Milawa

A Self-Verifying Theorem Prover

Jared Davis
Ph.D. Defense
Department of Computer Sciences
The University of Texas at Austin
September 18, 2009
jared@cs.utexas.edu

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)

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

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
$$\frac{}{\neg A \vee A}$$

Contraction
$$\frac{A \vee A}{A}$$

Expansion
$$\frac{A}{B \vee A}$$

Associativity
$$\frac{A \vee (B \vee C)}{(A \vee B) \vee C}$$

Cut
$$\frac{A \vee B \quad \neg A \vee C}{B \vee C}$$

Instantiation
$$\frac{A}{A/\sigma}$$

Induction

Reflexivity Axiom

$x = x$

Equality Axiom

$x_1 = y_1 \rightarrow x_2 = y_2 \rightarrow x_1 = x_2 \rightarrow y_1 = y_2$

Referential Transparency

$x_1 = y_1 \rightarrow \ldots \rightarrow x_n = y_n \rightarrow f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$

Beta Reduction

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

Base Evaluation

e.g., $1+2 = 3$

52 Lisp Axioms

e.g., $consp(cons(x, y)) = t$

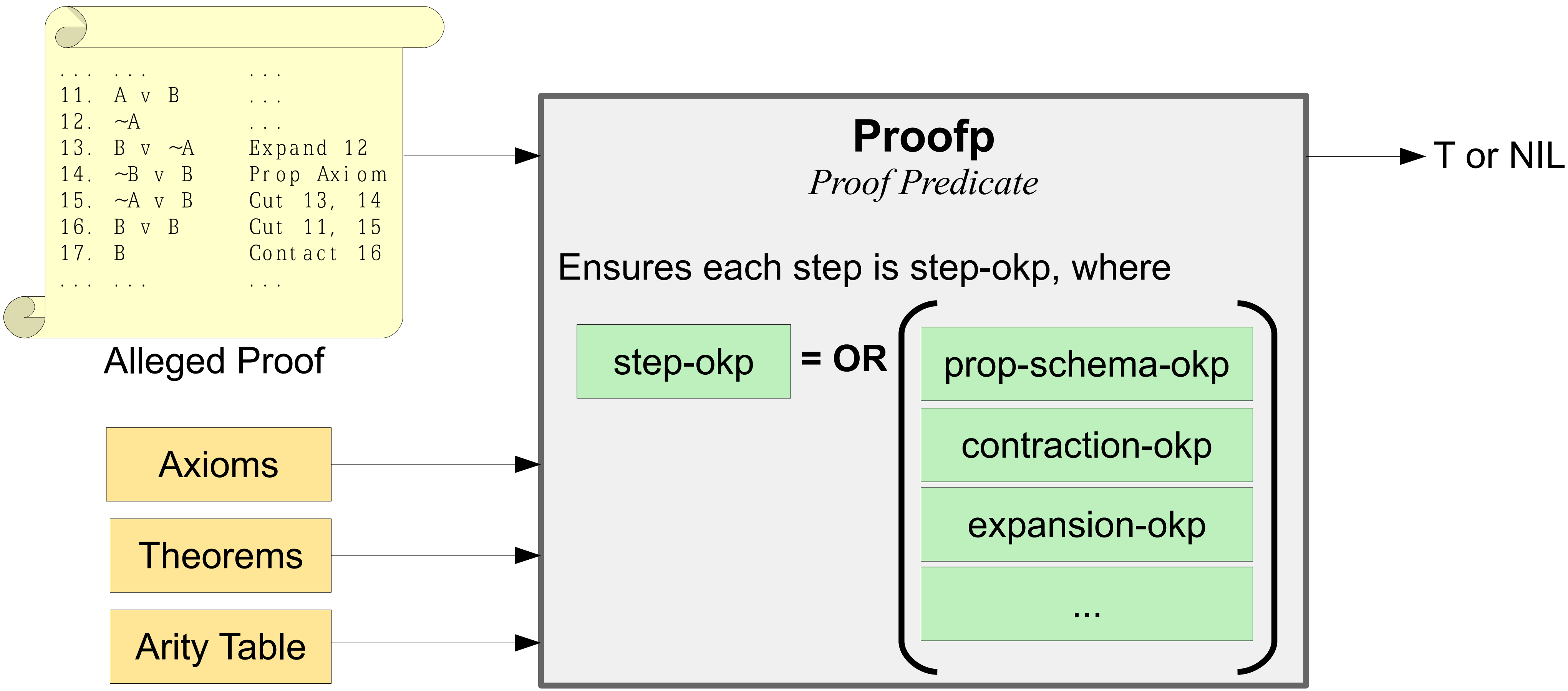
The logic as a programming language

Logical functions can be implemented in Lisp

Logic	Lisp
Naturals Symbols Ordered Pairs	Lisp Integers (arbitrary precision) Lisp Symbols Lisp Conses
if, equal natp, +, -, < symbolp, symbol-< consp, cons, car, cdr	`(defun MILAWA::car (x) (if (consp x) (car x) nil))`
Terminating, recursive functions	`(defun f (…) (… (f …) …))`
Skolem functions	`(defun skolem (…) (error “Called skolem function.”))`

