Steps Towards Verified Implementations of HOL Light

Magnus O. Myreen[1], Scott Owens[2], and Ramana Kumar[1]

[1] Computer Laboratory, University of Cambridge, UK
[2] School of Computing, University of Kent, UK

Abstract. This short paper describes our plans and progress towards construction of verified ML implementations of HOL Light: the first formally proved soundness result for an LCF-style prover. Building on Harrison's formalisation of the HOL Light logic and our previous work on proof-producing synthesis of ML, we have produced verified implementations of each of HOL Light's kernel functions. What remains is extending Harrison's soundness proof and proving that ML's module system provides the required abstraction for soundness of the kernel to relate to the entire theorem prover. The proofs described in this paper involve the HOL Light and HOL4 theorem provers and the OpenTheory toolchain.

1 Introduction

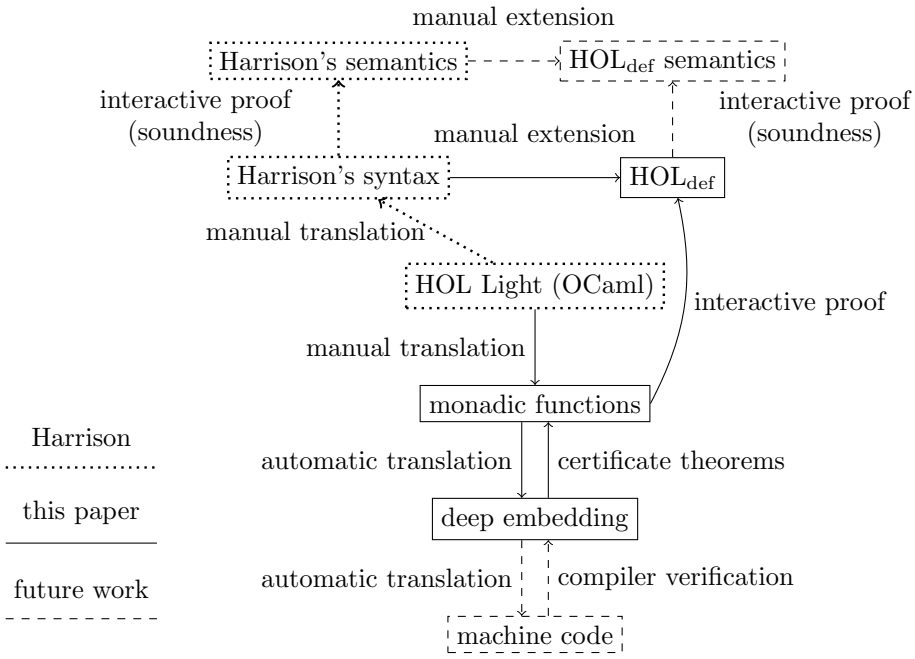
We are developing a new verification friendly dialect of ML, called CakeML. This ML dialect is approximately a subset of Standard ML carefully carved out to be convenient to program in and to reason about formally. We plan to build verified implementations of CakeML (a compiler, an implementation of a read-eval-print loop and possibly custom hardware) and also produce tools for generating and reasoning about CakeML programs (e.g. tools for synthesising CakeML from high-level specifications). One of our initial challenge examples is to construct and verify implementations of HOL Light [1] expressed in CakeML.

This short paper describes our plans and progress towards our HOL Light case study, which we believe could be the first formal proof of soundness for an implementation of an LCF-style prover. The theorem we are aiming for relates the semantics of higher-order logic (i.e. HOL) with the execution of the concrete machine code which runs the prover. We want to prove that only sound theorems can be derived in our CakeML implementations.

We build on previous work where Harrison [2] has formalised HOL in the HOL Light prover, and on our proof-producing synthesis tool [5] for MiniML, a pure version of CakeML. Throughout, we will simply write ML for CakeML.

2 Method

The following are the high-level steps we have taken in our effort to construct verified implementations of HOL Light.



1. **Extending Harrison's formalisation of HOL.** We started by adding support for user-defined constants, types and axioms to Harrison's specification of the syntactic inference rules of HOL. (We have yet to update Harrison's soundness proof, i.e. this extension of the inference rules has yet to be proved sound.) We will refer to this extension as HOL$_{\text{def}}$.

2. **Kernel as monadic functions.** Next, we took the sources for the HOL Light kernel (`fusion.ml` in the HOL Light code repository) and manually translated each HOL Light kernel function (written in OCaml) into a definition inside HOL. Since the OCaml code is stateful and uses exceptions, these HOL functions were written using a state-exception monad. We will refer to these functions as *the monadic functions*.

3. **Verification of the monadic functions.** We then proved that any computation the monadic functions can perform is something HOL$_{\text{def}}$ allows: if the monadic functions allow construction of a theorem *thm* then *thm* is also derivable in HOL$_{\text{def}}$. If HOL$_{\text{def}}$ is proved sound, then the monadic functions are also sound.

