in twenty minutes
How can we trust ACL2?

- 190,000 lines of Lisp (7.9 MB)
- Extensive use of program mode
- Hugely complex proof search
  - Rewriter includes integrated arithmetic and type reasoning, backward and forward chaining rules, and can call on user-written extensions (meta rules), etc.
Oh, please!

- Careful authors with domain expertise
- Massive regression suite to test against
- Many users working on non-trivial projects
- Soundness bugs are rare, tend to be esoteric
- Certainly a whole lot better than hand proofs
On the other hand...

- Extending ACL2 requires its authors
  - You or I need to “make a case” for why you need something new

- Soundness bug might be exploited to make false claims about some artifact
  - Probably only intentionally, but maybe even unintentionally
Milawa

• A small, trusted core proof checker
  - That really deals with formal proofs

• Written in its own logic
  - Which makes provability an idea we can talk about concretely

• With extensions that add capabilities
  - Like evaluation, rewriting, etc.
Soundness of Extensions

- Extensions written in the same logic
  - We can reason about them
  - We can prove they are sound

- Example: tautology checking
  - (tautologyp F) checks if F is a tautology
  - Goal:
    (tautologyp F) -> (provablep F)
Reflection

• “F is provable” is a proof of F

• Allows us to use our extensions

• Example:
  - Suppose (tautologyp F) is true.
  - Since (tautologyp F) -> (provablep F), it follows that (provablep F) is true.
  - By reflection, I conclude F is true.
Implementing Milawa

• Rigorously define the Milawa logic
  – Mostly done (still need definitions)

• Implement the core proof checker
  – Mostly done (everything but reflection)

• Implement some useful extensions
  – So far, I have tautology checking, substitution of equals for equals, evaluation of ground terms, and basic rewriting
The Hard Part

- Verify one of these extensions using only the core proof checker
  - Its rules of inference are extremely limited
  - How can we reasonably construct the formal proofs we need?

- I am focusing on tautology checking (TC) to begin with.
My Approach

• I implemented the core checker as an ACL2 program.

• I have an ACL2 proof of TC's soundness.
  – The logics are very similar
  – I can see now what lemmas I will need for the “real” proof.

• This is like sketching a proof before trying to push it through ACL2.
Attack From Both Sides

• Simplify the ACL2 proof so that fewer features of ACL2 are used.
  – For example, I have already eliminated all type reasoning, linear arithmetic, and forward chaining rules.

• Create functions to assist with building formal proofs.
  – For example, I have a function which will build a proof of a ground term's evaluation, a rewriter's simplification, etc.
Rinse and Repeat

- Prove the soundness of TC.

- Move on to the next extension.
  - But now we can use TC in our proofs, so our job will be somewhat easier.

- Work through the remaining extensions
  - At each step, we obtain a more capable program, which is nevertheless trustworthy
The Final Product

• A reasonably sophisticated proof checker

• That we can really trust

• That can be extended by users in complicated ways

• That is easy to port ACL2 proofs to?