The reflective Milawa theorem prover is sound

down to the machine code that runs it

ITP'14, Vienna

Magnus O. Myreen — Computer Laboratory, University of Cambridge, UK
Jared Davis — Centaur Technology, Inc., Austin TX, USA

Soundness

Soundness

We design our provers to be sound.

Soundness

We design our provers to be sound.

We verify programs with them.

Soundness

We design our provers to be sound.

We verify programs with them.

Why not prove the theorem provers sound?

Soundness

We design our provers to be sound.

We verify programs with them.

Why not prove the theorem provers sound?

This talk: explains how soundness was proved
for the Milawa theorem prover.

Previous work

Milawa

A self-verifying theorem prover

Jared Davis — PhD work

Previous work

Milawa

A self-verifying theorem prover

Jared Davis — PhD work

Jitawa
verified
LISP

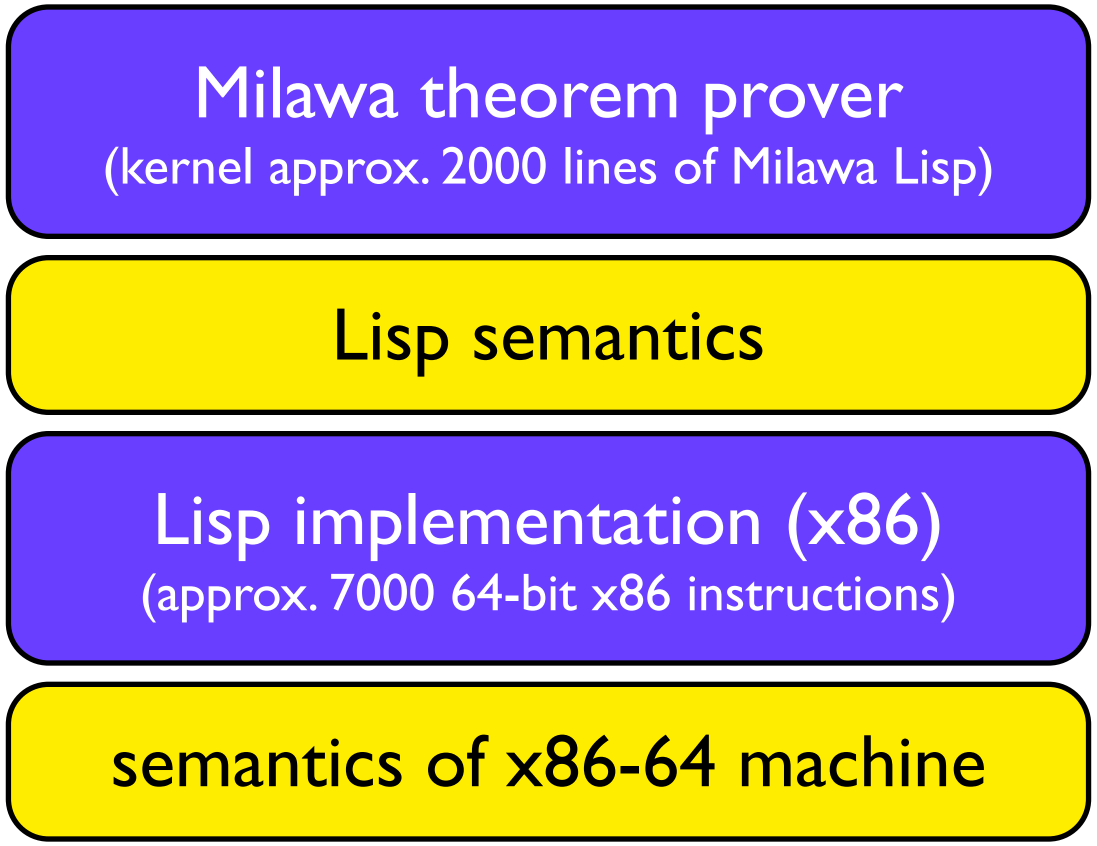
A verified runtime for a verified theorem prover

Magnus Myreen, Jared Davis — ITP'11

Proving Milawa sound

Milawa

Jitawa
verified
LISP



Milawa theorem prover
(kernel approx. 2000 lines of Milawa Lisp)

Lisp semantics

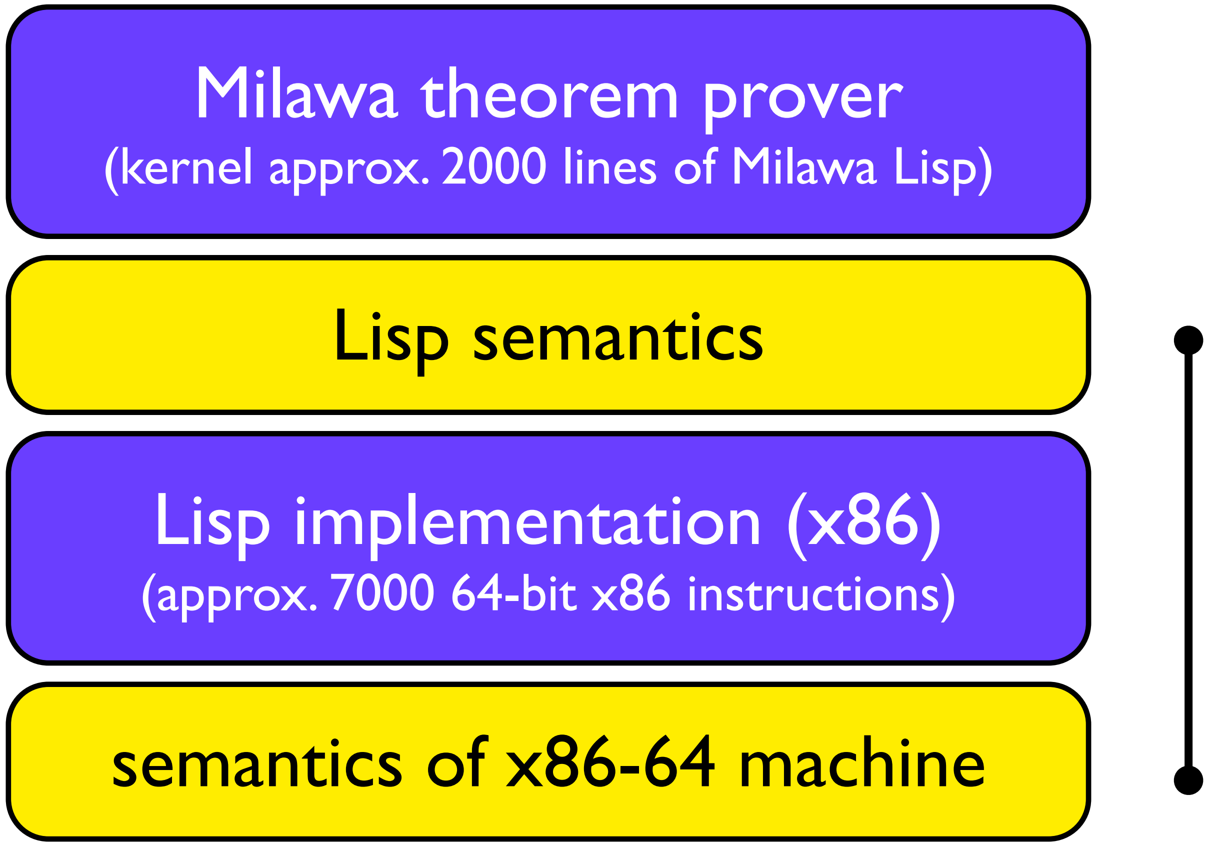
Lisp implementation (x86)
(approx. 7000 64-bit x86 instructions)

semantics of x86-64 machine

Proving Milawa sound

Milawa

Jitawa
verified
LISP



Milawa theorem prover
(kernel approx. 2000 lines of Milawa Lisp)

