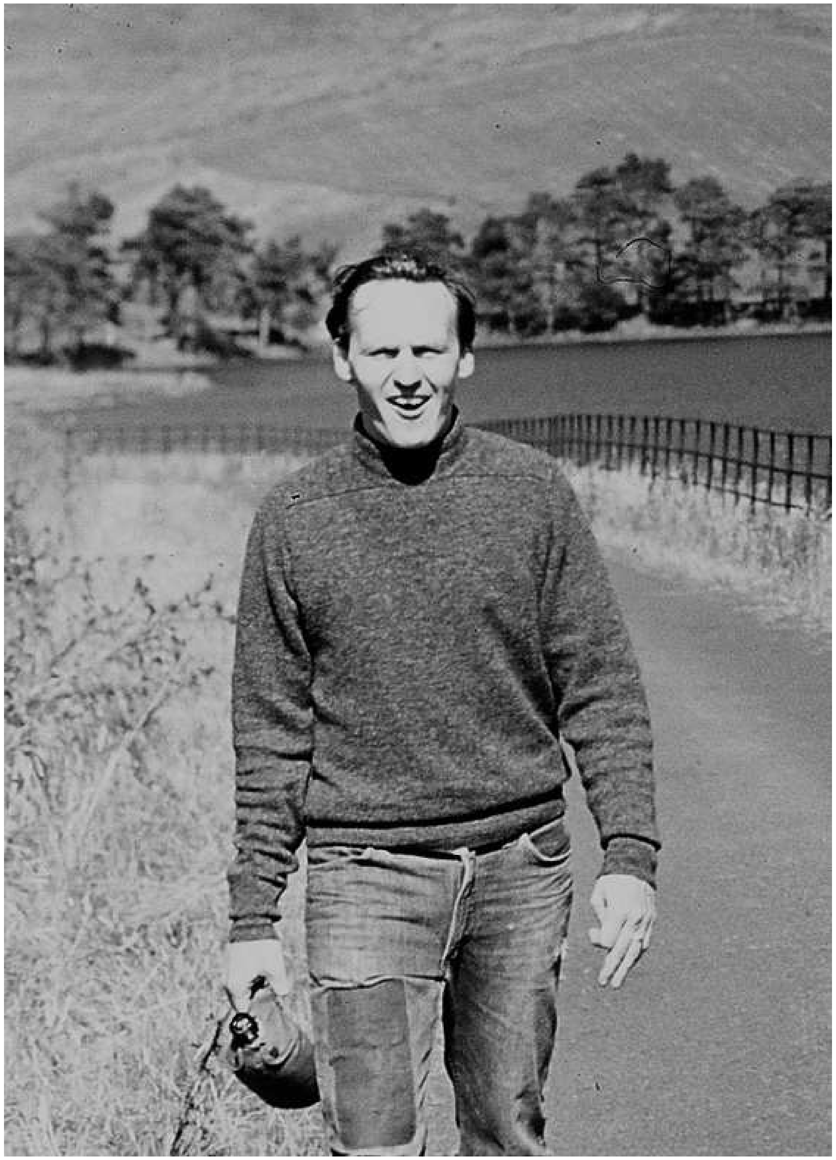
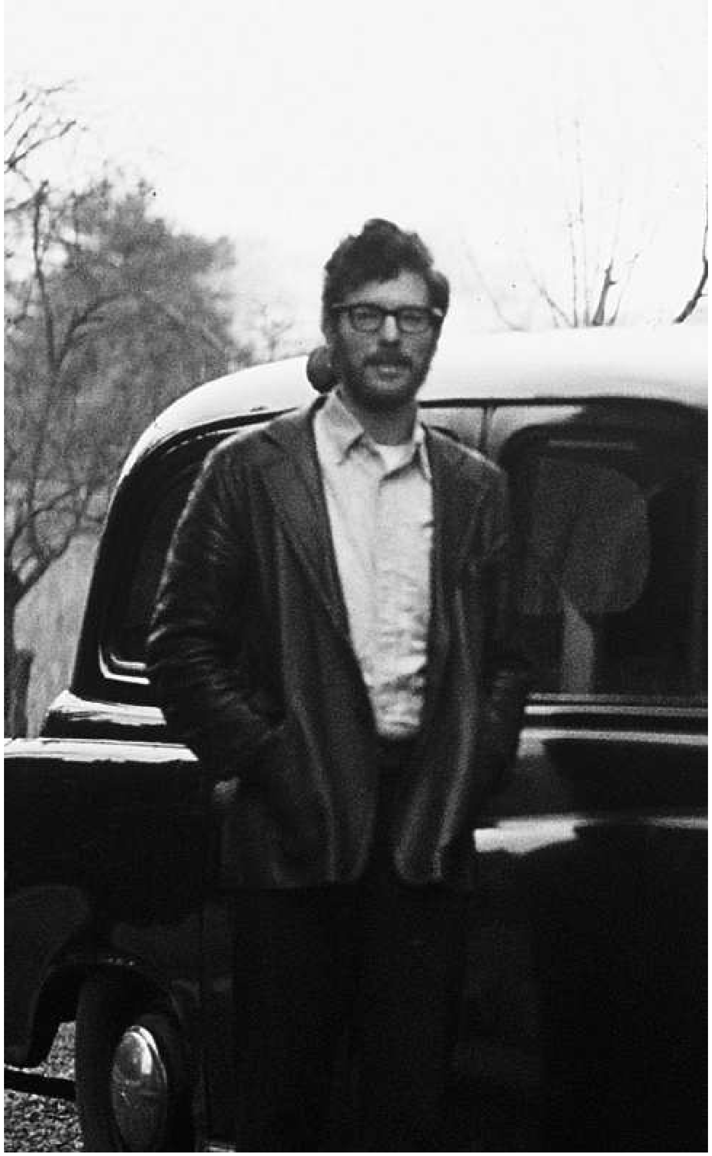
Lessons Learned over 45 Years in Theorem Proving

