Can we establish, in advance, that a useful theorem prover is sound?

Computer-checked proofs can be trusted if

1. We guard against computer mistakes
2. We have confidence in the logic
3. We believe our program is sound (it only accepts theorems)
The Milawa theorem prover

Goal-directed proof search
Rewriter with many features
  Assumptions system
  Calculation of ground terms
Case splitting into subgoals
  “Destructor elimination,” generalization, use of equalities
Induction

Has carried out large, complex proofs
To show Milawa is sound, we

1. Define provability for our logic

2. Model the theorem prover

3. Prove it only accepts theorems

Milawa finds the proof “Self-verifying”

A simple program checks the proof Avoids “I never lie”
A challenge

Formal proofs are long. Soundness is hard.

Is a formal proof possible?

We separate the challenge of finding the proof from constructing it.

To manage proof size, we develop and verify a series of increasingly capable proof checkers.
# Road Map

1. The simple program
   a. The Milawa logic
   b. Level 1 proof checker
   c. The command loop
   d. Higher-level proof checkers

2. Self-verification
   a. Building proofs
   b. Verifying proof techniques
   c. Planning the proof
   d. Following the plan

3. Building the proof
   a. Fully expansive Milawa
   b. Higher-level proof checkers
   c. Layering the proof
   d. Checking the proof

4. Conclusions
   a. Review of the proposal
   b. Related work
   c. Contributions
1-a. The Milawa logic

Objects
- Naturals
- Symbols
- Ordered Pairs

12 Primitives
- if, equal
- natp, +, -, <
- symbolp, symbol-<
- consp, cons, car, cdr

Terminating, recursive functions

Skolem functions

No type system, functions are total

Similar to the ACL2 logic
Rules of inference, axioms

Propositional Schema
\[ \neg A \lor A \]

Reflexivity Axiom
\[ x = x \]

Contraction
\[ A \lor A \]

Equality Axiom
\[ x_1 = y_1 \rightarrow x_2 = y_2 \rightarrow x_1 = x_2 \rightarrow y_1 = y_2 \]

Expansion
\[ A \]

Referential Transparency
\[ x_1 = y_1 \rightarrow \ldots \rightarrow x_n = y_n \rightarrow f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n) \]

Associativity
\[ A \lor (B \lor C) \]

Beta Reduction
\[ ((\lambda x_1 \ldots x_n. \beta) \ t_1 \ldots \ t_n) = \beta/[x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n] \]

Cut
\[ (A \lor B) \lor C \]

Base Evaluation
\[ e.g., 1+2 = 3 \]

Instantiation
\[ A \]

52 Lisp Axioms
\[ e.g., \ consp(\ cons(x, y)) = t \]

Induction
The logic as a programming language

Logical functions can be implemented in Lisp

Lisp Integers (arbitrary precision)
Lisp Symbols
Lisp Conses

(defun MILAWA::car (x)
  (if (consp x) (car x) nil))

(defun f (...)
  (... (f ...) ...))

(defun skolem (...)
  (error "Called skolem function."))
1-b. The level 1 proof checker

\[ \phi \text{ is provable when } \exists p : \text{Proofp}(p) \land \text{Conclusion}(p) = \phi \]
1-c. The command loop

Command File

(DEFINE F ...)
(SKOLEM F ...)
(VERIFY \( \phi \) ...)
(SWITCH ...)
(FINISH ...)

Command Loop

(VERIFY \( \phi \) "file.proof")

Proof establishes \( \phi \)?

Conclusion(\_) \( \equiv \) \( \phi \)

Proof is valid?

