The HOL-4 Trust Story

Konrad Slind

Rockwell Collins

August 12, 2010
• ADT of theorems, direct from LCF
• An inference rule is anything with ML type

\[ \tau_1 \rightarrow \cdots \rightarrow \tau_n \rightarrow \text{thm} \]

• This covers axioms, primitive rules, derived rules, primitive
definition principles, derived definition principles (recursive
types, recursive functions, inductive relations, ...)
• ML programming is used to compose inference steps
  arbitrarily while preserving safety
• Trust problem solved once and for all
• REALLY?
- ADT of theorems, direct from LCF
- An inference rule is anything with ML type

\[ \tau_1 \rightarrow \cdots \rightarrow \tau_n \rightarrow \text{thm} \]

- This covers axioms, primitive rules, derived rules, primitive definition principles, derived definition principles (recursive types, recursive functions, inductive relations, ...)
- ML programming is used to compose inference steps arbitrarily while preserving safety
- Trust problem solved once and for all
- REALLY?
The end result of a HOL-4 proof effort is a theory.
Theories are persistent, i.e., cached on disk in a readable format.
(In fact, HOL-4 theories are cached as ML modules.)
Can be read back in later sessions without replaying proofs.
This requires theorem creation (a primitive step)
Hence persistent theory import, export, and manipulation code is included in kernel
A theory could be maliciously altered while externally resident

For example it would be easy to add syntax that, when parsed back in, would result in $\vdash T = F$ under no assumptions.

Mitigated with tags (see later)

OR, one could arrange proof scripts in dependency order and execute them in order, in a single session.

No need then to import any theory, so this class of attacks avoided.
HOL-4 comes with 2 different prelogic implementations

- locally nameless (deBruijn terms + explicit substitutions)
- name-carrying

Both build the entire system + regressions
Which one is faster? It depends.
Which one is more trustworthy? We don’t know!

End introduction to HOL-4 kernel
HOL-4 comes with 2 different prelogic implementations
  - locally nameless (deBruijn terms + explicit substitutions)
  - name-carrying

Both build the entire system + regressions
Which one is faster? It depends.
Which one is more trustworthy? We don’t know!

End introduction to HOL-4 kernel
Richard Boulton’s PhD (early 90’s) was about making LCF-style provers more efficient.

- One idea was *lazy theorems*
- Essentially a thunkified theorem:

\[ \text{unit} \rightarrow \text{thm} \]

Except that it is also paired with the statement of the theorem:

\[ \text{l lazy_thm} = (\text{term list} \times \text{term}) \times (\text{unit} \rightarrow \text{thm}) \]

Thus a lazy inference rule has type

\[ \tau_1 \rightarrow \cdots \rightarrow \tau_n \rightarrow \text{l lazy_thm} \]
Lazy Theorems

- Allows some cheap exploratory term manipulation on the way to an actual proof. Only when a proof has been found does the thunk get invoked and a real theorem produced.
- Thus the actual proof is postponed until it is found.
- The technique resulted in some genuine speed-ups in performance-critical theorem proving code.
- Revisited by Amjad in his HOL-4 based model-checker (2005).
- **Trust impact**: none, since genuine theorems arising from real primitive inferences are ultimately produced.
HOL proofs have been formalized and generated, in a format suitable for external checking.

- von Wright, Wong (early 90’s), Skalberg, Obua (early 00’s), Hurd, Arthan (10’s)
- Are proofs doomed to be unfeasibly large? I used to think so, but work of H,A is encouraging.
- **Trust impact**: adds trust to LCF style (Pollack argument)
A more serious challenge for reasoning systems are proof techniques that require specialized term representations.

- Term representations in ITPs are quite general (e.g., first order terms, lambda terms)
- Typically pure
- How to incorporate efficient term representations for reasoning (often impure)?
- Case Study: BDDs
BDD Representations

- G,A constructed an LCF-style system connecting HOL terms to BDDs.
- A *Representation Judgement* is of the form (ignoring variable ordering clutter)
  \[ t \mapsto b \]
  and then propositional logic operations are paralleled by BDD operations, *e.g.*, 
  \[ t_1 \land t_2 \mapsto \text{BDD}_\text{AND}(b_1, b_2) \]
There are similar judgements for the other prop. operations.

