Introduction to ACL2

Matt Kaufmann
The University of Texas at Austin
Dept. of Computer Science, GDC 7.804

April 21-23, 2015

Outline

Introduction

Context

Prover Automation and Control

ACL2 Variants

Foundations

Implementation

Conclusion

Outline

Introduction

Context
- UT Mechanized Reasoning Group
- The ACL2 system
- Interactive theorem proving (ITP)
- Formal verification

Prover Automation and Control
- Simple demo of typical use: sum to n
- Prover automation
- Prover control

ACL2 Variants

Foundations

Implementation

Conclusion

Introduction

My goal for these two talks is to provide a sense of the ACL2 theorem proving system, including:

INTRODUCTION

My goal for these two talks is to provide a sense of the ACL2 theorem proving system, including:

- what can be done with it, and how (several demos);

Introduction

My goal for these two talks is to provide a sense of the ACL2 theorem proving system, including:

- what can be done with it, and how (several demos);
- why bother to use it; and

INTRODUCTION

My goal for these two talks is to provide a sense of the ACL2 theorem proving system, including:

- what can be done with it, and how (several demos);
- why bother to use it; and
- the nature of its implementation and foundations (time permitting).

INTRODUCTION

My goal for these two talks is to provide a sense of the ACL2 theorem proving system, including:

- what can be done with it, and how (several demos);
- why bother to use it; and
- the nature of its implementation and foundations (time permitting).

Short answer to "why bother": many organizations now *formally verify* digital systems.

INTRODUCTION

My goal for these two talks is to provide a sense of the ACL2 theorem proving system, including:

- what can be done with it, and how (several demos);
- why bother to use it; and
- the nature of its implementation and foundations (time permitting).

Short answer to "why bother": many organizations now *formally verify* digital systems.

In essence, they *prove* systems correct rather than run massive tests that are woefully incomplete.

INTRODUCTION

My goal for these two talks is to provide a sense of the ACL2 theorem proving system, including:

- what can be done with it, and how (several demos);
- why bother to use it; and
- the nature of its implementation and foundations (time permitting).

Short answer to "why bother": many organizations now *formally verify* digital systems.

In essence, they *prove* systems correct rather than run massive tests that are woefully incomplete.

Some of those use ACL2. Others don't yet....

Introduction (page 2)

Quoting Bill Gates, April 18, 2002. Keynote address at WinHec 2002
[http://research.microsoft.com/en-us/projects/slam/]

> *Things like even software verification, this has been the Holy Grail of computer science for many decades but now in some very key areas, for example, driver verification we're building tools that can do actual proof about the software and how it works in order to guarantee the reliability.*

Introduction (page 3)

NOTE: All demos are available, together with corresponding log files, via the gzipped tar file `demos.tgz` in the directory of these slides.

Introduction (page 3)

NOTE: All demos are available, together with corresponding log files, via the gzipped tar file `demos.tgz` in the directory of these slides.

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

Introduction (page 3)

NOTE: All demos are available, together with corresponding log files, via the gzipped tar file `demos.tgz` in the directory of these slides.

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

- ACL2 as an automatic theorem prover
  [DEMO]: file `insertion-sort.lsp`
  (log `insertion-sort-log.txt`)

Introduction (page 3)

NOTE: All demos are available, together with corresponding log files, via the gzipped tar file `demos.tgz` in the directory of these slides.

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

- ACL2 as an automatic theorem prover
  [DEMO]: file `insertion-sort.lsp`
  (log `insertion-sort-log.txt`)

- Interfaces
  - Emacs (my preferred)
  - ACL2 Sedan (Eclipse-based interface)
  - None?

Outline

Introduction

Context

Prover Automation and Control

ACL2 Variants

Foundations

Implementation

Conclusion

Outline

Introduction

Context
- UT Mechanized Reasoning Group
- The ACL2 system
- Interactive theorem proving (ITP)
- Formal verification

Prover Automation and Control
- Simple demo of typical use: sum to n
- Prover automation
- Prover control

ACL2 Variants

Foundations

Implementation

Conclusion

Context

Next, we'll step back and see how ACL2 sits in relation to UT and to the overall picture of formal verification.

CONTEXT

Next, we'll step back and see how ACL2 sits in relation to UT and to the overall picture of formal verification.

Moving from specific to general....

UT MECHANIZED REASONING GROUP

- The UT mechanized reasoning group sits on GDC 7S.

UT MECHANIZED REASONING GROUP

- The UT mechanized reasoning group sits on GDC 7S.
- An ACL2 seminar typically takes place weekly; you're invited!