Proofp

\[ \text{step-okp} = \text{OR} \begin{cases} \text{prop-schema-okp} \\ \text{contraction-okp} \\ \ldots \end{cases} \]

Then add \( \phi \) to Theorems

System State

Axioms
Theorems
Arity Table

Lisp Environment

Then add \( \phi \) to Theorems
1-d. Higher-level proof checkers

(SWITCH New-Proofp)

Soundness theorem for New-Proofp

If:

\[ \mathbf{P} \text{ is a proof structure concluding } \phi, \text{ and} \]
\[ \text{New-Proofp}(\mathbf{P}, \text{axioms, thms, atbl}) \]

Then:

\[ \text{Provablep}(\phi, \text{axioms, thms, atbl}) \]
# Road Map

## 1. The simple program
- a. The Milawa logic
- b. Level 1 proof checker
- c. The command loop
- d. Higher-level proof checkers

## 2. Self-verification
- a. Building proofs
- b. Verifying proof techniques
- c. Planning the proof
- d. Following the plan

## 3. Building the proof
- a. Fully expansive Milawa
- b. Higher-level proof checkers
- c. Layering the proof
- d. Checking the proof

## 4. Conclusions
- a. Review of the proposal
- b. Related work
- c. Contributions
2-a. Building proofs

Proof representation

Expansion \[ \frac{A}{B \lor A} \]

...
Primitive proof builders

Propositional Schema
\[ \neg A \lor A \]

\[
\text{build.propositional-schema}(A) =
\]

Cut
\[ A \lor B \quad \neg A \lor C \]
\[ \frac{\text{B} \lor C}{B \lor C} \]

\[
\text{build.cut(}
\begin{array}{c}
\ldots \\
A \lor B \\
\ldots
\end{array}
, 
\begin{array}{c}
\ldots \\
\neg A \lor C \\
\ldots
\end{array}
) =
\begin{array}{c}
\text{Cut} \\
B \lor C \\
nil
\end{array}
\]
Non-primitive builders

Commut Or
1. $A \lor B$  Given
2. $\neg A \lor A$  Propositional Schema
3. $B \lor A$  Cut 1, 2

build.commute-or(x) =

build.commute-or(x) = Let a = lhs(conclusion(x))
build.cut(x, build.propositional-schema(a))
The Three Theorems

Given suitable inputs, we prove each builder is

Well Typed: it builds a proof structure

Relevant: the proof has the desired conclusion

Sound: the proof is accepted by Proofp

These compose and allow us to treat builders as black boxes
2-b. Verifying proof techniques

Introduce the technique

\[ x \xrightarrow{\text{Ground Term Evaluator}} x' \]

Soundness claim \( x = x' \) is provable

Introduce a fully-expansive version
Establish it is well-typed, relevant, and sound

\[ x \xrightarrow{\text{Evaluation Builder}} \text{Proof of } x = x' \xrightarrow{\text{Proofp}} T \]

Many similarities to LCF systems
2-c. Planning the proof

We develop a plan of the proof in ACL2

A long sequence of events
- 2,700 definitions
- 11,600 theorems

Basic utilities (lists, arith, …)
- Logical concepts
- Builder library
- Clauses, clause splitting
- Rewriting
- Tactic system

(DEFUN ...)
(DEFUN ...)
(DEFTHM ...)
(DEFUN ...)
(DEFTHM ...)
(DEFUN ...)
(DEFTHM ...)
(DEFTHM ...)
(DEFTHM ...)
...
2-d. Following the plan

\[
\begin{align*}
&\text{(DEFUN \ldots)} \\
&\text{(DEFUN \ldots)} \\
&\text{(DEFSYM \ldots)} \\
\end{align*}
\]

\[
\begin{align*}
&\text{(DEFSYM \ldots)} \\
&\text{(DEFSYM \ldots)} \\
&\text{(DEFSYM \ldots)} \\
&\text{\ldots}
\end{align*}
\]
Road Map

1. The simple program
   a. The Milawa logic
   b. Level 1 proof checker
   c. The command loop
   d. Higher-level proof checkers

2. Self-verification
   a. Building proofs
   b. Verifying proof techniques
   c. Planning the proof
   d. Following the plan

3. Building the proof
   a. Fully expansive Milawa
   b. Higher-level proof checkers
   c. Layering the proof
   d. Checking the proof

4. Conclusions
   a. Review of the proposal
   b. Related work
   c. Contributions
3-a. Fully expansive Milawa

Recall how we verified our proof techniques

Easy to develop a **fully expansive** version of Milawa
A strategy for formalizing the proof

(DEFUN ...) (DEFUN ...) (DEFFTHM ...)

