ACL2 2007: Mechanized Metamathematics Revisited

Mechanized Metamathematics Revisited[a]

N. Shankar

shankar@csl.sri.com

URL: http://www.csl.sri.com/~shankar/

Computer Science Laboratory
SRI International
Menlo Park, CA

Background

Metamathematics studies the the properties of formal systems *(object languages and logics)* in a metalanguage.

Metamathematics is a killer app for the Boyer-Moore logic/prover family.

A few metamathematical proofs have already been formalized, but many issues remain to be explored.

This talk is meant to stimulate renewed interest in mechanized metamathematics, particularly in the ACL2 community.

Warning: Much of the talk is quite speculative.

What is Metamathematics?

"It's nothing like metaphysics".

Metamathematics is the mathematics of formalized mathematics.

Long history of work in metamathematics.

Most of the modern development is stimulated by Hilbert's programme for securing the foundations of mathematics through finitist metamathematics.

Gödel's incompleteness theorem demonstrated the implausibility of truly securing mathematics in this manner.

Metamathematics is a fertile medium for studying theoretical and engineering issues concerning logical and computational systems.

Why Metamethematics?

Metamathematics can be studied for its own sake as a source of many beautiful theorems and challenges.

Mechanized metamathematics can be exploited to build extensible proof checkers and verifiers.

Industrial-grade metamathematics is needed for dealing with formalization in the large.

Mechanized Metamathematics

Boyer and Moore's proof of a tautology checker for conditional expressions.

Turing-completeness of pure Lisp.

Tautology theorem: All tautologies are provable. Includes a tautology checker.

Church-Rosser theorem for untyped lambda calculus: β-reduction is confluent (repeated by Huet, Nipkow, Pfenning, Homeier, . . .).

First Incompleteness theorem: Existence of an unprovable sentence in hereditarily finite set theory $Z2$. Improved treatment in Coq by Russell O'Connor (7036 lines of specification; 37,906 lines of proof).

Other Examples

Interpreter-based proofs are very popular in ACL2.

FM9001 was verified with an explicit hardware representation.

The Piton assembler and micro-Gypsy compiler correctness proofs employ industry-grade metamathematics.

Recently, Xavier Leroy and others verified the correctness of a Clight compiler in Coq.

Programming Language Metatheory

The *Workshop on Mechanizing Metatheory* addresses this topic and contains several recent contributions.

The POPLMark Challenge: Transitivity of subtyping and type soundness for a subtype-polymorphic lambda calculus.

Naraschewski and Nipkow used Isabelle/HOL to check a proof of the W type inference algorithm.

Nipkow and his colleagues have verified the type safety for Java-like languages and multiple inheritance in a C++-like language.

Nipkow has also verified the soundness and completeness of a Hoare logic

Metamathematical Challenges in Computing

Computing is full of metamathematical challenges such as proving the correctness of

1. Parsers and type checkers
2. Optimizers and code generators (HW&SW)
3. Verification condition generators, model checkers, and static analyzers
4. Analyzers for information flow and security properties (Naumann)

Mathematical Logic

Frank Pfenning proved Gentzen's cut elimination for classical and intuitionistic logic in the Elf metalogical framework.

Harrison used HOL Light to verify the correctness of quantifier elimination for real-closed fields.

Norrish used HOL to develop proof producing versions of Cooper's algorithm and the Omega test.

Théry used Coq to verify Presburger's quantifier elimination procedure for first-order integer linear arithmetic.

Chaieb and Nipkow have also proved Cooper's algorithm and the Ferrante/Rackoff algorithm for the first-order theory of linear integer and real arithmetic.

Metamathematical Challenges in Logic

Many simple metatheorems of first-order logic have not yet been mechanically verified, e.g.:

1. Completeness: *Any consistent set of sentences has a model*
2. Compactness: *A set of sentences is satisfiable if all its finite subsets are.*)
3. Herbrand's theorem: *A prenex sentence is provable iff there is a quantifier-free Herbrand instance that is provable.*

A framework for systematically constructing soundness and completeness proofs for new and old logics would be a useful tool.

First Incompleteness Theorem

In formal systems that can represent their own syntax, it is easy to make self referential statements.

Suppose you have a pure Lisp object logic, then the metatheoretic expression `(P (SUBST '(P (SUBST X 'X (KWOTE X)) 'X (KWOTE '(P (SUBST X 'X (KWOTE X)))))))` is such a self-referential assertion.

