Theorem Provers Part 5: Reflections of Reflections

Thomas Sewell

Dec 2024

Theorem Provers & Reflections of Reflections

In part 5 of this lecture series, we'll look at the limits of the proof-by-reflection method from part 4.

To recall, proof by reflection involves:

- A special mechanism for producing theorems of the form `f x = y` by running the computation.
- A soundness/correctness proof for the function `f`.

Worked Recap

Based on some questions from the end of the part 4 lecture, let's look at a worked example.

Suppose that we are given $P \longrightarrow Q$ and we want to replace Q obligations with P, including under propositional connectives.

Let's say we have a goal G like this:

$$(P \longrightarrow Q \wedge 12 < 32) \wedge (P \vee R \longrightarrow R \vee Q)$$

Worked Recap: Deep Encoding

We need to split up our goal into some deeply-encoded syntax that our operation can run over. We need a datatype something like this:

$$(P \longrightarrow Q \wedge 12 < 32) \wedge (P \vee R \longrightarrow R \vee Q)$$

```
datatype prop_encoding =
         And_Encoding prop_encoding prop_encoding
       | Or_Encoding prop_encoding prop_encoding
       | Imp_Encoding prop_encoding prop_encoding
       | Leaf string
```

I worked through some of the steps of this in Isabelle/HOL in "Deep_Monotonic.thy".

The Big Question from Part 4

We can reflect to run a (silly) example proof step.

What about running a general hosted logic implementation?

- A tableau solver?
- A SAT solver?
- An implementation of a general-purpose logic?

Verifying a Verifier

Can we realistically verify a general-purpose logic checker?

- There is a long-running Meta-Coq project.
- Harrison's verification of HOL in HOL.
  - Towards self-verification of HOL Light
    `https://www.cl.cam.ac.uk/~jrh13/papers/holhol.pdf`
- Kumar & Abrahamsson's verification of Candle.

Harrison's HOL in HOL

Harrison explains that it is attractive (thanks to familiarity) to verify HOL-Light using HOL-Light.

- Verify the soundness of the conceptual logic, the inference rules and axioms.
- Verify the correctness of the OCaml implementation, showing that success of these functions implies conceptual steps.

What about Gödel?

- $I \vdash_{\text{HOL}} \mathit{Cons}(\text{HOL})$, given a new axiom I.
- $\vdash_{\text{HOL}} \mathit{Cons}(\text{HOL} - \infty)$ where the infinity axiom is dropped.

What is HOL?

HOL was originally proposed by Alonzo Church.

It is mostly just the simply-typed λ calculus with equality $=$.

- Standard rules about proof sequents.
- $=_\beta$, $=_\eta$ properties about λ.
- Transitivity and contextual properties about $=$.
- A boolean type, and $\alpha \to \beta$ as a type-former.
- A type subset mechanism.
- A way to name uninterpreted constants and types.

(See Harrison's paper for the HOL-Light versions.)

The interesting parts are:

- An extensionality principle, $\forall x.\ f\ x = g\ x \vdash f = g$.
- An ∞ axiom that constructs the naturals.
- A Hilbert choice constant.

HOL in ZF or HOL

There is a well-known interpretation of HOL into ZF set theory with choice.

To my knowledge this is only hand-written. Much of the work is to show that lambda-constructed functions behave like relation-constructed functions in set theory.

Harrison's work is similar, various technical steps justifying lambda rules and the substitution mechanism.

It also requires a **deep encoding**. What should the target type of the interpretation be?

The Inaccessible Cardinal

Pick a carrier type T in the verification. Every type α in the object logic will map to a subset $S_\alpha \subseteq T$.

We will also need an isomorphism from α to S_α for every type α that occurs.

This is possible (non-constructively), it just means that T needs to be really big, so big that it encompasses the types that can be constructed.