(DEFTHM ...)

Milawa Hints

(DEFUN ...) (DEFFTHM ...) (DEFFTHM ...)

...
3-b. Higher-level proof checkers

Alleged Proof

11. A v B
12. ~A
13. B v ~A Expand 12
14. ~B v B Prop Axiom
15. ~A v B Cut 13, 14
16. B v B Cut 11, 15
17. B Contact 16

Proofp
Proof Predicate

Ensures each step is step-okp, where

step-okp = OR

prop-schema-okp
contraction-okp
expansion-okp
...

Accepts only primitive rules
Good for trust, bad for proof size
Writing new proof checkers

Level2.Proofp
Ensures each step is Level2.step-okp, where

Level2.step-okp = OR

- commute-or-okp
- modus-ponens-okp
- ...
- Level1.step-okp

New Rules

Old Rules
Verifying higher-level proof checkers

Our simple program can't use the new proof checker until we prove it is sound

If:

P is a proof structure concluding \( \phi \), and

New-Proof\( p(P, \text{axioms, thms, atbl}) \)

Then:

Provable\( p(\phi, \text{axioms, thms, atbl}) \)

But now this is easy! (next slide)
Proving the soundness theorem

Show how to compile any high-level step into a Level 1 proof

Level 2 Proof

<table>
<thead>
<tr>
<th>Commute Or</th>
</tr>
</thead>
<tbody>
<tr>
<td>B ∨ A</td>
</tr>
<tr>
<td>nil</td>
</tr>
</tbody>
</table>

...  
A ∨ B  
...

Level 1 Proof

...  
A ∨ B  
...

Inductive Construction
Proving the soundness theorem

Show how to compile any high-level step into a Level 1 proof

Level 2 Proof

Commute Or

B ∨ A

nil

... A ∨ B ...

... Inductive Construction ...

Level 1 Proof

Cut

B ∨ A

nil

... A ∨ B ...

... Prop Schema

¬A ∨ A

nil

nil

build.commuter-or() =
Emitting high-level proofs

build.commute-or

Cut

B ∨ A

nil

... A ∨ B

... nil

Prop Schema

¬A ∨ A

nil

nil

Fully Expansive Milawa

Level 1 Proof

Level 2 Proof
3-c. Layering the proof

Level 1  The Primitives
Level 2  Propositional reasoning
Level 3  Rules about primitive functions
Level 4  Miscellaneous groundwork
Level 5  Assms. traces, updating clauses
Level 6  Factoring, splitting help
Level 7  Case splitting
Level 8  Rewriting traces
Level 9  Unconditional rewriting
Level 10 Conditional rewriting
Level 11 All tactics
Effects of layering

A “hard” lemma toward level 3

<table>
<thead>
<tr>
<th>Level</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search (s)</td>
<td>39.4</td>
<td>39.4</td>
<td>39.4</td>
<td>39.4</td>
<td>39.4</td>
<td>39.4</td>
<td>39.4</td>
<td>39.4</td>
<td>39.0</td>
<td>12.3</td>
<td>12.3</td>
</tr>
<tr>
<td>Build (s)</td>
<td>406</td>
<td>226</td>
<td>117</td>
<td>106</td>
<td>102</td>
<td>101</td>
<td>101</td>
<td>0.8</td>
<td>0.5</td>
<td>0.04</td>
<td>0.008</td>
</tr>
<tr>
<td>Size (MC)</td>
<td>3,681</td>
<td>441</td>
<td>234</td>
<td>62</td>
<td>53</td>
<td>38</td>
<td>36</td>
<td>76</td>
<td>76</td>
<td>.8</td>
<td>.8</td>
</tr>
<tr>
<td>Check (s)</td>
<td>11,440</td>
<td>2,968</td>
<td>914</td>
<td>433</td>
<td>408</td>
<td>342</td>
<td>332</td>
<td>50</td>
<td>50</td>
<td>12.8</td>
<td>12.6</td>
</tr>
</tbody>
</table>