J Strother Moore
Department of Computer Science
University of Texas at Austin

Hope Park Square, Edinburgh, 1971...

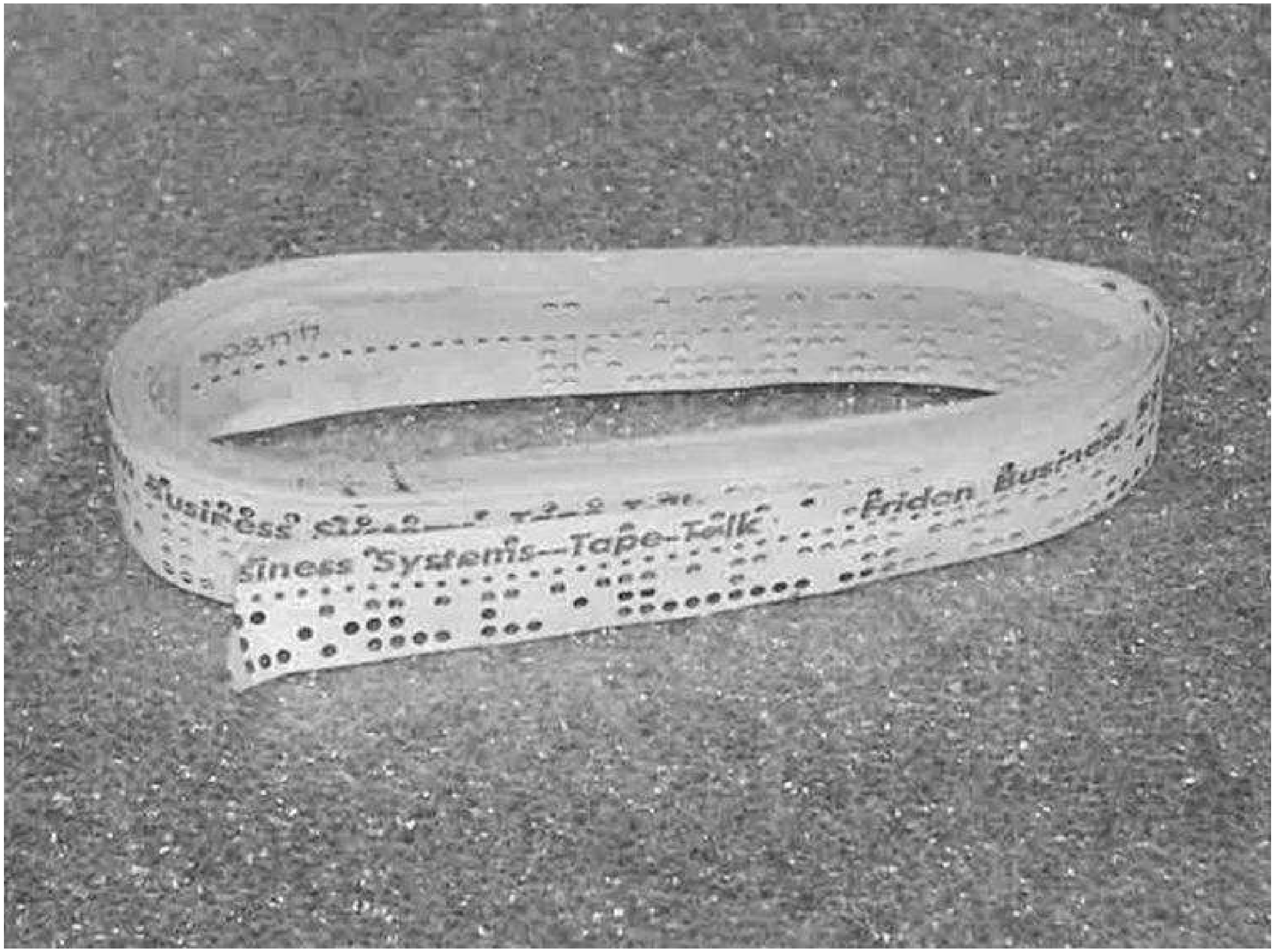




ICL 4130

64KB of RAM, paper tape input

Pop-2 CONS time: 400 microseconds


siness Systems--Tape-Talk