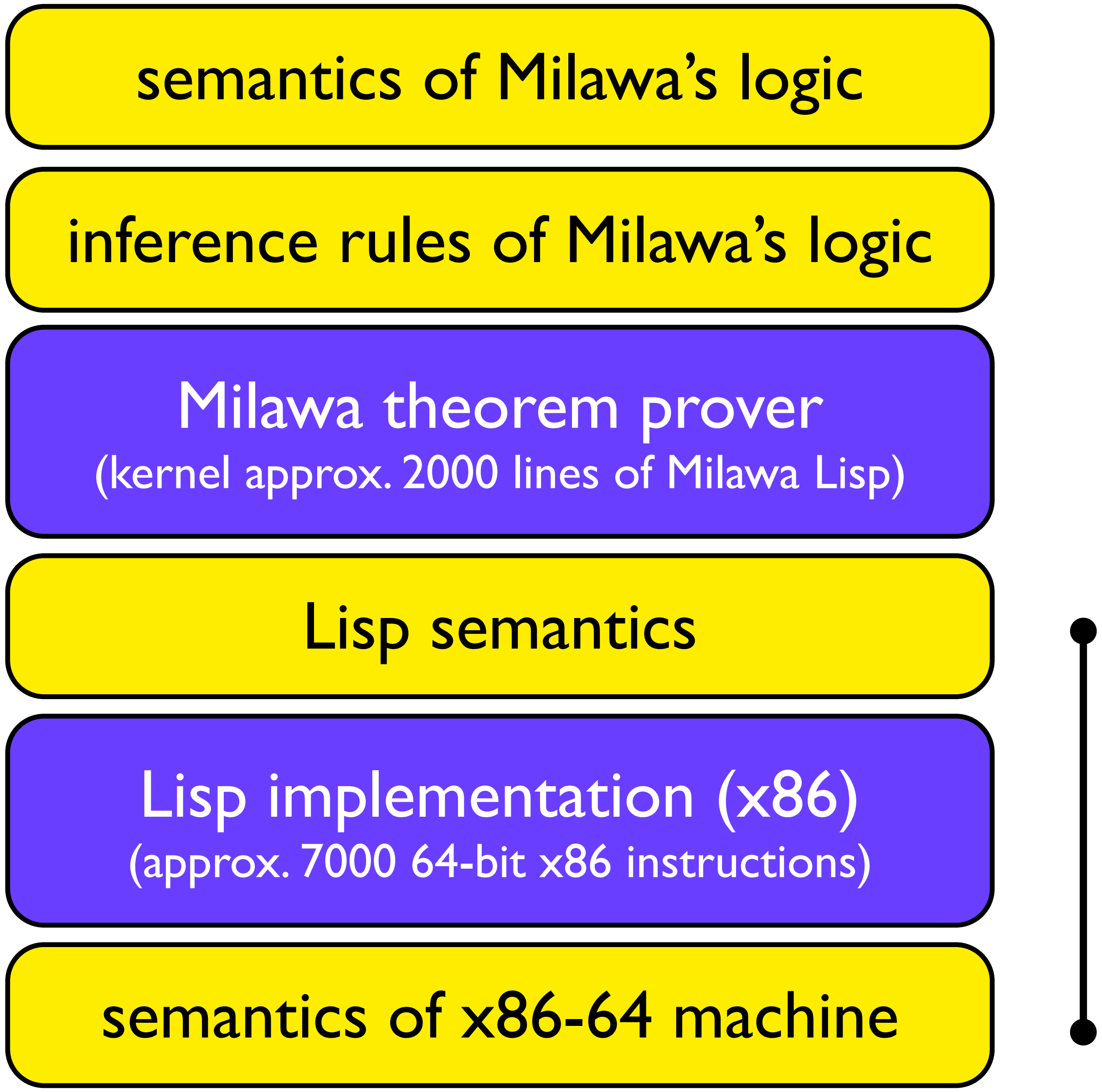
Lisp semantics

Lisp implementation (x86)
(approx. 7000 64-bit x86 instructions)

semantics of x86-64 machine

verification of a Lisp implementation [ITP'11]

Proving Milawa sound



semantics of Milawa's logic

inference rules of Milawa's logic

Milawa theorem prover
(kernel approx. 2000 lines of Milawa Lisp)

Lisp semantics

Lisp implementation (x86)
(approx. 7000 64-bit x86 instructions)

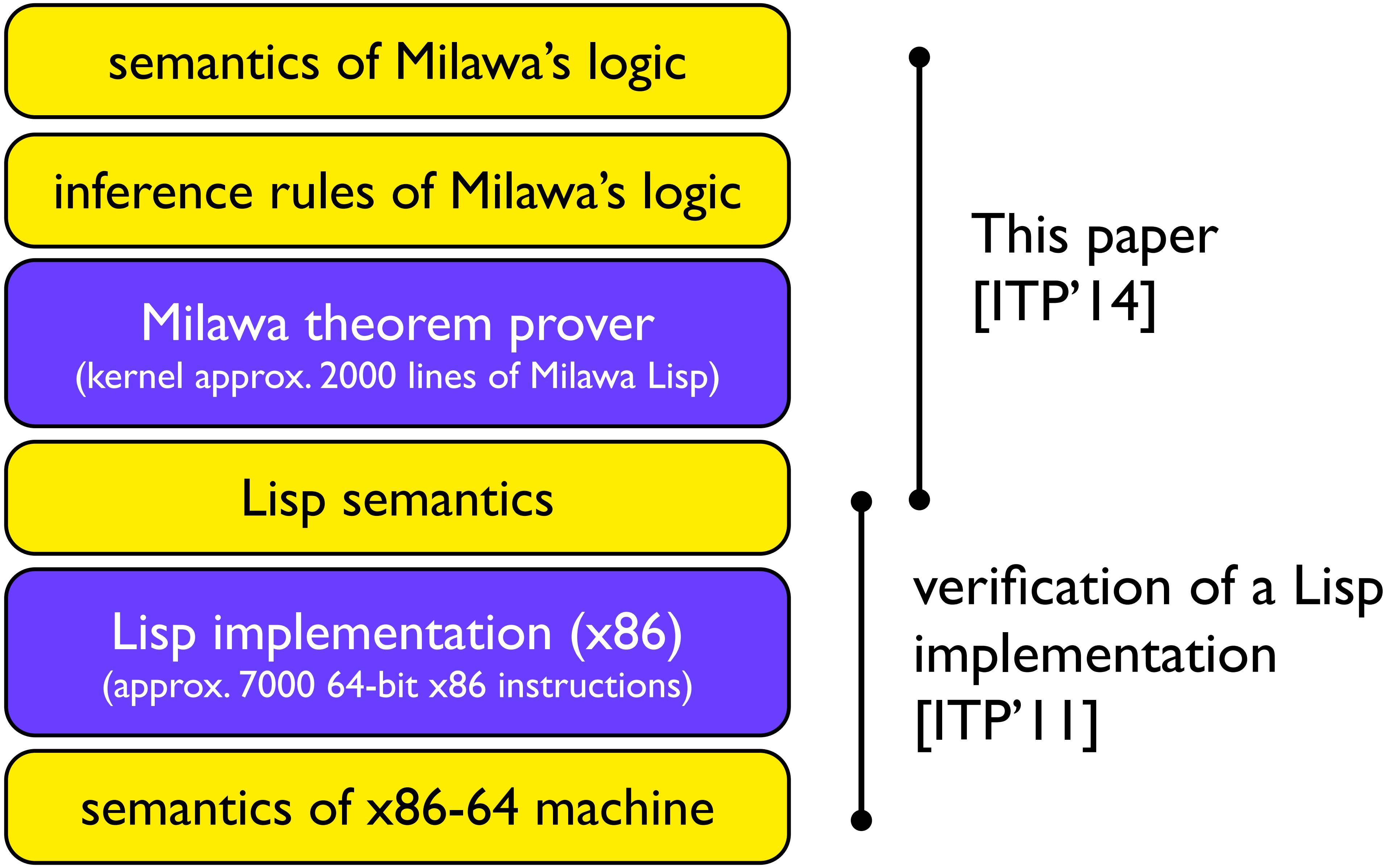
semantics of x86-64 machine

verification of a Lisp
implementation
[ITP'11]

Milawa

Jitawa
verified
LISP

Proving Milawa sound



semantics of Milawa's logic

inference rules of Milawa's logic

Milawa theorem prover
(kernel approx. 2000 lines of Milawa Lisp)

Lisp semantics

Lisp implementation (x86)
(approx. 7000 64-bit x86 instructions)

semantics of x86-64 machine

This paper
[ITP'14]

verification of a Lisp
implementation
[ITP'11]

Milawa

Jitawa
verified
LISP

work by Jared Davis

A very short introdution

Milawa

- Milawa is styled after theorem provers such as NQTHM and ACL2,
- has a small trusted logical kernel similar to LCF-style provers,
- ... but does not suffer the performance hit of LCF's fully expansive approach.

work by Jared Davis

Comparison with LCF approach

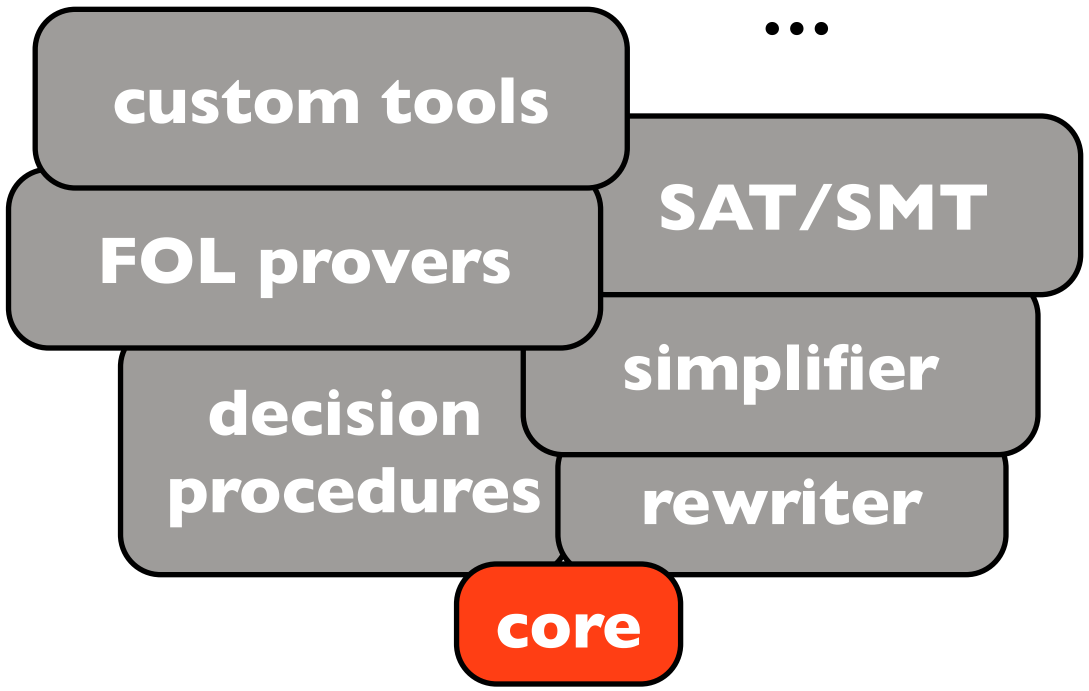
core

LCF-style approach

- all proofs pass through the core's primitive inferences
- extensions steer the core