A “moderate” lemma toward level 8

<table>
<thead>
<tr>
<th>Level</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search (s)</td>
<td>354</td>
<td>354</td>
<td>354</td>
<td>354</td>
<td>354</td>
<td>354</td>
<td>354</td>
<td>354</td>
<td>346</td>
<td>346</td>
<td></td>
</tr>
<tr>
<td>Build (s)</td>
<td>Ø</td>
<td>6,238</td>
<td>2,879</td>
<td>2,279</td>
<td>2,157</td>
<td>1,482</td>
<td>768</td>
<td>691</td>
<td>167</td>
<td>65</td>
<td>8</td>
</tr>
<tr>
<td>Size (MC)</td>
<td>Ø</td>
<td>8,289</td>
<td>4,310</td>
<td>1,117</td>
<td>1,049</td>
<td>426</td>
<td>222</td>
<td>171</td>
<td>129</td>
<td>58</td>
<td>27</td>
</tr>
<tr>
<td>Check (s)</td>
<td>Ø</td>
<td>31,451</td>
<td>5,323</td>
<td>2,816</td>
<td>3,120</td>
<td>2,737</td>
<td>1,874</td>
<td>1,430</td>
<td>440</td>
<td>457</td>
<td>163</td>
</tr>
</tbody>
</table>
3-d. Final checking of the proof

The proof files total 9 GB, uncompressed

We successfully checked all proofs on these machines, using Clozure Common Lisp

<table>
<thead>
<tr>
<th>Machine</th>
<th>Type</th>
<th>Processor</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jordan</td>
<td>My home computer</td>
<td>Intel Core 2 Duo</td>
<td>13.8 hrs</td>
</tr>
<tr>
<td>Cele</td>
<td>Apple MacBook</td>
<td>Intel Core 2 Duo</td>
<td>19.8 hrs</td>
</tr>
<tr>
<td>Lhug-3</td>
<td>HP server</td>
<td>AMD Opteron</td>
<td>31.2 hrs</td>
</tr>
</tbody>
</table>

Many proofs were also checked on these, and other machines, with different lisps
Road Map

1. The simple program
   a. The Milawa logic
   b. Level 1 proof checker
   c. The command loop
   d. Higher-level proof checkers

2. Self-verification
   a. Building proofs
   b. Verifying proof techniques
   c. Planning the proof
   d. Following the plan

3. Building the proof
   a. Fully expansive Milawa
   b. Higher-level proof checkers
   c. Layering the proof
   d. Checking the proof

4. Conclusions
   a. Review of the proposal
   b. Related work
   c. Contributions
4-a. Review of the proposal

The proposal describes
- The logic and proof checker
- The approach to building proofs
- The introduction and verification of extended proof checkers
- The verification of Level 2 proof checker (with one rule)

I proposed to
- Explain why the logic is reasonable and why the simple program is sound. *(See Chapters 2-4)*

Use this approach to verify a theorem prover that implements clausification (case splitting), evaluation, equality reasoning, conditional rewriting, destructor elimination. *(See Chapters 5-12)*
4-b. Related work

Other ways to develop theorem provers
- Boyer-Moore theorem provers
- LCF-style theorem provers
- Constructive type theory provers

Embedding proof checkers in a logic
- Gödel's proof, Shankar's formalization

Mechanically verifying proof checkers
- Harrison (HOL Light's core), von Wright (imperative proof checker)

Independently checking proofs
- McCune/Shumsky (Ivy), Obua/Skalberg (HOL to Isabelle/HOL)

Meta-reasoning in other systems
- Metafunctions, reducibly equal terms in Coq, ...
4-c. Contributions

A new approach to developing trustworthy theorem provers

Does not require fully expansive proofs

Demonstrates how Boyer-Moore theorem provers may be verified

Verified many theorem proving algorithms

Applications in other theorem provers with meta-reasoning capabilities
Additional contributions

A flexible proof representation

Many kinds of objects are treated as proofs
(rewrite traces, equivalence traces, proof skeletons)

An extensible proof representation

Verifying new kinds of proof steps can improve
efficiency of proof construction and checking

Potential target for other systems
Thanks!