Translating HOL-Light proofs to Coq

Frédéric Blanqui

Previous works & tools on HOL to Coq

- **Denney 2000:** translates HOL98 proofs to Coq **scripts** using some intermediate stack-based machine language
- **Wiedijk 2007:** describes a manual translation of HOL-Light proofs in Coq terms via a **shallow embedding** (no implem)
- **Keller & Werner 2010:** translates HOL-Light proofs to Coq terms via a **deep embedding** & computational reflection

Previous works & tools on HOL to Coq

- **Denney 2000:** translates HOL98 proofs to Coq **scripts** using some intermediate stack-based machine language
- **Wiedijk 2007:** describes a manual translation of HOL-Light proofs in Coq terms via a **shallow embedding** (no implem)
- **Keller & Werner 2010:** translates HOL-Light proofs to Coq terms via a **deep embedding** & computational reflection
- **B. 2023:** implements Wiedijk approach via a **shallow embedding** in Lambdapi using results and ideas from:
  - Assaf & Burel (translation of OpenTheory to Dedukti, 2015)
  - Kaliszyk & Krauss (translation of HOL-Light to Isabelle, 2013)

HOL-Light logic

Terms: simply typed λ-terms with prenex polymorphism (OCaml)

Rules:

$$\frac{}{\vdash t = t}\ \text{REFL} \qquad \frac{\Gamma \vdash s = t \quad \Delta \vdash t = u}{\Gamma \cup \Delta \vdash s = u}\ \text{TRANS}$$

$$\frac{\Gamma \vdash s = t \quad \Delta \vdash u = v}{\Gamma \cup \Delta \vdash su = tv}\ \text{MK\_COMB} \qquad \frac{\Gamma \vdash s = t}{\Gamma \vdash \lambda x, s = \lambda x, t}\ \text{ABS}$$

$$\frac{}{\vdash (\lambda x, t)x = t}\ \text{BETA} \qquad \frac{}{\{p\} \vdash p}\ \text{ASSUME}$$

$$\frac{\Gamma \vdash p = q \quad \Delta \vdash p}{\Gamma \cup \Delta \vdash q}\ \text{EQ\_MP}$$

$$\frac{\Gamma \vdash p \quad \Delta \vdash q}{(\Gamma - \{q\}) \cup (\Delta - \{p\}) \vdash p = q}\ \text{DEDUCT\_ANTISYM\_RULE}$$

$$\frac{\Gamma \vdash p}{\Gamma\theta \vdash p\theta}\ \text{INST} \qquad \frac{\Gamma \vdash p}{\Gamma\Theta \vdash p\Theta}\ \text{INST\_TYPE}$$

HOL-Light logic: connectives are defined from equality!

(Andrews Q0 logic)

$$
\begin{array}{rcl}
\top & =_{def} & (\lambda p.p) = (\lambda p.p) \\
\wedge & =_{def} & \lambda p.\lambda q.(\lambda f.fpq) = (\lambda f.f\top\top) \\
\Rightarrow & =_{def} & \lambda p.\lambda q.(p \wedge q) = p \\
\forall & =_{def} & \lambda p.p = (\lambda x.\top) \\
\exists & =_{def} & \lambda p.\forall q.(\forall x.px \Rightarrow q) \Rightarrow q \\
\vee & =_{def} & \lambda p.\lambda q.\forall r.(p \Rightarrow r) \Rightarrow (q \Rightarrow r) \Rightarrow r \\
\bot & =_{def} & \forall p.p \\
\neg & =_{def} & \lambda p.p \Rightarrow \bot
\end{array}
$$

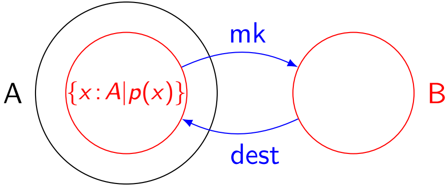
Term and type definitions in HOL-Light

- One can give a name `c` to a term `t` of type `A` by adding:
  - a typed constant `c:A`
  - an axiom `c = t`

Term and type definitions in HOL-Light

- One can give a name `c` to a term `t` of type `A` by adding:
  - a typed constant `c:A`
  - an axiom `c = t`

- One can give a name `B` to a type isomorphic to the set of terms of type `A` satisfying some predicate `p:A->bool` by adding:
  - a type constant `B`
  - a proof of ∃`a.p a`
  - a typed constant `mk:A->B`
  - a typed constant `dest:B->A`
  - an axiom ∀`b:B.mk(dest b) = b`
  - an axiom ∀`a:A.p a = (dest(mk a) = a)`



Step 1: extract proofs out of HOL-Light

HOL-Light uses the **LCF approach**:

it records provability and not proofs

```
type thm = Sequent of (term list * term      )
```

```
val REFL : term -> thm
val TRANS : thm -> thm -> thm
val MK_COMB : thm * thm -> thm
val ABS : term -> thm -> thm
val BETA : term -> thm
val ASSUME : term -> thm
val EQ_MP : thm -> thm -> thm
val DEDUCT_ANTISYM_RULE : thm -> thm -> thm
val INST_TYPE : (hol_type * hol_type) list -> thm -> thm
val INST : (term * term) list -> thm -> thm
```