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

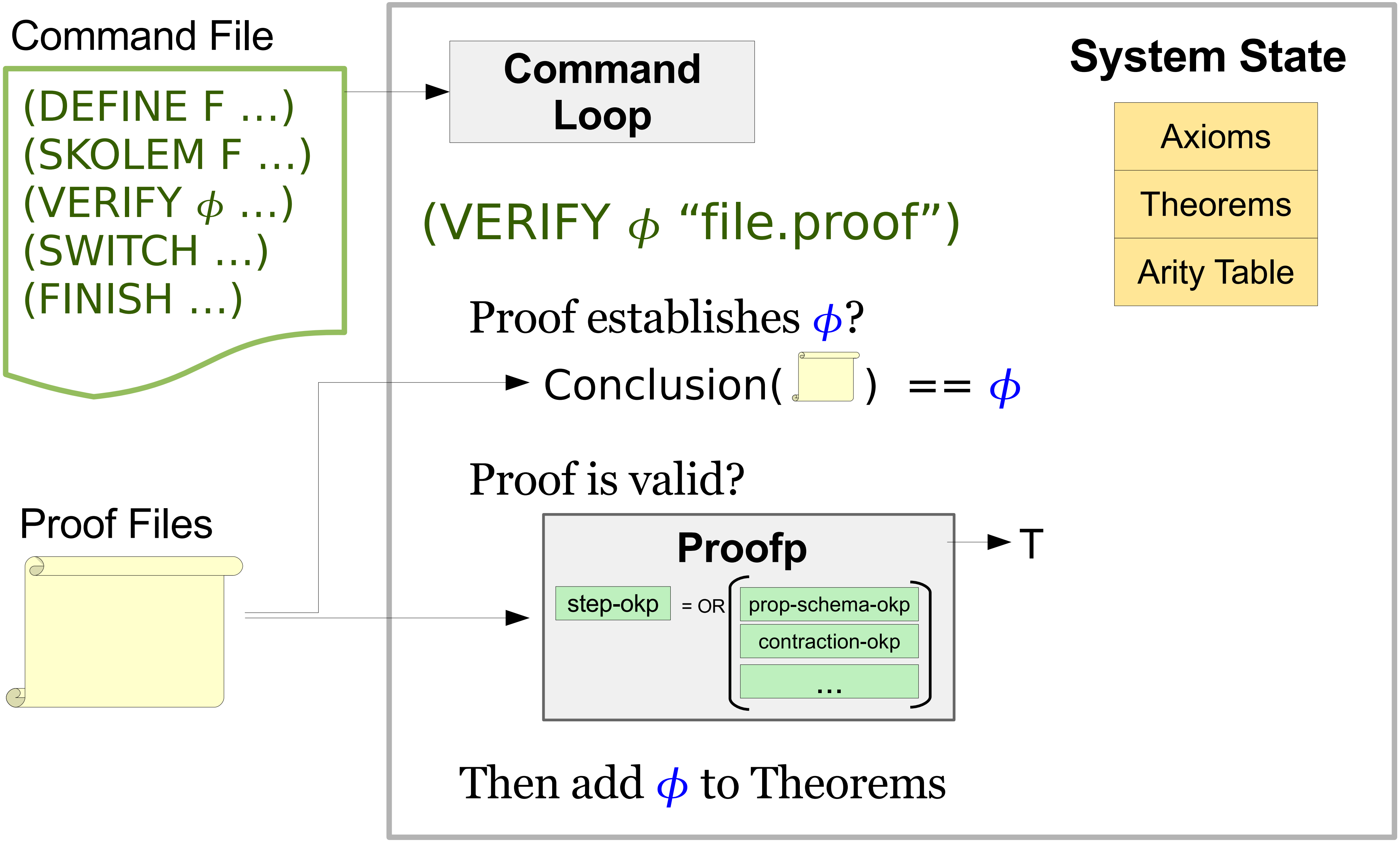
1-b. The level 1 proof checker



ϕ is provable when

$$\exists p : \text{Proofp}(p) \land \text{Conclusion}(p) = \phi$$

1-c. The command loop



Lisp Environment +

1-d. Higher-level proof checkers

(SWITCH New-Proofp)

Soundness theorem for New-Proofp

If:

P is a proof structure concluding ϕ, and
New-Proofp(**P**, *axioms*, *thms*, *atbl*)

Then:

Provablep(ϕ, *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

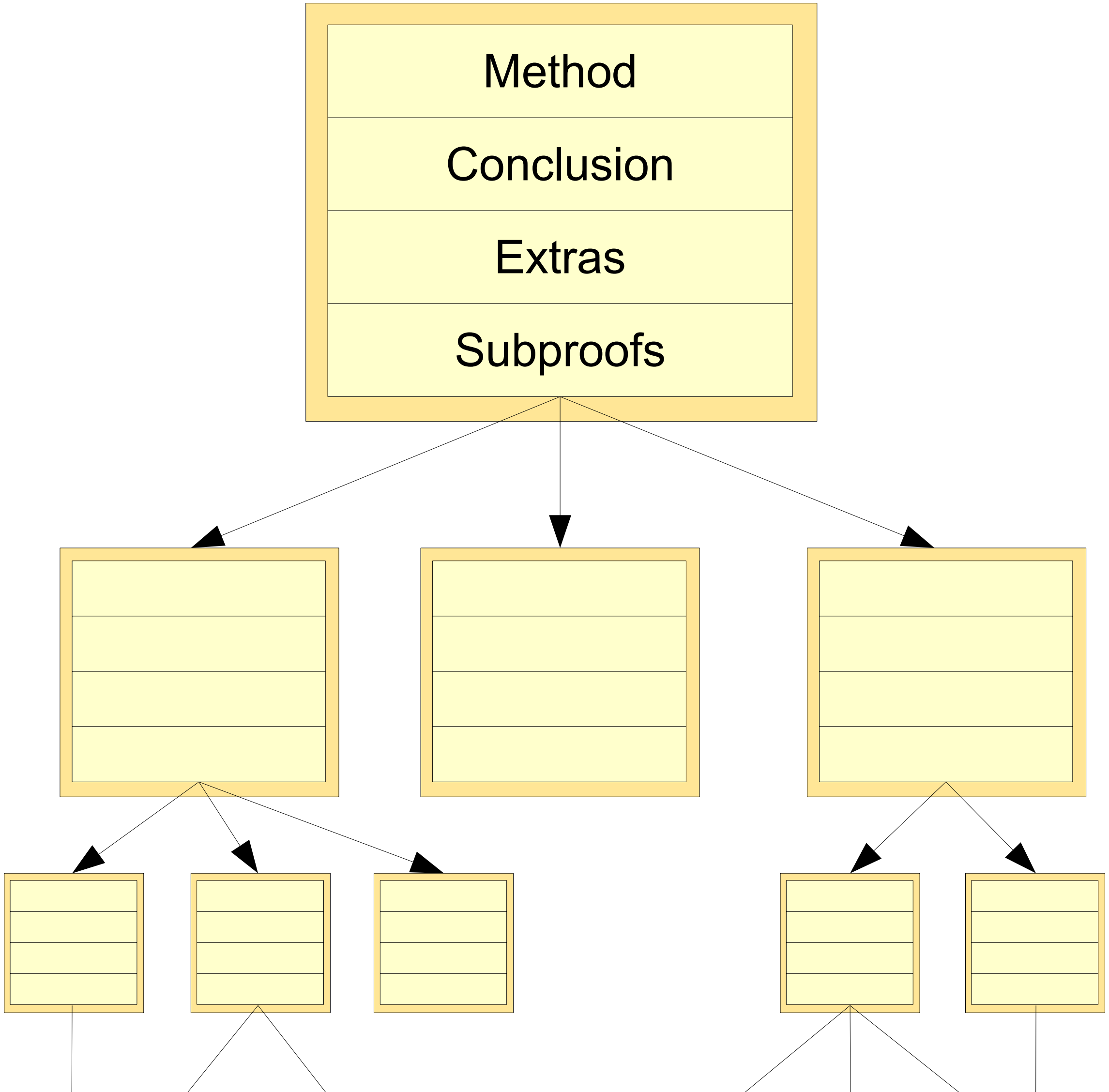
a. Fully expansive Milawa
b. Higher-level proof checkers
c. Layering the proof
d. Checking the proof

