{Z}$
- **Define** function `f`: $\texttt{f}(x, y) = \texttt{spec}(x + y)$
- **Define** "implementation function" `impl`: $\texttt{impl}(x) = 10 * x$
- Attach `impl` to `spec`:
  `(defattach spec impl)`

Result not provable from axioms for `f` and `spec`:

```
ACL2 !>(f 3 4)
```

DEFATTACH (1)

`Defattach` allows non-conservative extensions. Example:

- **Constraint for** “specification” function `spec`:
  $x \in \mathbb{Z} \implies \texttt{spec}(z) \in \mathbb{Z}$
- **Define** function `f`: $\texttt{f}(x, y) = \texttt{spec}(x + y)$
- **Define** “implementation function” `impl`: $\texttt{impl}(x) = 10 * x$
- Attach `impl` to `spec`:
  `(defattach spec impl)`

Result not provable from axioms for `f` and `spec`:

```
ACL2 !>(f 3 4)
70
ACL2 !>
```

DEFATTACH (2)

Issues to consider:

DEFATTACH (2)

Issues to consider:

- Is `(local (defattach ...))` supported?

DEFATTACH (2)

Issues to consider:

- Is `(local (defattach ...))` supported?
  YES, `local` is supported.

DEFATTACH (2)

Issues to consider:

- Is `(local (defattach ...))` supported?
  YES, `local` is supported.
- Then how do we deal with conservativity?

DEFATTACH (2)

Issues to consider:

- Is `(local (defattach ...))` supported?
  YES, `local` is supported.
- Then how do we deal with conservativity?
  Two theories: The usual *current theory* and a stronger *evaluation theory*, extended using `defattach`.

DEFATTACH (2)

Issues to consider:

- Is `(local (defattach ...))` supported?
  YES, `local` is supported.
- Then how do we deal with conservativity?
  Two theories: The usual *current theory* and a stronger *evaluation theory*, extended using `defattach`.
- Ah, but what about this?

```
(thm (equal (f 3 4) 70))
```

DEFATTACH (2)

Issues to consider:

- Is `(local (defattach ...))` supported?
  YES, `local` is supported.
- Then how do we deal with conservativity?
  Two theories: The usual *current theory* and a stronger *evaluation theory*, extended using `defattach`.
- Ah, but what about this?

  ```
  (thm (equal (f 3 4) 70))
  ```

  The proof fails! (Whew!)

DEFATTACH (2)

Issues to consider:

- Is `(local (defattach ...))` supported?
  YES, `local` is supported.
- Then how do we deal with conservativity?
  Two theories: The usual *current theory* and a stronger *evaluation theory*, extended using `defattach`.
- Ah, but what about this?

  ```
  (thm (equal (f 3 4) 70))
  ```

  The proof fails! (Whew!)
- Why is the evaluation theory consistent?

DEFATTACH (2)

Issues to consider:

- Is `(local (defattach ...))` supported?
  YES, `local` is supported.
- Then how do we deal with conservativity?
  Two theories: The usual *current theory* and a stronger *evaluation theory*, extended using `defattach`.
- Ah, but what about this?

  ```
  (thm (equal (f 3 4) 70))
  ```

  The proof fails! (Whew!)
- Why is the evaluation theory consistent?
  A key requirement is that the attachment relation is suitably acyclic.

DEFATTACH (2)

Issues to consider:

- Is `(local (defattach ...))` supported?
  YES, `local` is supported.
- Then how do we deal with conservativity?
  Two theories: The usual *current theory* and a stronger *evaluation theory*, extended using `defattach`.
- Ah, but what about this?

  ```
  (thm (equal (f 3 4) 70))
  ```

  The proof fails! (Whew!)
- Why is the evaluation theory consistent?
  A key requirement is that the attachment relation is suitably acyclic.

For details, including issues pertaining to evaluation, see the *Essay on Defattach* comment in the ACL2 sources.

Quantification, Choice, & Induction (1)

Quantification is implemented using what amounts to a choice operator. Example:

QUANTIFICATION, CHOICE, & INDUCTION (1)

Quantification is implemented using what amounts to a choice operator. Example:

When asked to define
$r(y,z) = (\exists x)(p(x,y,z) \wedge q(xyz))$
ACL2 generates the following.