- $I \vdash_{\text{HOL}} \mathit{Cons}(\text{HOL})$, given a new axiom I.
- $\vdash_{\text{HOL}} \mathit{Cons}(\text{HOL} - \infty)$ where the infinity axiom is dropped.

Candle

Candle is a project that pulls HOL-light into the CakeML ecosystem.

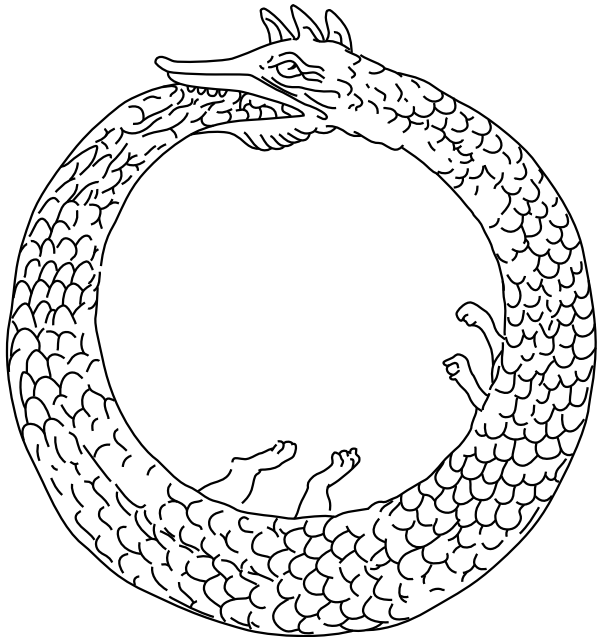
CAKEML
A Verified Implementation of ML

Borrowing an old slide again . . .

ML ITPs all the way down

A family of connected projects:

- ML, a language written for an ITP (LCF)
- HOL4, an ITP written in Standard ML
- CakeML, a compiler written in† HOL4
  - compiled by itself (CakeML)
- Candle (≈ HOL Light), an ITP
  - verified in HOL4
  - compiled by CakeML (CakeML)
- Silver, a simple ISA
  - written & verified in HOL4
  - a CakeML target



Candle: A Verified Implementation of HOL Light

Candle is a completed project now.

- An extension of CakeML to run in read-eval-print style.
  - Needed for Candle to be interactive and LCF style.
- A port of the HOL-Light verification to HOL4.
- A port of HOL-Light itself to the CakeML language.
  - Puns about cakes.
- A proof that only provable theorems are constructed.
- `https://cakeml.org/itp22-candle.pdf`

Candle: A Verified Implementation of HOL Light

Candle is a completed project now.

- An extension of CakeML to run in read-eval-print style.
  - Needed for Candle to be interactive and LCF style.
- A port of the HOL-Light verification to HOL4.
- A port of HOL-Light itself to the CakeML language.
  - Puns about cakes.
- A proof that only provable theorems are constructed.
- `https://cakeml.org/itp22-candle.pdf`

Candle has a code-evaluation oracle.

- Proves theorems of the form `f x = y`.
- Fully verified.
- `https://cakeml.org/itp23.pdf`

Candle: A Verified Implementation of HOL Light

Candle is a completed project now.

- An extension of CakeML to run in read-eval-print style.
  - Needed for Candle to be interactive and LCF style.
- A port of the HOL-Light verification to HOL4.
- A port of HOL-Light itself to the CakeML language.
  - Puns about cakes.
- A proof that only provable theorems are constructed.
- `https://cakeml.org/itp22-candle.pdf`

Candle has a code-evaluation oracle.

