Translating libraries from HOL Light to Lean4

Iván Martínez Comas

Supervisors: Frédéric Blanqui, INRIA (Deducteam)
Patrick Massot, U. Paris-Saclay

My academical background

- M2 Master Logique Mathématique et Fondements de l'Informatique. **U.Paris Cité, 2025**
- BAC+4 Mathématiques. **U. Saragosse, 2023**
- (Erasmus) M1 Math. Fondamentales. **U. Paris Saclay, 2022**

Past Projects:

- **Bachelor Thesis:** A text about the proofs of independence of AC and CH from ZF axioms.
- **M2 Internship at CEA:** Developing an effectul language based on Pure Subtype Theories and study its type-safety.
- **M2 Assignment** Formalization of a mini-compiler in Rocq.

What is a Proof Assistant?

A *proof assistant* is a software for developing and mechanically verifying formal specifications and proofs.

There is no single proof assistant: *many systems coexist*.

What is a Proof Assistant?

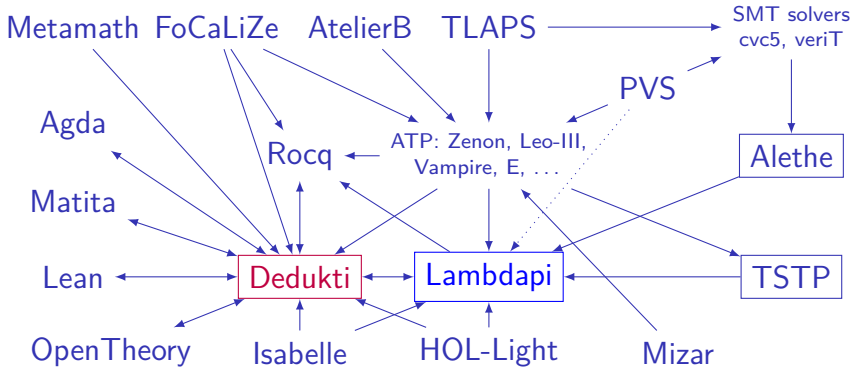
A *proof assistant* is a software for developing and mechanically verifying formal specifications and proofs.

There is no single proof assistant: *many systems coexist*.

- They may be based on different logical foundations or type theories.
- Provide different proof languages, automation mechanisms, and user workflows.
- Have distinct implementations.

As a consequence, formalization tools (and results) are often tied to a particular system.

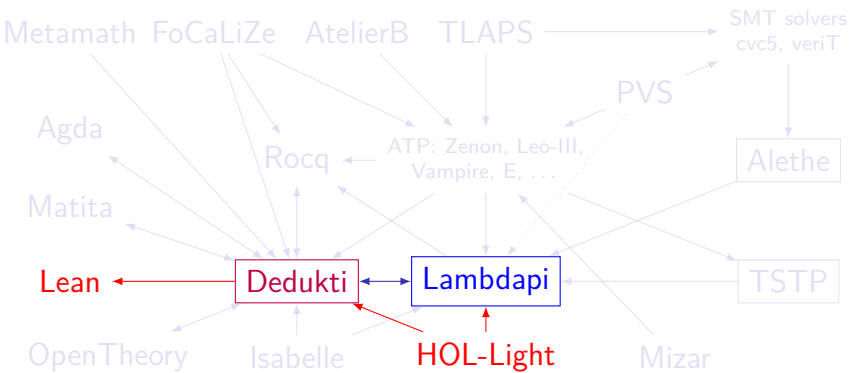
Motivation: The big picture



Lambdapi = Dedukti + implicit arguments/coercions, tactics, ...

`https://github.com/Deducteam/Dedukti`
`https://github.com/Deducteam/lambdapi`

My current goal



HOL Light

- Based on simple type theory (polymorphic λ-calculus).
- Classical higher-order logic (quantification over functions and predicates; types do not depend on terms).
- Heavily relies on Hilbert choice operator ε.
- Proofs are not represented as terms (LCF-style proof kernel).

Lean 4

- Based on dependent Calculus of Inductive Constructions.
- Natively implements dependent types and inductive types.
- Curry–Howard correspondence (proofs as terms, propositions as types).
- Proof irrelevance and extensionality principles (propositional and functional).