4. Conclusions

a. Review of the proposal
b. Related work
c. Contributions

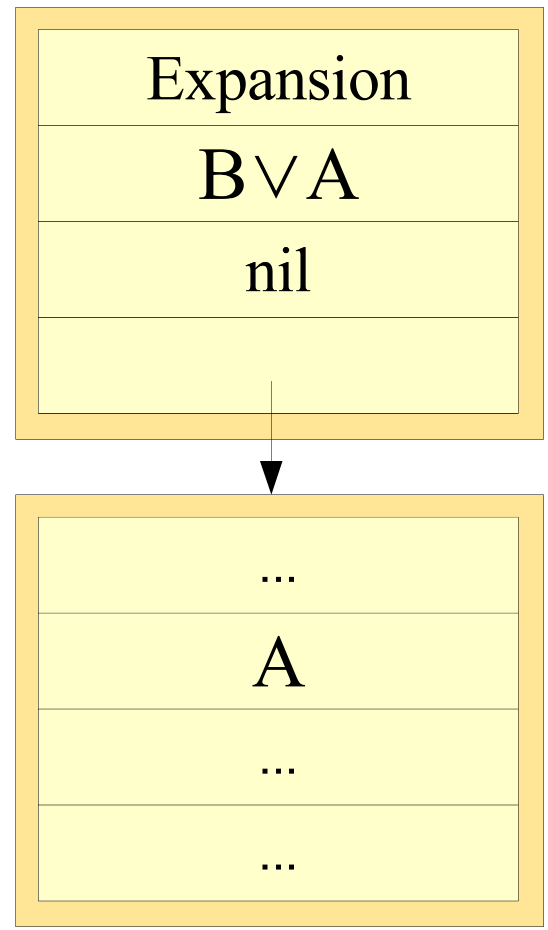
2-a. Building proofs

Proof representation



Expansion

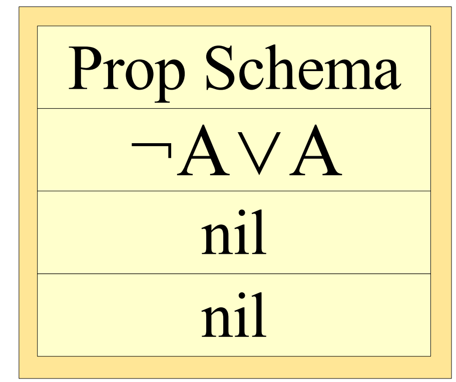
$$\frac{A}{B \vee A}$$



Primitive proof builders

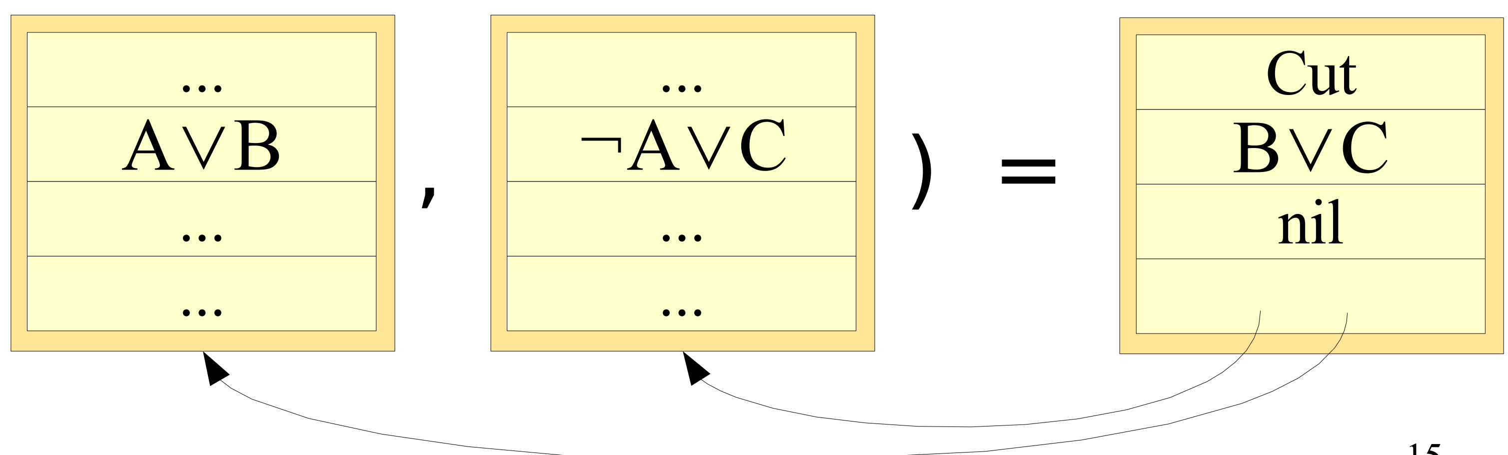
Propositional Schema $\dfrac{}{\neg A \lor A}$