By representing a proof checker for the formal system, in our case $Z2$, in $Z2$, we can construct a sentence U that asserts $\neg Pr(\overline{U})$.

Second Incompleteness Theorem

Given the representability of the metatheory, a predicate $Pr(y)$ can be defined in $Z2$ as $\exists x A_{\mathtt{PRF}}(x, y)$.

$Pr(\overline{U})$ is Σ_1, and $Z2$ can verify Σ_1-completeness (*every valid Σ_1-sentence is provable*).

Then

$$\vdash Pr(\overline{U}) \Rightarrow Pr(\overline{Pr(\overline{U})}).$$

But this says $\vdash Pr(\overline{U}) \Rightarrow Pr(\overline{\neg U})$.

Therefore $\vdash Con(Z2) \Rightarrow \neg Pr(\overline{U})$.

Hence $\neg \vdash Con(Z2)$, by first incompleteness theorem.

(Almost) Reflexive Soundness Proofs (Harrison)

Harrison used HOL Light to prove

1. The consistency of HOL - $\{$Infinity$\}$ in HOL Light.

2. The consistency of HOL in HOL Light plus the existence of a “large enough” cardinal.

This is a pretty good sanity check for a proof systems.

Gentzen and Gödel’s consistency proofs for arithmetic using induction up to ϵ_0 is a good challenge.

Is Metamathematics Useful?

Why Verify Inference Procedures?

Theorem provers need not be verified if we are merely using them to debug proofs.

However, if we want to trust the results, we need a sound foundation for proofs.

Low-level proof generation is a common approach to manifest correctness, e.g., LCF.

Proof-generating procedures can also contain bugs.

Proof generation is quite adequate for many uses but is impractical for the typical applications of decision procedures.

We need an approach that combines soundness and efficiency.

The Verified Reference (VR) Approach

The *Verified Reference* consists of a set of core procedures P that are reasonably efficient, proof producing, and verified.

We also have some untrusted procedures Q.

Any verification by Q can be efficiently checked with P by having Q generate some hints for P, e.g., variable orderings, case-splits, instantiations.

The procedures P have been verified using Q.

We can check this verification using P and generate proofs that are independently checked.

Inference Systems as a Uniform Framework for Decision Procedures

Inference Systems

Many decision procedures can be presented uniformly in terms of inference systems.

An inference system is a triple $\langle \Psi, \Lambda, \vdash \rangle$ of a set of logical states and an inference relation.

For each logical state ψ, $\Lambda(\psi)$ is a formula, and there is a special state $\bot$ where $\Lambda(\bot)$ is unsatisfiable.

The inference relation $\psi \vdash \psi'$ must be

1. *Conservative:* $\Lambda(\psi)$ and $\Lambda(\psi')$ are equisatisfiable
2. *Progressive:* $\psi' \preceq \psi$, and
3. *Canonizing:* If there is no ψ' such that $\psi \vdash \psi'$, then ψ is $\bot$ or $\Lambda(\psi)$ is satisfiable

Resolution

Input K is a set of clauses.

Atoms are ordered by $\succ$ which is lifted to literals so that $\neg p \succ p \succ \neg q \succ q$, if $p \succ q$.

Literals appear in clauses in decreasing order without duplication.

Tautologies, clauses containing both l and $\bar{l}$, are deleted from initial input.

Res	$\dfrac{K, l \vee \Gamma_1, \bar{l} \vee \Gamma_2}{K, l \vee \Gamma_1, \bar{l} \vee \Gamma_2, \Gamma_1 \vee \Gamma_2}$	$\Gamma_1 \vee \Gamma_2 \notin K$ $\Gamma_1 \vee \Gamma_2$ is not tautological
Contrad	$\dfrac{K}{\perp}$ if $p, \neg p \in K$ for some p	

Ordered Resolution: Example

$$
\begin{array}{c}
(K_0 =)\ \neg p \vee \neg q \vee r, \quad \neg p \vee q, \quad p \vee r, \quad \neg r \\
\hline
(K_1 =)\ \neg q \vee r, \quad K_0 \\
\hline
(K_2 =)\ q \vee r, \quad K_1 \\
\hline
(K_3 =)\ r, \quad K_2 \\
\hline
\bot
\end{array}
\begin{array}{l}
\\
\textsf{Res} \\
\textsf{Res} \\
\textsf{Res} \\
\textsf{Contrad}
\end{array}
$$

Correctness

Progress: Bounded number of clauses in the given literals. Each application of **Res** generates a new clause.

