Linking ACL2 and HOL: past achievements and future prospects

ABSTRACT. Over the years there have been several attempts to obtain the amazing automation and efficiency of ACL2 theorem proving within various versions of the HOL proof assistant. These include building a Boyer-Moore waterfall as a tactic, dynamically linking to ACL2 using the PROSPER Plug-In Interface and, most recently, embedding the ACL2 logic in the HOL logic and then using ACL2 as an trusted oracle. These activities have differed in goals and methods, e.g. placing different emphases on usability, efficiency and logical coherence. The talk will start with a critical historical overview, and will end with thoughts on possible future projects.

What's coming

- History overview
- Critical discussion
- What next (includes cool ideas from Matt)

History overview (partial)

- 1992: Richard Boulton “Boyer-Moore Automation for HOL”
  - implementation of waterfall from “A Computational Logic”
  - just a prototype ‘proof of concept’ experiment
- 1999: Mark Staples `ACL2PII`
  - linked HOL and ACL2 via the PROSPER Plug-In Interface
  - connect interactive HOL and ACL2 sessions
- 2004-2011: Gordon, Hunt, Kaufmann, Reynolds
  - ACL2 logic defined in HOL
  - called “`ACL2-in-HOL`” here

Related projects

- 1993 UC Davis PM proof manager (Archer, Fink & Yang)
  - linked other theorem provers to HOL using Emacs
  - not considered further
- Edinburgh-Cambridge CLAM-HOL project
  - partly suggested by Boulton’s work ... then hired him
  - Edinburgh proof planning influenced by Boyer-Moore ideas
- European Esprit PROSPER project
  - provided tools used in `ACL2PII` and CLAM-HOL
  - revealed challenges of prover-linking ‘middleware’

Boulton’s Boyer-Moore prover inside HOL

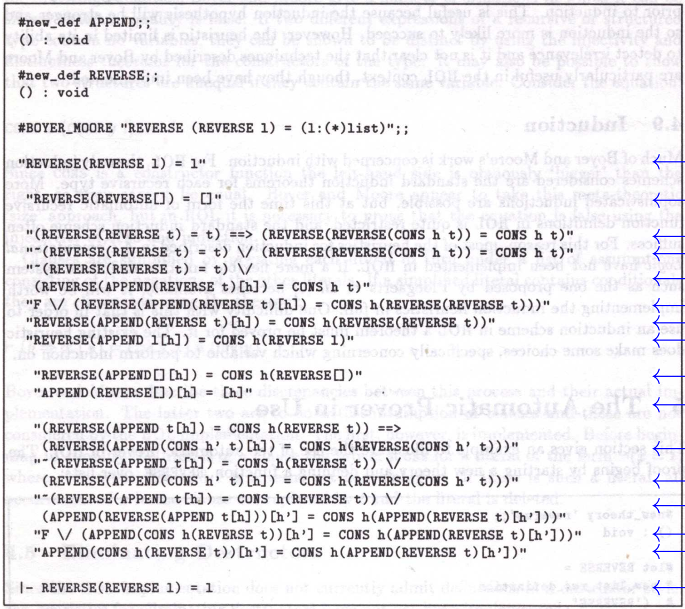
```
% ---------------------------------------------------------------------------- %
%                                                                              %
% DESCRIPTION: Boyer-Moore Automation for HOL                                  %
%                                                                              %
% AUTHORS: Richard J. Boulton                                                  %
%                                                                              %
% ADDRESS: University of Cambridge Computer Laboratory                         %
%          New Museums Site                                                    %
%          Pembroke Street                                                     %
%          Cambridge, CB2 3QG                                                  %
%          England                                                             %
%                                                                              %
%          email: rjb@cl.cam.ac.uk                                             %
%                                                                              %
% DATE: 92.10.16                                                               %
% ---------------------------------------------------------------------------- %

This directory contains an implementation for the HOL system (HOL88
Version 2.01) of Boyer and Moore's automatic proof heuristics. The
code is based on the description in `A Computational Logic' and so
does not reflect the advances made to the Boyer-Moore theorem prover
since 1979. There are many limitations, the most significant of which
is the assumption that recursive functions are defined in a
`constructor-style' according to the restrictions of the automatic
definition tools currently available in HOL. The code was written more
as an experiment than as a practical tool. However, it may be found to
be useful. Minimal documentation can be found below.

Richard Boulton, 16th October 1992.
```

http://
bazaar.launchpad.net/~ubuntu-branches/ubuntu/trusty/hol88/trusty/files/head:/contrib/boyer-moore/