Friden Business

Lessons

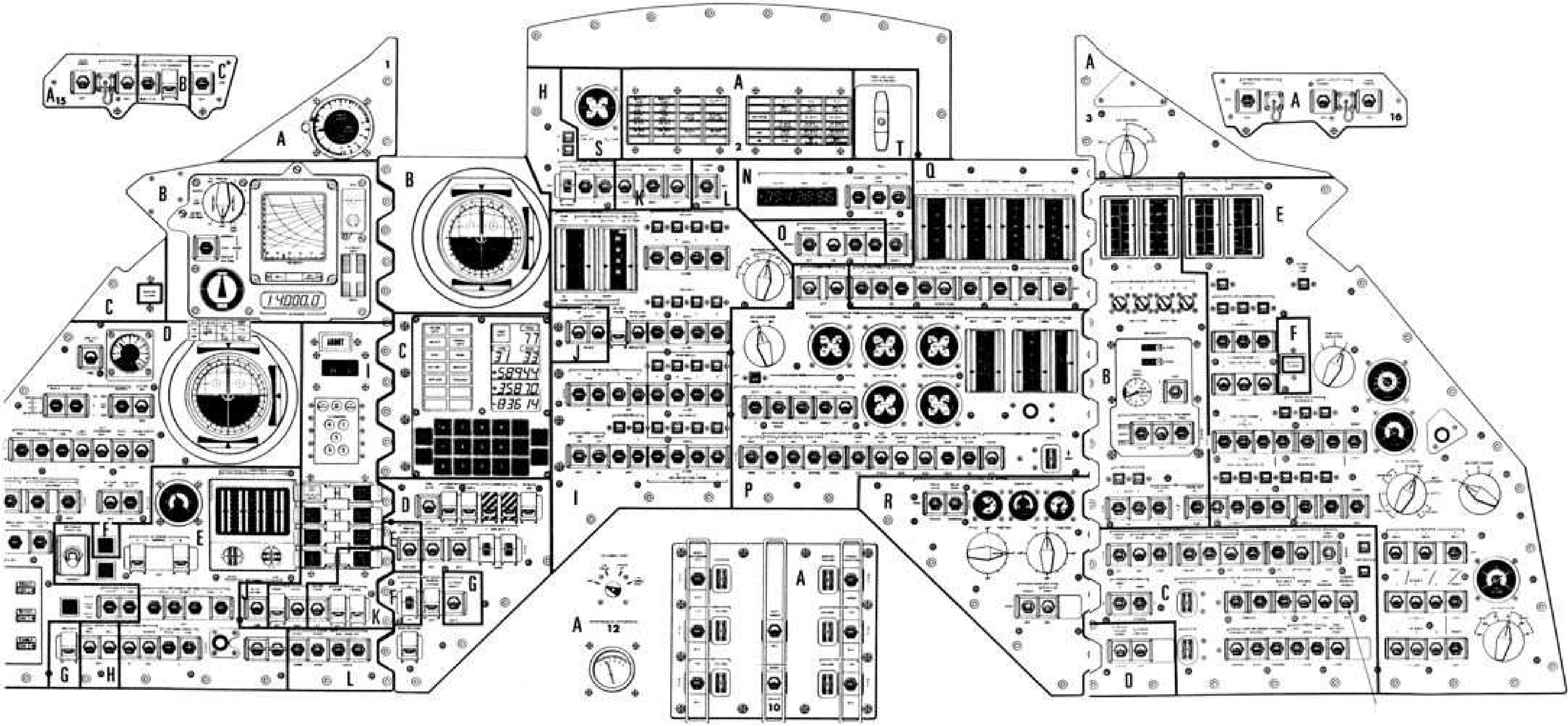
- any project worth doing can be done by two people
- soundness is paramount
- keep a regression suite and test new heuristics against it
- beware of bound variables
- being outside the mainstream is ok