Quantification, Choice, & Induction (1)

Quantification is implemented using what amounts to a choice operator. Example:

When asked to define
$r(y,z) = (\exists x)(p(x,y,z) \wedge q(xyz))$
ACL2 generates the following.

Conservatively introduce $w(y,z)$ and $r(y,z)$ *using local witness*
$w(y,z) = (\varepsilon\ x)(p(x,y,z) \wedge q(x,y,z))$
to prove these axioms:

- $r(y,z) = (p(w(y,z),y,z) \wedge q(w(y,z),y,z))$
- $(p(x,y,z) \wedge q(x,y,z)) \implies r(y,z)$

QUANTIFICATION, CHOICE, & INDUCTION (2)

This sort of thing is clearly conservative (assuming the Axiom of Choice or at least well-orderable models). . .

Quantification, Choice, & Induction (2)

This sort of thing is clearly conservative (assuming the Axiom of Choice or at least well-orderable models)...

... IF we ignore induction!

Quantification, Choice, & Induction (2)

This sort of thing is clearly conservative (assuming the Axiom of Choice or at least well-orderable models)…

… IF we ignore induction!

Conservativity *with* induction follows from a model-theoretic forcing argument.

META-THEORETIC REASONING (1)

In ACL2, you can:

META-THEORETIC REASONING (1)

In ACL2, you can:

- code a simplifier,

META-THEORETIC REASONING (1)

In ACL2, you can:

- code a simplifier,
- prove that it is sound, and

META-THEORETIC REASONING (1)

In ACL2, you can:

- code a simplifier,
- prove that it is sound, and
- direct its use during later proofs.

META-THEORETIC REASONING (2)

ACL2 supports a notion of “eval”, together with this sort of *meta* theorem, directing the use of `fn` to transform terms that are calls of `nth` or of `foo`.

META-THEORETIC REASONING (2)

ACL2 supports a notion of "eval", together with this sort of *meta* theorem, directing the use of `fn` to transform terms that are calls of `nth` or of `foo`.

```
(defthm fn-correct-1
  (equal (evl x a)
         (evl (fn x) a))
  :rule-classes ((:meta :trigger-fns (nth foo))))
```

META-THEORETIC REASONING (2)

ACL2 supports a notion of "eval", together with this sort of *meta* theorem, directing the use of `fn` to transform terms that are calls of `nth` or of `foo`.

```
(defthm fn-correct-1
  (equal (evl x a)
         (evl (fn x) a))
  :rule-classes ((:meta :trigger-fns (nth foo))))
```

More complex forms are supported, including:

Meta-theoretic Reasoning (2)

ACL2 supports a notion of "eval", together with this sort of *meta* theorem, directing the use of `fn` to transform terms that are calls of `nth` or of `foo`.

```
(defthm fn-correct-1
  (equal (evl x a)
         (evl (fn x) a))
  :rule-classes ((:meta :trigger-fns (nth foo))))
```

More complex forms are supported, including:

- extended-metafunctions that take `STATE` and contextual inputs;

META-THEORETIC REASONING (2)

ACL2 supports a notion of "eval", together with this sort of *meta* theorem, directing the use of `fn` to transform terms that are calls of `nth` or of `foo`.

```
(defthm fn-correct-1
  (equal (evl x a)
         (evl (fn x) a))
  :rule-classes ((:meta :trigger-fns (nth foo))))
```

More complex forms are supported, including:

- extended-metafunctions that take `STATE` and contextual inputs;
- transformations at the goal level; and

META-THEORETIC REASONING (2)

ACL2 supports a notion of "eval", together with this sort of *meta* theorem, directing the use of `fn` to transform terms that are calls of `nth` or of `foo`.

```
(defthm fn-correct-1
  (equal (evl x a)
         (evl (fn x) a))
  :rule-classes ((:meta :trigger-fns (nth foo))))
```

More complex forms are supported, including:

- extended-metafunctions that take `STATE` and contextual inputs;
- transformations at the goal level; and
- hypotheses that extract known information from the logical world.