UT MECHANIZED REASONING GROUP

- The UT mechanized reasoning group sits on GDC 7S.
- An ACL2 seminar typically takes place weekly; you're invited!
- Personnel
  - Dr. Marijn Heule (SAT expert)
  - Prof. Warren Hunt (Group leader)
  - Prof. J Moore (ACL2 co-author; retired but very active)
  - Dr. Bill Young (Lecturer and researcher)
  - Dr. Matt Kaufmann (ACL2 co-author)
  - 5 Ph.D. students

UT MECHANIZED REASONING GROUP

- The UT mechanized reasoning group sits on GDC 7S.
- An ACL2 seminar typically takes place weekly; you're invited!
- Personnel
  - Dr. Marijn Heule (SAT expert)
  - Prof. Warren Hunt (Group leader)
  - Prof. J Moore (ACL2 co-author; retired but very active)
  - Dr. Bill Young (Lecturer and researcher)
  - Dr. Matt Kaufmann (ACL2 co-author)
  - 5 Ph.D. students
- **Contact us if you're interested in research opportunities.**

UT MECHANIZED REASONING GROUP

- The UT mechanized reasoning group sits on GDC 7S.
- An ACL2 seminar typically takes place weekly; you're invited!
- Personnel
  - Dr. Marijn Heule (SAT expert)
  - Prof. Warren Hunt (Group leader)
  - Prof. J Moore (ACL2 co-author; retired but very active)
  - Dr. Bill Young (Lecturer and researcher)
  - Dr. Matt Kaufmann (ACL2 co-author)
  - 5 Ph.D. students
- **Contact us if you're interested in research opportunities.**
  - Example: Nathan Wetzler is completing his Ph.D. on "Efficient, Mechanically-Verified Validation of Satisfiability Solvers" (proofs about SAT using ACL2)

THE ACL2 SYSTEM

- Freely available, including libraries of *certifiable books*

THE ACL2 SYSTEM

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

THE ACL2 SYSTEM

- Freely available, including libraries of *certifiable books*
- Let's explore the ACL2 home page.
- *Bleeding edge* for libraries (*community books*) and the ACL2 system are available from Github:
  `https://github.com/acl2/acl2`

THE ACL2 SYSTEM

- Freely available, including libraries of *certifiable books*
- Let's explore the ACL2 home page.
- *Bleeding edge* for libraries (*community books*) and the ACL2 system are available from Github:
  `https://github.com/acl2/acl2`
- Workshop series: #13 is here at UT, Oct. 1-2, ACES 2.402:
  `http://www.cs.utexas.edu/users/moore/acl2/workshop-2015`
  - The ACL2 Workshop 2015 chairs anticipate some scholarships being available for student registration fees.

THE ACL2 SYSTEM

- Freely available, including libraries of *certifiable books*
- Let's explore the ACL2 home page.
- *Bleeding edge* for libraries (*community books*) and the ACL2 system are available from Github:
  `https://github.com/acl2/acl2`
- Workshop series: #13 is here at UT, Oct. 1-2, ACES 2.402:
  `http://www.cs.utexas.edu/users/moore/acl2/workshop-2015`
  - The ACL2 Workshop 2015 chairs anticipate some scholarships being available for student registration fees.
- History