Step 1: extract proofs out of HOL-Light

HOL-Light uses the **LCF approach**:

it records provability and not proofs

we need to patch it to export proofs (Obua 2005, Polu 2019):

```
type thm = Sequent of (term list * term * int)
                                        (* theorem identifier *)
val REFL : term -> thm
val TRANS : thm -> thm -> thm
val MK_COMB : thm * thm -> thm
val ABS : term -> thm -> thm
val BETA : term -> thm
val ASSUME : term -> thm
val EQ_MP : thm -> thm -> thm
val DEDUCT_ANTISYM_RULE : thm -> thm -> thm
val INST_TYPE : (hol_type * hol_type) list -> thm -> thm
val INST : (term * term) list -> thm -> thm
```

```
type proof = Proof of (thm * proof_content)
and proof_content =
| Prefl of term
| Ptrans of int * int
| ...
```

Step 2: simplify HOL-Light proofs

the number of generated proof steps can be reduced by:

- **instrumenting** connectives intro/elim rules and α-equivalence

Step 2: simplify HOL-Light proofs

the number of generated proof steps can be reduced by:

- **instrumenting** connectives intro/elim rules and α-equivalence
- **rewriting** proofs:

$$
\begin{array}{rcl}
\mathsf{SYM(REFL(t))} & \hookrightarrow & \mathsf{REFL(t)} \\
\mathsf{SYM(SYM(p))} & \hookrightarrow & \mathsf{p} \\
\mathsf{TRANS(REFL(t),p)} & \hookrightarrow & \mathsf{p} \\
\mathsf{TRANS(p,REFL(t))} & \hookrightarrow & \mathsf{p} \\
\mathsf{CONJUNCT1(CONJ(p,\_))} & \hookrightarrow & \mathsf{p} \\
\mathsf{CONJUNCT2(CONJ(\_,p))} & \hookrightarrow & \mathsf{p} \\
\mathsf{MKCOMB(REFL(t),REFL(u))} & \hookrightarrow & \mathsf{REFL(t(u))} \\
\mathsf{EQMP(REFL(\_),p)} & \hookrightarrow & \mathsf{p}
\end{array}
$$

Step 2: simplify HOL-Light proofs

the number of generated proof steps can be reduced by:

- **instrumenting** connectives intro/elim rules and α-equivalence
- **rewriting** proofs:

$$
\begin{array}{rcl}
\mathsf{SYM}(\mathsf{REFL}(t)) & \hookrightarrow & \mathsf{REFL}(t) \\
\mathsf{SYM}(\mathsf{SYM}(p)) & \hookrightarrow & p \\
\mathsf{TRANS}(\mathsf{REFL}(t),p) & \hookrightarrow & p \\
\mathsf{TRANS}(p,\mathsf{REFL}(t)) & \hookrightarrow & p \\
\mathsf{CONJUNCT1}(\mathsf{CONJ}(p,\_)) & \hookrightarrow & p \\
\mathsf{CONJUNCT2}(\mathsf{CONJ}(\_,p)) & \hookrightarrow & p \\
\mathsf{MKCOMB}(\mathsf{REFL}(t),\mathsf{REFL}(u)) & \hookrightarrow & \mathsf{REFL}(t(u)) \\
\mathsf{EQMP}(\mathsf{REFL}(\_),p) & \hookrightarrow & p
\end{array}
$$

- **removing** useless proof steps (because of tactic failures)

Step 2: simplify HOL-Light proofs

the number of generated proof steps can be reduced by:

- **instrumenting** connectives intro/elim rules and α-equivalence
- **rewriting** proofs:

$$
\begin{array}{rcl}
\mathsf{SYM(REFL(t))} & \hookrightarrow & \mathsf{REFL(t)} \\
\mathsf{SYM(SYM(p))} & \hookrightarrow & \mathsf{p} \\
\mathsf{TRANS(REFL(t),p)} & \hookrightarrow & \mathsf{p} \\
\mathsf{TRANS(p,REFL(t))} & \hookrightarrow & \mathsf{p} \\
\mathsf{CONJUNCT1(CONJ(p,\_))} & \hookrightarrow & \mathsf{p} \\
\mathsf{CONJUNCT2(CONJ(\_,p))} & \hookrightarrow & \mathsf{p} \\
\mathsf{MKCOMB(REFL(t),REFL(u))} & \hookrightarrow & \mathsf{REFL(t(u))} \\
\mathsf{EQMP(REFL(\_),p)} & \hookrightarrow & \mathsf{p}
\end{array}
$$

- **removing** useless proof steps (because of tactic failures)

initial number of steps for hol.ml	with basic tactics instrumentation	and simplification and purge
14.3 M	8.6 M (-40%)	3.5 M (-76%)

Step 3: represent HOL-Light terms and proofs in Lambdapi (Assaf & Burel, 2015)

```
/* Encoding of HOL-Light types as terms of type Set */
constant symbol Set : TYPE;
constant symbol bool : Set;
constant symbol fun : Set → Set → Set;
```

Step 3: represent HOL-Light terms and proofs in Lambdapi (Assaf & Burel, 2015)

```
/* Encoding of HOL-Light types as terms of type Set */
constant symbol Set : TYPE;
constant symbol bool : Set;
constant symbol fun : Set → Set → Set;
```

```
/* Interpretation of HOL-Light types as Lambdapi types */
injective symbol El : Set → TYPE;
rule El(fun $a $b) ↪ El $a → El $b;
```

Step 3: represent HOL-Light terms and proofs in Lambdapi (Assaf & Burel, 2015)

```
/* Encoding of HOL-Light types as terms of type Set */
constant symbol Set : TYPE;
constant symbol bool : Set;
constant symbol fun : Set → Set → Set;
```

```
/* Interpretation of HOL-Light types as Lambdapi types */
injective symbol El : Set → TYPE;
rule El(fun $a $b) ↪ El $a → El $b;
```

```
/* HOL-Light primitive constants */
constant symbol = [A] : El(fun A (fun A bool));
symbol ε [A] : El (fun (fun A bool) A);
```