Meta-theoretic Reasoning (2)

ACL2 supports a notion of "eval", together with this sort of *meta* theorem, directing the use of `fn` to transform terms that are calls of `nth` or of `foo`.

```
(defthm fn-correct-1
  (equal (evl x a)
         (evl (fn x) a))
  :rule-classes ((:meta :trigger-fns (nth foo))))
```

More complex forms are supported, including:

- extended-metafunctions that take `STATE` and contextual inputs;
- transformations at the goal level; and
- hypotheses that extract known information from the logical world.

For details, including issues pertaining to evaluation, see the *Essay on Correctness of Meta Reasoning* comment in the ACL2 sources.

META-THEORETIC REASONING (2)

ACL2 supports a notion of "eval", together with this sort of *meta* theorem, directing the use of `fn` to transform terms that are calls of `nth` or of `foo`.

```
(defthm fn-correct-1
  (equal (evl x a)
         (evl (fn x) a))
  :rule-classes ((:meta :trigger-fns (nth foo))))
```

More complex forms are supported, including:

- extended-metafunctions that take `STATE` and contextual inputs;
- transformations at the goal level; and
- hypotheses that extract known information from the logical world.

For details, including issues pertaining to evaluation, see the *Essay on Correctness of Meta Reasoning* comment in the ACL2 sources. *Attachments* provide a challenge.

Other Logical Challenges

Here are some other challenges in the foundations of ACL2.

Other Logical Challenges

Here are some other challenges in the foundations of ACL2.

- *Functional instantiation* allows the replacement of functions $f_1, \ldots, f_k$ by other functions $g_1, \ldots, g_k$ such that the g_i satisfy the axioms introducing the f_i.

OTHER LOGICAL CHALLENGES

Here are some other challenges in the foundations of ACL2.

- *Functional instantiation* allows the replacement of functions $f_1, \ldots, f_k$ by other functions $g_1, \ldots, g_k$ such that the g_i satisfy the axioms introducing the f_i.
- *Packages* provide namespaces — *e.g.*, `PKG1::F` and `PKG2::F` are distinct. But packages introduce axioms such as `symbol-package-name(PKG1::F) = "PKG1"`. So package introduction is *not conservative* and hence must be recorded.

OTHER LOGICAL CHALLENGES

Here are some other challenges in the foundations of ACL2.

- *Functional instantiation* allows the replacement of functions $f_1, \ldots, f_k$ by other functions $g_1, \ldots, g_k$ such that the g_i satisfy the axioms introducing the f_i.
- *Packages* provide namespaces — *e.g.*, `PKG1::F` and `PKG2::F` are distinct. But packages introduce axioms such as `symbol-package-name(PKG1::F) = "PKG1"`. So package introduction is *not conservative* and hence must be recorded.
- One can specify a *measure* in order to admit a recursive definition. But what if the measure is defined in terms of a function whose definition is `LOCAL`?

Outline

Outline

CONCLUSION

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

CONCLUSION

- ACL2 has a 25 (or 44) year history and is used in industry.
  - People are actually *paid* to prove theorems with ACL2.

Conclusion

- ACL2 has a 25 (or 44) year history and is used in industry.
  - People are actually *paid* to prove theorems with ACL2.
- As an ITP system, it relies on user guidance for large problems but enjoys scalability.

CONCLUSION

- ACL2 has a 25 (or 44) year history and is used in industry.
  - People are actually *paid* to prove theorems with ACL2.
- As an ITP system, it relies on user guidance for large problems but enjoys scalability.
- Mechanizing a logic, for efficient and flexible evaluation and proof, can present challenges.

CONCLUSION

- ACL2 has a 25 (or 44) year history and is used in industry.
  - People are actually *paid* to prove theorems with ACL2.
- As an ITP system, it relies on user guidance for large problems but enjoys scalability.
- Mechanizing a logic, for efficient and flexible evaluation and proof, can present challenges.
- For more information, see the ACL2 home page, in particular links to The Tours and Publications, which links to introductory material.

Conclusion

- ACL2 has a 25 (or 44) year history and is used in industry.
  - People are actually *paid* to prove theorems with ACL2.
- As an ITP system, it relies on user guidance for large problems but enjoys scalability.
- Mechanizing a logic, for efficient and flexible evaluation and proof, can present challenges.
- For more information, see the ACL2 home page, in particular links to The Tours and Publications, which links to introductory material.

THANK YOU!