work by Jared Davis

Comparison with LCF approach

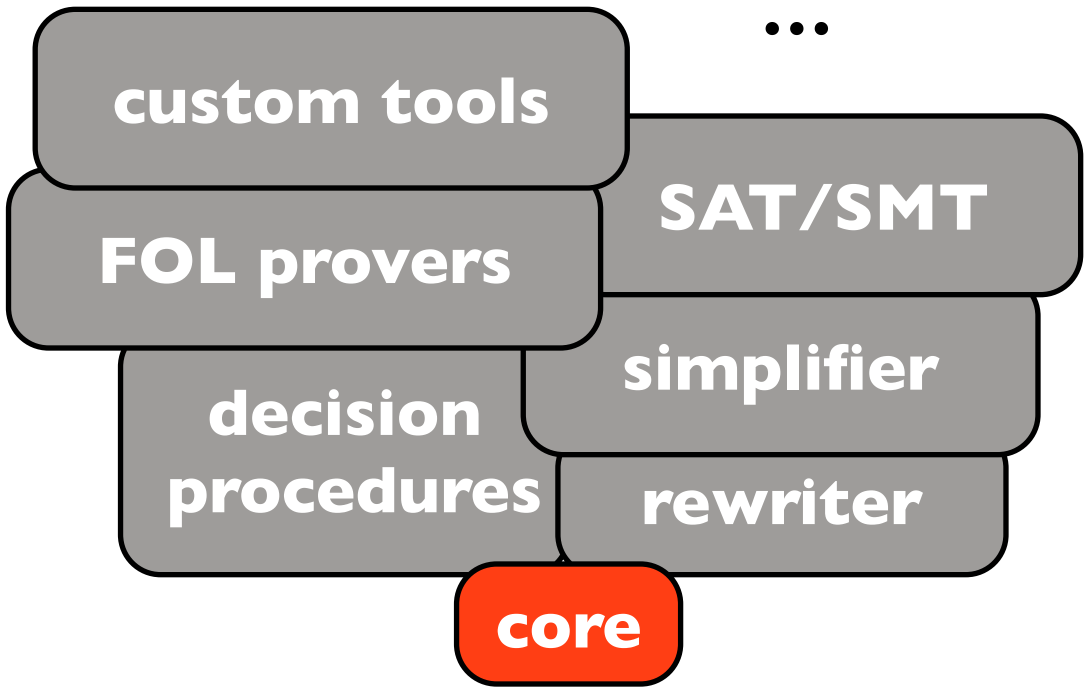


LCF-style approach

- all proofs pass through the core's primitive inferences
- extensions steer the core

work by Jared Davis

Comparison with LCF approach



LCF-style approach

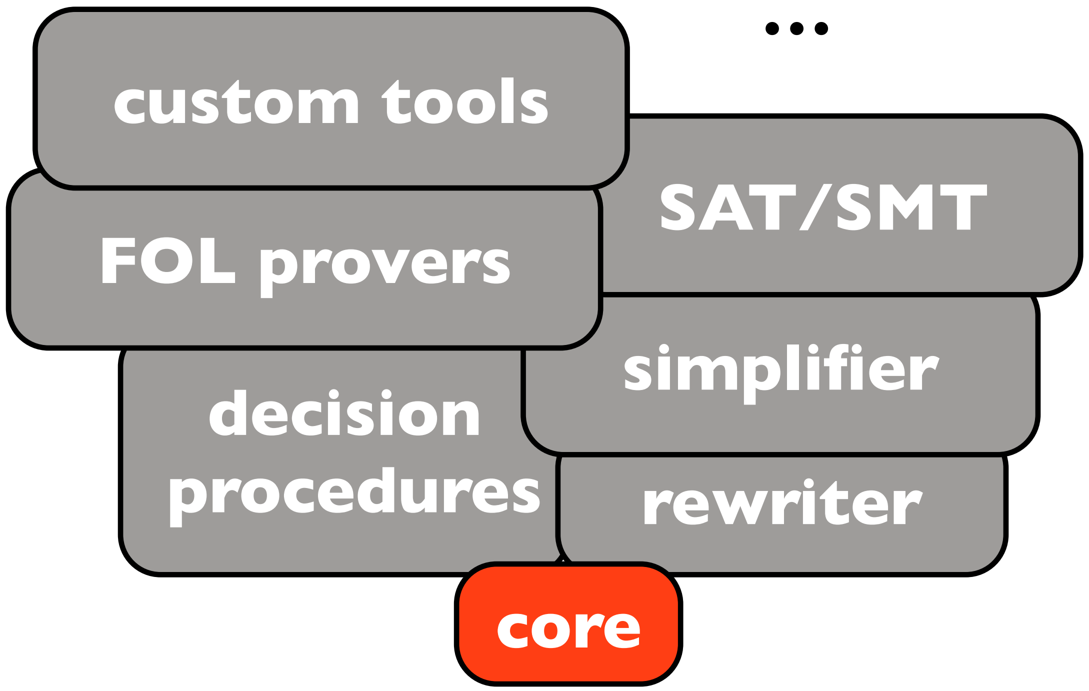
- all proofs pass through the core's primitive inferences
- extensions steer the core

the Milawa approach

- all proofs must pass the core
- the core proof checker can be replaced at runtime

work by Jared Davis

Comparison with LCF approach



LCF-style approach

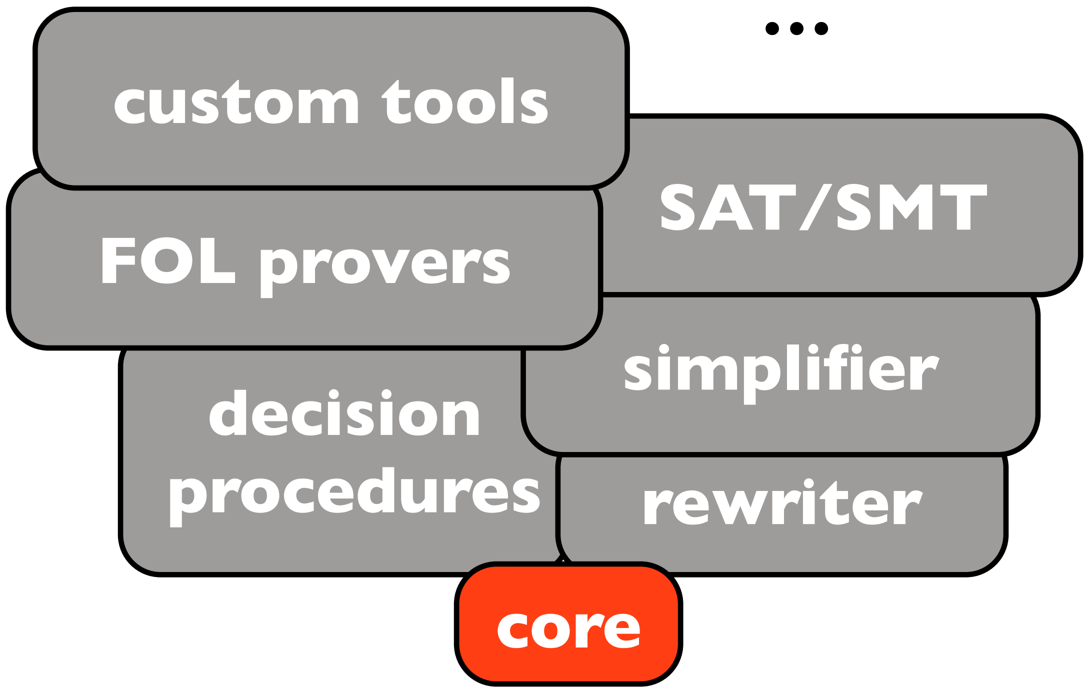
- all proofs pass through the core's primitive inferences
- extensions steer the core

the Milawa approach

- all proofs must pass the core
- the core proof checker can be replaced at runtime

work by Jared Davis

Comparison with LCF approach



…

custom tools

SAT/SMT

FOL provers

simplifier

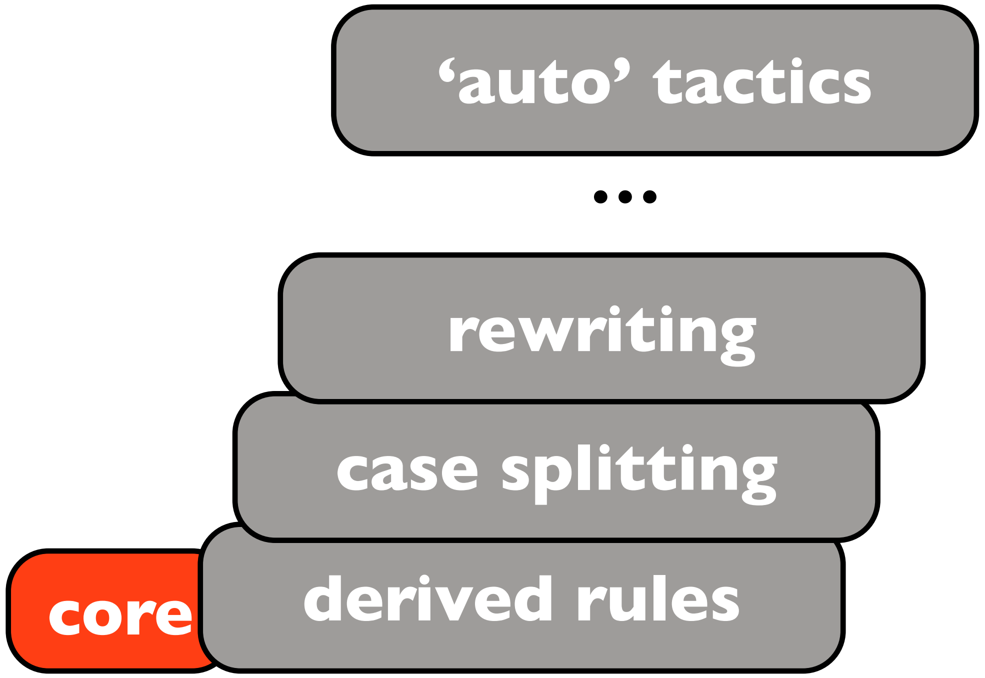
decision procedures

rewriter

core

LCF-style approach

- all proofs pass through the core's primitive inferences
- extensions steer the core