Step 3: represent HOL-Light terms and proofs in Lambdapi (Assaf & Burel, 2015)

```
/* Encoding of HOL-Light types as terms of type Set */
constant symbol Set : TYPE;
constant symbol bool : Set;
constant symbol fun : Set → Set → Set;
```

```
/* Interpretation of HOL-Light types as Lambdapi types */
injective symbol El : Set → TYPE;
rule El(fun $a $b) ↪ El $a → El $b;
```

```
/* HOL-Light primitive constants */
constant symbol = [A] : El(fun A (fun A bool));
symbol ε [A] : El (fun (fun A bool) A);
```

```
/* Interpretation of HOL-Light propositions as Lambdapi types
  (Curry-Howard correspondence to be defined) */
injective symbol Prf : El bool → TYPE;
```

Step 3: represent HOL-Light terms and proofs in Lambdapi (Assaf & Burel, 2015)

```
/* HOL-Light axioms and rules */
symbol REFL [a] (t : El a) : Prf(= t t);
symbol MK_COMB [a b] [s t : El(fun a b)] [u v : El a] :
  Prf(= s t) → Prf(= u v) → Prf(= (s u) (t v));
symbol EQ_MP [p q] : Prf(= p q) → Prf p → Prf q;
symbol fun_ext [a b] [f g : El (fun a b)] :
  (Π x, Prf (= (f x) (g x))) → Prf (= f g);
symbol prop_ext [p q] :
  (Prf p → Prf q) → (Prf q → Prf p) → Prf (= p q);
```

Step 3: represent HOL-Light terms and proofs in Lambdapi (Assaf & Burel, 2015)

```
/* HOL-Light axioms and rules */
symbol REFL [a] (t : El a) : Prf(= t t);
symbol MK_COMB [a b] [s t : El(fun a b)] [u v : El a] :
  Prf(= s t) → Prf(= u v) → Prf(= (s u) (t v));
symbol EQ_MP [p q] : Prf(= p q) → Prf p → Prf q;
symbol fun_ext [a b] [f g : El (fun a b)] :
  (Π x, Prf (= (f x) (g x))) → Prf (= f g);
symbol prop_ext [p q] :
  (Prf p → Prf q) → (Prf q → Prf p) → Prf (= p q);
```

```
/* HOL-Light derived connectives */
constant symbol ⇒ : El (fun bool (fun bool bool));
rule Prf(⇒ $p $q) ↪ Prf $p → Prf $q;
constant symbol ∀ [A] : El (fun (fun A bool) bool);
rule Prf(∀ $p) ↪ Π x,Prf($p x);
...
```

Step 3: represent HOL-Light terms and proofs in Lambdapi (Assaf & Burel, 2015)

```
/* HOL-Light axioms and rules */
symbol REFL [a] (t : El a) : Prf(= t t);
symbol MK_COMB [a b] [s t : El(fun a b)] [u v : El a] :
  Prf(= s t) → Prf(= u v) → Prf(= (s u) (t v));
symbol EQ_MP [p q] : Prf(= p q) → Prf p → Prf q;
symbol fun_ext [a b] [f g : El (fun a b)] :
  (Π x, Prf (= (f x) (g x))) → Prf (= f g);
symbol prop_ext [p q] :
  (Prf p → Prf q) → (Prf q → Prf p) → Prf (= p q);
```

```
/* HOL-Light derived connectives */
constant symbol ⇒ : El (fun bool (fun bool bool));
rule Prf(⇒ $p $q) ↪ Prf $p → Prf $q;
constant symbol ∀ [A] : El (fun (fun A bool) bool);
rule Prf(∀ $p) ↪ Π x,Prf($p x);
...
```

```
/* Natural deduction rules */
symbol ∧i [p] : Prf p → Π[q],Prf q → Prf(∧ p q);
symbol ∧e1 [p q] : Prf(∧ p q) → Prf p;
symbol ∧e2 [p q] : Prf(∧ p q) → Prf q;
symbol ∃i [a] (p : El a → El bool) t : Prf(p t) → Prf(∃ p);
symbol ∃e [a] [p : El a → El bool] :
  Prf(∃(λ x,p x)) → Π[r],(Π x:El a,Prf(p x) → Prf r) → Prf r;
```

Step 4: from Lambdapi to Coq

the translation is purely syntactic:

- the symbols El and Prf are removed
- some symbols are replaced by Coq expr. wrt a user-defined map:

HOL-Light	Lambdapi	Coq
hol_type	Set	{type:>Type; el:type}
fun	arr	->
bool	bool	Prop
=	=	eq
Prefl	REFL	eq_refl
==>	⇒	->
/\	∧	and
num	num	nat
+	+	add
<=	<=	le
...	...	...

example output:

```
Lemma thm_DIV_MOD : forall m : nat, forall n : nat,
  forall p : nat, (MOD (DIV m n) p) = (DIV (MOD m (mul n p)) n).
```

Step 5: alignment of definitions

- One can give a name `c` to a term `t` of type `A` by adding:
  - a typed constant `c:A`
  - an axiom `c = t`

Step 5: alignment of definitions

- One can give a name `c` to a term `t` of type `A` by adding:
  - a typed constant `c:A`
  - an axiom `c = t`

  to replace `c` by the Coq expression `c'`, we need to do in Coq:
  - prove `c' = t`