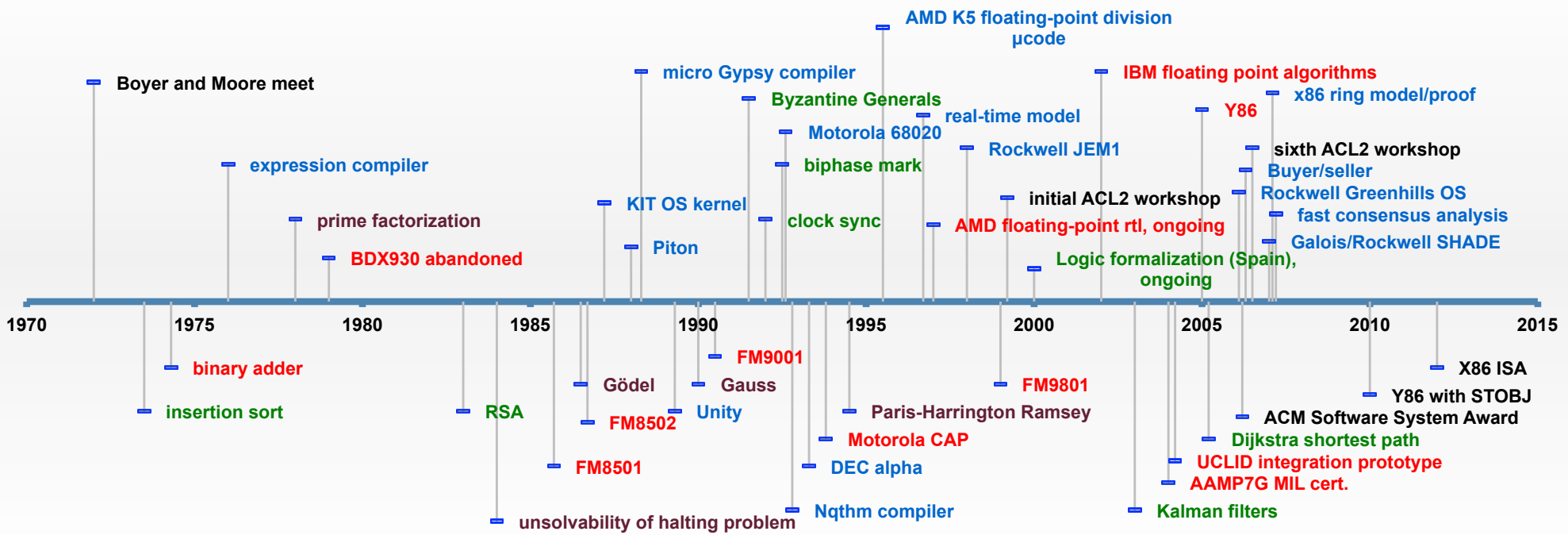
THE ACL2 SYSTEM

- Freely available, including libraries of *certifiable books*
- Let's explore the ACL2 home page.
- *Bleeding edge* for libraries (*community books*) and the ACL2 system are available from Github:
  `https://github.com/acl2/acl2`
- Workshop series: #13 is here at UT, Oct. 1-2, ACES 2.402:
  `http://www.cs.utexas.edu/users/moore/acl2/workshop-2015`
  - The ACL2 Workshop 2015 chairs anticipate some scholarships being available for student registration fees.
- History
  - Bob Boyer and J Moore started ACL2 in 1989. I joined and Bob dropped out in 1993. J and I continue its development.

THE ACL2 SYSTEM

- Freely available, including libraries of *certifiable books*
- Let's explore the ACL2 home page.
- *Bleeding edge* for libraries (*community books*) and the ACL2 system are available from Github:
  `https://github.com/acl2/acl2`
- Workshop series: #13 is here at UT, Oct. 1-2, ACES 2.402:
  `http://www.cs.utexas.edu/users/moore/acl2/workshop-2015`
  - The ACL2 Workshop 2015 chairs anticipate some scholarships being available for student registration fees.
- 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.

Partial timeline



Interactive Theorem Proving (ITP)

Interactive Theorem Proving (ITP)

- ITP is typically more scalable than automatic theorem proving, but requires some human assistance.

Interactive Theorem Proving (ITP)

- ITP is typically more scalable than automatic theorem proving, but requires some human assistance.
  - For large problems, such as encountered in industry, it's important to control the proof effort.

Interactive Theorem Proving (ITP)

- ITP is typically more scalable than automatic theorem proving, but requires some human assistance.
  - For large problems, such as encountered in industry, it's important to control the proof effort.
  - Many ITP systems, including ACL2, can send sub-problems to automatic proof tools, e.g., SAT solvers.

Interactive Theorem Proving (ITP)

- ITP is typically more scalable than automatic theorem proving, but requires some human assistance.
  - For large problems, such as encountered in industry, it's important to control the proof effort.
  - Many ITP systems, including ACL2, can send sub-problems to automatic proof tools, e.g., SAT solvers.
- The longest-standing well-known ITP systems in use today include ACL2, HOL4, Isabelle, Coq, and PVS. But there are many others.

Interactive Theorem Proving (ITP)

- ITP is typically more scalable than automatic theorem proving, but requires some human assistance.
  - For large problems, such as encountered in industry, it's important to control the proof effort.
  - Many ITP systems, including ACL2, can send sub-problems to automatic proof tools, e.g., SAT solvers.
- The longest-standing well-known ITP systems in use today include ACL2, HOL4, Isabelle, Coq, and PVS. But there are many others.
  - One famous use: Coq, to verify proof of the four-color theorem.

Interactive Theorem Proving (ITP)

- ITP is typically more scalable than automatic theorem proving, but requires some human assistance.
  - For large problems, such as encountered in industry, it's important to control the proof effort.
  - Many ITP systems, including ACL2, can send sub-problems to automatic proof tools, e.g., SAT solvers.
- The longest-standing well-known ITP systems in use today include ACL2, HOL4, Isabelle, Coq, and PVS. But there are many others.
  - One famous use: Coq, to verify proof of the four-color theorem.
- Yearly ITP conference (formerly TPHOLs)

ITP (PAGE 2)

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.*

ITP (PAGE 2)

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.*

ACL2 has a long history of automating deductions.

ITP (PAGE 2)

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.*

ACL2 has a long history of automating deductions.

Other ITP systems also automate reasoning, to various degrees.

Formal Verification

> *In the context of hardware and software systems, formal verification is the act of proving or disproving the correctness of intended algorithms underlying a system with respect to a certain formal specification or property, using formal methods of mathematics.*

– Quoting Wikipedia
[Sanghavi, Alok (21 May 2010). “What is formal verification?”. EE Times_Asia.]

Formal Verification

> *In the context of hardware and software systems, formal verification is the act of proving or disproving the correctness of intended algorithms underlying a system with respect to a certain formal specification or property, using formal methods of mathematics.*

– Quoting Wikipedia
[Sanghavi, Alok (21 May 2010). "What is formal verification?". EE Times_Asia.]

Formal tools include:

- equivalence checkers
- model checkers
- theorem provers (including ACL2)
- SAT solvers and SMT solvers
- static analysis tools (e.g. COMPASS, Blast, Slam)
- ...