build.propositional-schema(A) =



Cut $\dfrac{A \lor B \quad \neg A \lor C}{B \lor C}$

build.cut(

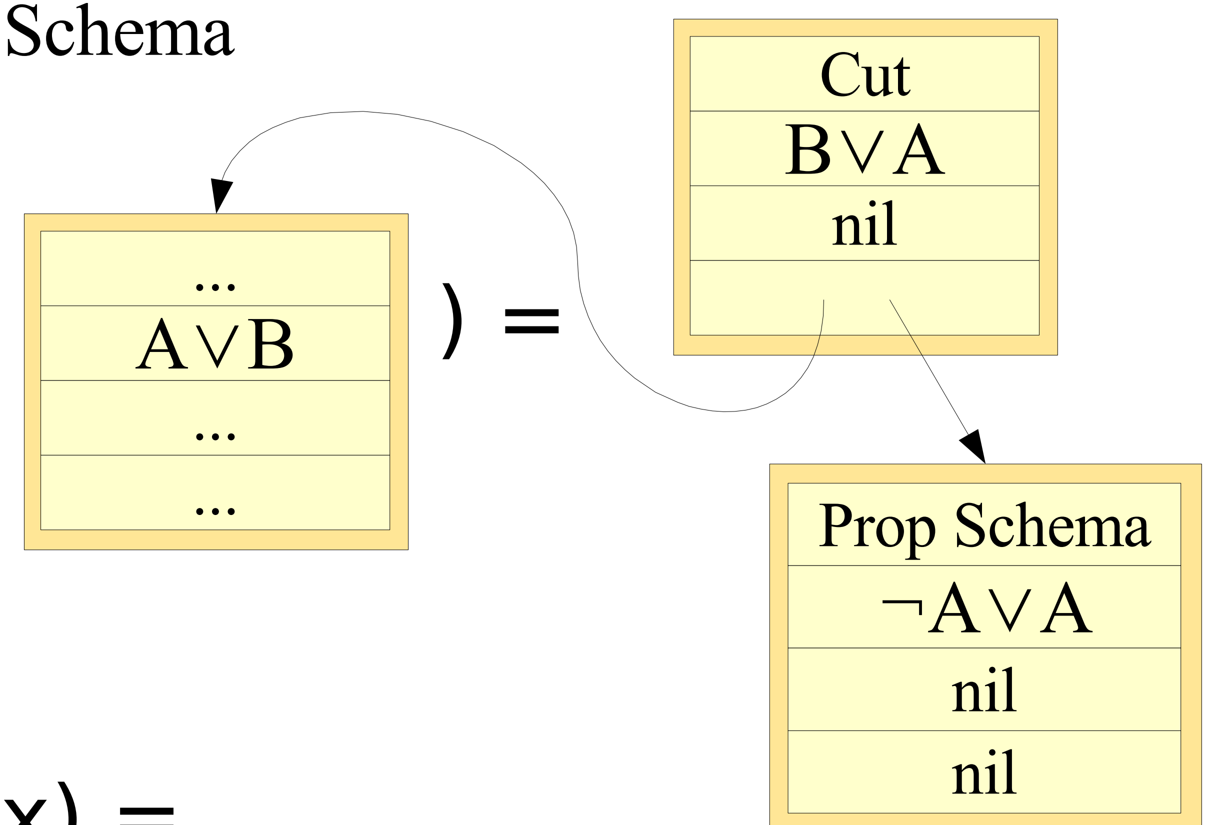


Non-primitive builders

Commute Or

1. A∨B Given
2. ¬A∨A Propositional Schema
3. B ∨ A Cut 1, 2

build.commute-or( ) =



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

The Three Theorems

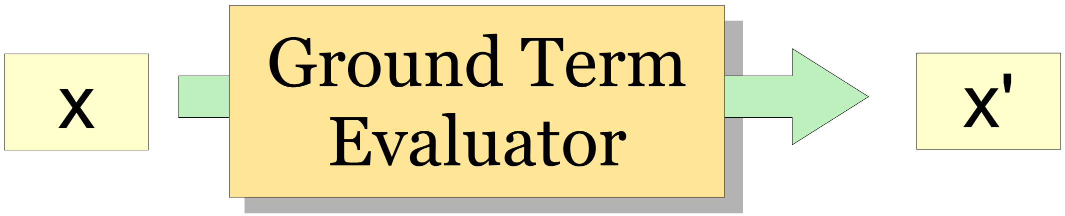
Given suitable inputs, we prove each builder is

- 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

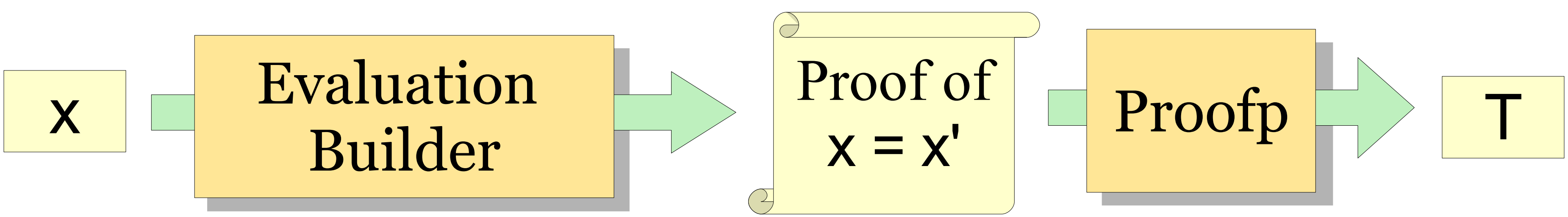
2-b. Verifying proof techniques

Introduce the technique



Soundness claim
x = x' is provable

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



Many similarities to LCF systems

2-c. Planning the proof

We develop a plan of the proof in ACL2

```
(DEFUN ...)
(DEFUN ...)
(DEFTHM ...)
(DEFUN ...)
(DEFTHM ...)
(DEFTHM ...)
(DEFUN ...)
(DEFTHM ...)
(DEFTHM ...)
(DEFTHM ...)
...
```

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

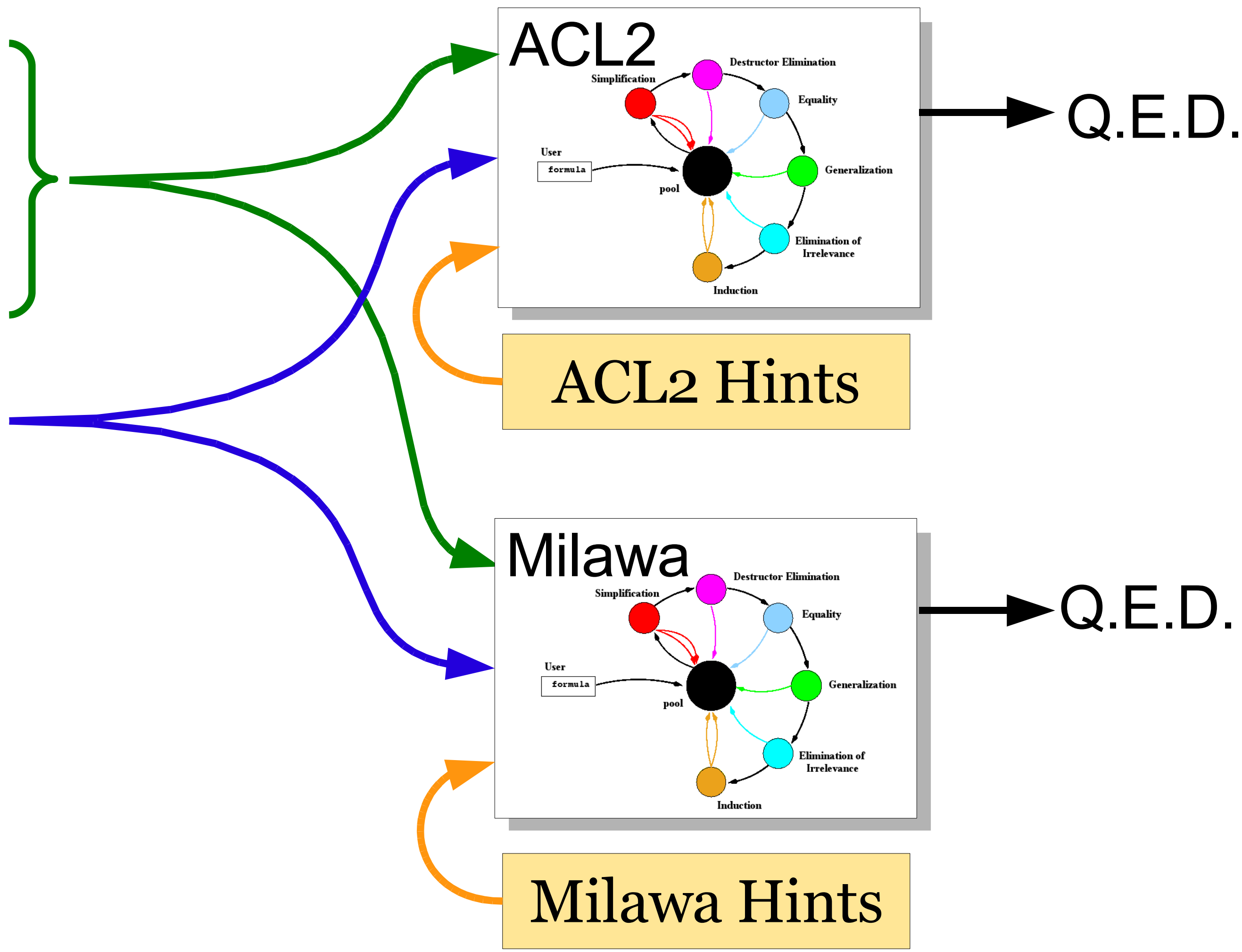
Basic utilities (lists, arith, …)
Logical concepts
Builder library
Clauses, clause splitting
Rewriting
Tactic system