Step 5: alignment of definitions

- One can give a name `c` to a term `t` of type `A` by adding:
  - a typed constant `c:A`
  - an axiom `c = t`

  to replace `c` by the Coq expression `c'`, we need to do in Coq:
  - prove `c' = t`

- One can give a name `B` to a type isomorphic to the set of terms of type `A` satisfying some predicate `p:A->bool` by adding:
  - a type constant `B`
  - a proof of `∃a.p a`
  - a typed constant `mk:A->B`
  - a typed constant `dest:B->A`
  - an axiom `∀b:B.mk(dest b) = b`
  - an axiom `∀a:A.p a = (dest(mk a) = a)`

  to replace `B` by the Coq expression `B'`, we need to do in Coq:
  - define `mk:A->B'`
  - define `dest:B'->A`
  - prove `∀b:B', mk(dest b) = b`
  - prove `∀a:A, p a = (dest(mk a) = a)`

Alignments already proved

- **connectives**
- **unit** type
- **product** type constructor
- type of **natural numbers**, addition, substraction, multiplication, division, power, ordering, min, max, mod, even, odd, ...
- **option** type constructor
- **sum** type constructor
- **list** type constructor, head, tail, concatenation, reverse, length, map, forall, membership, ... (thanks to Anthony Bordg)

and we are currently working on the type of **real** numbers

HOL-Light library in Coq

available on Opam:

https://github.com/deducteam/coq-hol-light/

currently contains 667 lemmas on logic, arithmetic and lists mainly

usage in Coq:

```
Require Import HOLLight.hol_light.
```

Axioms required in Coq

```
Axiom classic (P : Prop) : P \/ ~ P.

Axiom constructive_indefinite_description (A : Type) P :
  (exists x, P x) -> {x : A | P x}.

Axiom fun_ext {A B: Type} {f g: A -> B}:
  (forall x, f x = g x) -> f = g.

Axiom prop_ext {P Q : Prop} : (P -> Q) -> (Q -> P) -> P = Q.

Axiom proof_irrelevance (P:Prop) (p1 p2 : P) : p1 = p2.
```

Performances

The translations (HOL-Light to Lambdapi, and Lambdapi to Coq) and the verification by Coq can be done **in parallel** by generating a Lambdapi/Coq file for each HOL-Light user-defined theorem

To scale up, we also need to **share** types and terms

On a machine with 32 processors i9-13950HX and 64Go RAM:

HOL-Light file	dump-simp	dump size	proof steps	nb theorems
hol.ml	3m57s	3 Go	5 M	5679
topology.ml	48m	52 Go	52 M	18866

HOL-Light file	make -j32 lp	make -j32 v	v files size	make -j32 vo
hol.ml	51s	55s	1 Go	18m4s
topology.ml	22m22s	20m16s	68 Go	8h

Tools: hol2dk and lambdapi

- https://github.com/Deducteam/hol2dk

  – provides a small patch for HOL-Light to export proofs

  improves ProofTrace [Polu 2019] by reducing memory consumption and adding on-the-fly writing on disk

  – translates HOL-Light proofs to Dedukti and Lambdapi

- https://github.com/Deducteam/lambdapi

  – allows to converts dk/lp files using some encodings of HOL into Coq files

Exporting dk/lp files to Coq using Lambdapi

```
lambdapi export -o stt_coq \
  --encoding encoding.lp \
  --renaming renaming.lp \
  --erasing erasing.lp \
  --requiring coq.v \
  [--use-notations] \
  file.[dk|lp]
```

`encoding.lp`: tell lambdapi which symbols are used for the encoding of higher-order logic

`renaming.lp`: map some lambdapi identifiers that are not valid in Coq to valid Coq identifiers

`erasing.lp`: map some lambdapi identifiers to Coq expressions, and remove their declarations

`coq.v`: file imported at the beginning of each generated coq file

encoding.lp for HOL-Light

```
// symbols needed for encoding simple type theory

builtin "Set" ≔ Set;
builtin "prop" ≔ bool;
builtin "arr" ≔ fun;

builtin "imp" ≔ ⇒;
builtin "all" ≔ ∀;
builtin "eq" ≔ =;
builtin "or" ≔ ∨;
builtin "and" ≔ ∧;
builtin "ex" ≔ ∃;
builtin "not" ≔ ¬;

builtin "El" ≔ El;
builtin "Prf" ≔ Prf;
```

HOL-Light types

HOL-Light comes with 2 type constructors:

```
let the_type_constants = ref ["bool",0; "fun",2]
```

HOL-Light types must be inhabited

This is represented in Lambdapi by having the axiom

```
symbol el [A] : El A;
```

HOL-Light types in Coq

HOL-Light types are mapped to elements of:

```
Record Type' := { type :> Type; el : type }.
```

Examples:

```
Definition bool' := {| type := bool; el := true |}.
Canonical bool'.

Definition arr a (b:Type') :=
  {| type := a -> b; el := fun _ => el b |}.
Canonical arr.
```

We use **canonical structures** for Coq to automatically infer the declared canonical element of Type' from a given element of Type

`erasing.lp`:

```
    builtin "Type'" ≔ Set;
    builtin "el" ≔ el;
    builtin "arr" ≔ fun;
```

Alignment of the type of propositions and connectives

HOL-Light assumes:

```
let the_term_constants =
 ref ["=",Tyapp("fun",[aty;Tyapp("fun",[aty;bool_ty])])]
```