Example: proving **|- REVERSE (REVERSE l) = l**



```
#new_def APPEND;;
() : void

#new_def REVERSE;;
() : void

#BOYER_MOORE "REVERSE (REVERSE l) = (l:(*)list)";;

"REVERSE(REVERSE l) = l"

 "REVERSE(REVERSE[]) = []"

 "(REVERSE(REVERSE t) = t) ==> (REVERSE(REVERSE(CONS h t)) = CONS h t)"
 "~(REVERSE(REVERSE t) = t) \/ (REVERSE(REVERSE(CONS h t)) = CONS h t)"
 "~(REVERSE(REVERSE t) = t) \/
  (REVERSE(APPEND(REVERSE t)[h]) = CONS h t)"
 "F \/ (REVERSE(APPEND(REVERSE t)[h]) = CONS h(REVERSE(REVERSE t)))"
 "REVERSE(APPEND(REVERSE t)[h]) = CONS h(REVERSE(REVERSE t))"
 "REVERSE(APPEND l[h]) = CONS h(REVERSE l)"

  "REVERSE(APPEND[][h]) = CONS h(REVERSE[])"
  "APPEND(REVERSE[])[h] = [h]"

  "(REVERSE(APPEND t[h]) = CONS h(REVERSE t)) ==>
   (REVERSE(APPEND(CONS h' t)[h]) = CONS h(REVERSE(CONS h' t)))"
  "~(REVERSE(APPEND t[h]) = CONS h(REVERSE t)) \/
   (REVERSE(APPEND(CONS h' t)[h]) = CONS h(REVERSE(CONS h' t)))"
  "~(REVERSE(APPEND t[h]) = CONS h(REVERSE t)) \/
   (APPEND(REVERSE(APPEND t[h]))[h'] = CONS h(APPEND(REVERSE t)[h']))"
  "F \/ (APPEND(CONS h(REVERSE t))[h'] = CONS h(APPEND(REVERSE t)[h']))"
  "APPEND(CONS h(REVERSE t))[h'] = CONS h(APPEND(REVERSE t)[h'])"

|- REVERSE(REVERSE l) = l
```

goal

base case

step case
clausal form
expand definitions

simplify
simplify
generalise "REVERSE t" to "l"

new induction: base case

expand definitions

new induction: step case

clausal form
expand definitions

simplify
simplify

expand definitions - theorem proved

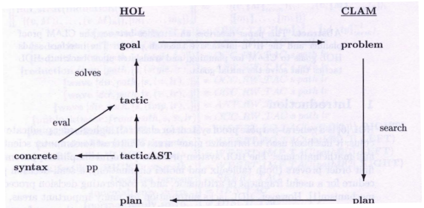
Comments on Boulton’s Boyer-Moore automation

- Not nearly as sophisticated as real Boyer-Moore provers
- An early “fully expansive” automated prover
  - Boulton invented the phrase “fully expansive”
    (he also invented “deep embedding“, “shallow embedding”
    and the first fully-expansive arithmetic decision procedure)
- Not much used — possible reasons:
  - simple inductive proofs not a problem for users
  - not good at remembering and using pre-proved lemmas
  - Boulton and research thread shifted to CLAM-HOL project

    The investigation described in this paper has shown that automation in the style of Boyer and Moore can be achieved within the HOL system. However, the question remains as to whether this is a good approach. An alternative is to interface HOL to another system such as the Boyer-Moore theorem prover or the proof planner CLAM.

CLAM-HOL Project

- Not directly related to ACL2, but relevant
  - it's what Richard Boulton did next
  - influenced Staples' `ACL2PII`
- CLAM produces 'proof plans' for Martin Löf type theory
- CLAM-HOL uses CLAM to plan HOL proofs



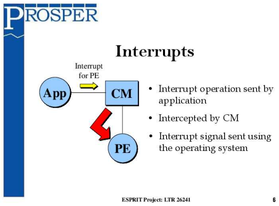
  - HOL goal converted to CLAM constructive logic problem
  - CLAM searches for a proof plan
  - found CLAM plan converted to a HOL tactic
  - tactic executed in HOL to solve goal
- Could an ACL2 proof trace be a proof plan?
  - maybe like Isabelle's Sledgehammer