Formal Verification: Growing its use

"Top 10" list from my talk, *My Top Ten Things to do for more Empirically Successful Computerized Reasoning*, ESCoR Workshop, FLoC, Seattle, Aug 21, 2006.

Formal Verification: Growing its use

"Top 10" list from my talk, *My Top Ten Things to do for more Empirically Successful Computerized Reasoning*, ESCoR Workshop, FLoC, Seattle, Aug 21, 2006.

10. Automation

Formal Verification: Growing its use

"Top 10" list from my talk, *My Top Ten Things to do for more Empirically Successful Computerized Reasoning*, ESCoR Workshop, FLoC, Seattle, Aug 21, 2006.

10. Automation
9. Apply to problems that people care about

Formal Verification: Growing its use

"Top 10" list from my talk, *My Top Ten Things to do for more Empirically Successful Computerized Reasoning*, ESCoR Workshop, FLoC, Seattle, Aug 21, 2006.

10. Automation
9. Apply to problems that people care about
8. Soundness

Formal Verification: Growing its use

"Top 10" list from my talk, *My Top Ten Things to do for more Empirically Successful Computerized Reasoning*, ESCoR Workshop, FLoC, Seattle, Aug 21, 2006.

10. Automation
9. Apply to problems that people care about
8. Soundness
7. Support for being a friendly "proof companion"

Formal Verification: Growing its use

"Top 10" list from my talk, *My Top Ten Things to do for more Empirically Successful Computerized Reasoning*, ESCoR Workshop, FLoC, Seattle, Aug 21, 2006.

10. Automation
9. Apply to problems that people care about
8. Soundness
7. Support for being a friendly "proof companion"
6. Get on Oprah

Formal Verification: Growing its use

"Top 10" list from my talk, *My Top Ten Things to do for more Empirically Successful Computerized Reasoning*, ESCoR Workshop, FLoC, Seattle, Aug 21, 2006.

10. Automation
9. Apply to problems that people care about
8. Soundness
7. Support for being a friendly "proof companion"
6. Get on Oprah
5. Education

Formal Verification: Growing its use

"Top 10" list from my talk, *My Top Ten Things to do for more Empirically Successful Computerized Reasoning*, ESCoR Workshop, FLoC, Seattle, Aug 21, 2006.

10. Automation
9. Apply to problems that people care about
8. Soundness
7. Support for being a friendly "proof companion"
6. Get on Oprah
5. Education
4. Tools to communicate with designers in their own language

Formal Verification: Growing its use

"Top 10" list from my talk, *My Top Ten Things to do for more Empirically Successful Computerized Reasoning*, ESCoR Workshop, FLoC, Seattle, Aug 21, 2006.

10. Automation
9. Apply to problems that people care about
8. Soundness
7. Support for being a friendly "proof companion"
6. Get on Oprah
5. Education
4. Tools to communicate with designers in their own language
3. Scalability

Formal Verification: Growing its use

"Top 10" list from my talk, *My Top Ten Things to do for more Empirically Successful Computerized Reasoning*, ESCoR Workshop, FLoC, Seattle, Aug 21, 2006.

10. Automation
9. Apply to problems that people care about
8. Soundness
7. Support for being a friendly "proof companion"
6. Get on Oprah
5. Education
4. Tools to communicate with designers in their own language
3. Scalability
2. Find bugs (but only actual bugs – soundness!): gets attention

Formal Verification: Growing its use

"Top 10" list from my talk, *My Top Ten Things to do for more Empirically Successful Computerized Reasoning*, ESCoR Workshop, FLoC, Seattle, Aug 21, 2006.

10. Automation
9. Apply to problems that people care about
8. Soundness
7. Support for being a friendly "proof companion"
6. Get on Oprah
5. Education
4. Tools to communicate with designers in their own language
3. Scalability
2. Find bugs (but only actual bugs – soundness!): gets attention
1. Connections/infiltration, including management positions [i.e., social network]

Formal Verification with ACL2

ACL2 is used **in industry** at

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

Formal Verification with ACL2

ACL2 is used **in industry** at

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

as well as the **U.S. Government** and **universities**,

including UT:

x86 modeling project, with x86 interpreter defined in ACL2.

Formal Verification: ACL2 modeling

Typical ACL2-based approaches to software and hardware verification:

Formal Verification: ACL2 modeling

Typical ACL2-based approaches to software and hardware verification:

- Using a *translator*: Map programs to ACL2 functions.

Formal Verification: ACL2 modeling

Typical ACL2-based approaches to software and hardware verification:

- Using a *translator*: Map programs to ACL2 functions.
  - We did this at AMD for rtl verification.
  - Sometimes called a *shallow embedding*.