All the other connectives are defined from =

These definitions equal those of Coq if `bool` is mapped to `Prop`:

```
Lemma or_def :
  or = (fun p => fun q => forall r, (p -> r) -> (q -> r) -> r).
Proof.
  apply fun_ext; intro p; apply fun_ext; intro q. apply prop_ext.
    intros pq r pr qr. destruct pq. apply (pr H). apply (qr H).
    intro h. apply h.
      intro hp. left. exact hp.
      intro hq. right. exact hq.
Qed.
```

`erasing.lp`:

```
builtin "Prop" ≔ bool;
builtin "eq" ≔ =;
builtin "or" ≔ ∨;
builtin "or_def" ≔ ∨_def;
```

Definition of natural numbers in HOL-Light (part 1)

HOL-Light assumes one type `ind` and the existence of a function `f:ind -> ind` that is injective but not surjective

```
let INFINITY_AX = new_axiom
  `?f:ind->ind. ONE_ONE f /\ ~(ONTO f)`;;
```

This leads to:
– an element `IND_0` that is not in the image of `f` and
– a function `IND_SUC` that is injective

Definition of natural numbers in HOL-Light (part 2)

The type of natural numbers `num` is axiomatized as being isomorphic to the smallest subset `NUM_REP` of `ind` containing `IND_0` and stable by `IND_SUC`:

```
let NUM_REP_RULES,NUM_REP_INDUCT,NUM_REP_CASES =
  new_inductive_definition
   `NUM_REP IND_0 /\
    (!i. NUM_REP i ==> NUM_REP (IND_SUC i))`;;

let num_tydef = new_basic_type_definition
  "num" ("mk_num","dest_num")
    (CONJUNCT1 NUM_REP_RULES);;
```

The translation to Coq generates several axioms:

```
Axiom dest_num : num -> ind.
Axiom mk_num : ind -> num.
Axiom axiom_7 : forall (a : num), (mk_num (dest_num a)) = a.
Axiom axiom_8 :
  forall (r : ind), (NUM_REP r) = ((dest_num (mk_num r)) = r).
```

Alignment of the types of natural numbers (part 1)

These axioms can be eliminated if we map `num` to `nat'`:

```
Fixpoint dest_num (n:nat) : ind :=
  match n with
  | O => IND_0
  | S p => IND_SUC (dest_num p)
  end.

Definition mk_num_pred i n := i = dest_num n.

Definition mk_num i := ε (mk_num_pred i).

Lemma axiom_7 : forall (a : nat), (mk_num (dest_num a)) = a.
Proof. exact mk_num_dest_num. Qed.

Lemma axiom_8 :
  forall (r : ind), (NUM_REP r) = ((dest_num (mk_num r)) = r).
Proof.
  intro r. apply prop_ext.
    apply dest_num_mk_num.
    intro h. rewrite <- h. apply NUM_REP_dest_num.
Qed.
```

Alignment of the types of natural numbers (part 2)

We can then add in `erasing.lp`:

```
builtin "nat" ≔ num;
builtin "mk_num" ≔ mk_num;
builtin "dest_num" ≔ dest_num;
builtin "axiom_7" ≔ axiom_7;
builtin "axiom_8" ≔ axiom_8;
```

Remark: because `num` is defined out of `ind` we need to define `ind`, `IND_0`, `IND_SUC` and prove some properties about them too

Remark: we map `ind` to `nat` to eliminate the axiom of infinity

Alignment of functions on natural numbers (part 1)

```
let ZERO_DEF = new_definition
 `_0 = mk_num IND_0`;;

let SUC_DEF = new_definition
 `SUC n = mk_num(IND_SUC(dest_num n))`;;
```

is initially translated to Coq as:

```
Definition _0 : num := mk_num IND_0.

Lemma _0_def : _0 = (mk_num IND_0).
Proof. exact (eq_refl _0). Qed.

Definition SUC : num -> num :=
  fun _2104 : num => mk_num (IND_SUC (dest_num _2104)).

Lemma SUC_def :
  SUC = (fun _2104 : num => mk_num (IND_SUC (dest_num _2104))).
Proof. exact (eq_refl SUC). Qed.
```

Alignment of functions on natural numbers (part 2)

to replace _0 by 0 and SUC by S, we need to prove that the lemmas _0_def and SUC_def still hold after the replacement:

```
Lemma _0_def : 0 = (mk_num IND_0).
Proof.
  symmetry. unfold mk_num. set (P := mk_num_pred IND_0).
  assert (h: exists n, P n). exists 0. reflexivity.
  generalize (ε_spec h). set (i := ε P). unfold P, mk_num_pred. in
  apply dest_num_inj. simpl. symmetry. exact e.
Qed.

Lemma SUC_def : S = (fun _2104 : nat => mk_num (IND_SUC (dest_num
Proof.
  symmetry. apply fun_ext; intro x. rewrite mk_num_S. 2: apply NUM
  apply f_equal. apply axiom_7.
Qed.
```

then we can add in `erasing.lp`:

```
    builtin "0" ≔ _0;
    builtin "_0_def" ≔ _0_def;
    builtin "S" ≔ SUC;
    builtin "SUC_def" ≔ SUC_def;
```