Mark Staples' `ACL2PII`

- Transfer theorems from ACL2 to HOL in 'real-time'
- Ad-hoc translation: pattern matching + type 'guessing'
  - user creates HOL types to approximate ACL2 'types'
  - user defines heuristic translation rules
  - ACL2 $\texttt{NIL} \overset{?}{\rightsquigarrow} \texttt{F:bool}$ or $\texttt{[]:}\alpha\ \texttt{list}$ or $\texttt{NONE:}\alpha\ \texttt{option}$
  - default translation to values of a HOL type `sexp`
  - unverified trusted rules and implementation
- HOL oracle theorems tagged with `"ACL2"`
- Uses of `ACL2PII`
  - importing 'small machine' theory from ACL2
  - an example in Susanto's PhD supervised by Tom Melham (*A Verification Platform for System on Chip*)
- Built with rather complex PROSPER Plug-In Interface (PII)

PROSPER and ACL2PII (illustration of complexity – not an explanation)

- PROSPER is ‘middleware’ for linking to HOL



PROSPER

Interrupts

Interrupt for PE

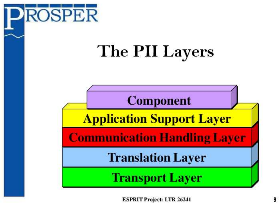
App

CM

PE

- Interrupt operation sent by application
- Intercepted by CM
- Interrupt signal sent using the operating system

ESPRIT Project: LTR 26241 6



PROSPER

The PII Layers

Component

Application Support Layer

Communication Handling Layer

Translation Layer

Transport Layer

ESPRIT Project: LTR 26241 9

- ACL2PII connects running HOL and ACL2 sessions

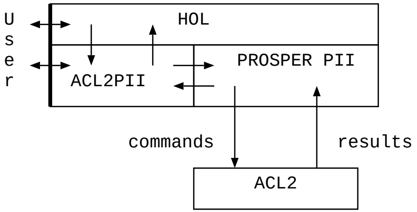
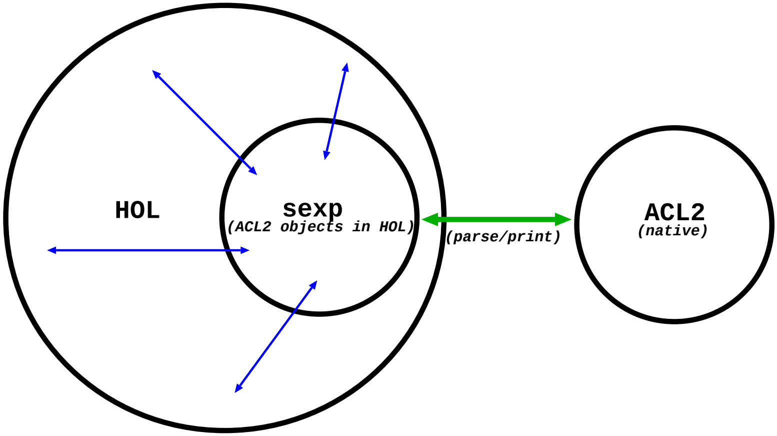


Figure 1: How ACL2 Talks with Hol

From `ACL2PII` to `ACL2-in-HOL`

- `ACL2PII` was pioneering, but:
  - soundness dependent on user-supplied translation rules
  - no way to verify this soundness other than by intuition
  - only supported one-way translation from ACL2 to HOL
  - used complex no-longer-supported PROSPER toolkit
- `ACL2-in-HOL` addressed these issues:
  - built on `ACL2PII` default translation to HOL `sexp` values
  - initial experiments by Gordon, Hunt and Kaufmann
  - James Reynolds' PhD research
  - case study by Gordon, Kaufmann and Ray

`ACL2-in-HOL`: ACL2-HOL link by reading/writing files



- File-based interface indicated by green arrow
- Trustworthiness of $\longleftrightarrow$ depends on its simplicity
- Blue arrows are proof verified translations inside HOL

Defining of S-expressions in HOL

- Datatype `sexp` has five constructors

Category	ACL2	HOL term representation
Symbol	*sym*	`ACL2_SYMBOL` *pkg sym*
String	*str*	`ACL2_STRING` *str*
Character	*chr*	`ACL2_CHARACTER` *chr*
Number	*n*	`ACL2_NUMBER` *n*
Dotted pair	$(s_1.s_2)$	`ACL2_PAIR` s_1 s_2