Conservation: For any model M, if $M \models l \vee \Gamma_1$ and $M \models \bar{l} \vee \Gamma_2$, then $M \models \Gamma_1 \vee \Gamma_2$.

Canonicity: Given an irreducible non-$\perp$ configuration K in the atoms $p_1, \ldots, p_n$ with $p_i \prec p_{i+1}$ for $1 \leq i \leq n$, build a series of partial interpretations M_i as follows:

1. Let $M_0 = \emptyset$
2. If p_{i+1} is the maximal literal in a clause $p_{i+1} \vee \Gamma \in K$ and $M_i \not\models \Gamma$, then let $M_{i+1} = M_i\{p_{i+1} \mapsto \top\}$.

   Otherwise, let $M_{i+1} = M_i\{p_{i+1} \mapsto \perp\}$.

Each M_i satisfies all the clauses in K in the atoms $p_1, \ldots, p_i$.

SAT with DPLL Search

DPLL looks for a satisfying assignment for a set of clauses K by building a *partial assignment* M in levels and a set of implied *conflict clauses* C.

Partial assignment M up to level l has the form $M_0; M_1; \ldots; M_l$.

M_i has the form $d_i : l_1[\Gamma_1], \ldots, l_n[\Gamma_n]$ with decision literal d_i and implied literals l_i with source clause Γ_i from $K \cup C$.

SAT with DPLL Search

If M is a total assignment that does not falsify any clauses in K, we report satisfiability.

Otherwise, we *propagate* M to $K \cup C$ to find implied literals that are added to M.

Or, we find a conflict, i.e., a clause $\kappa \in K \cup C$ falsified by M, and either report unsatisfiability (at level 0), or we *analyze* the conflict and *backjump* with an implied literal and new *conflict clause* added to C.

Otherwise, there are no new implied literals or conflicts, so we continue the search at the next level by adding a *selected* unassigned literal as the decision literal.

DPLL Pseudocode

$$
dpll(l, M, K, C) = \begin{cases}
dpll(l+1, (M'; k), K, C), \text{ if } & M' = propagate(M, K, C) \neq \bot \\
& k, \neg k \notin dom(M') \\
M', \text{ if } \overline{dom(M)} = \emptyset & \\
dpll(l', M', K, C'), \text{ if } & l > 0 \\
& \bot[\kappa] = propagate(M, K, C) \\
& \kappa' = analyze(\kappa, M, K, C) \\
& \langle l', M' \rangle = backjump(\kappa', l, M) \\
& C' = C \cup \{\kappa\} \\
\bot, \text{ if } & l = 0 \\
& \bot[\kappa] = propagate(M, K, C)
\end{cases}
$$

Example

Let K be $\{p \vee q, \neg p \vee q, p \vee \neg q, s \vee \neg p \vee q, \neg s \vee p \vee \neg q, \neg p \vee r, \neg q \vee \neg r\}$.

step	h	M	K	C	Γ
choose s	1	$;s$	K	$\emptyset$	_
choose r	2	$;s;r$	K	$\emptyset$	_
propagate	2	$;s;r,\neg q[\neg q \vee \neg r]$	K	$\emptyset$	_
propagate	2	$;s;r,\neg q,p[p \vee q]$	K	$\emptyset$	_
conflict	2	$;s;r,\neg q,p$	K	$\emptyset$	$\neg p \vee q$
analyze	0	$\emptyset$	K	q	_
propagate	0	$q[q]$	K	q	_
propagate	0	$q,p[p \vee \neg q]$	K	q	_
propagate	0	$q,p,r[\neg p \vee r]$	K	q	_
conflict	0	q,p,r	K	q	$\neg q \vee \neg r$

Correctness

Conservation: In each inference step from $\langle l, M, K, C\rangle$ to $\langle l', M', K, C'\rangle$, the conjunction of M_0, K, C is equivalent to the conjunction of $M_0'; K; C'$.

Progress: The weight of M given by $\Sigma_{i=0}^{n}|M_i| * (n+1)^{(n-i)}$ increases to a bound $(n+1)^{(n+1)}$.

Canonicity: A non-$\perp$ irreducible configuration is a satisfying total assignment.

Proof Generation

Resolution proofs can be easily extracted from the DPLL search procedure.

Each conflict clause in C has an associated proof.

The proof of the final conflict can be derived by resolution from the clauses in $K \cup C$.

Most solvers do not maintain proofs due to the time/space overhead.

SMT

SMT deals with formulas with theory atoms like $x = y$, $x \neq y$, $x - y \leq 3$, and $select(store(A, i, v), j) = w$.