Alignment of functions on natural numbers (part 3)

```
let ADD = new_recursive_definition num_RECURSION
 `(!n. 0 + n = n) /\
  (!m n. (SUC m) + n = SUC(m + n))`;;
```

is initially translated to Coq as:

```
Definition add : num -> num -> num :=
  @ε (num -> num -> num -> num)
  (fun add' : num -> num -> num -> num =>
    forall _2155 : num,
      (forall n : num, (add' _2155 (NUMERAL _0) n) = n)
        /\ (forall m : num, forall n : num,
          (add' _2155 (SUC m) n) = (SUC (add' _2155 m n))))
  (NUMERAL (BIT1 (BIT1 (BIT0 (BIT1 (BIT0 (BIT1 _0))))))).

Lemma add_def :
  add = @ε (num -> num -> num -> num)
        (fun add' : num -> num -> num -> num =>
          forall _2155 : num,
            (forall n : num, (add' _2155 (NUMERAL _0) n) = n)
              /\ (forall m : num, forall n : num,
                (add' _2155 (SUC m) n) = (SUC (add' _2155 m n))))
        (NUMERAL (BIT1 (BIT1 (BIT0 (BIT1 (BIT0 (BIT1 _0))))))).
Proof. exact (eq_refl add). Qed.
```

Alignment of functions on natural numbers (part 4)

to replace add by `Nat.add`, we need to prove that the lemma add_def still holds after the replacement:

```
Lemma add_def : add = @ε (num -> num -> num -> num)
        (fun add' : num -> num -> num -> num => forall _2155 : num
            (forall n : num, (add' _2155 (NUMERAL _0) n) = n)
              /\ (forall m : num, forall n : num,
                (add' _2155 (SUC m) n) = (SUC (add' _2155 m n))))
        (NUMERAL (BIT1 (BIT1 (BIT0 (BIT1 (BIT0 (BIT1 _0))))))).
Proof.
  generalize ( (BIT1 (BIT1 (BIT0 (BIT1 (BIT0 (BIT1 0))))))). intro
  match goal with [|- _ = ε ?x _] => set (Q := x) end.
  assert (i : exists q, Q q). exists (fun _ => Nat.add). split; re
  generalize (ε_spec i a). intros [h0 hs].
  apply fun_ext; intro x. apply fun_ext; intro y.
  induction x; simpl. rewrite h0. reflexivity. rewrite hs, IHx. re
Qed.
```

then we can add in `erasing.lp`:

```
    builtin "Nat.add" ≔ +;
    builtin "add_def" ≔ +_def;
```

Definition of real numbers in HOL-Light (part 1)

Step 1: subset `nadd` of nearly additive sequences of nats

$x : \mathbb{N} \to \mathbb{N}$ is nearly additive if $\exists B, \forall m, \forall n, |mx_n - nx_m| \leq B(m+n)$

```
let is_nadd = new_definition
  `is_nadd x <=> (?B. !m n. dist(m * x(n),n * x(m)) <= B * (m + n))`;;

let nadd_abs,nadd_rep =
  new_basic_type_definition "nadd" ("mk_nadd","dest_nadd") is_nadd_0;;

override_interface ("fn",`dest_nadd`);;
override_interface ("afn",`mk_nadd`);;
```

Definition of real numbers in HOL-Light (part 2)

Step 2: definition on `nadd` of $\leq$, $+$, $\times$, injection of $\mathbb{N}$, $^{-1}$, $/$, and proof of some properties including:

- $+$ is commutative, associative, monotone wrt $\leq$, and has 0 as neutral element
- $\times$ is commutative, associative, monotone wrt $\leq$, distributes overs $+$, and has 1 as neutral element and $^{-1}$ as inverse
- $\leq$ is total
- `nadd` is Archimedian
- `nadd` is complete: every non-empty bounded subset has a lub

Definition of real numbers in HOL-Light (part 3)

Step 3: quotient of `nadd` by $x \equiv y$ iff $\exists B, \forall n, |x_n - y_n| \leq B$

```
let nadd_eq = new_definition
  `x === y <=> ?B. !n. dist(fn x n,fn y n) <= B`;;

let hreal_tybij =
  define_quotient_type "hreal" ("mk_hreal","dest_hreal") `(===)`;;
```

Step 4: lift all operations and properties from `nadd` to `hreal`

Definition of real numbers in HOL-Light (part 4)

Step 5: lift all operations and properties to `hreal * hreal`

Step 6: quotient of `hreal * hreal` by

```
let treal_eq = new_definition
  `(x1,y1) treal_eq (x2,y2) <=> (x1 + y2 = x2 + y1)`;;

let real_tybij =
  define_quotient_type "real" ("mk_real","dest_real") `(treal_eq)`
```

Step 7: lift all operations and properties to `real`

How to align HOL-Light reals with Coq reals ?

we need to map every axiomatized type used in the construction of `real` to actual Coq type definitions

How to align HOL-Light reals with Coq reals ?

we need to map every axiomatized type used in the construction of `real` to actual Coq type definitions

we need a general translation of subsets and quotients

How to align HOL-Light reals with Coq reals ?

we need to map every axiomatized type used in the construction of `real` to actual Coq type definitions

we need a general translation of subsets and quotients

we need to define an isomorphism between the obtained type and the Coq types of reals that we want to use:

- standard library
- fourcolor library
- mathcomp-analysis library
- corn library

fortunately, all models of real numbers are isomorphic

How to align HOL-Light reals with Coq reals ?

we need to map every axiomatized type used in the construction of `real` to actual Coq type definitions

we need a general translation of subsets and quotients

we need to define an isomorphism between the obtained type and the Coq types of reals that we want to use:

- standard library
- fourcolor library
- mathcomp-analysis library
- corn library

fortunately, all models of real numbers are isomorphic

a theorem already proved in **corn** and **fourcolor**

HOL-Light subsets in Coq

```
Section Subtype.

  Variables (A : Type) (P : A -> Prop) (a : A) (h : P a).

  Definition subtype := {| type := {x : A | P x}; el := exist P a h |}.

  Definition dest : subtype -> A := fun x => proj1_sig x.

  Definition mk : A -> subtype :=
    fun x => COND_dep (P x) subtype (exist P x) (fun _ => exist P a h).

  Lemma dest_mk_aux x : P x -> (dest (mk x) = x).
  Proof.
    intro hx. unfold mk, COND_dep. destruct excluded_middle_informative.
    reflexivity. contradiction.
  Qed.

  Lemma dest_mk x : P x = (dest (mk x) = x).
  Proof.
    apply prop_ext. apply dest_mk_aux.
    destruct (mk x) as [b i]. simpl. intro e. subst x. exact i.
  Qed.

  Lemma mk_dest x : mk (dest x) = x.
  Proof.
    unfold mk, COND_dep. destruct x as [b i]; simpl.
    destruct excluded_middle_informative.
    rewrite (proof_irrelevance _ p i). reflexivity.
    contradiction.
  Qed.

End Subtype.
```

HOL-Light quotients in Coq

```
Section Quotient.

  Variables (A : Type') (R : A -> A -> Prop).

  Definition is_eq_class X := exists a, X = R a.

  Definition class_of x := R x.

  Lemma is_eq_class_of x : is_eq_class (class_of x).
  Proof. exists x. reflexivity. Qed.

  Local Definition a := el A.

  Definition quotient := subtype (is_eq_class_of a).

  Definition mk_quotient : (A -> Prop) -> quotient := mk (is_eq_class_of a).
  Definition dest_quotient : quotient -> (A -> Prop) := dest (is_eq_class_of a).

  Lemma mk_dest_quotient : forall x, mk_quotient (dest_quotient x) = x.
  Proof. exact (mk_dest (is_eq_class_of a)). Qed.

  Lemma dest_mk_aux_quotient : forall x, is_eq_class x -> (dest_quotient (mk_quotient x)
  Proof. exact (dest_mk_aux (is_eq_class_of a)). Qed.

  Lemma dest_mk_quotient : forall x, is_eq_class x = (dest_quotient (mk_quotient x) = x)
  Proof. exact (dest_mk (is_eq_class_of a)). Qed.

End Quotient.
```

fourcolor definition of models of real numbers

```
Record structure : Type := Structure {
   val : Type;                  (* type of real (denotation) values *)
   set := val -> Prop;          (* type of real (denotation) sets *)
   rel := val -> set;           (* type of real (denotation) relations *)
   le : rel;                    (* real order (less than or equal) relation *)
   sup : set -> val;            (* supremum of (nonempty, bounded) real sets *)
   add : val -> val -> val;     (* addition of real values *)
   zero : val;                  (* real zero *)
   opp : val -> val;            (* opposite of a real value *)
   mul : val -> val -> val;     (* multiplication of real values *)
   one : val;                   (* real one *)
   inv : val -> val             (* inverse of a (nonzero) real value *) }.

Definition eq R : rel R := fun x y => le x y /\ le y x.

Record axioms R : Prop := Axioms {
  le_reflexive (x : val R) : le x x;
  le_transitive (x y z : val R) : le x y -> le y z -> le x z;
  sup_upper_bound (E : set R) : has_sup E -> ub E (sup E);
  sup_total (E : set R) (x : val R) : has_sup E -> down E x \/ le (sup E) x;
  add_monotone (x y z : val R) : le y z -> le (add x y) (add x z);
  add_commutative (x y : val R) : eq (add x y) (add y x);
  add_associative (x y z : val R) : eq (add x (add y z)) (add (add x y) z);
  add_zero_left (x : val R) : eq (add (zero R) x) x;
  add_opposite_right (x : val R) : eq (add x (opp x)) (zero R);
  mul_monotone x y z : le (zero R) x -> le y z -> le (mul x y) (mul x z);
  mul_commutative (x y : val R) : eq (mul x y) (mul y x);
  mul_associative (x y z : val R) : eq (mul x (mul y z)) (mul (mul x y) z);
  mul_distributive_right (x y z : val R) : eq (mul x (add y z)) (add (mul x y) (mul x z)
  mul_one_left (x : val R) : eq (mul (one R) x) x;
  mul_inverse_right (x : val R) : ~ eq x (zero R) -> eq (mul x (inv x)) (one R);
  one_nonzero : ~ eq (one R) (zero R) }.

Record model : Type := Model {
  model_structure : structure; model_axioms : axioms model_structure }.
```

fourcolor theorem of categoricity of the theory of reals

```
Record morphism R S (phi : val R -> val S) : Prop := Morphism {
  morph_le x y : le (phi x) (phi y) <-> le x y;
  morph_sup (E : set R) : has_sup E -> eq (phi (sup E)) (sup (image phi E));
  morph_add x y : eq (phi (add x y)) (add (phi x) (phi y));
  morph_zero : eq (phi (zero R)) (zero S);
  morph_opp x : eq (phi (opp x)) (opp (phi x));
  morph_mul x y : eq (phi (mul x y)) (mul (phi x) (phi y));
  morph_one : eq (phi (one R)) (one S);
  morph_inv x : ~ eq x (zero R) -> eq (phi (inv x)) (inv (phi x))
}.

Section CanonicalRealMorphism.
  Variable R S : Real.model.
  ...
  Definition Rmorph_to x := ...
  ...
End CanonicalRealMorphism.

Theorem Rmorph_to_inv (R S : Real.model) x : Rmorph_to R (Rmorph_to S x) == x.
Proof. ... Qed.
```