- Values of type `sexp` correspond to ACL2 S-expressions
  - after macro expansion and other preprocessing
- Definition of `sexp` in HOL:

```
(*****************************************************************************)
(* ACL2 S-expressions defined as a HOL datatype.                             *)
(* Definition below adapted from Mark Staples' code.                         *)
(*****************************************************************************)
val _ = Hol_datatype
 `sexp = ACL2_SYMBOL    of packagename => name     (* only curried for style *)
       | ACL2_STRING    of string
       | ACL2_CHARACTER of char
       | ACL2_NUMBER    of complex_rational
       | ACL2_PAIR      of sexp => sexp`;          (* only curried for style *)
```

ACL2 and HOL logics

- ACL2 logic is first-order axiomatic theory in `axioms.lisp`
- Axiomatizes Lisp atoms `nil`, `t` and 31 ACL2 primitives:

```
acl2-numberp bad-atom<= binary-* binary-+ unary-- unary-/
< car cdr char-code characterp code-char complex
complex-rationalp coerce cons consp denominator equal if
imagpart integerp intern-in-package-of-symbol numerator
pkg-witness rationalp realpart stringp symbol-name
symbol-package-name symbolp
```

- HOL theory SEXP defines these as constants
- ACL2 formulae are S-expressions, HOL's are `bool` terms
- ACL2 term `p` is true if and only if it is not `nil`
- HOL formula $\models$ `p` defined to mean that `p` is true:

```
⊢ ∀p. (⊨ p)  =  ¬(p = nil)
```

- Axiom `car-cdr-elim` is verified in HOL by proving:

```
⊢ ∀x. ⊨ implies (consp x) (equal (cons (car x) (cdr x)) x)
```

- Many axioms in `axioms.lisp` verified in HOL4

HOL theory SEXP is the ACL2 ‘Standard Model’

- HOL definitions formalise ACL2 Standard Model
- ACL2 logic is first-order so also has non-standard models
  - ACL2(r) depends on this
- Exist formulas true in standard model but not in all models
- Exist formulas provable in HOL theory but not in ACL2
- HOL theory is stronger – not equivalent to ACL2
  - but can’t prove things inconsistent with the ACL2 axioms

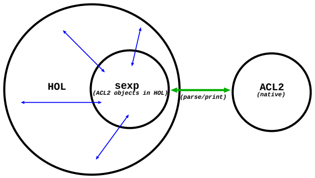
From higher-order logic to first-order S-expressions

- ACL2 is not typed but HOL is typed
- Constants defined in SEXP must have a HOL type

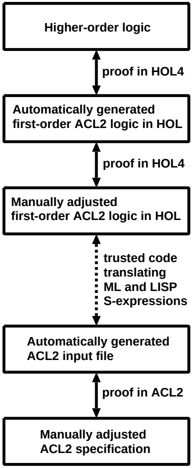
HOL constants	**HOL type**
`t, nil`	`sexp`
$\models$	`sexp` $\rightarrow$ `bool`
`car, cdr`	`sexp` $\rightarrow$ `sexp`
`cons, equal, implies`	`sexp` $\rightarrow$ `sexp` $\rightarrow$ `sexp`
`acl2_if`	`sexp` $\rightarrow$ `sexp` $\rightarrow$ `sexp` $\rightarrow$ `sexp`

- Reynolds implemented translation tools (blue arrows below)
- Recursive: encodes and decodes sub-functions
- Builds library for encoding/decoding primitive functions
- HOL datatypes are encoded as s-expressions
- HOL functions as first-order functions on s-expressions

$$\begin{tikzcd}
\boxed{\sigma} \arrow[r, "f"] \arrow[d, "\text{encode\_}\sigma"'] & \boxed{\tau} \\
\boxed{\text{sexp}} \arrow[r, "\text{encode\_}\sigma\tau\ (f)"] & \boxed{\text{sexp}} \arrow[u, "\text{decode\_}\tau"']
\end{tikzcd}$$



Flow between HOL and ACL2


Higher-order logic
proof in HOL4
Automatically generated first-order ACL2 logic in HOL
proof in HOL4
Manually adjusted first-order ACL2 logic in HOL
trusted code translating ML and LISP S-expressions
Automatically generated ACL2 input file
proof in ACL2
Manually adjusted ACL2 specification

- arrows preserve semantic equivalence
- solid arrows checked by proof
- dotted arrows trusted
- validated by 'round-trip' testing:
  $\vdash_{ACL2}\ x\ =\ \text{toACL2}(\text{toHOL}\ x)$

ACL2-in-HOL: discussion

- Initial goal was fast trustworthy execution
- Results below are only illustrative
  - involves 3 different systems: Moscow ML, MLton , ACL2
  - not clear if the same things are being timed
  - HOL EVAL timed out!