- being outside the mainstream is ok
  - not predicate calculus
  - not uniform proof procedure
  - not complete
  - not typed lambda calculus
  - not Fortran or COBOL
  - not inductive assertions and vcg

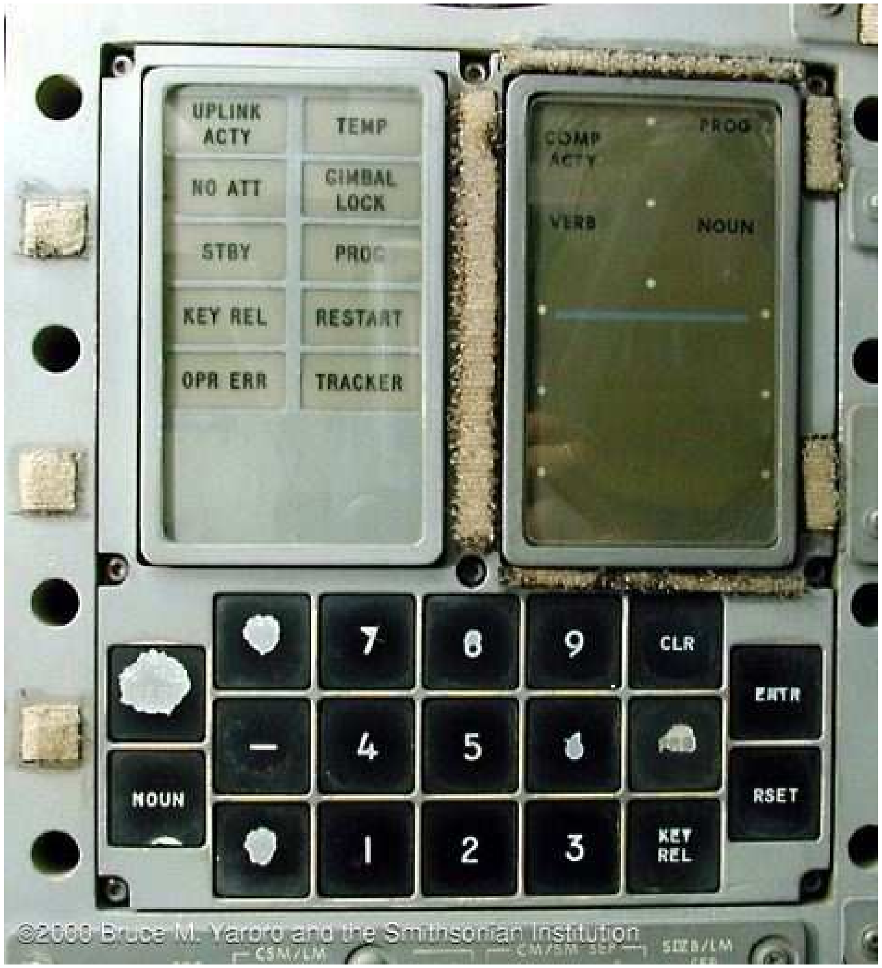
- Heroes are worth having:
  - John McCarthy - Lisp as a programming and spec language
  - Woody Bledsoe - heuristic theorem prover

