The Milawa Theorem Prover is Sound
down to the x86 machine code that runs it

Jared Davis
Centaur Technology, USA

Magnus Myreen
University of Cambridge, UK
Correctness in Computer Science

**Artifacts**
- Program
- Hardware module
- Algorithm
- Protocol
- Type system

**Goals**
- Implements the C standard
- Correctly divides floats
- Finds the minimum spanning tree
- Transactions are secret
- Subtyping is transitive
Formal Verification with Theorem Provers

- Artifact
  - Modeling Effort
  - Guidance
- Model
- Theorem Prover
- Proofs
- Goals
  - Modeling Effort
  - Properties
Focus: Soundness of the Theorem Prover

- Guidance
- Model
- Theorem Prover
- Properties
- Proofs

- Goals
- Artifact
- Modeling
- Effort
Scope of the Theorem Prover

850K line design

Translator
125K loc

Model

350K loc

Proofs

Human Effort

Properties

Heavy, heavy books
Scope of the Theorem Prover

850K line design

Translator

125K loc

Model

350K loc

Properties

300K lisp lines

70K c/asm lines

Human Effort

Heavy, heavy books
This Talk

850K line design

Translator

Model

350K loc

300K lisp lines

70K c/asm lines

2-4M lines

6-15M lines

125K loc

Human Effort

Properties

Heavy, heavy books
What is Milawa?

User Interface

Interactive, command-line program
Define functions
Propose theorems
Manage proof attempts

Unsound, Unverified

MILAWA !> (%defun in (a x)
  (if (consp x)
    (or (equal a (car x))
      (in a (cdr x))
    nil)
  :measure (len x))
Two goals remain.

1. `(EQUAL (ORDP (LEN X)) 'T)`

2. `(IF (EQUAL (ORD< (LEN (CDR X)) (LEN X)) 'T) 'T (IF (EQUAL (CONSP X) 'NIL) 'T (EQUAL (NOT (EQUAL A (CAR X))) 'NIL)))`

MILAWA !>
Termination Proof

MILAWA !>(%split)
; Splitting clause 2.
; Splitting clause 1.
Two goals remain.

1. (EQUAL (ORDP (LEN X)) 'T)

2. (IMPLIES (AND (NOT (EQUAL A (CAR X)))
               (CONSP X))
               (EQUAL (ORD< (LEN (CDR X)) (LEN X))
                       'T))

MILAWA !>
MILAWA !>(%crewrite default)

; Rewrote clause #2 in 0.001999 seconds (proved) ...
; Rewrote clause #1 in 0.038994 seconds (proved) ...
; Rewrote 2 clauses; 0 (+ 0 forced) remain.
All goals have been proven.
MILAWA !(%rewrite default)
; Rewrote clause #2 in 0.001999 seconds (proved) ...
; Rewrote clause #1 in 0.038994 seconds (proved) ...
; Rewrote 2 clauses; 0 (+ 0 forced) remain.
All goals have been proven.

MILAWA !(%admit)
; Compiling worlds for IN...
; Compiling proofs for IN...
...
;; Preparing to admit IN.
;; Proof sizes total: 3,409,472 conses ...
; Checking the proofs...
...
; Proof-checking completed.
;; Proofs accepted. Saving as user/admit-in.proofs
...
New rule: IN

MILAWA !>
Example Theorem: (not (in a a))

Let's prove lists can't be in themselves

This can't happen, no circular conses

Lists have to be *bigger* than their elements
The Key Lemma

Lists have to be bigger than their elements:

\[
\text{MILAWA }!>(\%\text{defthm rank-when-in})
\]
\[
\text{\hspace{1em} (implies (in a x)}
\]
\[
\hspace{1em} \hspace{1em} (\hspace{1em} (\text{< (rank a) (rank x)})\hspace{1em}))
\]

One goal remains.