Formal Verification: ACL2 modeling

Typical ACL2-based approaches to software and hardware verification:

- Using a *translator*: Map programs to ACL2 functions.
  - We did this at AMD for rtl verification.
  - Sometimes called a *shallow embedding*.
- Using an *interpreter*:

Formal Verification: ACL2 modeling

Typical ACL2-based approaches to software and hardware verification:

- Using a *translator*: Map programs to ACL2 functions.
  - We did this at AMD for rtl verification.
  - Sometimes called a *shallow embedding*.
- Using an *interpreter*:
  - Has been done for many years.
  - Currently used for rtl verification at Centaur.
  - Sometimes called a *deep embedding*.

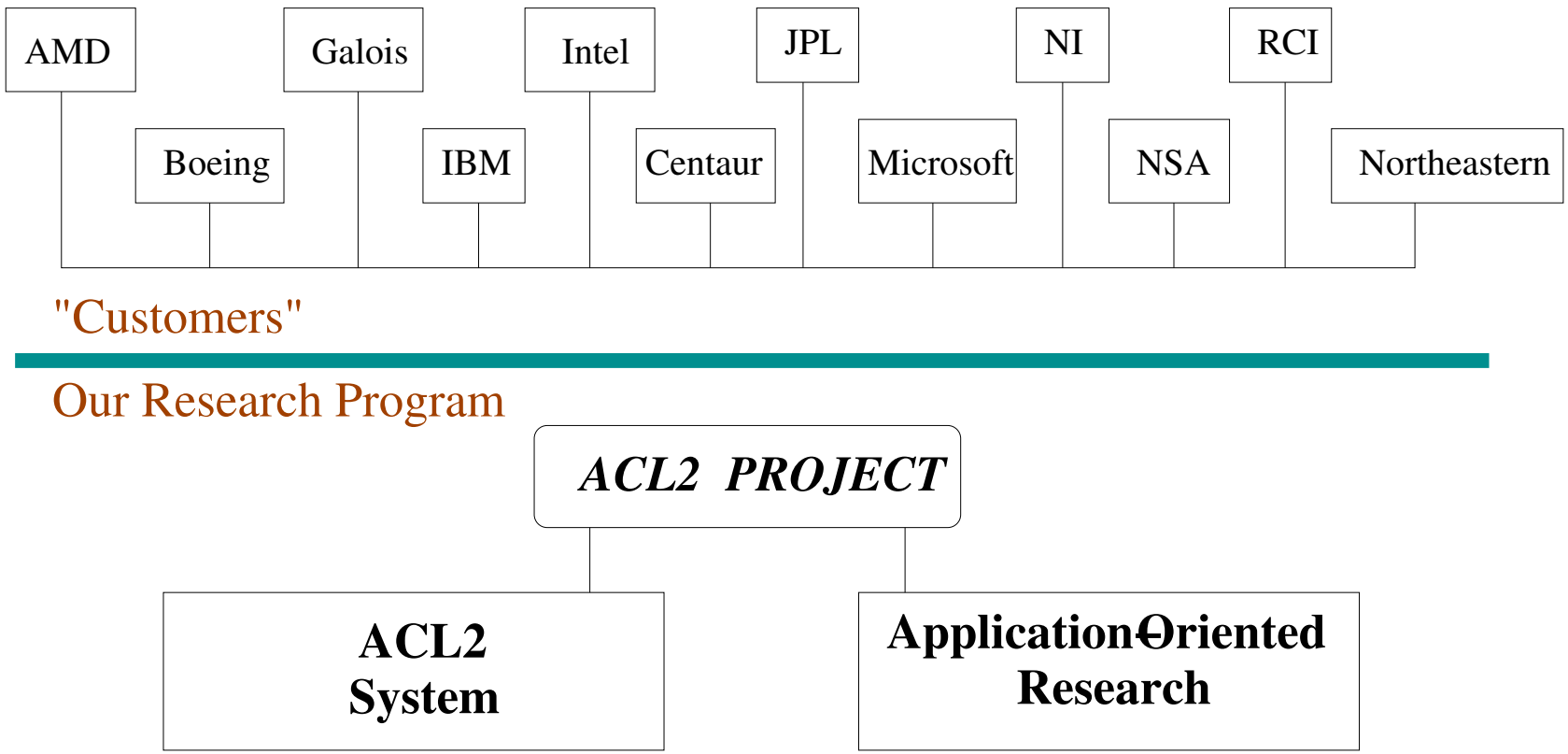
Formal Verification: ACL2 modeling

Typical ACL2-based approaches to software and hardware verification:

- Using a *translator*: Map programs to ACL2 functions.
  - We did this at AMD for rtl verification.
  - Sometimes called a *shallow embedding*.
- Using an *interpreter*:
  - Has been done for many years.
  - Currently used for rtl verification at Centaur.
  - Sometimes called a *deep embedding*.

```
(defun run (st n)
  (if (zp n)              ; n is 0
      st
    (run (run1 st)        ; run one instruction
         (- n 1))))
```

THE ACL2 "ECOSYSTEM"



Outline

Introduction

Context

Prover Automation and Control

ACL2 Variants

Foundations

Implementation

Conclusion

Outline

Introduction

Context

- UT Mechanized Reasoning Group
- The ACL2 system
- Interactive theorem proving (ITP)
- Formal verification

Prover Automation and Control

- Simple demo of typical use: sum to n
- Prover automation
- Prover control

ACL2 Variants

Foundations

Implementation

Conclusion

Prover Automation and Control

How does the prover operate, and how does one operate the prover?