'auto' tactics

…

rewriting

case splitting

core

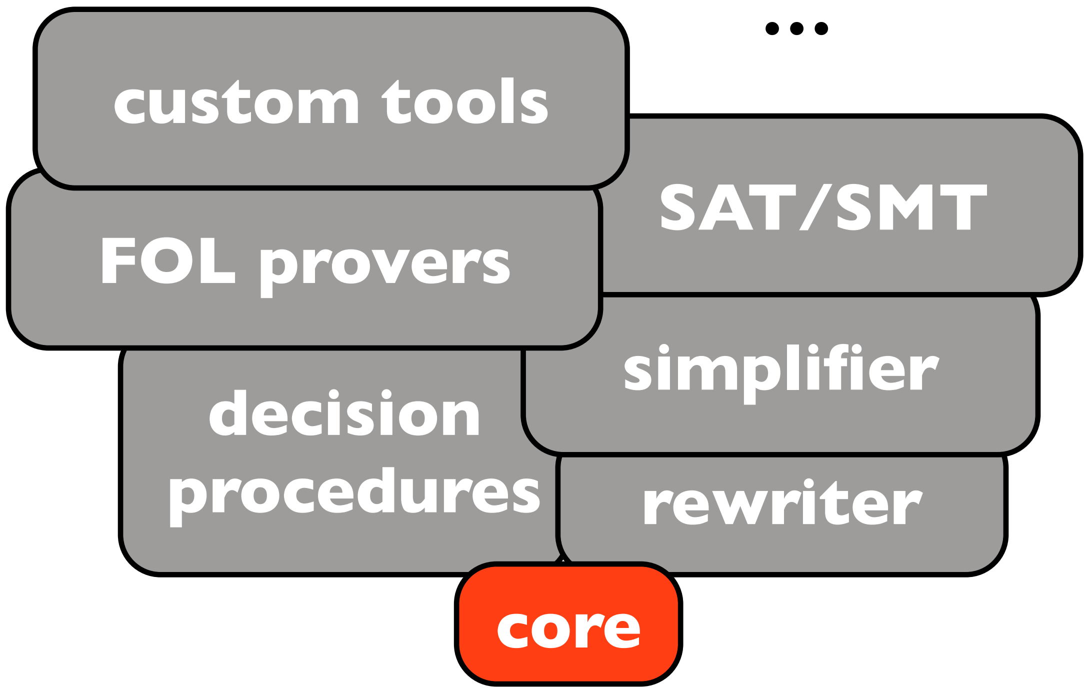
derived rules

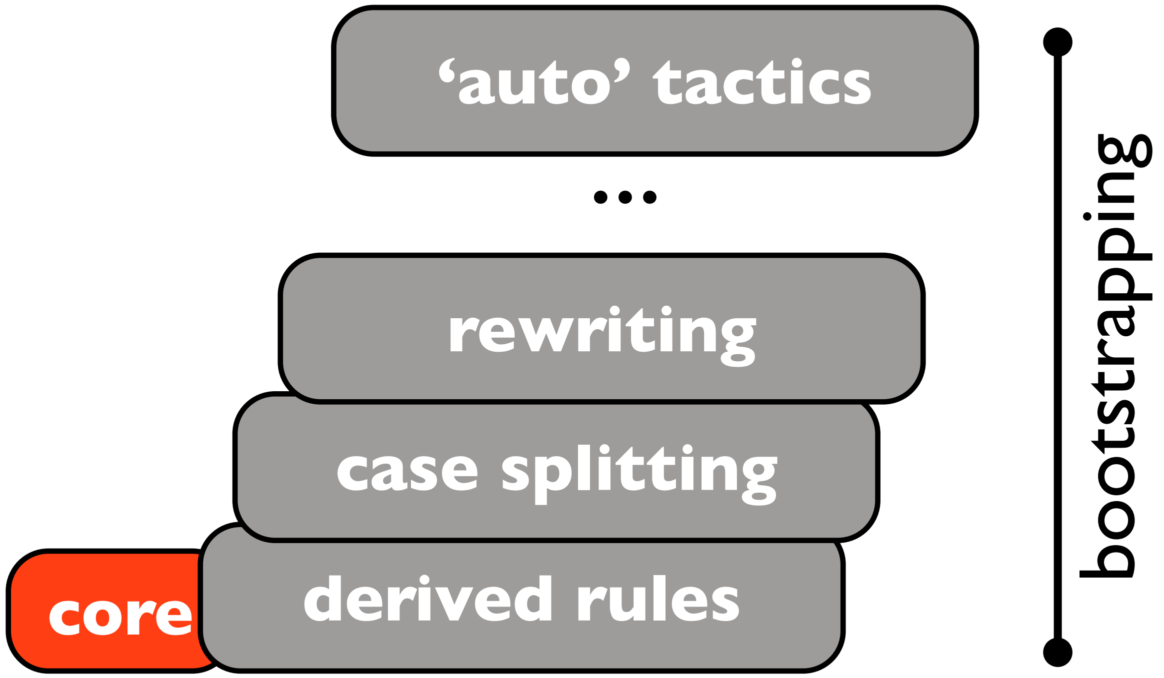
the Milawa approach

- all proofs must pass the core
- the core proof checker can be replaced at runtime

work by Jared Davis

Comparison with LCF approach





LCF-style approach

- all proofs pass through the core's primitive inferences
- extensions steer the core

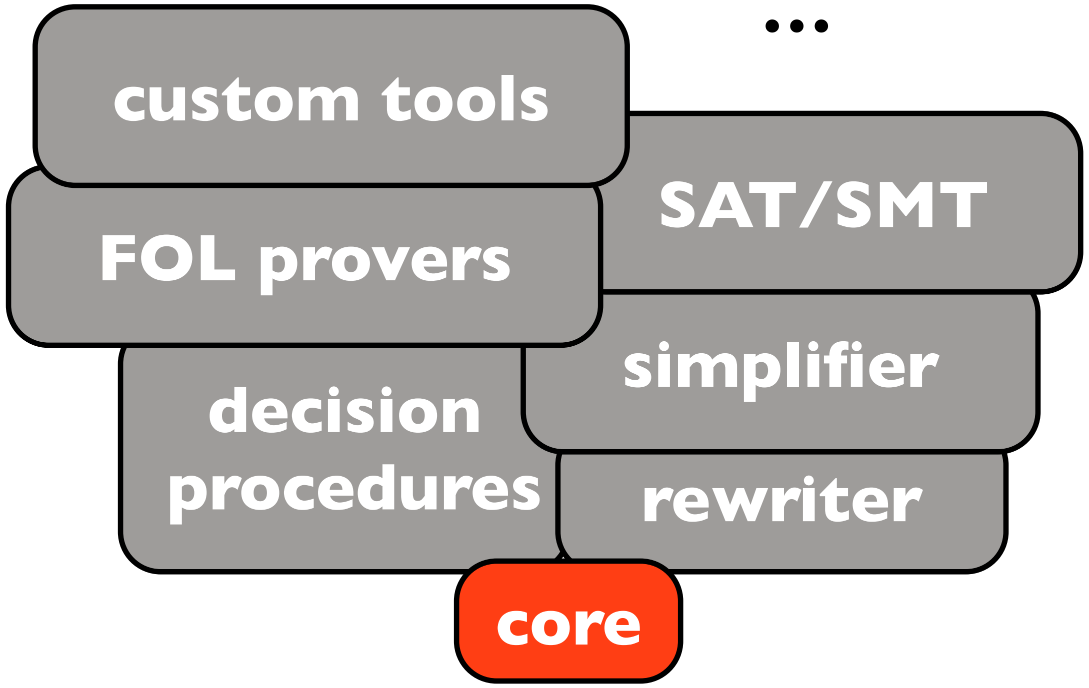
the Milawa approach

- all proofs must pass the core
- the core proof checker can be replaced at runtime

work by Jared Davis

Comparison with LCF approach

The Milawa theorem prover


...

custom tools

SAT/SMT

FOL provers

simplifier

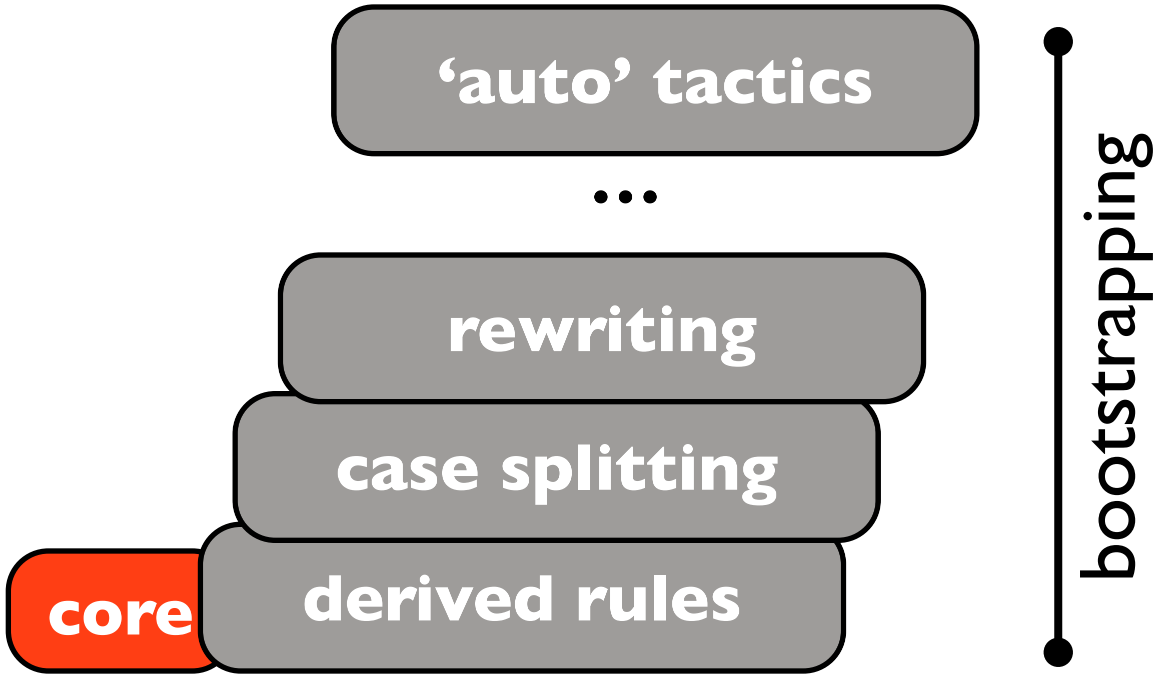
decision procedures

rewriter

core

LCF-style approach

- all proofs pass through the core's primitive inferences
- extensions steer the core


'auto' tactics

...

rewriting

case splitting

core

derived rules

bootstrapping

the Milawa approach

- all proofs must pass the core
- the core proof checker can be replaced at runtime

Steps

Steps

A. formalise Milawa’s logic

- syntax, semantics, inference, soundness

Steps

A. formalise Milawa’s logic

- syntax, semantics, inference, soundness

B. prove that Milawa's kernel is faithful to the logic

- run the Lisp parser (in the logic) on Milawa’s kernel
- translate (with proof) deep embedding into shallow
- prove that Milawa’s (reflective) kernel is faithful to logic

Steps

A. formalise Milawa’s logic

- syntax, semantics, inference, soundness

B. prove that Milawa's kernel is faithful to the logic

- run the Lisp parser (in the logic) on Milawa’s kernel
- translate (with proof) deep embedding into shallow
- prove that Milawa’s (reflective) kernel is faithful to logic

C. connect the verified Lisp implementation

- compose with the correctness thm from ITP’11

Steps

A. formalise Milawa’s logic

- syntax, semantics, inference, soundness

B. prove that Milawa's kernel is faithful to the logic

- run the Lisp parser (in the logic) on Milawa’s kernel
- translate (with proof) deep embedding into shallow
- prove that Milawa’s (reflective) kernel is faithful to logic

C. connect the verified Lisp implementation

- compose with the correctness thm from ITP’11

A—C combine to a top-level theorem that relates the logic’s semantics with the execution of the x86 machine code.