The DPLL search is augmented with a theory state S in addition to the partial assignment.

Total assignments are *checked* for theory satisfiability.

When a literal is added to M by propagation, it is *asserted* to S.

When a literal is implied by S (*ask*), it is added to M.

Analysis is as in SAT, but backjumping needs to *retract* any literals from S that are removed from M.

Variable Equality: Union

The variable equality inference system is similar to the one presented earlier.

The state consists of a *find* structure F, the E-graph, that maintains equivalence classes and the input disequalities D.

Initially, $F(x) = x$ for each variable x.

The equality $x = y$ is processed by merging distinct equivalence classes using the *union* operation below.

$$union(F)(x,y) \quad = \quad \begin{cases} F[x' := y'], y' \prec x' \\ F[y' := x'], \text{ otherwise} \end{cases}$$

$$\text{where } x' \equiv F^*(x) \not\equiv F^*(y) \equiv y'$$

Merging Input Equalities

$$addeqlit(x = y, F, D) \qquad (skip)$$

$$:= \quad \langle F, D\rangle, \text{ if}$$

$$F^*(x) \equiv F^*(y)$$

$$addeqlit(x = y, F, D) \qquad (union)$$

$$:= \begin{cases} \bot, \text{ if} \\ \quad F'^*(u) \equiv F'^*(v) \text{ for some } u \neq v \in D \\ \langle F', D\rangle, \text{otherwise} \end{cases}$$

where

$$x' = F^*(x) \not\equiv F^*(y) = y',$$

$$F' = union(F)(x, y)$$

Adding Disequalities

$$\begin{array}{lcll}
addeqlit(x \neq y, F, D) & := & \bot,\ \text{if } F^*(x) \equiv F^*(y) & (contrad) \\
addeqlit(x \neq y, F, D) & := & \langle F, D\rangle,\ \text{if} & (skipdiseq) \\
 & & F^*(x) \equiv F^*(x'), & \\
 & & F^*(y) \equiv F^*(y'), & \\
 & & \text{for } x' \neq y' \in D & \\
addeqlit(x \neq y, F, D) & := & \langle F, \{x \neq y\} \cup D\rangle,\ \text{otherwise.} & (adddiseq)
\end{array}$$

Correctness

Progress: The *find* trees are *rooted* so that for any x, $F(F^i(x)) = F^i(x)$ for some i.

This means the $F^*(x)$ operation is always well defined and terminating.

Conservation: In $addeqlit(l, F, D) = \langle F', D' \rangle$, the two sides are equisatisfiable, as is also the case when $addeqlit(l, F, D) = \bot$.

Canonicity: When $addeqlit(l, F, D) = \langle F', D' \rangle$, the state $\langle F', D' \rangle$ can be used to construct a term model M with $|M| = \{x \mid F'(x) = x\}$ and $M(x) = {F'}^*(x)$.

TDPLL example

Input is $y = z, \quad x = y \vee x = z, \quad x \neq y \vee x \neq z$

Step	M	F	D	C
Propagate	$y = z$	$\{y \mapsto z\}$	$\emptyset$	$\emptyset$
Select	$y = z; x \neq y$	$\{y \mapsto z\}$	$\{x \neq y\}$	$\emptyset$
Scan	$\ldots, x \neq z$ $[x \neq z \vee y \neq z \vee x = y]$	$\{y \mapsto z\}$	$\{x \neq y\}$	$\emptyset$
Propagate	$\ldots$	$\{y \mapsto z\}$	$\{x \neq y\}$	
Analyze	$\ldots$	$\{y \mapsto z\}$	$\{x \neq y\}$	$\{y \neq z \vee x = y\}$
Backjump	$y = z, x = y$	$\{y \mapsto z\}$	$\{x \neq y\}$	$\{y \neq z \vee x = y\}$
Assert	$y = z, x = y$	$\{x \mapsto y, y \mapsto z\}$	$\{x \neq y\}$	$\{y \neq z \vee x = y\}$
Scan	$\ldots, x = z$ $[x = z \vee x \neq y \vee y \neq z]$	$\{x \mapsto y, y \mapsto z\}$	$\{x \neq y\}$	$\{y \neq z \vee x = y\}$
Conflict				

Theory Combination

Practical satisfiability problems involve multiple theories: arithmetic, arrays, datatypes, bit-vectors, and uninterpreted function symbols.