2-d. Following the plan

```
(DEFUN ...)
(DEFUN ...)
(DEFTHM ...)
```

```
(DEFTHM ...)
```

```
(DEFUN ...)
(DEFTHM ...)
(DEFTHM ...)
(DEFTHM ...)
...
```



Road Map

1. The simple program

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

3. Building the proof

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

2. Self-verification

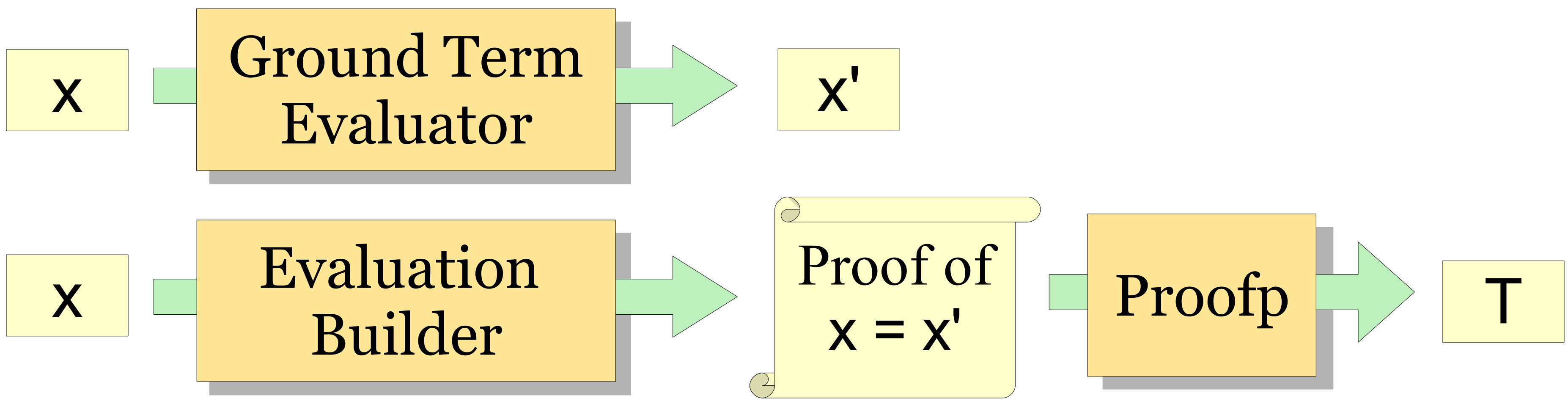
a. Building proofs
b. Verifying proof techniques
c. Planning the proof
d. Following the plan

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

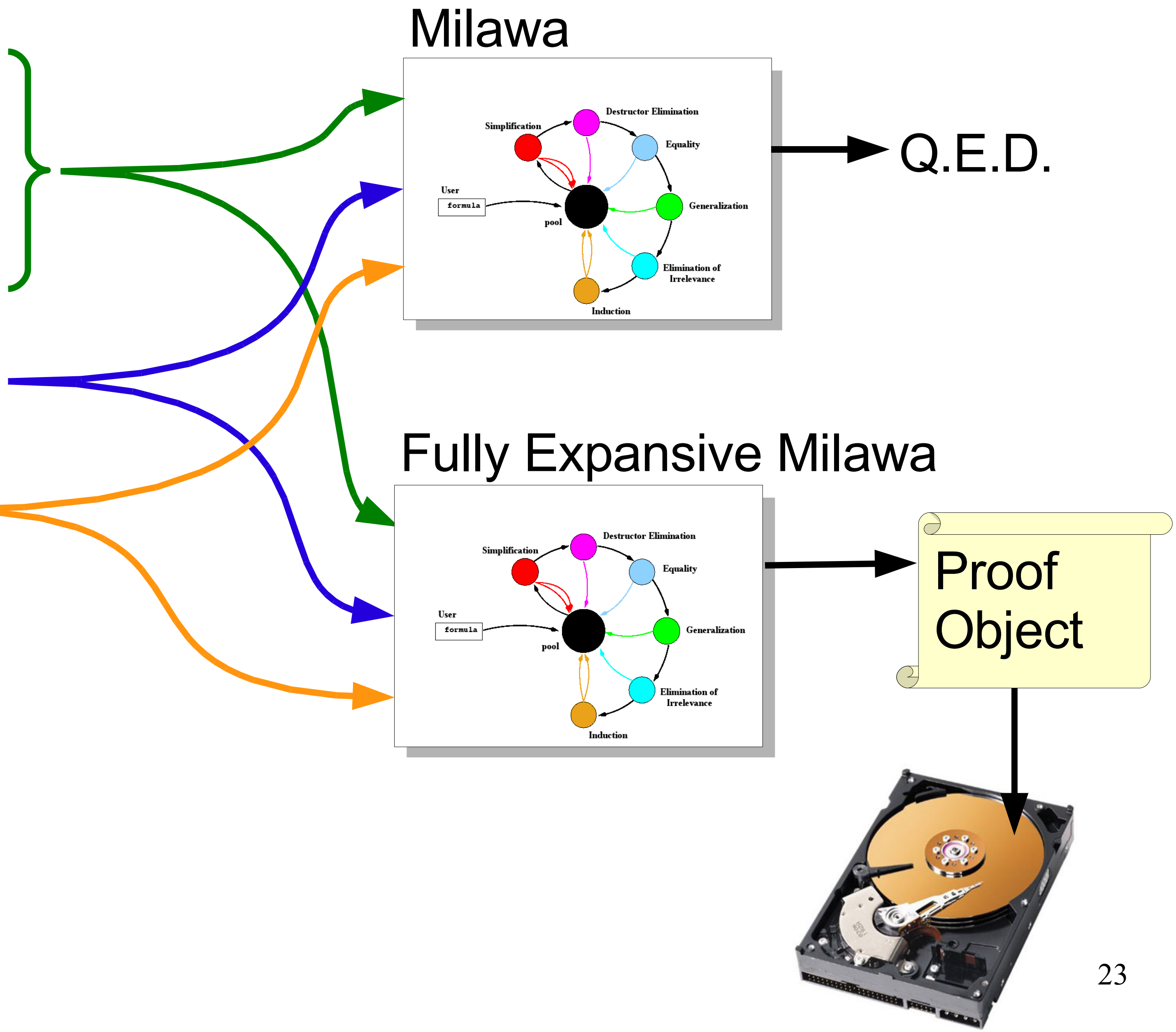
A strategy for formalizing the proof

(DEFUN ...)
(DEFUN ...)
(DEFTHM ...)