SIMPLE DEMO OF TYPICAL USE: SUM TO N

[DEMO]: file `demo-2.lsp`
(log `demo-2-log.txt`)

- Illustrates recursive definition, automated proof, rewriting
- Note that prover operation is controlled by proving theorems, which are typically stored as rules (to be applied automatically).
- The basic interaction model is "The Method":
  write functions, prove lemmas, react to unproved subgoals by proving rewrite rules.

Prover automation

- Most important:
  - **simplification** (especially, using **rewriting**, but also linear arithmetic, boolean reasoning, . . .)
  - **induction**

Prover automation

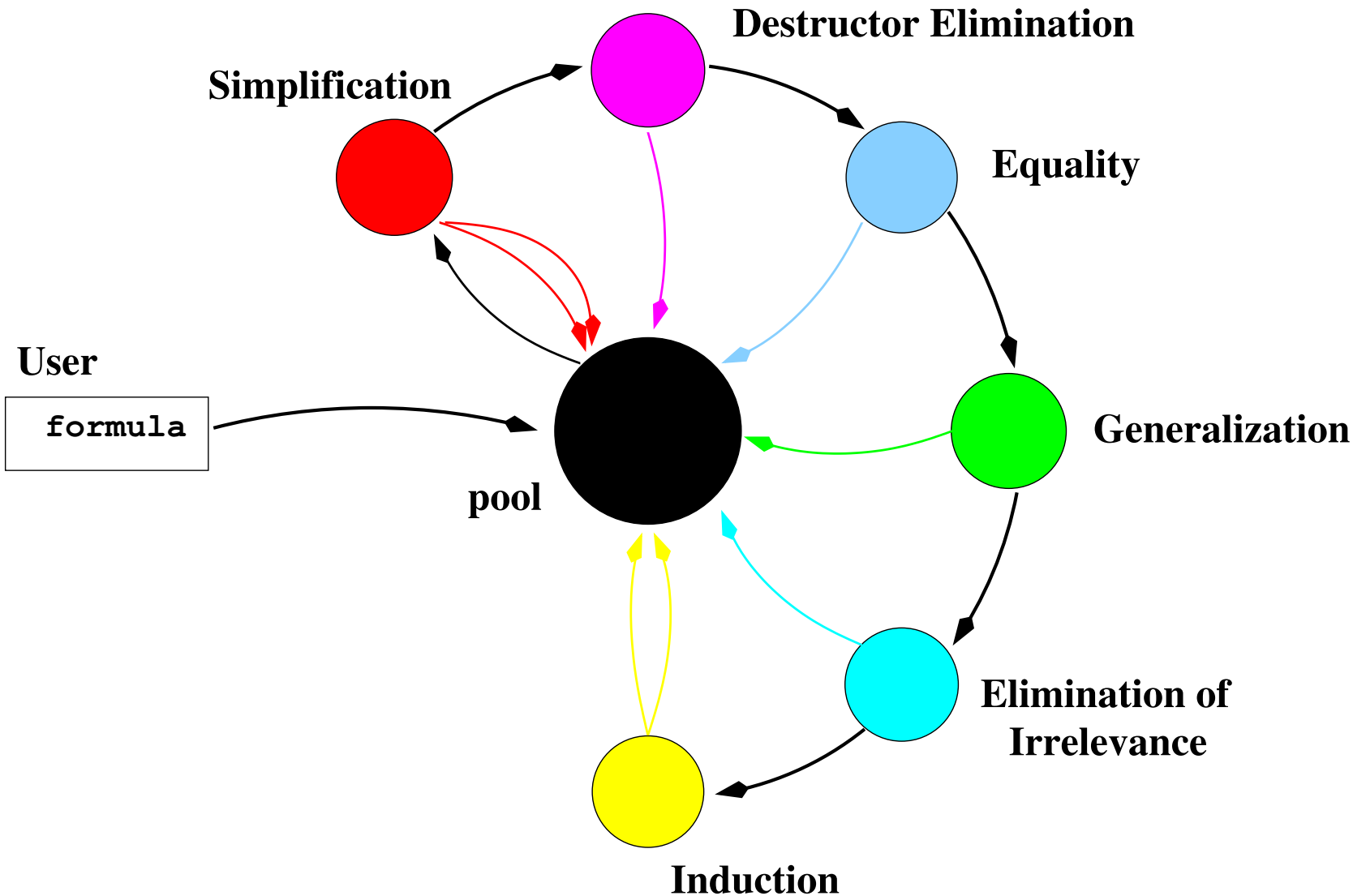
- Most important:
  - **simplification** (especially, using **rewriting**, but also linear arithmetic, boolean reasoning, . . .)
  - **induction**
- Other processes: destructor elimination, heuristic use of equalities, generalization, and elimination of irrelevance.