This talk

A. formalise Milawa’s logic

B. prove that Milawa's kernel is faithful to the logic

C. connect the verified Lisp implementation

This talk

A. formalise Milawa’s logic

B. prove that Milawa's kernel is faithful to the logic

C. connect the verified Lisp implementation

Syntax

sexp	$::=$	$\mathsf{Val}\ \mathit{num} \mid \mathsf{Sym}\ \mathit{string} \mid \mathsf{Dot}\ \mathit{sexp}\ \mathit{sexp}$	S-expression
prim	$::=$	$\mathsf{If} \mid \mathsf{Equal} \mid \mathsf{Not} \mid \mathsf{Symbolp} \mid \mathsf{Symbol\_less}$	
	$\mid$	$\mathsf{Natp} \mid \mathsf{Add} \mid \mathsf{Sub} \mid \mathsf{Less} \mid \mathsf{Consp} \mid \mathsf{Cons}$	
	$\mid$	$\mathsf{Car} \mid \mathsf{Cdr} \mid \mathsf{Rank} \mid \mathsf{Ord\_less} \mid \mathsf{Ordp}$	
func	$::=$	$\mathsf{PrimitiveFun}\ \mathit{prim}$	primitive functions
	$\mid$	$\mathsf{Fun}\ \mathit{string}$	user-defined
term	$::=$	$\mathsf{Const}\ \mathit{sexp}$	constant S-expression
	$\mid$	$\mathsf{Var}\ \mathit{string}$	variable
	$\mid$	$\mathsf{App}\ \mathit{func}\ (\mathit{term}\ \mathsf{list})$	function application
	$\mid$	$\mathsf{LamApp}\ (\mathit{string}\ \mathsf{list})\ \mathit{term}\ (\mathit{term}\ \mathsf{list})$	λ formals body actuals
$\mathit{formula}$	$::=$	$\neg \mathit{formula}$	negation
	$\mid$	$\mathit{formula} \lor \mathit{formula}$	disjunction
	$\mid$	$\mathit{term} = \mathit{term}$	term equality

Context

Syntax, semantics and inference rules depend on a context.

Context

Syntax, semantics and inference rules depend on a context.

A context is a finite partial map
from $string$ to $string$ list $\times$ $func\_body$ $\times$ $(sexp$ list $\rightarrow$ $sexp)$

Context

Syntax, semantics and inference rules depend on a context.

A context is a finite partial map
from string to string list $\times$ $\mathit{func\_body}$ $\times$ $(\mathit{sexp}$ list $\rightarrow$ $\mathit{sexp})$

parameters

Context

Syntax, semantics and inference rules depend on a context.

A context is a finite partial map
from $string$ to $string$ list $\times$ $func\_body$ $\times$ $(sexp$ list $\rightarrow$ $sexp)$

parameters

syntax of body

Context

Syntax, semantics and inference rules depend on a context.

A context is a finite partial map
from $string$ to $string$ list × $func\_body$ × ($sexp$ list $\rightarrow$ $sexp$)

parameters

syntax of body

$func\_body ::=$	Body $term$	concrete term (e.g. recursive function)
$\mid$	Witness $term$ $string$	property, element name
$\mid$	None	no function body given

Context

Syntax, semantics and inference rules depend on a context.

A context is a finite partial map
from $string$ to $\underbrace{string\ \text{list}}_{\text{parameters}} \times \underbrace{func\_body}_{\text{syntax of body}} \times \underbrace{(sexp\ \text{list} \to sexp)}_{\text{semantic interpretation}}$

$func\_body ::=$	Body $term$	concrete term (e.g. recursive function)
$\mid$	Witness $term$ $string$	property, element name
$\mid$	None	no function body given

Semantics

Semantics

$$(\models_\pi p) \quad = \quad \mathsf{formula\_ok}_\pi\ p\ \wedge\ \forall i.\ \mathsf{eval\_formula}\ i\ \pi\ p$$

Semantics

$$(\models_\pi p) \quad = \quad \mathsf{formula\_ok}_\pi\ p \ \wedge\ \forall i.\ \mathsf{eval\_formula}\ i\ \pi\ p$$

syntax makes sense

Semantics

$$(\models_\pi p) \quad = \quad \textsf{formula\_ok}_\pi\ p \ \wedge\ \forall i.\ \textsf{eval\_formula}\ i\ \pi\ p$$

syntax makes sense

truth value

Semantics

$$(\models_\pi p) \quad = \quad \text{formula\_ok}_\pi\ p\ \wedge\ \forall i.\ \text{eval\_formula}\ i\ \pi\ p$$

syntax makes sense

truth value

$$\begin{aligned}
\text{eval\_formula}\ i\ \pi\ (\neg p) &\quad = \quad \neg(\text{eval\_formula}\ i\ \pi\ p) \\
\text{eval\_formula}\ i\ \pi\ (p \vee q) &\quad = \quad \text{eval\_formula}\ i\ \pi\ p \vee \text{eval\_formula}\ i\ \pi\ q \\
\text{eval\_formula}\ i\ \pi\ (x \mathbin{\mathsf{=}} y) &\quad = \quad (\text{eval\_term}\ i\ \pi\ x = \text{eval\_term}\ i\ \pi\ y)
\end{aligned}$$

Semantics

$$(\models_\pi p) \quad = \quad \mathsf{formula\_ok}_\pi\ p \ \wedge\ \forall i.\ \mathsf{eval\_formula}\ i\ \pi\ p$$

syntax makes sense

truth value

$$\begin{array}{lcl}
\mathsf{eval\_formula}\ i\ \pi\ (\neg p) & = & \neg(\mathsf{eval\_formula}\ i\ \pi\ p) \\
\mathsf{eval\_formula}\ i\ \pi\ (p \vee q) & = & \mathsf{eval\_formula}\ i\ \pi\ p \vee \mathsf{eval\_formula}\ i\ \pi\ q \\
\mathsf{eval\_formula}\ i\ \pi\ (x \mathbin{\mathsf{=}} y) & = & (\mathsf{eval\_term}\ i\ \pi\ x = \mathsf{eval\_term}\ i\ \pi\ y)
\end{array}$$

$$\begin{array}{lcl}
\mathsf{eval\_term}\ i\ \pi\ (\mathsf{Const}\ c) & = & c \\
\mathsf{eval\_term}\ i\ \pi\ (\mathsf{Var}\ v) & = & i(v) \\
\mathsf{eval\_term}\ i\ \pi\ (\mathsf{App}\ f\ xs) & = & \mathsf{eval\_app}\ (f, \mathsf{map}\ (\mathsf{eval\_term}\ i\ \pi)\ xs, \pi) \\
\mathsf{eval\_term}\ i\ \pi\ (\mathsf{LambdaApp}\ vs\ x\ xs) & = & \mathsf{let}\ ys = \mathsf{map}\ (\mathsf{eval\_term}\ i\ \pi)\ xs\ \mathsf{in} \\
 & & \quad \mathsf{eval\_term}\ [vs \mapsto ys]\ \pi\ x
\end{array}$$

Semantics

$$(\models_\pi p) \quad = \quad \text{formula\_ok}_\pi\ p\ \wedge\ \forall i.\ \text{eval\_formula}\ i\ \pi\ p$$

syntax makes sense

truth value

$$\begin{aligned}
\text{eval\_formula}\ i\ \pi\ (\neg p) &= \neg(\text{eval\_formula}\ i\ \pi\ p)\\
\text{eval\_formula}\ i\ \pi\ (p \vee q) &= \text{eval\_formula}\ i\ \pi\ p \vee \text{eval\_formula}\ i\ \pi\ q\\
\text{eval\_formula}\ i\ \pi\ (x \text{ = } y) &= (\text{eval\_term}\ i\ \pi\ x = \text{eval\_term}\ i\ \pi\ y)
\end{aligned}$$

$$\begin{aligned}
\text{eval\_term}\ i\ \pi\ (\text{Const}\ c) &= c\\
\text{eval\_term}\ i\ \pi\ (\text{Var}\ v) &= i(v)\\
\text{eval\_term}\ i\ \pi\ (\text{App}\ f\ xs) &= \text{eval\_app}\ (f, \text{map}\ (\text{eval\_term}\ i\ \pi)\ xs, \pi)\\
\text{eval\_term}\ i\ \pi\ (\text{LambdaApp}\ vs\ x\ xs) &= \text{let}\ ys = \text{map}\ (\text{eval\_term}\ i\ \pi)\ xs\ \text{in}\\
&\qquad \text{eval\_term}\ [vs \mapsto ys]\ \pi\ x
\end{aligned}$$

$$\begin{aligned}
\text{eval\_app}\ (\text{PrimitiveFun}\ p, args, \pi) &= \text{eval\_primitive}\ p\ args\\
\text{eval\_app}\ (\text{Fun}\ name, args, \pi) &= \text{let}\ (\_, \_, interp) = \pi(name)\ \text{in}\\
&\qquad interp(args)
\end{aligned}$$