Executing a million instructions on a toy interpreter					
HOL EVAL	Moscow ML	MLton	ACL2	Common Lisp	ACL2 + stobj
∞?	5.8	1.26	8.16	5.6	0.06

- Suggests executing Fox's ARM model worth exploring
  - intended use co-simulation with actual ARM
  - bug finding: not soundness-critical
  - ARM not done; Reynolds' VFP benchmarks encouraging
  - VFP: ARM Vector Floating-point Coprocessor

James Reynolds' arithmetic and FP results

- James Reynolds' PhD: export HOL model of VFP to ACL2
- Ground arithmetic benchmarks (optimised Poly/ML HOL4)
  - numbers are triples: (*sign*, *exponent*, *fraction*)

a:	b:	result:	Time(s):
(0,15,524288)+	(0,15,262144)=	(0,16,393216)	0.374
(0,125,4194303)+	(0,125,2097151)=	(0,126,3145727)	0.399
(0,251,8388607)+	(0,251,4194303)=	(0,252,6291455)	0.441

- ACL2 enormously faster

a:	b:	result:	Time(s)
(0,15,524288)+	(0,15,262144)=	(0,16,393216)	6.460×10^{-4}
(0,125,4194303)+	(0,125,2097151)=	(0,126,3145727)	2.460×10^{-3}
(0,251,8388607)+	(0,251,4194303)=	(0,252,6291455)	3.860×10^{-3}

- Benchmarks for FP calculations in VFP less spectacular

Operation:	HOL (s):	ACL2 (s):	Ratio:
Multiply Accumulate, Single, *len*= 8	6.71	0.0539	124
Multiply Accumulate, Double, *len*= 4	11.9	0.0836	142
Square Root, Single, *len*= 8	4.19	0.0543	77.1
Square Root, Double, *len*= 4	15.7	0.0936	167

Another application: cone of influence reduction

- Started with an ACL2-proved first-order theorem
- ACL2 used non-standard finite-path LTL semantics
- HOL can express standard infinite path semantics
- Proof done in HOL by importing main lemma from ACL2
- Combined imported lemma with HOL LTL semantics
- Result is a HOL theorem in ACL2 standard model
- Details in 2011 JAR paper (Gordon, Kaufmann, Ray)
- Title:

  The Right Tools for the Job: Correctness of Cone of Influence Reduction Proved Using ACL2 and HOL4
- Abstract:

  We present a case study illustrating how to exploit the expressive power of higher-order logic to complete a proof whose main lemma is already proven in a first-order theorem prover. Our proof exploits a link between the HOL4 and ACL2 proof systems to show correctness of a cone of influence reduction algorithm, implemented in ACL2, with respect to the classical semantics of linear temporal logic, formalized in HOL4.

Is there a future for linking ACL2 and HOL?

- Incremental projects with existing `ACL2-in-HOL`
  - convert HOL ARM, MIPS, RISC-V models to ACL2
  - convert ACL2 JVM model to HOL
  - validate Cryptol-to-ACL2 translation with HOL semantics
  - hard to motivate without some wider research context
- Solving HOL goals by trusting native ACL2 automation
  - an ‘ACL2 tactic’ in HOL
  - convert HOL goal G to ACL2 formula G_{ACL2}
  - import HOL definitions used in G to ACL2 definitions
  - use imported context to prove G_{ACL2}
  - if success, import G_{ACL2} into HOL (tagged ACL2)
  - this trusts ACL2; doesn’t replay in HOL (cf. CLAM-HOL)
- $\text{HOL} \longleftrightarrow \underbrace{\{\text{L3, SAIL, LEM}\}}_{\text{formal specification DSLs}} \longleftrightarrow \text{ACL2 ?}$
  - Matt has experimented with translating L3 to ACL2