The translation

The translation

- Generates a Lambdapi file per proof
- Stores "type–arities"

https://github.com/Deducteam/hol2dk

Translates each generated file to Lean using the auxilary files:

- encoding.lp
- renaming.lp
- mappings.lp
- arities.tvs

HOL Light definition

```
let ADD_SYM = prove
 (`!m n. m + n = n + m`,
  INDUCT_TAC THEN ASM_REWRITE_TAC[ADD_CLAUSES]);;
```

Lean 4 Translation (Only statement)

```
theorem thm_ADD_SYM :  \forall m : Nat, \forall n : Nat,
(ADD m n) = (ADD n m)
```

HOL Light definition

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

HOL Light definition

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

Lean 4 Translation

```
noncomputable def ADD : Nat -> Nat -> Nat :=
@Classical.epsilon (Nat -> Nat -> Nat -> Nat) _
    (fun +' : Nat -> Nat -> Nat -> Nat =>
        \forall _2155 : Nat, (\forall n : Nat, = (+' _2155 0 n) n)
        \and
        (\forall m : Nat, \forall n : Nat, = (+' _2155 (SUC m) n)
        (SUC (+' _2155 m n))))
(NUMERAL (BIT1 (BIT1 (BIT0 (BIT1 (BIT0 (BIT1 _0)))))))
```

Examples: Addition on Natural numbers

```
theorem add_def : Nat.add = ADD := by
  epsilon_tac -- custom tactic to eliminate epsilon
  . lia
  . intros f _ _
    funext _ m _
    induction m <;> grind
```

```
theorem add_def : Nat.add = ADD := by
  epsilon_tac -- custom tactic to eliminate epsilon
  . lia
  . intros f _ _
    funext _ m _
    induction m <;> grind
theorem thm_ADD_SYM : \forall m : Nat, \forall n : Nat,
    (m + n) = (n + m)
```

We call this last step alignment!

The (full) translation

Translates each generated file to Lean using the auxilary files:

- encoding.lp
- renaming.lp
- mappings.lp
- arities.tvs
- **mappings.lean**

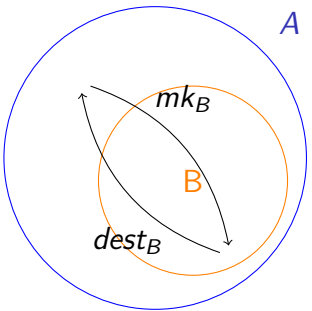
Alignment of types

Given a type A and a predicate P, HOL Light defines the type B of elements in A satisfying P.

Alignment of types

Given a type A and a predicate P, HOL Light defines the type B of elements in A satisfying P.
This generates two functions $\texttt{mk\_B} : A \to B$ and $\texttt{dest\_B} : B \to A$ such that:

- $\forall b : B, (\texttt{mk\_B}\ (\ \texttt{dest\_B}\ b)) = b$
- $\forall a : A, (P\, a = (\texttt{dest\_B}\ (\texttt{mk\_B}\ a) = a))$



In HOL Light, the type of natural numbers is defined under the predicate num_rep

```
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))`;;
```

Where

```
`IND_SUC = @f:ind->ind. ?z. (!x1 x2. (f x1 = f x2) = (x1 = x2)) /\
                                (!x. ~(f x = z))`
`IND_0 = @z:ind. (!x1 x2. IND_SUC x1 = IND_SUC x2 <=> x1 = x2) /\
                  (!x. ~(IND_SUC x = z))`
```

So to prove the equivalence we show that for Lean's natural numbers we have

```
theorem axiom_7 : \forall (a : Nat), (mk_num (dest_num a)) = a
theorem axiom_8 : \forall (i : ind),
                          (NUM_REP i) = ((dest_num (mk_num i)) = i)
```

So to prove the equivalence we show that for Lean's natural numbers we have

```
theorem axiom_7 : \forall (a : Nat), (mk_num (dest_num a)) = a
theorem axiom_8 : \forall (i : ind),
                          (NUM_REP i) = ((dest_num (mk_num i)) = i)
```

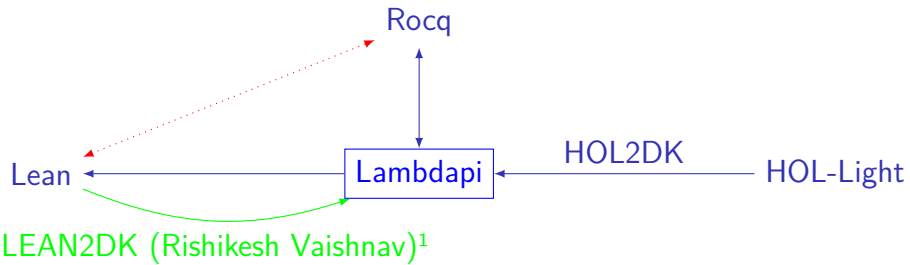
For some suitable definitions in Lean of

```
def ind := Nat
noncomputable def IND_SUC := ...
noncomputable def IND_0 := ...
noncomputable def dest_num : Nat -> ind := ...
noncomputable def mk_num : ind -> Nat := ...
```

Alignments proved so far

- Connectives
- Quotient types
- unit type
- product type
- The type of natural numbers, arithmetical operations, even and odd (natural) numbers, min, max
- Option type.
- Sum type and its functions.
- List type and most of its functions.
- The type of Real numbers and its operations.

Future Work



`https://github.com/Deducteam/lean2dk`

Other future projects:

- A formalization of Balinski's theorem in Lean.
- Translation of verbose-lean to Spanish.