1. (IMPLIES (AND (IN A X))
\[
\hspace{1em} (\text{IFF (}\hspace{1em} (\text{< (RANK A) (RANK X)}) \hspace{1em} 'T))
\]
One goal remains.

1. \( (\text{IMPLIES} \ (\text{AND} \ (\text{IN} \ A \ X)) \ (\text{IFF} \ (< \ (\text{RANK} \ A) \ (\text{RANK} \ X)) \ 'T)) \)

MILAWA !>(%cdr-induction x)
... five subgoals ... 

MILAWA !>(%auto)
... various progress messages ...

Two goals remain.

1. \( (\text{IMPLIES} \ (\text{AND} \ (\text{NOT} \ (\text{CONSP} \ X))) \ (\text{NOT} \ (\text{IN} \ A \ X))) \)

2. \( (\text{IMPLIES} \ (\text{AND} \ (\text{IN} \ A \ (\text{CONS} \ X1 \ X2)) \ (\text{NOT} \ (\text{IN} \ A \ X2))) \ (< \ (\text{RANK} \ A) \ (+ \ '1 \ (+ \ (\text{RANK} \ X1) \ (\text{RANK} \ X2)))))) \)
One goal remains.

1. \( \text{IMPLIES} \ (\text{AND} \ (\text{IN} \ A \ X)) \ (\text{IFF} \ (< \ (\text{RANK} \ A) \ (\text{RANK} \ X)) \ 'T)) \)

MILAWA !>(%cdr-induction x)
... five subgoals ...

MILAWA !>(%auto)
... various progress messages ...

Two goals remain.

1. \( \text{IMPLIES} \ (\text{AND} \ (\text{NOT} \ (\text{CONSP} \ X))) \ (\text{NOT} \ (\text{IN} \ A \ X))) \)

A good rule!

2. \( \text{IMPLIES} \ (\text{AND} \ (\text{IN} \ A \ (\text{CONS} \ X1 \ X2)) \ (\text{NOT} \ (\text{IN} \ A \ X2))) \ (< \ (\text{RANK} \ A) \ (+ \ '1 \ (+ \ (\text{RANK} \ X1) \ (\text{RANK} \ X2)))))) \)

A good rule!
MILAWA !>(%qed)
; Compiling worlds for RANK-WHEN-IN...
...
; Preparing to check RANK-WHEN-IN.
; Checking the proof.
...
;; Proof accepted. Saving as user/thm-rank-when-in.proof
New rule: RANK-WHEN-IN
Our Goal Theorem

MILAWA !>(%defthm not-in-self
  (not (in a a)))

One goal remains.

1. (EQUAL (IN A A) 'NIL)

MILAWA !>(%use (%instance (%thm rank-when-in)
  (x a)))

... one goal with messy ifs ...
MILAWA !>(%split) ;; to clean it up
One goal remains.

1. (IMPLIES (AND (IFF (< (RANK A) (RANK A)) 'T))
  (NOT (IN A A)))
MILAWA !>(%crewrite default)
; Rewrote clause #1 in 0.001 seconds (proved), [...] 
; Rewrote 1 clauses; 0 (+ 0 forced) remain.
All goals have been proven.

MILAWA !>(%qed)
; Compiling worlds for NOT-IN-SELF...
...
;; Proof accepted. Saving as user/thm-not-in-self.proof
New rule: NOT-IN-SELF
Unsound, Unverified

User Interface

= 

Interfacing Nonsense
5,000 lines of ACL2 macros

Theorem Proving Tactics
2000 functions, 100,000 lines**
(Defined in the Logic)

- crewrite
- split
- use
- ...

The Milawa Interface
This Talk

850K line design

Translator
125K loc

Model

350K loc

300K lisp lines
70K c/asm lines

Properties

+ clozure

2-4M
lines

GCC

6-15M
lines

Human Effort

Heavy, heavy books
Milawa: A First Approximation

Unsound, Unverified
User Interface

Proof Files
Kernel

The Logic

Jitawa Machine Code

C

ACL2

GCC

GCC
Foreshadowing

Doesn't need to be sound...