Semantics

$$(\models_\pi p) \quad = \quad \underbrace{\mathsf{formula\_ok}_\pi\ p}_{\text{syntax makes sense}} \ \wedge\ \underbrace{\forall i.\ \mathsf{eval\_formula}\ i\ \pi\ p}_{\text{truth value}}$$

$$\begin{aligned}
\mathsf{eval\_formula}\ i\ \pi\ (\neg p) &\quad = \quad \neg(\mathsf{eval\_formula}\ i\ \pi\ p)\\
\mathsf{eval\_formula}\ i\ \pi\ (p \vee q) &\quad = \quad \mathsf{eval\_formula}\ i\ \pi\ p \vee \mathsf{eval\_formula}\ i\ \pi\ q\\
\mathsf{eval\_formula}\ i\ \pi\ (x\ \mathsf{=}\ y) &\quad = \quad (\mathsf{eval\_term}\ i\ \pi\ x = \mathsf{eval\_term}\ i\ \pi\ y)
\end{aligned}$$

$$\begin{aligned}
\mathsf{eval\_term}\ i\ \pi\ (\mathsf{Const}\ c) &\quad = \quad c\\
\mathsf{eval\_term}\ i\ \pi\ (\mathsf{Var}\ v) &\quad = \quad i(v)\\
\mathsf{eval\_term}\ i\ \pi\ (\mathsf{App}\ f\ xs) &\quad = \quad \mathsf{eval\_app}\ (f, \mathsf{map}\ (\mathsf{eval\_term}\ i\ \pi)\ xs, \pi)\\
\mathsf{eval\_term}\ i\ \pi\ (\mathsf{LambdaApp}\ vs\ x\ xs) &\quad = \quad \mathsf{let}\ ys = \mathsf{map}\ (\mathsf{eval\_term}\ i\ \pi)\ xs\ \mathsf{in}\\
&\qquad\qquad \mathsf{eval\_term}\ [vs \mapsto ys]\ \pi\ x
\end{aligned}$$

$$\begin{aligned}
\mathsf{eval\_app}\ (\mathsf{PrimitiveFun}\ p, args, \pi) &\quad = \quad \mathsf{eval\_primitive}\ p\ args\\
\mathsf{eval\_app}\ (\mathsf{Fun}\ name, args, \pi) &\quad = \quad \mathsf{let}\ (\_, \_, interp) = \pi(name)\ \mathsf{in}\\
&\qquad\qquad interp(args)
\end{aligned}$$

$$\begin{aligned}
\mathsf{eval\_primitive}\ \mathsf{Add}\ [\mathsf{Val}\ 2, \mathsf{Val}\ 3] &= \mathsf{Val}\ 5\\
\mathsf{eval\_primitive}\ \mathsf{Add}\ [\mathsf{Val}\ 2, \mathsf{Sym}\ \texttt{"a"}] &= \mathsf{Val}\ 2\\
\mathsf{eval\_primitive}\ \mathsf{Cons}\ [\mathsf{Val}\ 2, \mathsf{Sym}\ \texttt{"a"}] &= \mathsf{Dot}\ (\mathsf{Val}\ 2)\ (\mathsf{Sym}\ \texttt{"a"})
\end{aligned}$$

Well-formedness of context

Semantics only makes sense for well-formed contexts.

Well-formedness of context

Semantics only makes sense for well-formed contexts.

For every entry,

$$\pi(name) \;=\; (formals, \mathsf{Body}\ body, interp)$$

it must be that:

- the formals are all distinct
- the body is well-formed w.r.t. the context
- the interpretation satisfies the defining equation:

$$\forall i.\;\; interp(\mathsf{map}\ i\ formals) \;=\; \mathsf{eval\_term}\ i\ \pi\ body$$

Well-formedness of context

Semantics only makes sense for well-formed contexts.

For every entry,

$$\pi(name) \;=\; (formals, \textsf{Body}\ body, interp)$$

it must be that:

- the formals are all distinct
- the body is well-formed w.r.t. the context
- the interpretation satisfies the defining equation:

$$\forall i.\;\; interp(\textsf{map}\ i\ formals) \;=\; \textsf{eval\_term}\ i\ \pi\ body$$

Similarly for the other function types.

(a few of the) Inference rules

$$\frac{\vdash_\pi a \vee (b \vee c)}{\vdash_\pi (a \vee b) \vee c} \text{ (associativity)}$$

$$\frac{a \in \textsf{milawa\_axioms}}{\vdash_\pi a} \text{ (basic axiom)}$$

$$\frac{\pi(name) = (formals, \textsf{Body}\ body, interp)}{\vdash_\pi \textsf{App (Fun}\ name\textsf{) (map Var}\ formals\textsf{) =}\ body}$$

(a few of the) Inference rules

$$\frac{\vdash_\pi a \vee (b \vee c)}{\vdash_\pi (a \vee b) \vee c}\ \text{(associativity)}$$

facts about Lisp primitives

$$\frac{a \in \mathsf{milawa\_axioms}}{\vdash_\pi a}\ \text{(basic axiom)}$$

$$\frac{\pi(name) = (formals, \mathsf{Body}\ body, interp)}{\vdash_\pi \mathsf{App}\ (\mathsf{Fun}\ name)\ (\mathsf{map\ Var}\ formals)\ \mathsf{=}\ body}$$

(a few of the) Inference rules

$$\frac{\vdash_\pi a \lor (b \lor c)}{\vdash_\pi (a \lor b) \lor c}\ \text{(associativity)}$$

facts about Lisp primitives

$$\frac{a \in \textsf{milawa\_axioms}}{\vdash_\pi a}\ \text{(basic axiom)}$$

function definition in context

$$\frac{\pi(name) = (formals, \textsf{Body}\ body, interp)}{\vdash_\pi \textsf{App}\ (\textsf{Fun}\ name)\ (\textsf{map Var}\ formals)\ \textsf{=}\ body}$$

(a few of the) Inference rules

$$\frac{\vdash_\pi a \lor (b \lor c)}{\vdash_\pi (a \lor b) \lor c}\ \text{(associativity)}$$

facts about Lisp primitives

$$\frac{a \in \mathsf{milawa\_axioms}}{\vdash_\pi a}\ \text{(basic axiom)}$$

function definition in context

body of function

$$\frac{\pi(name) = (formals, \mathsf{Body}\ body, interp)}{\vdash_\pi \mathsf{App}\ (\mathsf{Fun}\ name)\ (\mathsf{map}\ \mathsf{Var}\ formals) \mathrel{\mathsf{=}} body}$$

(a few of the) Inference rules

$$\frac{\vdash_\pi a \vee (b \vee c)}{\vdash_\pi (a \vee b) \vee c}\ \text{(associativity)}$$

facts about Lisp primitives

$$\frac{a \in \mathsf{milawa\_axioms}}{\vdash_\pi a}\ \text{(basic axiom)}$$

function definition in context

body of function

$$\frac{\pi(name) = (formals, \mathsf{Body}\ body, interp)}{\vdash_\pi \mathsf{App}\ (\mathsf{Fun}\ name)\ (\mathsf{map}\ \mathsf{Var}\ formals)\ \mathsf{=}\ body}$$

defining equation

Soundness of logic

Soundness of inference rules:

$$\forall \pi\ p.\ \ \mathsf{context\_ok}\ \pi \wedge (\vdash_\pi p) \Longrightarrow (\models_\pi p)$$

Soundness of logic

Soundness of inference rules:

$$\forall \pi\ p.\quad \mathsf{context\_ok}\ \pi \wedge (\vdash_\pi p) \implies (\models_\pi p)$$

- induction rule most interesting, Kaufmann&Slind [TPHOLs'07]

Soundness of logic

Soundness of inference rules:

$$\forall \pi\ p.\ \ \mathsf{context\_ok}\ \pi \wedge (\vdash_\pi p) \Longrightarrow (\models_\pi p)$$

- induction rule most interesting, Kaufmann&Slind [TPHOLs'07]

Soundness of definition mechanism:

$$\begin{aligned}&\forall \pi\ \mathit{name}\ \mathit{formals}\ \mathit{body}.\\ &\quad \mathsf{context\_ok}\ \pi \wedge \mathsf{definition\_ok}\ (\mathit{name}, \mathit{formals}, \mathit{body}, \pi)\ \Longrightarrow\\ &\quad \mathsf{context\_ok}\ (\pi[\mathit{name} \mapsto (\mathit{formals}, \mathit{body}, \mathsf{new\_interp}\ \pi\ \mathit{name}\ \mathit{formals}\ \mathit{body})])\end{aligned}$$

Soundness of logic

Soundness of inference rules:

$$\forall \pi\ p.\quad \textsf{context\_ok}\ \pi \wedge (\vdash_\pi p) \implies (\models_\pi p)$$

- induction rule most interesting, Kaufmann&Slind [TPHOLs'07]

Soundness of definition mechanism:

$$\begin{array}{l} \forall \pi\ \mathit{name}\ \mathit{formals}\ \mathit{body}. \\ \quad \textsf{context\_ok}\ \pi \wedge \textsf{definition\_ok}\ (\mathit{name}, \mathit{formals}, \mathit{body}, \pi)\ \implies \\ \quad \textsf{context\_ok}\ (\pi[\mathit{name} \mapsto (\mathit{formals}, \mathit{body}, \textsf{new\_interp}\ \pi\ \mathit{name}\ \mathit{formals}\ \mathit{body})]) \end{array}$$

- req. proving that termination conditions imply that a semantic interpretation exists as a function in HOL

This talk

A. formalise Milawa’s logic

B. prove that Milawa's kernel is faithful to the logic

C. connect the verified Lisp implementation

This talk

A. formalise Milawa’s logic

B. prove that Milawa's kernel is faithful to the logic

C. connect the verified Lisp implementation

This talk

A. formalise Milawa’s logic

B. prove that Milawa's kernel is faithful to the logic

- run the Lisp parser (in the logic) on Milawa’s kernel
- translate (with proof) deep embedding into shallow
- prove that Milawa’s (reflective) kernel is faithful to logic

C. connect the verified Lisp implementation

Proving Milawa faithful to its logic

Verification must be w.r.t. semantics of Lisp [ITP’11].