- “If you can build a better prover, do it.”
- self-worth is independent of what other people think

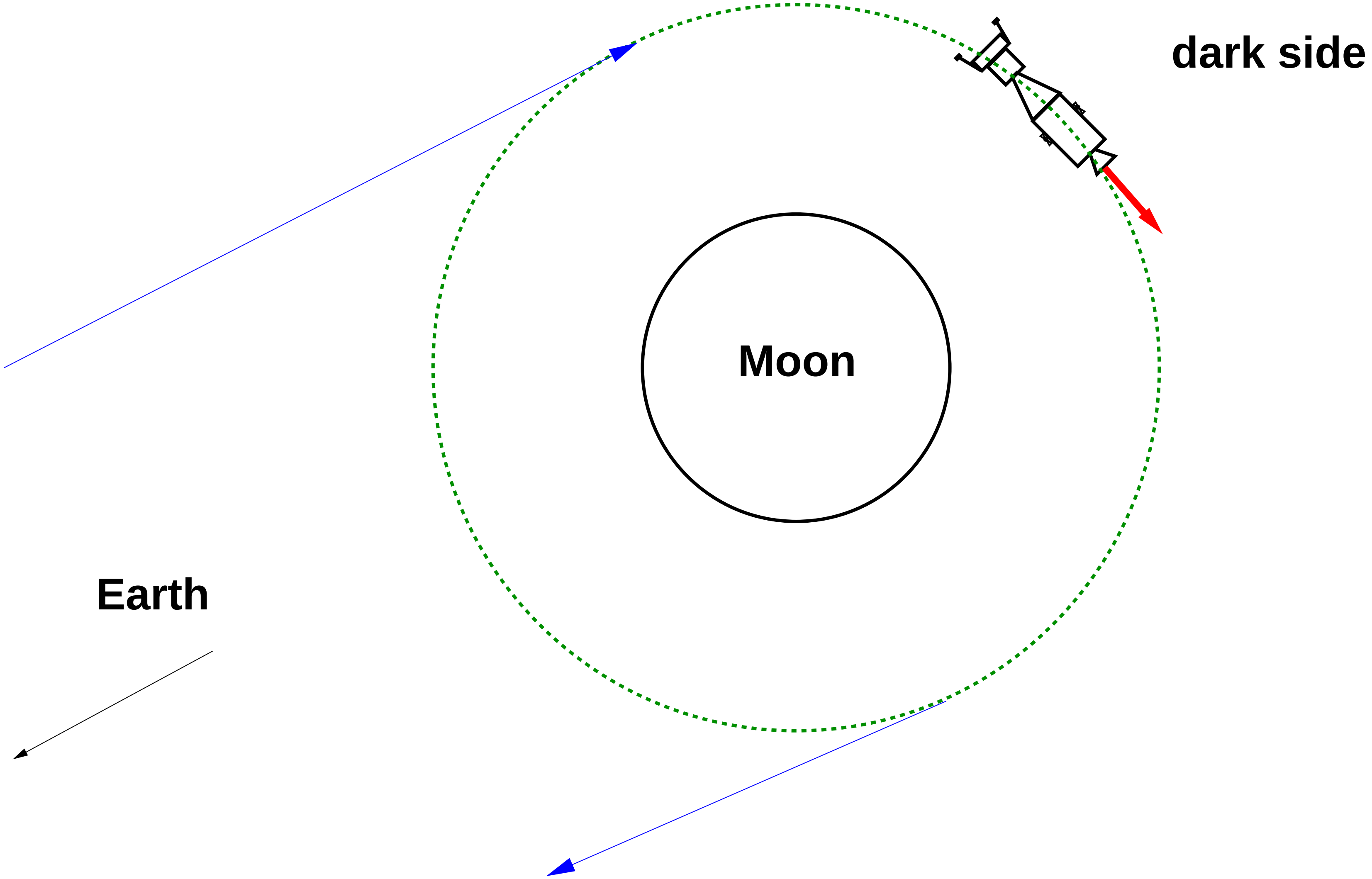
My Summer Job, 1968



APOLLO COMMAND MODULE MAIN CONTROL PANEL



©2008 Bruce M. Yarbro and the Smithsonian Institution


dark side
Moon
Earth

We put men into orbit around the Moon in 1968 and landed men on the Moon in 1969.