Unsound, Unverified

User Interface

Proof Files

The Logic

Kernel

Jitawa Machine Code

Formally Verified with HOL4
So what's in the Kernel?

Kernel (Lisp Program)
165 functions, 2000 lines incl. PC

Proof Checker
100 functions, 800 lines
(Defined in the Logic)

- Define a function
- Prove a theorem
- Save your progress (checkpoint)
- Switch to a new proof checker

= Teeny Tiny Proof Steps
But there's kind of a catch...

Great Big Proof Files!

Teeny Tiny Proof Steps

Great Big Proof Files!
But there's kind of a catch...

Unsound, Unverified

User Interface

Too big to Store

Too big to Build

Too big to Check

The Logic

Kernel

Jitawa
Machine Code

C

GCC

ACL2

Applicative Common Lisp
Reflection and Self-Verification

Interfacing Nonsense
5,000 lines of ACL2 macros

Theorem Proving Tactics
2000 functions, 100,000 lines**
(Defined in the Logic)
- crewrite
- split
- use
- ...

Kernel (Lisp Program)
165 functions, 2000 lines incl. PC

Proof Checker
100 functions, 800 lines
(Defined in the Logic)
- Define a function
- Prove a theorem
- Save your progress (checkpoint)
- Switch to a new proof checker
A Self-Verifying Theorem Prover

Interfacing Nonsense

Theorem Proving Tactics
2000 functions, 100,000 lines**
(Defined in the Logic)

Find, Writes

Bootstrapping Proofs
13,000 theorems, 8 GB on disk

“The tactics can only prove formulas that the proof checker accepts.”

Kernel (Lisp Program)
165 functions, 2000 lines incl. PC

Proof Checker
100 functions, 800 lines
(Defined in the Logic)

Define a function
Prove a theorem
Save your progress (checkpoint)
Switch to a new proof checker
A Self-Verifying Theorem Prover

**Theorem Proving Tactics**
2000 functions, 100,000 lines**
(Defined in the Logic)

"The tactics can only prove formulas that the proof checker accepts."

**Interfacing Nonsense**

**Kernel (Lisp Program)**
165 functions, 2000 lines incl. PC

**Proof Checker**
100 functions, 800 lines
(Defined in the Logic)

Define a function
Prove a theorem
Save your progress (checkpoint)
Switch to a new proof checker

**Bootstrapping Proofs**
13,000 theorems, 8 GB on disk

Find, Writes

Checks
The tactics can only prove formulas that the proof checker accepts.

A Self-Verifying Theorem Prover

Theorem Proving Tactics
2000 functions, 100,000 lines**
(Defined in the Logic)

Kernel (Lisp Program)
165 functions, 2000 lines incl. PC

Proof Checker
100 functions, 800 lines
(Defined in the Logic)

Define a function
Prove a theorem
Save your progress (checkpoint)
Switch to a new proof checker

Interfacing Nonsense

Find, Writes

Bootstrapping Proofs
13,000 theorems, 8 GB on disk

“The tactics can only prove formulas that the proof checker accepts.”

Becomes

(Verified) Theorem Prover
Bootstrapping

Level 2
Core

Level 3
Propositional reasoning

Level 4
Rules about basic functions

Level 5
Miscellaneous groundwork
Assumptions and clauses

Level 6
Clause factoring, splitting groundwork

Level 7
Clause splitting

Level 8
Rewrite traces

Level 9
Evaluation, unconditional rewriting

Level 10
Conditional rewriter

Level 11
All other tactics
This Talk

850K line design

Translator
125K loc

Model

350K loc

Properties

300K lisp lines
70K c/asm lines

Heavy, heavy books

Human Effort

2-4M
lines

6-15M
lines

Intel® 64 and IA-32 Architectures Optimization Reference Manual

ACL2

Applicative Common Lisp

+ clozure associates

+ GCC

Informal Specification

Artifact

125K loc

Human Effort

850K line design
The Soundness Story, So Far

- **Kernel**
  - **Proof Checker**
  - Theorem Proving Tactics
  - Proven correct by the kernel