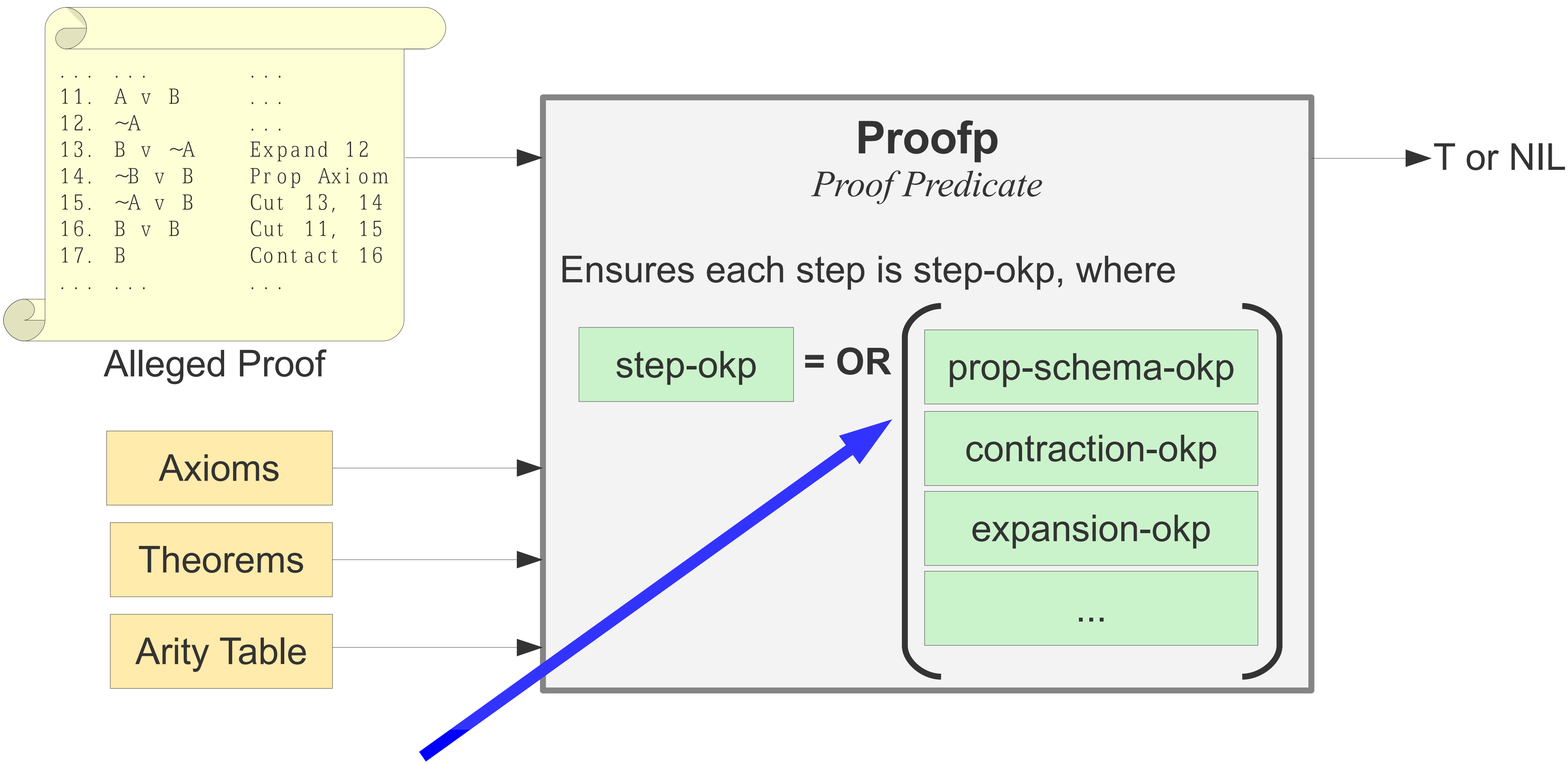
(DEFTHM ...)

Milawa Hints

(DEFUN ...)
(DEFTHM ...)
(DEFTHM ...)
(DEFTHM ...)
...