   Note that, Harrison's formalisation lives within the HOL Light theorem prover, but the rest of our work lives within the HOL4 theorem prover [7]. To bridge this gap, we transported our extension of Harrison's development from HOL Light into HOL4 using the OpenTheory toolchain [3]. OpenTheory replays the primitive inferences from one prover inside another.

4. **Verified kernel in ML.** Next, we constructed an actual ML implementation, a *deep embedding*, from the monadic functions. We constructed this

ML implementation using an *automatic* shallow-to-deep-embedding translator [5] which, for each translation, proves a certificate theorem w.r.t. a formal specification of the operational semantics of ML. These certificate theorems allowed us to carry over the correctness results for each monadic function (shallow embedding) over to the ML code (the deep embedding).

Our original translator only produced pure ML functions. For this work, we extended our previously developed translator to map the state-exception monads to appropriate stateful ML constructs.

The final — and currently missing — steps lift our kernel verification to a soundness result for the entire theorem prover running on a verified ML runtime:

5. **Verified theorem prover in ML.** We then hope to package up the deep embedding constructed above into an ML module (the HOL Light kernel) and prove that ML's module system provides the necessary restrictions which imply that only the kernel can construct values of type theorem (`thm`).
6. **Verified theorem prover running on a verified ML runtime.**

The rest of this paper describes these steps and discusses related work.

2.1 Formalising HOL with a definition mechanism

As mentioned above, we build on Harrison's formalisation of HOL inside HOL. We extended his model of the syntax with support for user-defined type operators (`Tyapp`) and term constants (`Const`):

```
type = Tyvar string | Bool | Ind | Fun type type
     | Tyapp string (type list)                    We added this line

term = Var string type | Equal type | Select type
     | Comb term term | Abs string type term
     | Const string type                           ... and this line.
```

We also define the 'state' of the logic. The state consists of a list of definitions: a definition defines a new constant, type or axiom:

```
def = Constdef string term                 term name, expression
    | Typedef string term string string    type name, prop, abs, rep
    | Axiomdef term                        statement of axiom
```

The inference rules were extended to include the new state component. Each judgement `hyps` $\vdash$ `concl` is now `defs, hyps` $\vdash$ `concl`, where `defs` is a list of definitions, i.e. `defs` has HOL type `def list`. There are also five new inference rules: one which allows extension of the definitions with a new definition,

```
defs, asl ⊢ p ∧ def_ok d defs  ⟹  (CONS d defs), asl ⊢ p
```

and four inference rules which provide theorems that arise from the definitions. For example, the following inference rule provides a description of a term definition. The constant `name` is equal to term `tm`, if the constant is defined as such in the list of definitions `defs` (which must be well-formed).

```
context_ok defs ∧ MEM (Constdef name tm) defs
 ⟹ defs, [] ⊢ Const name (typeof tm) === tm
```

Every attempt was made to be as minimal as possible in the extension of Harrison's work. The hope is that his semantics and soundness proof can be updated to work with this extension of his original formalisation. No attempt has yet been made to extend his semantics or soundness proof.

2.2 Defining the HOL Light kernel in HOL using monads

We use implementation friendly versions of the main datatypes when defining the kernel of HOL Light as functions in HOL.

```
hol_type = Tyvar string | Tyapp string (hol_type list)

hol_term = Var string hol_type | Const string hol_type
         | Comb hol_term hol_term | Abs hol_term hol_term

thm = Sequent (hol_term list) hol_term
```

The kernel of HOL Light makes use of exceptions and maintains state. The state consists of three references: `the_type_constants`, `the_term_constants`, `the_axioms`. When defining the kernel of HOL Light as functions in HOL (the monadic functions), we model the state using a record. This record contains two new components: `the_definitions` keeps track of the 'state' of the logic; and `the_clash_var` is used to hold data that should be carried in an exception.[3]

```
hol_refs = <| the_type_constants : (string # num) list ;
              the_term_constants : (string # hol_type) list ;
              the_axioms         : thm list ;
              the_definitions    : def list ;
              the_clash_var      : hol_term |>
```

We then defined each function of HOL Light's kernel using a state-exception monad based on this record type. We make use of HOL4's special syntax for monads (due to Michael Norrish). For example, HOL Light's `mk_const` function

```
let mk_const(name,theta) =
  let uty = try get_const_type name with Failure _ ->
    failwith "mk_const: not a constant name" in
  Const(name,type_subst theta uty)
```

is defined in HOL as follows:

```
mk_const(name,theta) =
  do uty <- try get_const_type name
       "mk_const: not a constant name" ;
     return (Const name (type_subst theta uty))
  od
```

[3] At the time of writing, CakeML did not support carrying of arbitrary information in exceptions. This use of an extra reference is our temporary workaround.