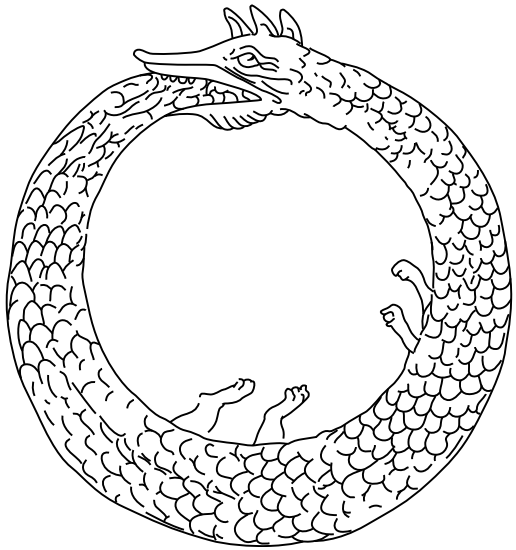
- Proves theorems of the form `f x = y`.
- Fully verified.
- `https://cakeml.org/itp23.pdf`
- This runs in an interpreter, not the CakeML eval environment.

Theorems about Theorem Provers

Candle is verified in HOL4, and runs in CakeML.

Candle can describe functional programs, and evaluate them.

Can this process recurse?

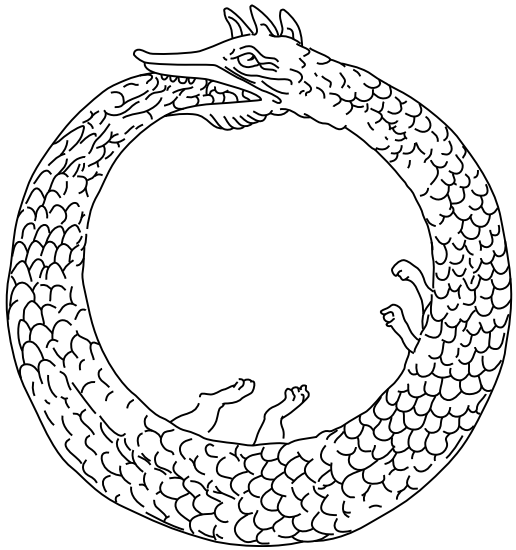


Theorems about Theorem Provers

Candle is verified in HOL4, and runs in CakeML.

Candle can describe functional programs, and evaluate them.

Can this process recurse?



Maybe yes, but there is a problem. With each reflection we're adding limitations.

Systems within Systems

Can you run a VM within your VM?

What happens if you compile Agda $\rightarrow$ Haskell $\rightarrow$ WebASM $\rightarrow$ your browser?

What happens if you interpret your type-theory in a classical logical universe.

Systems within Systems

Can you run a VM within your VM?

What happens if you compile Agda $\rightarrow$ Haskell $\rightarrow$ WebASM $\rightarrow$ your browser?

What happens if you interpret your type-theory in a classical logical universe.

What tends to happen is that the limitations accumulate.

Systems within Systems

Can you run a VM within your VM?

What happens if you compile Agda $\rightarrow$ Haskell $\rightarrow$ WebASM $\rightarrow$ your browser?

What happens if you interpret your type-theory in a classical logical universe.

What tends to happen is that the limitations accumulate.
The amazing thing about Lisp & ACL2 is that each layer of meta-programming keeps the same logic & programming language.

A Manifesto, of Sorts

Recall from part 3 that the Hilbert program is sunk.

- There is no ideal base logic that establishes the consistency of proves the consistency of everything we want to do.

But what about its computational counterpart?

- An ideal base computational logic that lets you run whatever logic you like above it.
- Needs to be logically minimal.
- Needs to reflect most of the computational power of the computer into its computational fragment.

A Manifesto, of Sorts

Recall from part 3 that the Hilbert program is sunk.

- There is no ideal base logic that establishes the consistency of proves the consistency of everything we want to do.

But what about its computational counterpart?

- An ideal base computational logic that lets you run whatever logic you like above it.
- Needs to be logically minimal.
- Needs to reflect most of the computational power of the computer into its computational fragment.

Future work.

Recap

Recap. What is a theorem prover? Logic joined with software.

- Proofs may be programs.
- Proofs may be elaborated by programs (LCF-style).
- Proofs may be about programs.
- Reflection ties the knot.

That's all. Thanks for listening.