- **Common Lisp Runtime** (CCL, GCL, ...)
  - Is the logic sound?
  - Is the program faithful to it?
  - The program is short
  - “Social proof”

- **Operating System** (Linux, Mac, ...)
  - Practically have to trust
    - (no verified options)
    - Use multiple systems, at least

- **Hardware** (Intel, AMD, ...)
  - Fundamentally have to trust
    - Use multiple systems, at least
Two Projects Meet

**Milawa**
Self-verifying theorem prover

Jared Davis, UT Austin, 2009

verified **LISP** on
x86, ARM, PowerPC

Magnus Myreen, Cambridge, 2008
So can we do this?

Practically have to trust
Use multiple systems, at least

Theorem Proving Tactics

Kernel

Proof Checker

Common Lisp Runtime (CCL, GCL, ...)

verified LISP on
x86, ARM, PowerPC

Operating System (Linux, Mac, ...)

Hardware (Intel, AMD, ...)

Is the logic sound?
Is the program faithful to it?

The program is short
“Social proof”

Fundamentally have to trust
Use multiple systems, at least
Well, no.

**Bootstrapping Proofs**

½ billion unique conses
16 hours on CCL
8 GB on disk

**verified LISP on**
x86, ARM, PowerPC

Interpreted, slow
32-bit, memory limited

Magnus set out to develop **Jitawa**, a new Lisp runtime for Milawa.
What does Milawa need?

**Theorem Prover**

First-order, recursive functions  
Naturals, symbols, conses  

12 Primitive Functions  
\texttt{cons car cdr consp}  
\texttt{+ - < natp}  
\texttt{symbolp symbol-<}  
\texttt{if equal}  

11 Macros  
\texttt{and or list cond}  
\texttt{let let*}  
\texttt{first ... fifth}  

**Kernel**

Destructive updates  
Hash tables  
File reading  
Timing, status messages  
Checkpointing  
Function compilation  
Dynamic function calls  
Runtime errors  

**I/O Requirements**

½ billion unique conses  
8 GB on disk  
Abbreviations are critical
What does Milawa really need?

**Theorem Prover**
- First-order, recursive functions
- Naturals, symbols, conses

**Kernel**
- Destructive updates
- Hash tables
- File reading
- Timing, status messages
- Checkpointing
- Function compilation
- Dynamic function calls
- Runtime errors

**I/O Requirements**
- ½ billion unique conses
- 8 GB on disk
- 4 GB input file

**Abbreviations are critical**

**12 Primitive Functions**
- cons car cdr consp
- +    -   <   natp
- symbolp symbol-<
- if    equal

**11 Macros**
- and or list cond
- let let*
- first ... fifth
Jitawa – A Scaled Up, Verified Lisp

**Verified Core**
- 7500 lines of verified x86 machine code
- Just-in-time (JIT) compiler to 64-bit x86
- Copying garbage collector
- Up to $2^{31}$ conses (16 GB), big stacks
- Efficient parser with #1=... abbreviations
- Always exits gracefully
- Calls C routines for I/O

**Unverified C Wrapper**
- 200 lines (with #if debug)
- Parse command line
- Allocate memory
- Initialize IO function pointers
- Invoke verified core
  - read_line
  - print_string
  - report_error

Far simpler than a full Common Lisp implementation
Jitawa – A Scaled Up, Verified Lisp

Verified Core
- 7500 lines of verified x86 machine code
- Just-in-time (JIT) compiler to 64-bit x86
- Copying garbage collector
- Up to 2^31 conses (16 GB), big stacks
- Efficient parser, with #1= style abbreviations
- Graceful exit in all circumstances
- Calls C routines for I/O

Unverified C Wrapper
- 200 lines (with #if debug)
- Parses command line
- Allocates memory
- Initializes IO function pointers
- Invokes verified core

Examples:
- `> '3`
  - `3`
- `> (cons '5 '(6 7))`
  - `(5 6 7)`
- `> (define 'increment '(n) '(+ n '1))`
  - `NIL`
- `> (increment '5)`
  - `6`

Implements an ordinary read-eval-print loop!
How is it Verified?