The monad-bind operator that hides under the syntactic sugar propagates the state and exceptions appropriately. In some cases, the monadic version is necessarily more verbose than the original OCaml code, e.g.

```
let REFL tm = Sequent([],mk_eq(tm,tm))
```

must be split with a semicolon since `mk_eq` is a monadic function:

```
REFL tm = do eq <- mk_eq(tm,tm); return (Sequent [] eq) od
```

For each of these functions (`mk_const`, `REFL`, etc.) we proved that the types, terms, theorems and states they produce are wellformed, given wellformed inputs. A theorem is wellformed if it is derivable ($\vdash$) in our extension of Harrison's formalisation of HOL w.r.t. the current list of definitions (`the_definitions`).

2.3 Proof-producing synthesis of stateful ML

The HOL Light kernel, as defined above, carries around state. In the generated ML, we implement this state using five references, one for each component of the state record from above. In order to use our previously developed proof-producing synthesis tool [5], we had to extend it with support for making use of such top-level references.

The extension essentially just threads a state (from the monadic functions) and reference store (from the ML semantics) through the entire development. At each point, the state and the reference store must agree according to a refinement invariant which relates the two representations of state.

The new state-aware synthesis tool produces deep embeddings and certificate theorems much like the original tool. For example, the monadic function `REFL` from above turns into the following ML code. Bind is translated into ML `let`.

```
val REFL = fun tm =>
  let val eq = mk_eq (Pair tm tm) in Sequent ([], eq) end;
```

The automatically proved certificate theorem for `REFL` makes a statement about the generated ML code (deep embedding) w.r.t. the operational semantics of ML: if the kernel has been loaded, then the name `"REFL"` refers to an ML function (deep embedding), which given an input, returns an output and accesses the state in a manner that exactly follows the monadic function (shallow embedding).

The details of this extension of our synthesis tool will be described in a forthcoming extension of the original conference publication [5].

3 Results, discussion and related work

At the time of writing, we have a verified ML implementation (deep embedding) for each function in HOL Light's kernel. We have proved that all types, terms and theorems this ML code produces are wellformed w.r.t. our extension of Harrison's formalisation of HOL. What remains is: proving HOL_{def} sound; proving that the

module system successfully prevents construction of theorems (values of type `thm`) outside of the kernel; and construction of verified implementations of ML.

Why not verify HOL light as it is? Such a proof would require dealing with a semantics of OCaml [6]. Real OCaml includes problematic features such as mutable strings and unsafe primitives (e.g. `Obj.magic`), which can be used to seemingly or actually produce unsoundness in HOL Light. As mentioned at the beginning, our interest lies in developing a verification friendly ML dialect.

Would Wiedijk's stateless version of HOL light [8] have been easier to verify? Wiedijk's version of HOL is very neat. However, the fact that Harrison's HOL Light is stateful is not a major hurdle and Harrison's work on formalising HOL inside HOL fits better with his version of HOL Light. Our initial efforts concentrate on Harrison's stateful version, but we are also looking into constructing verified implementations of Wiedijk's stateless version.

What is the most closely related project? Our previous project [4] on proving soundness of Davis' ACL2-inspired Milawa theorem prover had similar aims: to prove that every theorem admitted by the Milawa system (when run on our implementation of Lisp) must be true by the semantics of the Milawa logic.

Acknowledgements. Freek Wiedijk initially got us started by asking: "Can you do for HOL Light what you did for Milawa?" We are also grateful for encouragement from John Harrison and appreciate comments received from Mike Gordon and Dan Synek on drafts this paper. The first author was funded by the Royal Society, UK, and the third author was funded by the Gates Cambridge Trust, UK.

References

1. Harrison, J.: HOL Light: An overview. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (eds.) Theorem Proving in Higher Order Logics (TPHOLs). LNCS, Springer, `http://www.cl.cam.ac.uk/~jrh13/hol-light/`
2. Harrison, J.: Towards self-verification of HOL Light. In: Furbach, U., Shankar, N. (eds.) International Joint Conference on Automated Reasoning (IJCAR). LNAI, Springer (2006)
3. Hurd, J.: The OpenTheory standard theory library. In: Bobaru, M., Havelund, K., Holzmann, G.J., Joshi, R. (eds.) NASA Formal Methods. LNCS, Springer (2011)
4. Myreen, M.O., Davis, J.: The reflective Milawa theorem prover is sound (2012), `http://www.cl.cam.ac.uk/~mom22/jitawa/`
5. Myreen, M.O., Owens, S.: Proof-producing synthesis of ML from higher-order logic. In: Thiemann, P., Findler, R.B. (eds.) International Conference on Functional Programming (ICFP). ACM (2012)
6. Owens, S.: A sound semantics for OCaml light. In: Drossopoulou, S. (ed.) European Symposium on Programming (ESOP). LNCS, Springer (2008)
7. Slind, K., Norrish, M.: A brief overview of HOL4. In: Mohamed, O.A., Muñoz, C.A., Tahar, S. (eds.) Theorem Proving in Higher Order Logics (TPHOLs). LNCS, Springer (2008)
8. Wiedijk, F.: Stateless HOL. In: Hirschowitz, T. (ed.) Types for Proofs and Programs (TYPES). EPTCS (2009)