Matt's idea: `HOL-in-ACL2`

- HOL implemented in ACL2 with ACL2 as metalanguage
  - Matt emphasises ideas inspired by Jared Davis's Milawa
  - represent HOL terms in ACL2
  - define `provable-p` in ACL2
  - fast evaluator `hol-eval`
    $\vdash_{ACL2}$ `(provable-p(make-equality x (hol-eval x ...)))`
- Potential customers
  - HOL users wanting to use ACL2 for fast evaluation
  - HOL users wanting to use ACL2 to prove lemmas
  - ACL2 users wanting to do higher-order proof developments
- Example target application: fast evaluation for CakeML
  - see: `cakeml.org`
  - compiler is written in HOL then translated to CakeML
  - perform bootstrap ground evaluation
    `(hol-eval(CakeMLCompile ⌜CakeMLCompile⌝) ...)`

Matt's plan and what he's done so far

- Define a notion of HOL term in ACL2
  - STATUS: simplified version done, e.g., no polymorphism
- Define a function `(hol-eval x defs depth)`
  - `x` an ACL2 representation of a HOL term
  - `(hol-eval x defs depth)` provably equal to `x`
  - STATUS: initial version of `hol-eval` done
    (the "provably equal" part is only barely begun)
- Write tool to import HOL developments into ACL2
  - import ACL2 representations of HOL definitions
  - import ACL2 representations of HOL theorems
  - maybe use OpenTheory?
  - STATUS: TO DO
- Collaborators sought

Defining HOL provability in ACL2

- Get Matt to explain this after the talk!
- Model HOL proof theory in ACL2 with:
  `(provable-p x gamma defs axioms)`
- Inference rules correctness from definition of `provable-p`
- Idea not new – similar thing done by Jared Davis
- Prove correctness of hol-eval:

```
(defthm hol-eval-correct
  (let ((y (hol-eval x defs depth)))
    (implies (and (def-list-p defs)
                  (hol-term-p x defs)
                  (hol-ground-term-p x)
                  (subsetp-equal (base-axioms) axioms)
                  (not (equal y *out-of-time*)))
             (provable-p (make-hol-binary-ap '= x y :bool)
                         gamma defs axioms))))
```

- Optional: prove a completeness theorem for `hol-eval`

```
(terminating-p defs ...)==>
(exists depth) (hol-value-p (hol-eval x defs depth))
```

- Collaborators sought

Cool stuff becomes possible with `HOL-in-ACL2`

- Verify HOL rule optimisations with ACL2
  - install verified derived rules of inference
- Matt's example (STATUS: DONE)
  - Matt coded a HOL primitive rule in ACL2

$$\frac{\vdash a_0 = b_0, \quad \vdash a_1 = b_1}{\vdash a_0(a_1) = b_0(b_1)}$$

  - Matt also coded a derived rule in ACL2

$$\frac{\vdash a_0 = b_0, \quad \vdash a_1 = b_1, \quad \ldots \quad \vdash a_k = b_k}{\vdash a_0\ a_1\ \cdots\ a_k = b_0\ b_1\ \cdots\ b_k}$$

  - Matt then proved that:
    - if there is a proof using the derived rule
    - then there is a proof using only the primitive rule
    - uses an ACL2 representation of HOL proofs
    - ask Matt about ACL2 details
  - verifies one can trust the non-fully-expansive ACL2 code
- Collaborators sought

More ideas from Matt

- Verify other kinds of derived rules of inference
  - BDDs, symbolic execution, ACL2(h)
- Implement even faster evaluation
  - replace `hol-eval` interpreter by direct ACL2 evaluation (preliminary benchmark: 2 seconds $\longrightarrow$ 1/100 second)
  - use stobjs?
- Employ native ACL2 reasoning to do HOL proofs
  - e.g. prove HOL theorem
    $\vdash$ `(hol-app (hol-app x y) z) = (hol-app x (hol-app y z))`
    by converting to trivial associativity-of-append in ACL2
- Build infrastructure for seamless mixing of HOL and ACL2 (suggested by Warren Hunt)
  - rework *Right Tools for the Job* in `HOL-in-ACL2`
- Collaborators sought

Final thoughts

- Activities possible at several levels
  - undergraduate: experiment with `hol-eval`
  - masters: verify and use derived HOL rules inside ACL2
  - PhD: build infrastructure and study trust challenges
  - funded international projects: above + major applications
- Many extremely cool possibilities ahead!
- Collaborators sought

THE END