Proving Milawa faithful to its logic

Verification must be w.r.t. semantics of Lisp [ITP'11].

Semantics of Lisp's read-eval-print loop:

Proving Milawa faithful to its logic

Verification must be w.r.t. semantics of Lisp [ITP'11].

Semantics of Lisp's read-eval-print loop:

1. parse ASCII characters into s-expressions
2. translate s-expressions into program AST
3. evaluate program AST
4. print results, goto **1.**

Proving Milawa faithful to its logic

Verification must be w.r.t. semantics of Lisp [ITP'11].

Semantics of Lisp's read-eval-print loop:

1. parse ASCII characters into s-expressions
2. translate s-expressions into program AST
3. evaluate program AST
4. print results, goto **1.**

Need to verify program down to concrete source code.

Steps towards an easier verification

- run the Lisp parser (in the logic) on Milawa’s kernel

Steps towards an easier verification

- run the Lisp parser (in the logic) on Milawa's kernel

Each top-level function definition in ASCII

```
(defun lookup-safe (a x)
  (if (consp x)
      (if (equal a (car (car x)))
          (if (consp (car x))
              (car x)
            (cons (car (car x)) (cdr (car x))))
        (lookup-safe a (cdr x)))
    nil))
```

Steps towards an easier verification

- run the Lisp parser (in the logic) on Milawa's kernel

Each top-level function definition in ASCII

```
(defun lookup-safe (a x)
  (if (consp x)
      (if (equal a (car (car x)))
          (if (consp (car x))
              (car x)
            (cons (car (car x)) (cdr (car x))))
        (lookup-safe a (cdr x)))
    nil))
```

becomes a program AST

App Define [Const (Sym `"LOOKUP-SAFE"`), Const (...), Const (...)]

Steps towards an easier verification

$\text{App Define}\ [\text{Const}\ (\text{Sym}\ \texttt{"LOOKUP-SAFE"}), \text{Const}\ (...), \text{Const}\ (...)]$

Steps towards an easier verification

When

$$\text{App Define } [\text{Const } (\text{Sym } \texttt{"LOOKUP-SAFE"}), \text{Const } (\ldots), \text{Const } (\ldots)]$$

is evaluated, the op. sem. adds a definition to its context:

Steps towards an easier verification

When

```
App Define [Const (Sym "LOOKUP-SAFE"), Const (...), Const (...)]
```

is evaluated, the op. sem. adds a definition to its context:

```
function name:   "LOOKUP-SAFE"
parameter list:  "A", "X"
function body:   If (App (PrimitiveFun Consp) [Var "X"])
                   (If (App (PrimitiveFun Equal) [...])
                      (If (App (PrimitiveFun Consp) [...] (...) (...))
                      (App (Fun "LOOKUP-SAFE") [...]))
                   (Const (Sym "NIL"))
```

Steps towards an easier verification

When

App Define [Const (Sym "LOOKUP-SAFE"), Const (...), Const (...)]

is evaluated, the op. sem. adds a definition to its context:

```
function name:   "LOOKUP-SAFE"
parameter list:  "A", "X"
function body:   If (App (PrimitiveFun Consp) [Var "X"])
                    (If (App (PrimitiveFun Equal) [...])
                        (If (App (PrimitiveFun Consp) [...] (...) (...))
                        (App (Fun "LOOKUP-SAFE") [...]))
                    (Const (Sym "NIL"))
```

We could do verification over this deep embedding.

Steps towards an easier verification

When

```
App Define [Const (Sym "LOOKUP-SAFE"), Const (...), Const (...)]
```

is evaluated, the op. sem. adds a definition to its context:

```
function name:   "LOOKUP-SAFE"
parameter list:  "A", "X"
function body:   If (App (PrimitiveFun Consp) [Var "X"])
                    (If (App (PrimitiveFun Equal) [...])
                        (If (App (PrimitiveFun Consp) [...] (...) (...))
                        (App (Fun "LOOKUP-SAFE") [...]))
                    (Const (Sym "NIL"))
```

We could do verification over this deep embedding.

…but a shallow embedding is easier to work with.

Steps towards an easier verification

```
function name:   "LOOKUP-SAFE"
parameter list:  "A", "X"
function body:   If (App (PrimitiveFun Consp) [Var "X"])
                   (If (App (PrimitiveFun Equal) [...])
                       (If (App (PrimitiveFun Consp) [...] (...) (...))
                       (App (Fun "LOOKUP-SAFE") [...]))
                   (Const (Sym "NIL"))
```

We translate deep embedding into convenient shallow emb.
[ITP'12]

```
lookup_safe a x  =  if consp x then
                      if a = car (car x) then
                        if consp (car x) then
                          car x
                        else cons (car (car x)) (cdr (car x))
                      else lookup_safe a (cdr x)
                    else Sym "NIL"
```

Steps towards an easier verification

We translate deep embedding into convenient shallow emb. [ITP'12]

```
lookup_safe $a$ $x$  =  if consp $x$ then
                          if $a$ = car (car $x$) then
                            if consp (car $x$) then
                              car $x$
                            else cons (car (car $x$)) (cdr (car $x$))
                          else lookup_safe $a$ (cdr $x$)
                        else Sym "NIL"
```

and produce a certificate theorem relating the deep and shallow embeddings.

$$\ldots \implies (\mathsf{Fun}\ \texttt{"LOOKUP-SAFE"}, [a, x], \mathit{state}) \xrightarrow{\mathsf{ap}} (\mathsf{lookup\_safe}\ a\ x, \mathit{state})$$

Steps towards an easier verification

We translate deep embedding into convenient shallow emb. [ITP'12]

$$\begin{array}{lcl}
\mathsf{lookup\_safe}\ a\ x & = & \mathsf{if\ consp}\ x\ \mathsf{then} \\
 & & \quad \mathsf{if}\ a = \mathsf{car}\ (\mathsf{car}\ x)\ \mathsf{then} \\
 & & \quad\quad \mathsf{if\ consp}\ (\mathsf{car}\ x)\ \mathsf{then} \\
 & & \quad\quad\quad \mathsf{car}\ x \\
 & & \quad\quad \mathsf{else\ cons}\ (\mathsf{car}\ (\mathsf{car}\ x))\ (\mathsf{cdr}\ (\mathsf{car}\ x)) \\
 & & \quad \mathsf{else\ lookup\_safe}\ a\ (\mathsf{cdr}\ x) \\
 & & \mathsf{else\ Sym}\ \texttt{"NIL"}
\end{array}$$

and produce a certificate theorem relating the deep and shallow embeddings.

$$\ldots \Longrightarrow (\mathsf{Fun}\ \texttt{"LOOKUP-SAFE"}, [a, x], \mathit{state}) \xrightarrow{\mathsf{ap}} (\mathsf{lookup\_safe}\ a\ x, \mathit{state})$$

name in deep embedding

Steps towards an easier verification

We translate deep embedding into convenient shallow emb. [ITP'12]

```
lookup_safe $a$ $x$  =  if consp $x$ then
                          if $a$ = car (car $x$) then
                            if consp (car $x$) then
                              car $x$
                            else cons (car (car $x$)) (cdr (car $x$))
                          else lookup_safe $a$ (cdr $x$)
                        else Sym "NIL"
```

and produce a certificate theorem relating the deep and shallow embeddings.

$$\ldots \Longrightarrow (\mathsf{Fun}\ \texttt{"LOOKUP-SAFE"}, [a, x], \mathit{state}) \xrightarrow{\mathsf{ap}} (\mathsf{lookup\_safe}\ a\ x, \mathit{state})$$

name in deep embedding

shallow embedding

Steps towards an easier verification

We translate deep embedding into convenient shallow emb. [ITP'12]

```
lookup_safe a x  =  if consp x then
                      if a = car (car x) then
                        if consp (car x) then
                          car x
                        else cons (car (car x)) (cdr (car x))
                      else lookup_safe a (cdr x)
                    else Sym "NIL"
```

and produce a certificate theorem relating the deep and shallow embeddings.

$$\ldots \implies (\text{Fun } \texttt{"LOOKUP-SAFE"}, [a, x], \mathit{state}) \xrightarrow{\text{ap}} (\text{lookup\_safe } a\ x, \mathit{state})$$

name in deep embedding

shallow embedding

Lisp semantics

Verification proof

- prove that Milawa’s (reflective) kernel is faithful to logic

A routine verification exercise.

Verification proof

- prove that Milawa’s (reflective) kernel is faithful to logic

A routine verification exercise.

Points of interest:

Milawa’s initial proof checker was a large function

Top-level loop has complicated invariant, relates:

- program state
- current Lisp op.sem. state
- logical context

Verification proof

- prove that Milawa's (reflective) kernel is faithful to logic

A routine verification exercise.

Points of interest:

Milawa's initial proof checker was a large function

Top-level loop has complicated invariant, relates:

- program state
- current Lisp op.sem. state
- logical context

Bugs found?

Verification proof

- prove that Milawa's (reflective) kernel is faithful to logic

A routine verification exercise.