stdlib reals is a fourcolor model of reals

```
Import Real.

Definition R_struct : structure := {| ... |}.

Lemma R_axioms : axioms R_struct.
Proof.
  apply Axioms.
  apply Rle_refl.
  apply Rle_trans.
  apply Rsup_upper_bound.
  apply Rsup_total.
  apply Rplus_le_compat_l.
  intros x y. rewrite eq_R_struct. apply Rplus_comm.
  intros x y z. rewrite eq_R_struct. rewrite Rplus_assoc. reflexivity.
  intro x. rewrite eq_R_struct. apply Rplus_0_l.
  intro x. rewrite eq_R_struct. apply Rplus_opp_r.
  apply Rmult_le_compat_l.
  intros x y. rewrite eq_R_struct. apply Rmult_comm.
  intros x y z. rewrite eq_R_struct. rewrite Rmult_assoc. reflexivity.
  intros x y z. rewrite eq_R_struct. apply Rmult_plus_distr_l.
  intro x. rewrite eq_R_struct. apply Rmult_1_l.
  intro x. rewrite eq_R_struct. apply Rinv_r.
  rewrite eq_R_struct. apply R1_neq_R0.
Qed.

Definition R_model : model := {|
  model_structure := R_struct;
  model_axioms := R_axioms;
|}.
```

HOL-Light reals is a fourcolor model of reals

```
Definition real_struct : structure := {| ... |}.

Lemma real_axioms : axioms real_struct.
Proof.
  apply Axioms.
  apply REAL_LE_REFL.
  intros x y z xy yz; apply (REAL_LE_TRANS x y z (conj xy yz)).
  apply real_sup_upper_bound.
  apply real_sup_total.
  intros x y z yz; rewrite REAL_LE_LADD; exact yz.
  intros x y. rewrite eq_real_struct. apply REAL_ADD_SYM.
  intros x y z. rewrite eq_real_struct. apply REAL_ADD_ASSOC.
  intro x. rewrite eq_real_struct. apply REAL_ADD_LID.
  intro x. rewrite eq_real_struct. rewrite REAL_ADD_SYM. apply REAL_ADD_LINV.
  intros x y z hx yz. apply REAL_LE_LMUL. auto.
  intros x y. rewrite eq_real_struct. apply REAL_MUL_SYM.
  intros x y z. rewrite eq_real_struct. apply REAL_MUL_ASSOC.
  intros x y z. rewrite eq_real_struct. apply REAL_ADD_LDISTRIB.
  intro x. rewrite eq_real_struct. apply REAL_MUL_LID.
  intro x. rewrite eq_real_struct. rewrite REAL_MUL_SYM. apply REAL_MUL_LINV.
  unfold one, zero. simpl. rewrite eq_real_struct, REAL_OF_NUM_EQ. auto.
Qed.

Definition real_model : model := {|
  model_structure := real_struct;
  model_axioms := real_axioms;
|}.
```

Alignment of the types of reals

```
Require Import fourcolor.realcategorical.

Definition R_of_real := @Rmorph_to real_model R_model.
Definition real_of_R := @Rmorph_to R_model real_model.

Lemma R_of_real_of_R r : R_of_real (real_of_R r) = r.
Proof. rewrite <- eq_R_model. apply (@Rmorph_to_inv R_model real_model). Qed.

Lemma real_of_R_of_real r : real_of_R (R_of_real r) = r.
Proof. rewrite <- eq_real_model. apply (@Rmorph_to_inv real_model R_model). Qed.

Definition mk_real : (prod hreal hreal -> Prop) -> R := fun x => R_of_real (mk_real x).

Definition dest_real : R -> prod hreal hreal -> Prop := fun x => dest_real (real_of_R x)

Lemma axiom_23 : forall (a : R), mk_real (dest_real a) = a.
Proof. intro a. unfold mk_real, dest_real. rewrite axiom_23. apply R_of_real_of_R. Qed.

Lemma axiom_24 : forall (r : prod hreal hreal -> Prop),
  (exists x : prod hreal hreal, r = treal_eq x) = (dest_real (mk_real r) = r).
Proof.
  intro c. unfold dest_real, mk_real. rewrite real_of_R_of_real, <- axiom_24.
  reflexivity.
Qed.
```

problem: we need to use the properties of HOL-Light reals

$\Rightarrow$ we need to interleave translation and mapping (TODO)

Alignment of the theory functions and predicates

```
Lemma real_le_def : Rle = (fun x1 : R => fun y1 : R =>
  @ε Prop (fun u : Prop => exists x1' : prod hreal hreal,
    exists y1' : prod hreal hreal,
      ((treal_le x1' y1') = u) /\ ((dest_real x1 x1') /\ (dest_real y1 y1')))).
Proof.
  apply fun_ext; intro x. apply fun_ext; intro y.
  unfold dest_real. rewrite le_morph_R.
  generalize (real_of_R x); clear x; intro x.
  generalize (real_of_R y); clear y; intro y.
  reflexivity.
Qed.

Lemma real_add_def : Rplus = (fun x1 : R => fun y1 : R =>
  mk_real (fun u : prod hreal hreal => exists x1' : prod hreal hreal,
    exists y1' : prod hreal hreal,
      (treal_eq (treal_add x1' y1') u) /\ ((dest_real x1 x1') /\ (dest_real y1 y1')))).
Proof.
  apply fun_ext; intro x. apply fun_ext; intro y.
  rewrite add_eq. unfold mk_real. apply f_equal. reflexivity.
Qed.

...
```

Conclusion/future work

- need to find a way to interleave translation and mapping for not having to prove the properties of HOL-Light definitions in Coq again
- translate the HOL-Light analysis library to Coq soon (>20,000 theorems!)
- still need to align function definitions on real numbers (e.g. exp, sinus, etc.)