Two more operations provide a bridge between HOL and BDD:

\[ \frac{\vdash t_1 = t_2 \quad t_1 \mapsto b}{t_2 \mapsto b} \quad \frac{t \mapsto \text{BDD_TRUE}}{\vdash t} \]

Then verifying modelcheckers for CTL and $\mu$-calculus built on top (Amjad thesis)
ML was used as the unifying environment to maintain the two judgement systems (BDD-land and HOL-land) ‘side-by-side’ while also orchestrating the passage back and forth between the representations.

BDD packages can be trusted by social process argument (heavy usage, few bugs). The transformation of BDD results to theorems occurs via a simple and small interface (ADTs again). Results are tagged.

**Trust impact**: Trust weakened by reliance on BDD package, but dependencies clear and interfaces clean, i.e., no other alien components.
ACL2 (and other systems?) supports logic definitions being exported to corresponding meta-language definitions and then executed, even to the point of using the results of evaluation in theorems.

HOL-4 also allows definitions to be exported to meta-language.

The generated code is completely separate from the theorem prover.

We currently do not systematically incorporate execution results back into proof (read-back uses type-based translation)

**Trust impact**: none. Could use tags.
Execution

Question: What is the view in other systems?
Is incorporation of execution results trivially OK, or not?
Theorems by fiat

- **mk_thm** coerces a formula into a theorem. Extremely useful!
- Generalized oracle facility:

\[
\text{mk_oracle_thm} : \text{tag} \rightarrow \text{term list} \ast \text{term} \rightarrow \text{thm}
\]

- From this, obtain **mk_thm** and **mk_axiom** by creating a separate tag for each.
- **Trust impact**: complete loss of trust
- Loss of trust can be monitored by suitable propagation of tags
A *tag* is extra information attached to a theorem that is useful to some external agent (person or program).

- Doesn’t influence the meaning of the theorem.
- Kalvala proposed using annotations (tags) systematically. Hutter explored their use in automated proof (unification, resolution)
- Tags come in two flavours: meta-language and object-language.
Object-Language Tags

Most are introduced by logical definitions of the form

\[ \vdash \text{Tag}_1 \, x = x \]
\[ \vdash \text{Tag}_2 \, x \, y = x \]

- Can attach any kind of information to any subterm in a semantically transparent way
- \text{Tag}_2 \, M \, N puts tag \( N \) on term \( M \) and has the same type and meaning as \( M \).
- Useful for some applications, e.g., control of rewriting, rippling, origin tracking
- \textbf{Trust impact:} none
Object-Language Tags

- Such tags are not a panacea (consider using OL tag for tracking formal proofs)
- Also easy to remove such tags by rewriting with the above definitions.
- The absence of such a tag does not mean that the term was not once tagged!
- Crucial property for tracking oracle usage
Consider the HOL-4 kernel code:

```
fun MP (THM(o1,Gamma,c)) (THM(o2,Delta,A')) =
    let val (A,B) = dest_imp c
    in if aconv A A'
        then THM (Tag.merge o1 o2,
                      union_hyp Gamma Delta,
                      B)
        else raise MP_Failed
    end
```
A HOL-4 theorem has the form \texttt{THM}(\textit{tag}, \textit{H}, \textit{c})

- An external function \texttt{Tag.merge} uniformly merges tags. (Currently takes unions.)
- Design currently being generalized.
- \textbf{Trust impact}: none. Tag processing does not interfere with the production of the theorem.
- Also, tags only accumulate through inference, infecting each theorem produced from a tagged theorem.
- Important: a theorem with an empty tag means that no oracle invocation was explicitly or implicitly used in the derivation of the theorem, \textit{i.e.}, it has a proof in the HOL logic.
Hunt, Kaufmann, Gordon, Reynolds have built and applied a logically justified connection between HOL and ACL2.

- ACL2 s-expressions formalized as HOL datatype
- ACL2 operations imported and defined over `sexp`
- ACL2 axioms identified and then proved
- So ACL2 logic is sound, having a model
- So if ACL2 proves something, then there is a HOL proof of the corresponding `sexp` formula
HOL<->ACL2

- Provides a logically sound link between the two systems
- Has been used by Reynolds in his PhD, K,G have re-done correctness proof for an LTL model-checking algorithm
- Major Benefit: No need to send proofs!
- **Trust impact** None, modulo faithfulness of transmission mechanisms.
- Prover A can use prover B to get a trusted result, without proof translation or verification of B or checker verification. Formal proof done once and forall.
John Harrison formalized something close to the implementation of the HOL-Light kernel, and proved it correct.

This might give a path to reflection of new inference rules into an LCF-style kernel, simply by showing that a proposed inference rule is equal to an existing derived rule.
THE END