Points of interest:

Milawa's initial proof checker was a large function

Top-level loop has complicated invariant, relates:

- program state
- current Lisp op.sem. state
- logical context

Bugs found? Yes, two very minor (no soundness bugs)

Verification proof

Theorem:

$$\begin{array}{l}
\exists ans\ k\ output\ ok.\\
\quad \mathsf{milawa\_main}\ cmds\ \mathsf{init\_state} = (ans, (k, output, ok)) \ \wedge\\
\quad (ok \Longrightarrow (ans = \mathsf{Sym}\ \texttt{"SUCCESS"}) \ \wedge\\
\qquad\qquad\quad \mathsf{let}\ result = \mathsf{compute\_output}\ cmds\ \mathsf{in}\\
\qquad\qquad\qquad \mathsf{every\_line}\ \mathsf{line\_ok}\ result \ \wedge\\
\qquad\qquad\qquad output = \mathsf{output\_string}\ result)
\end{array}$$

where

$$\begin{array}{lcl}
\mathsf{line\_ok}\ (\pi, l) & = & (l = \texttt{"NIL"}) \vee\\
 & & (\exists n.\ (l = \texttt{"(PRINT (}n\ \texttt{...))"}) \wedge \mathsf{is\_number}\ n) \ \vee\\
 & & (\exists \phi.\ (l = \texttt{"(PRINT (THEOREM }\phi\texttt{))"}) \wedge \mathsf{context\_ok}\ \pi \wedge \models_{\pi} \phi)
\end{array}$$

This talk

A. formalise Milawa’s logic

B. prove that Milawa's kernel is faithful to the logic

C. connect the verified Lisp implementation

Correctness of Jitawa Lisp [ITP'11]

Top-level correctness theorem:

$$\{\,\mathsf{init\_state}\ input * \mathsf{pc}\ pc * \langle \mathsf{terminates\_for}\ input\rangle\,\}$$
$$pc : \mathsf{code\_for\_entire\_jitawa\_implementation}$$
$$\{\,\mathsf{error\_message} \vee \exists output.\ \langle([], input) \xrightarrow{\mathsf{exec}} (output, \mathsf{true})\rangle * \mathsf{final\_state}\ output\,\}$$

Correctness of Jitawa Lisp [ITP'11]

There must be enough memory and I/O assumptions must hold.

ness theorem:

$$\{\,\text{init\_state}\ input * \text{pc}\ pc * \langle \text{terminates\_for}\ input\rangle\,\}$$

$$pc : \text{code\_for\_entire\_jitawa\_implementation}$$

$$\{\,\text{error\_message} \vee \exists output.\ \langle([], input) \xrightarrow{\text{exec}} (output, \text{true})\rangle * \text{final\_state}\ output\,\}$$

Correctness of Jitawa Lisp [ITP'11]

There must be enough memory and I/O assumptions must hold.

ness theorem:

$$\{\,\text{init\_state}\ \mathit{input} * \text{pc}\ \mathit{pc} * \langle \text{terminates\_for}\ \mathit{input}\rangle\,\}$$

$$\mathit{pc} : \text{code\_for\_entire\_jitawa\_implementation}$$

$$\{\,\text{error\_message} \vee \exists \mathit{output}.\ \langle([], \mathit{input}) \xrightarrow{\text{exec}} (\mathit{output}, \text{true})\rangle * \text{final\_state}\ \mathit{output}\,\}$$

Each execution is allowed to fail with an error message.

Correctness of Jitawa Lisp [ITP'11]

There must be enough memory and I/O assumptions must hold.

ness theorem:

$$
\begin{array}{l}
\{\, \text{init\_state}\ input * \text{pc}\ pc * \langle \text{terminates\_for}\ input \rangle \,\} \\
\ pc : \text{code\_for\_entire\_jitawa\_implementation} \\
\{\, \text{error\_message} \vee \exists output.\ \langle ([\,], input) \xrightarrow{\text{exec}} (output, \text{true}) \rangle * \text{final\_state}\ output \,\}
\end{array}
$$

Each execution is allowed to fail with an error message.

If there is no error message, then the result is described by the high-level op. semantics.

Correctness of Jitawa Lisp [ITP'11]

There must be enough memory and I/O assumptions must hold.

This machine-code Hoare triple holds only for terminating executions.

ne

$$\{\, \mathsf{init\_state}\ \mathit{input} * \mathsf{pc}\ \mathit{pc} * \langle \mathsf{terminates\_for}\ \mathit{input} \rangle \,\}$$
$$\mathit{pc} : \mathsf{code\_for\_entire\_jitawa\_implementation}$$
$$\{\, \mathsf{error\_message} \vee \exists \mathit{output}.\ \langle ([], \mathit{input}) \xrightarrow{\mathsf{exec}} (\mathit{output}, \mathsf{true}) \rangle * \mathsf{final\_state}\ \mathit{output} \,\}$$

Each execution is allowed to fail with an error message.

If there is no error message, then the result is described by the high-level op. semantics.

Correctness of Jitawa Lisp [ITP'11]

There must be enough memory and I/O assumptions must hold.

ne

This machine-code Hoare triple holds only for terminating executions.

$$\{\,\mathsf{init\_state}\ input * \mathsf{pc}\ pc * \langle \mathsf{terminates\_for}\ input\rangle\,\}$$
$$pc : \mathsf{code\_for\_entire\_jitawa\_implementation}$$
$$\{\,\mathsf{error\_message} \lor \exists output.\ \langle([], input) \xrightarrow{\mathsf{exec}} (output, \mathsf{true})\rangle * \mathsf{final\_state}\ output\,\}$$

list of numbers

Each execution is allowed to fail with an error message.

If there is no error message, then the result is described by the high-level op. semantics.

Theorem: Milawa is sound down to x86

$$
\begin{array}{l}
\forall \mathit{input}\ \mathit{pc}.\\
\quad \{\,\mathsf{init\_state}\ (\mathsf{milawa\_implementation} \mathbin{++} \texttt{"(milawa-main '}\mathit{input}\texttt{)"}) * \mathsf{pc}\ \mathit{pc}\,\}\\
\quad\ \mathit{pc} : \mathsf{code\_for\_entire\_jitawa\_implementation}\\
\quad \{\,\mathsf{error\_message} \lor (\mathsf{let}\ \mathit{result} = \mathsf{compute\_output}\ (\mathsf{parse}\ \mathit{input})\ \mathsf{in}\\
\qquad\qquad\qquad\qquad \langle \mathsf{every\_line}\ \mathsf{line\_ok}\ \mathit{result}\rangle\ *\\
\qquad\qquad\qquad\qquad \mathsf{final\_state}\ (\mathsf{output\_string}\ \mathit{result} \mathbin{++} \texttt{"SUCCESS"}))\,\}
\end{array}
$$

Theorem: Milawa is sound down to x86

There must be enough memory and input is Milawa's kernel followed by call to main for some *input*.

$$
\begin{array}{l}
\forall \mathit{input}\ \mathit{pc}.\\
\quad \{\, \mathsf{init\_state}\ (\mathsf{milawa\_implementation} \mathbin{++} \texttt{"(milawa-main '}\mathit{input}\texttt{)"}) * \mathsf{pc}\ \mathit{pc} \,\}\\
\quad\ \mathit{pc} : \mathsf{code\_for\_entire\_jitawa\_implementation}\\
\quad \{\, \mathsf{error\_message} \vee (\mathsf{let}\ \mathit{result} = \mathsf{compute\_output}\ (\mathsf{parse}\ \mathit{input})\ \mathsf{in}\\
\qquad\qquad\qquad\qquad\quad \langle \mathsf{every\_line}\ \mathsf{line\_ok}\ \mathit{result} \rangle *\\
\qquad\qquad\qquad\qquad\quad \mathsf{final\_state}\ (\mathsf{output\_string}\ \mathit{result} \mathbin{++} \texttt{"SUCCESS"})) \,\}
\end{array}
$$

Theorem: Milawa is sound down to x86

There must be enough memory and input is Milawa's kernel followed by call to main for some *input*.

$$
\begin{aligned}
&\forall input\ pc.\\
&\quad \{\, \mathsf{init\_state}\ (\mathsf{milawa\_implementation} \mathbin{++} \texttt{"(milawa-main '}input\texttt{)"}) * \mathsf{pc}\ pc \,\}\\
&\quad\ pc : \mathsf{code\_for\_entire\_jitawa\_implementation}\\
&\quad \{\, \mathsf{error\_message} \vee (\mathsf{let}\ result = \mathsf{compute\_output}\ (\mathsf{parse}\ input)\ \mathsf{in}\\
&\qquad\qquad\qquad\qquad \langle \mathsf{every\_line}\ \mathsf{line\_ok}\ result \rangle *\\
&\qquad\qquad\qquad\qquad \mathsf{final\_state}\ (\mathsf{output\_string}\ result \mathbin{++} \texttt{"SUCCESS"}))\,\}
\end{aligned}
$$

Machine code terminates either with error message, or …

Theorem: Milawa is sound down to x86

There must be enough memory and input is Milawa's kernel followed by call to main for some *input*.

$$
\begin{array}{l}
\forall \mathit{input}\ \mathit{pc}.\\
\quad \{\, \mathsf{init\_state}\ (\mathsf{milawa\_implementation} \mathbin{++} \texttt{"(milawa-main '}\mathit{input}\texttt{)"}) * \mathsf{pc}\ \mathit{pc} \,\}\\
\quad \mathit{pc} : \mathsf{code\_for\_entire\_jitawa\_implementation}\\
\quad \{\, \mathsf{error\_message} \vee (\mathsf{let}\ \mathit{result} = \mathsf{compute\_output}\ (\mathsf{parse}\ \mathit{input})\ \mathsf{in}\\
\qquad\qquad\qquad\qquad \langle \mathsf{every\_line}\ \mathsf{line\_ok}\ \mathit{result} \rangle *\\
\qquad\qquad\qquad\qquad \mathsf{final\_state}\ (\mathsf{output\_string}\ \mathit{result} \mathbin{++} \texttt{"SUCCESS"}))\,\}
\end{array}
$$

Machine code terminates either with error message, or …

… output lines that are all true w.r.t. the semantics of the logic.

Summary

Summary

The top-level theorem:

relates the logic's semantics
with the execution of the x86 machine code.

Summary

The top-level theorem:

relates the logic’s semantics
with the execution of the x86 machine code.

Steps:

A. formalise Milawa’s logic
  - syntax, semantics, inference, soundness

B. prove that Milawa's kernel is faithful to the logic
  - run the Lisp parser (in the logic) on Milawa’s kernel
  - translate (with proof) deep embedding into shallow
  - prove that Milawa’s (reflective) kernel is faithful to logic

C. connect the verified Lisp implementation
  - compose with the correctness thm from ITP’11

Summary

The top-level theorem:

Questions?

relates the logic’s semantics
with the execution of the x86 machine code.

Steps:

A. formalise Milawa’s logic

- syntax, semantics, inference, soundness

B. prove that Milawa's kernel is faithful to the logic

- run the Lisp parser (in the logic) on Milawa’s kernel
- translate (with proof) deep embedding into shallow
- prove that Milawa’s (reflective) kernel is faithful to logic

C. connect the verified Lisp implementation

- compose with the correctness thm from ITP’11