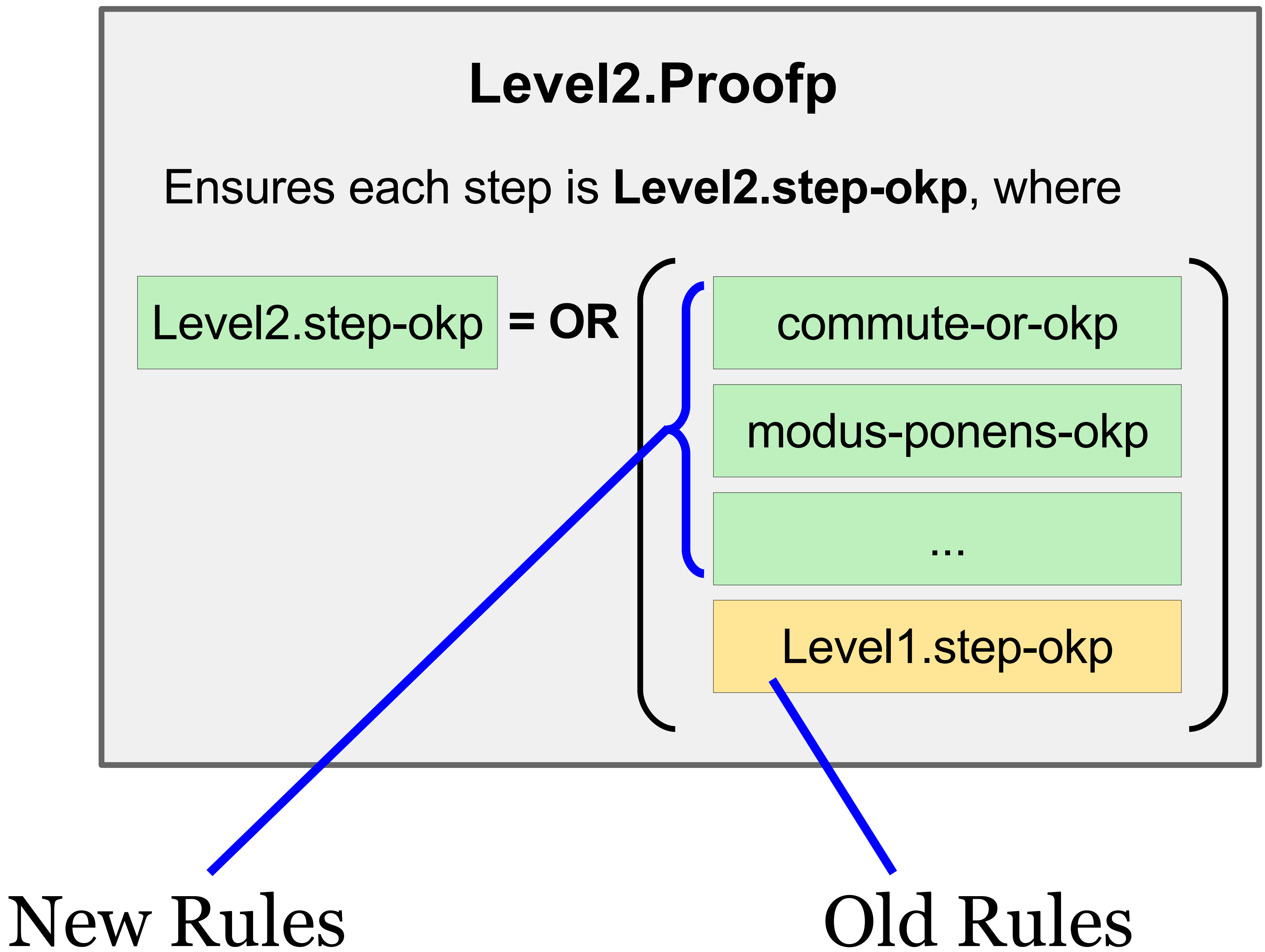
3-b. Higher-level proof checkers



Accepts only primitive rules

Good for trust, bad for proof size

Writing new proof checkers



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 ϕ, and
New-Proofp(**P**, *axioms*, *thms*, *atbl*)

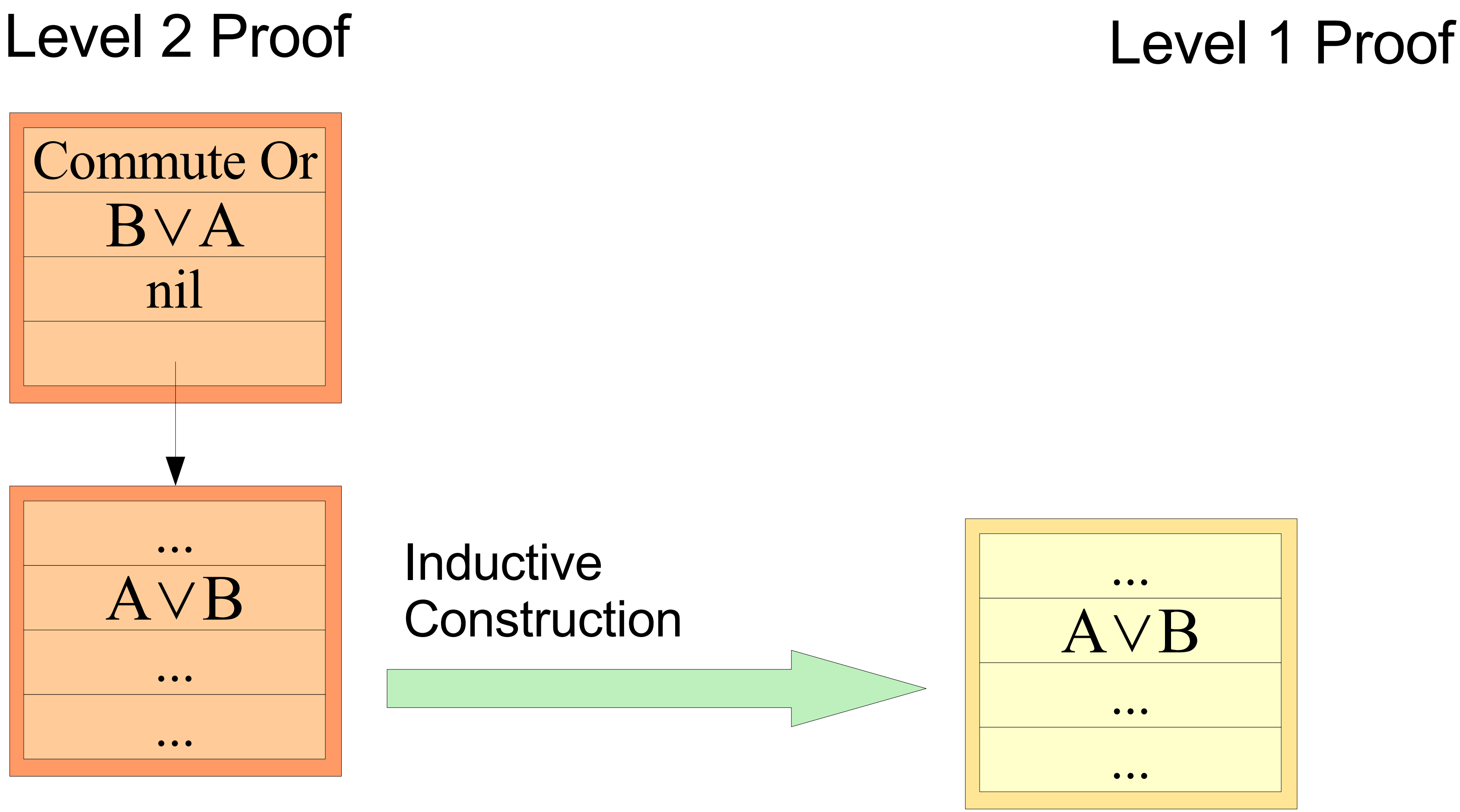
Then:

Provablep(ϕ, *axioms*, *thms*, *atbl*)

But now this is easy! (next slide)