Prover automation

- Most important:
  - **simplification** (especially, using **rewriting**, but also linear arithmetic, boolean reasoning, . . .)
  - **induction**
- Other processes: destructor elimination, heuristic use of equalities, generalization, and elimination of irrelevance.
  [DEMO]: file `rev-rev-1.lsp`
  (log `rev-rev-1-log.txt`)

For more on rewriting, see the documentation:
ACL2
ACL2-tutorial
Introduction-to-the-theorem-prover
introduction-to-rewrite-rules-part-1

THE ACL2 WATERFALL



Prover control

- Hints

Prover control

- Hints
- Rules, especially rewrite rules (about a dozen and a half kinds of rules)

PROVER CONTROL

- Hints
- Rules, especially rewrite rules (about a dozen and a half kinds of rules)
  [DEMO]: file `rev-rev-2.lsp`
  (log `rev-rev-2-log.txt`)

PROVER CONTROL (CONT.)

- Many more ways to control the prover: Meta reasoning, macros, rule-classes, . . .

Prover control (cont.)

- Many more ways to control the prover: Meta reasoning, macros, rule-classes, ...
- Documentation helps, e.g.:
  - THE-METHOD
  - INTRODUCTION-TO-THE-THEOREM-PROVER
  - DEBUGGING

Prover control (cont.)

- Many more ways to control the prover: Meta reasoning, macros, rule-classes, . . .
- Documentation helps, e.g.:
  - THE-METHOD
  - INTRODUCTION-TO-THE-THEOREM-PROVER
  - DEBUGGING
- Mailing lists available from the ACL2 home page include **acl2-help**.

Prover control (cont.)

- Many more ways to control the prover: Meta reasoning, macros, rule-classes, ...
- Documentation helps, e.g.:
  - THE-METHOD
  - INTRODUCTION-TO-THE-THEOREM-PROVER
  - DEBUGGING
- Mailing lists available from the ACL2 home page include **acl2-help**.
- [DEMO]: file `rotate.lsp`
  (log `rotate-log.txt`)

PROVER CONTROL (CONT.)

- Many more ways to control the prover: Meta reasoning, macros, rule-classes, . . .
- Documentation helps, e.g.:
  - THE-METHOD
  - INTRODUCTION-TO-THE-THEOREM-PROVER
  - DEBUGGING
- Mailing lists available from the ACL2 home page include **acl2-help**.
- [DEMO]: file `rotate.lsp`
  (log `rotate-log.txt`)

  (for another proof, see `rotate-alt.lsp`)

Outline

Introduction

Context

Prover Automation and Control

ACL2 Variants

Foundations

Implementation

Conclusion

Outline

Introduction

Context

- UT Mechanized Reasoning Group
- The ACL2 system
- Interactive theorem proving (ITP)
- Formal verification

Prover Automation and Control

- Simple demo of typical use: sum to n
- Prover automation
- Prover control

ACL2 Variants

Foundations

Implementation

Conclusion

ACL2 Variants

- ACL2(r): support for real numbers (Ruben Gamboa)

ACL2 Variants

- ACL2(r): support for real numbers (Ruben Gamboa)
- ACL2(p): support for parallel evaluation and reasoning (David Rager)

ACL2 Variants

- ACL2(r): support for real numbers (Ruben Gamboa)
- ACL2(p): support for parallel evaluation and reasoning (David Rager)
- ACL2(h): hash cons, function memoization, and applicative hash tables (Bob Boyer, Jared Davis, Warren Hunt, and Sol Swords)
  - Now part of ACL2

ACL2 Variants

- ACL2(r): support for real numbers (Ruben Gamboa)
- ACL2(p): support for parallel evaluation and reasoning (David Rager)
- ACL2(h): hash cons, function memoization, and applicative hash tables (Bob Boyer, Jared Davis, Warren Hunt, and Sol Swords)
  - Now part of ACL2
  - The following demo shows that ACL2 executes efficiently, but can be yet much faster when using function memoization.
    [DEMO]: file `fibonacci.lsp`
    (log `fibonacci-log.txt`)

Outline

Introduction

Context

Prover Automation and Control

ACL2 Variants

Foundations

Implementation

Conclusion

Outline

Introduction

Context
- UT Mechanized Reasoning Group
- The ACL2 system
- Interactive theorem proving (ITP)
- Formal verification

Prover Automation and Control
- Simple demo of typical use: sum to n
- Prover automation
- Prover control

ACL2 Variants

Foundations

Implementation

Conclusion

FOUNDATIONS

- The ACL2 logic is first-order logic with induction (actually epsilon-0 induction; see ORDINALS)

FOUNDATIONS

- The ACL2 logic is first-order logic with induction (actually epsilon-0 induction; see ORDINALS)
- Evolving theories: conservative extensions

FOUNDATIONS

- The ACL2 logic is first-order logic with induction (actually epsilon-0 induction; see ORDINALS)
- 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