**Jitawa Specification**
- 400 lines of HOL
- Parsing, Evaluation, Printing

**X86-64 Model**
- (in HOL)
- Not a full x86 model
- Just the relevant instructions

**Compiler, GC, Parsing, Printing**
- Algorithms Defined in HOL

**Jitawa Implementation**
- X86-64 Machine Code

HOL Proof Effort

Proof producing synthesis

Testing
Jitawa Specification (400 lines of HOL)

Current State

Out

Defs

In

Defs

“>”

(g x) = x
(h x) = (+ x 1)

“(print 3) (g 5) ...”

EV

Out

“(>3)”

ans

(Sym “nil”)

Next State

Out''

Def''s

In'

“>3 nil”

(g x) = x
(h x) = (+ x 1)

“(g 5) ...”

App (Fun “print”)
(Const (Val 3))

sexp

Dot (Sym “print”) (Val 3)

rest

“(g 5) ...”
An Improved Soundness Story

- Theorem Proving Tactics
  - Proven correct by the kernel

- Kernel
  - Proof Checker
  - Is the logic sound?
  - Is the program faithful to it?
  - The program is short
    - "Social proof"

- Jitawa
  - Verified Core
    - Proven correct down to the Machine Code in HOL4

- Operating System (Linux, Mac, ...)
  - Practically have to trust
    - Use multiple systems, at least

- Hardware (Intel, AMD, ...)
  - Fundamentally have to trust
    - Use multiple systems, at least
Lifting

Simplified Kernel
1700 Lines of Jitawa Lisp

Jitawa Specification
400 lines of HOL
Parsing, Evaluation, Printing
Lifting

Simplified Kernel
1700 Lines of Jitawa Lisp

Jitawa Specification
400 lines of HOL
Parsing, Evaluation, Printing

Automated (HOL4 tactics)

Nice HOL4 Model of the Milawa Kernel
Faithfulness

Nice HOL4 Model of the Milawa Kernel

Milawa Logic Formalized in HOL

- Syntax
- Axioms
- Inference Rules
- Ext. Principles
Soundness

Milawa Logic
Formalized in HOL

Syntax
Axioms
Inference Rules
Ext. Principles

Milawa Logic
Semantics

\( \forall \pi \ p. \ context\_ok \ \pi \land (\vdash_{\pi} \ p) \implies (\models_{\pi} \ p) \)

Syntactically provable

Semantically true
Putting it all Together

- Milawa Logic Semantics
- Milawa Logic Mechanics
- Nice HOL4 Model of the Milawa Kernel
- Jitawa Specification
- Simplified Kernel Lisp Code
- X86-64 Model
- Jitawa Implementation X86-64 Machine Code
\( \forall \text{input } p. \)

\[
\{ \text{init\_state (milawa\_implementation ++ "(milawa\_main 'input)") \* pc pc \} } \\
\text{pc} : \text{code\_for\_entire\_jitawa\_implementation} \\
\{ \text{error\_message V } \exists \text{output. } (\text{every\_line line\_ok output} \* \\
\text{final\_state (output ++ "SUCCESS") } \}
\]
The New Soundness Story

Theorem Proving Tactics

Kernel
  Proof Checker

Jitawa
  Verified Core

Operating System (Linux, Mac, ...)

Hardware (Intel, AMD, ...)

Proven correct by the kernel
Proven sound down to the Machine Code in HOL4

Fundamentally have to trust
Use multiple systems, at least

Practically have to trust
Use multiple systems, at least
Thanks!

Questions?