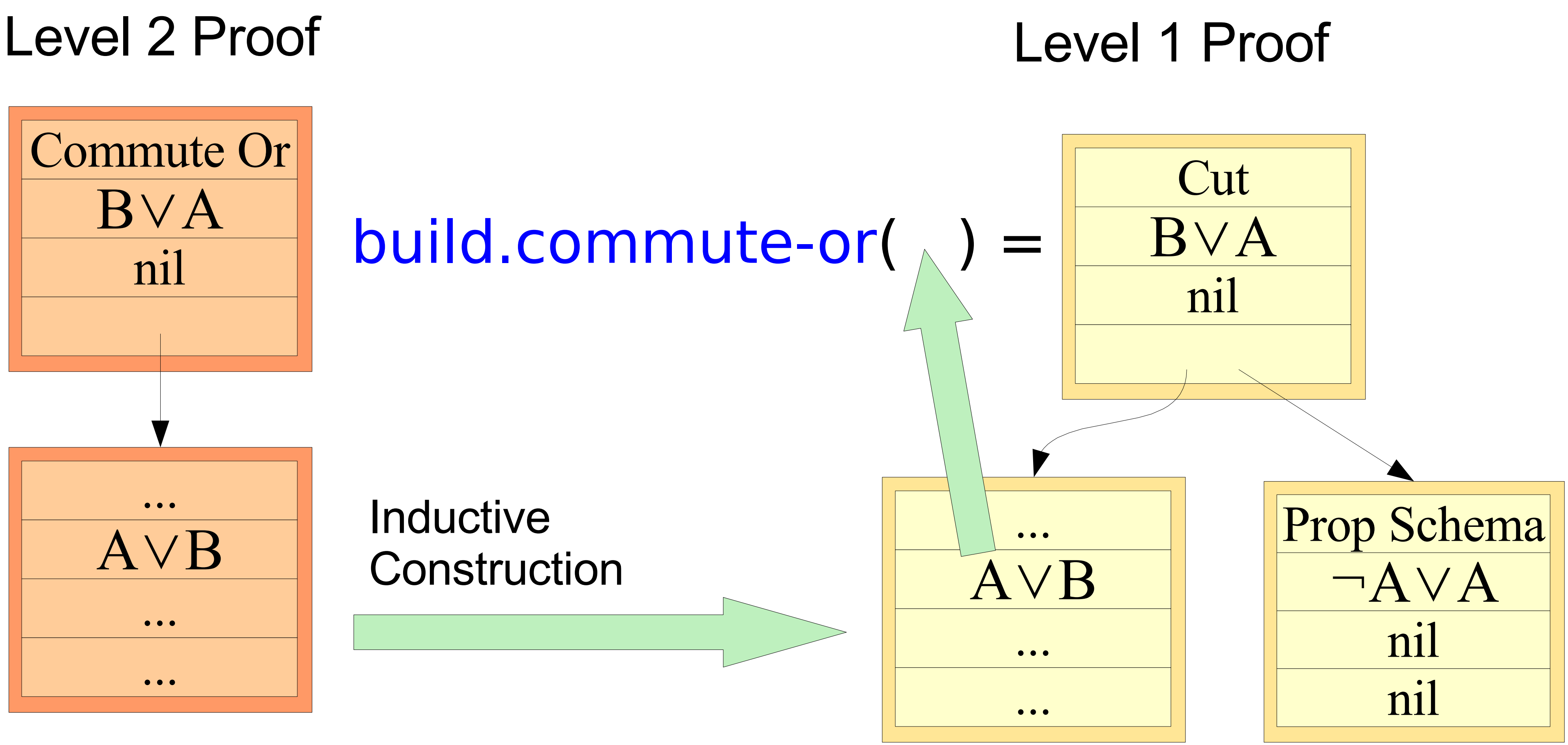
Proving the soundness theorem

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



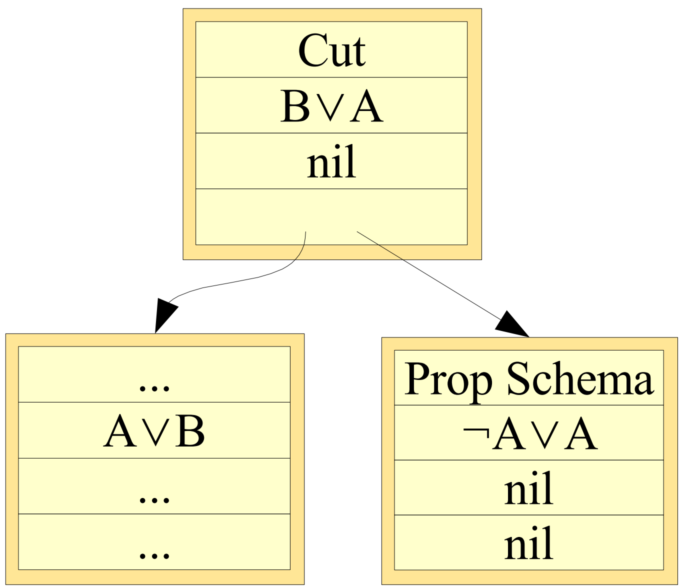
Proving the soundness theorem

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

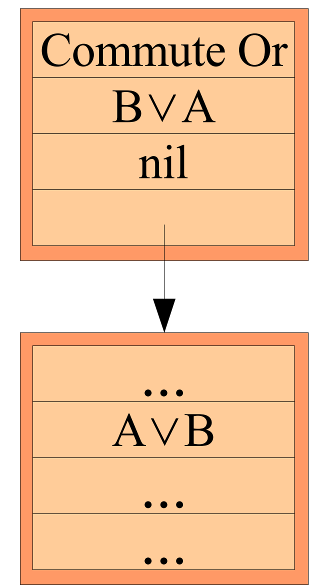


Emitting high-level proofs

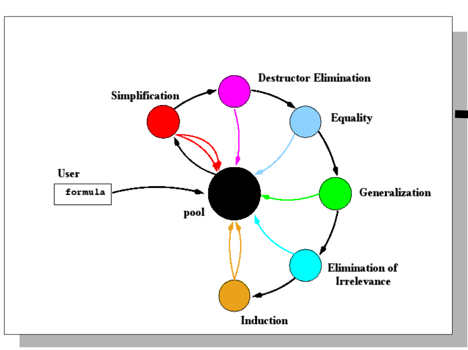
build.commute-or

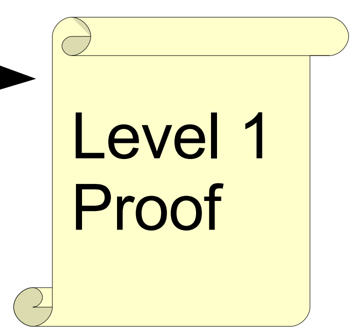


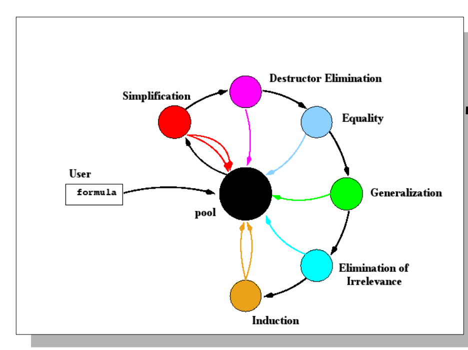
build.commute-or-high

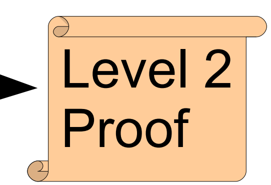


Fully Expansive Milawa









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

Level	1	2	3	4	5	6	7	8	9	10	11
Search (s)	39.4	39.4	39.4	39.4	39.4	39.4	39.4	39.4	39.0	12.3	12.3
Build (s)	406	226	117	106	102	101	101	0.8	0.5	.04	.008
Size (MC)	3,681	441	234	62	53	38	36	76	76	.8	.8
Check (s)	11,440	2,968	914	433	408	342	332	50	50	12.8	12.6

A "moderate" lemma toward level 8

Level	1	2	3	4	5	6	7	8	9	10	11
Search (s)	354	354	354	354	354	354	354	354	354	346	346
Build (s)	$\varnothing$	6,238	2,879	2,279	2,157	1,482	768	691	167	65	8
Size (MC)	$\varnothing$	8,289	4,310	1,117	1,049	426	222	171	129	58	27
Check (s)	$\varnothing$	31,451	5,323	2,816	3,120	2,737	1,874	1,430	440	457	163

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

Jordan	My home computer	Intel Core 2 Duo	13.8 hrs
Cele	Apple MacBook	Intel Core 2 Duo	19.8 hrs
Lhug-3	HP server	AMD Opteron	31.2 hrs

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!