- The ACL2 logic is first-order logic with induction (actually epsilon-0 induction; see ORDINALS)
- 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.
    - Extensions by definition are conservative

FOUNDATIONS

- The ACL2 logic is first-order logic with induction (actually epsilon-0 induction; see ORDINALS)
- 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.
    - Extensions by definition are conservative
      – even by recursive definition, when termination is provable

FOUNDATIONS

- The ACL2 logic is first-order logic with induction (actually epsilon-0 induction; see ORDINALS)
- 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.
    - Extensions by definition are conservative
      – even by recursive definition, when termination is provable
  - Importance: need to introduce new concepts to do program verification, but must be done conservatively in order to believe the results

FOUNDATIONS

- The ACL2 logic is first-order logic with induction (actually epsilon-0 induction; see ORDINALS)
- 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.
    - Extensions by definition are conservative
      – even by recursive definition, when termination is provable
  - Importance: need to introduce new concepts to do program verification, but must be done conservatively in order to believe the results
- [DEMO]: *books* `rotate.lisp` and `rotate-proof.lisp` (log rotate-certification-log.txt)

FOUNDATIONS

- The ACL2 logic is first-order logic with induction (actually epsilon-0 induction; see ORDINALS)
- 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.
    - Extensions by definition are conservative
      – even by recursive definition, when termination is provable
  - Importance: need to introduce new concepts to do program verification, but must be done conservatively in order to believe the results
- [DEMO]: *books* `rotate.lisp` and `rotate-proof.lisp` (log rotate-certification-log.txt)
- Correctness of LOCAL and ENCAPSULATE:
  M. Kaufmann and J Moore, "Structured Theory Development for a Mechanized Logic." *Journal of Automated Reasoning* 26, no. 2 (2001) 161-203.

Outline

Introduction

Context

Prover Automation and Control

ACL2 Variants

Foundations

Implementation

Conclusion

Outline

Introduction

Context
- UT Mechanized Reasoning Group
- The ACL2 system
- Interactive theorem proving (ITP)
- Formal verification

Prover Automation and Control
- Simple demo of typical use: sum to n
- Prover automation
- Prover control

ACL2 Variants

Foundations

Implementation

Conclusion

Implementation

ACL2 is written mostly in itself (!).

IMPLEMENTATION

ACL2 is written mostly in itself (!).

Example, time permitting: we'll look at the code for a substitution function, `sublis-var`.

Outline

Introduction

Context

Prover Automation and Control

ACL2 Variants

Foundations

Implementation

Conclusion

Outline

- Introduction
- Context
  - UT Mechanized Reasoning Group
  - The ACL2 system
  - Interactive theorem proving (ITP)
  - Formal verification
- Prover Automation and Control
  - Simple demo of typical use: sum to n
  - Prover automation
  - Prover control
- ACL2 Variants
- Foundations
- Implementation
- Conclusion

Conclusion

- ACL2 has a long history and is now being used in industry.

Conclusion

- ACL2 has a long history and is now being used in industry.
- As an ITP system, it relies on user guidance for large problems but enjoys scalability.

Conclusion

- ACL2 has a long history and is now being used in industry.
- As an ITP system, it relies on user guidance for large problems but enjoys scalability.
- For more information:

Conclusion

- ACL2 has a long history and is now being used in industry.
- As an ITP system, it relies on user guidance for large problems but enjoys scalability.
- 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 long history and is now being used in industry.
- As an ITP system, it relies on user guidance for large problems but enjoys scalability.
- For more information:
  - See the ACL2 home page, in particular links to The Tours and publications, which links to introductory material.
  - Come to the the ACL2 seminar

Conclusion

- ACL2 has a long history and is now being used in industry.
- As an ITP system, it relies on user guidance for large problems but enjoys scalability.
- For more information:
  - See the ACL2 home page, in particular links to The Tours and publications, which links to introductory material.
  - Come to the the ACL2 seminar
- **See us about research opportunities**:
  Marijn Heule, GDC 7.714, marijn@cs.utexas.edu
  Warren Hunt, GDC 7.818, hunt@cs.utexas.edu
  Matt Kaufmann, GDC 7.804, kaufmann@cs.utexas.edu

Conclusion

- ACL2 has a long history and is now being used in industry.
- As an ITP system, it relies on user guidance for large problems but enjoys scalability.
- For more information:
  - See the ACL2 home page, in particular links to The Tours and publications, which links to introductory material.
  - Come to the the ACL2 seminar
- **See us about research opportunities**:
  Marijn Heule, GDC 7.714, marijn@cs.utexas.edu
  Warren Hunt, GDC 7.818, hunt@cs.utexas.edu
  Matt Kaufmann, GDC 7.804, kaufmann@cs.utexas.edu

Bill Gates again, this time at the dedication of our building, the Gates Dell Complex: 1 minute 33 seconds on how the greatest challenge for CS in the years ahead is "verifying correctness":
`https://www.youtube.com/watch?v=UOPWydeC6a0#t=2219`