Nelson and Oppen give an elegant method for combining theory solvers:

1. The overall state of the combined solvers consists of a *core E-graph* S_0 and the individual theory states: $S_1; \ldots; S_m$.

2. A mixed input literal l is purifed into individual literals of the form $l', x_1 = t_1, \ldots, x_n = t_n$, where the literal l' is in the core and each t_i is a pure term in some theory.

3. Add each literal to the appropriate theory to obtain S_i'.

4. If there is an arrangement A of shared variables into equivalence classes such that $A \cup S_0' \cup S_i'$ is satisfiable for each theory i, then the literal l is satisfiable with respect to S.

Verifying Solvers in PVS

In 2002, Jonathan Ford & S. verified the basic Shostak combination

We have started an effort to verify solvers in PVS.

We have verified solvers for equality (union-find) and difference arithmetic.

Marc Vaucher (École Polytechnique) & S. verified the termination of a DPLL procedure.

Reflection

Reflection was first introduced in the seventies with Davis/Schwarz, Weyhrauch, and Boyer/Moore's metafunctions.

There are two types of reflection: computational and proof.

In computational reflection, we have an *internal representation* of the syntax of some fragment of the logic, e.g., arithmetic expressions, an *interpreter* for this fragment, e.g., an evaluator, and a *verified simplifier*.

Computational reflection can be directly implemented in any logic that supports syntactic representation and evaluation.

Proof Reflection

The system Kurt, built in 1988, is a small (< 700 lines) reflexive proof checker for a Boyer-Moore logic with a provability predicate Pr.

The system has a built-in rule interpreter for inference rules of the form

$$\vdash Pr(f(x)) \Rightarrow Pr(x)$$

The primitive inference rules are given as axioms (1000 lines), and derived ones are proved as theorems about Pr.

When given a goal G, the rule interpreter applies an axiom or derived rule to reduce it to the subgoal obtained from evaluating $f(G)$.

Is Kurt Sound?

Kurt is clearly inspired by the way `EVAL` works in Lisp.

With a reflexive `EVAL`, at worst, you risk non-termination.

But, with proof reflection you risk falling into the Gödel-Löb tarpit.

But, the Pr predicate does not satisfy the third derivability condition

$$\vdash Pr(\overline{p}) \Rightarrow Pr(\overline{Pr(\overline{p})}).$$

We have an open world assumption for the axioms about Pr so it is not known to the logic as a Σ_1 predicate.

But at this point, the soundness is still a conjecture rather than a claim. (I'd be happy to share the code with anyone who is interested.)

Harrison's Critique/Challenge

An unpublished 1996 paper by John Harrison contains a trenchant critique of reflection.

The basic objections are

1. Reflection poses semantic and engineering challenges
2. No natural examples where reflection theoretically beats proof generation, although theoretically this is the case.
3. No evidence that reflection works in practice.

Harrison's claims are still unrefuted, but can be seen as a challenge.

Harrison's Overreach

Harrison argues that proof generation as implemented in LCF is the most pragmatic approach to trusted extensibility.

Here, his argument is somewhat weaker.

Tactics are a very nice way to structure and extend interactive provers.

But LCF's approach to proof generation is fairly heavy-handed and the practical overhead is quite significant.

For example, Chaieb and Nipkow recently showed that the reflected quantifier elimination procedures ran 60 to 130 times faster than the corresponding tactic.

The sizes of proofs generated by modern SMT solvers could easily take up terabytes.

Validation through SAT solving

Propositional reasoning is the major contributor to proof size, and rewriting might be significant as well.

The proposal here is to factor proof checking into SAT solving plus lemma checking.

These lemmas are propositional formulas corresponding to facts generated by theory solvers or rewriters.

The lemmas are checked by the verified theory solvers and rewriters, and even rechecked with proof generation.

—: It is restricted to classical logic, whereas proof generation is general.

Validation through SAT solving

The SAT-based approach has many advantages.

Many different verification tools can easily generate evidence in the form of SAT problems.

These include model checkers and static analyzers.

The verification can be done in layers where each tool generates evidence that is checked by lower-level tools, e.g., SMT solvers, and eventually through SAT solving with lemma checking.

Conclusions

With the right tools, mechanizing proofs in metamathematics is great fun.

Many problems in programming languages, hardware, and logic require mechanized metamathematics.

Metamathematics can and should be leveraged to enhance prover capabilities and performance.

Proof generation as an approach to extensible theorem proving has several practical limitations.

Metamathematically verified reference procedures can be used to reconcile speed and trust in extensible provers.

Reflexive proof checking is a nice challenge.