And the whole program was canceled less than 5 years later because the American public had lost interest in spaceflight.

One of the most talented engineering teams ever assembled was disbanded and scattered.

But it doesn’t reflect on the importance of their achievement or their talent.

Why Is ACL2 Successful?

Reason 1: Our mathematical logic is an executable programming language.

- Many very efficient heavy-duty implementations
- Supported on many platforms

- Many independently provided programming/system development tools and environments.

Imagine that in 1971 Boyer and I had chosen *any other programming language of the time*.

Reason 2: Our community has invested 45 years

- supporting efficient execution *and* proof (so models are dual-purpose)
- integrating a wide variety of proof techniques (so proofs are more automatic)
- engineering for industrial scale formulas
- documenting the system

- developing reusable books
- interfacing to other tools (e.g., IBM Sixth Sense, ABC, SAT, MC) (so embedded theorem proving can glue disparate fragments together), and
- supporting verification tool building (so users can build, verify, and then efficiently execute special-purpose tools)

Reason 3: The theorem prover is semi-automatic:

- finds many straightforward proofs fully automatically, including inductive proofs
- is primarily guided by rules expressed as theorems (facilitating *proof maintenance*)
- admits interactive proof checking – but discourages it except for exploring proof strategies

At its best, ACL2 puts the human in the loop where the human is most effective.

Reason 4: We have an integrated environment in which users can

- prototype models
- execute programs
- prove theorems
- develop useful libraries
- develop other verification and analysis tools

Reason 5: We have chosen the right problems. In our applications, the models

- are bit- and cycle-accurate, not “toys”,
- are useful as pre-fab simulation engines, and
- permit mathematical abstraction supported by proof.

Reason 6: Our user community is *very* talented.

Reason 6: Our user community is *very* talented.

> “The reason the Boyer-Moore theorem is so ‘good’ is that only smart people use it!” – *anonymous critic, early 1980s*

Reason 7: Industry has no other alternative than to use mechanized reasoning; their artifacts are too complicated to analyze accurately any other way.

But ARE We Succeeding?

Our community is *very* small.

Is Lisp a help or a hindrance?

Is “first-order” a help or a hindrance?

And does the size of the community matter much if ours is still the best tool for doing what we do?

But ARE We Succeeding? (con't)

Verification of useful software is *still* too hard!

ACL2 is not automatic enough!

At its worst, the burden on the user is too heavy!

Future Work

Here are some of the scientific challenges to building a better ACL2:

- create better books, especially for machine arithmetic
- integrate FSM and SMT decision procedures
- exploit the parallelizability of function programs

- do more (non-combinatoric) search
- encorporate heuristics for discovering invariants
- provide counterexamples
- exploit examples to guide search
- support interactive steering and visualization

The Real Challenge

Why are our imaginations so limited?

Most of these ideas are *minor* compared to the problem of truly automating software verification.

The Dream

I work with ACL2 all day, trying various approaches to a problem.

I quit for the day.

During my downtime, ACL2 analyzes everything it can get its hands on and the next morning it greets me with “I proved that[1], so now what?”

[1] Not by a brute force all night run, but by decomposing it properly – something it should also be able to explain to me.

Is ACL2 the right platform to experiment with?

Does the weight of industrial use and a legacy regression suite preclude radical experimentation?

E.g., what would happen if we just threw ACL2 away and worked on a toy prover that learns?

Who Pays?

What is the funding model for improvements to ACL2?

Is work on ACL2

- *a hobby of Matt's and J's*,
- *research*,
- *development*, or
- *maintenance*?

Sustainability

The 45 year "Boyer-Moore Project" was built primarily on the *passion* and *dedication* of a few individuals who just kept going.

But does ACL2's success discourage the development better provers?

Do we want ACL2 to outlive us?