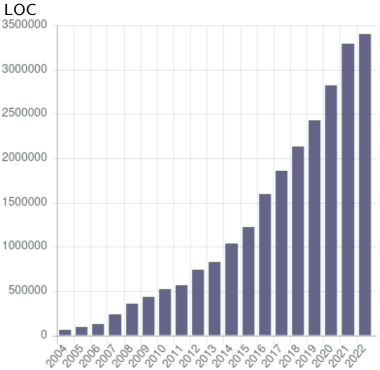
Translating HOL-Light proofs to Coq

Frédéric Blanqui

Libraries of formal proofs today

Library	Nb files	Nb objects*
Coq Opam	35,000	1,200,000
Isabelle AFP	7,500	280,000
Lean Mathlib	4,200	210,000
Mizar Mathlib	1,400	77,000
HOL-Light	635	37,000
...	...	...

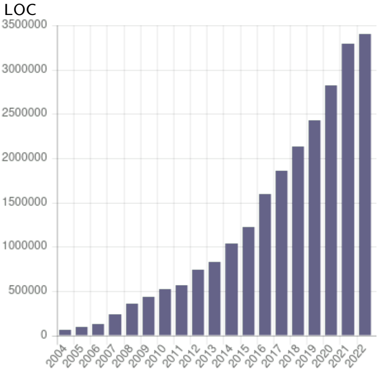
* type, definition, theorem, ...



Libraries of formal proofs today

Library	Nb files	Nb objects*
Coq Opam	35,000	1,200,000
Isabelle AFP	7,500	280,000
Lean Mathlib	4,200	210,000
Mizar Mathlib	1,400	77,000
HOL-Light	635	37,000
...	...	...

* type, definition, theorem, ...



- Every system has its own basic libraries on integers, lists, reals, ...
- Some definitions/theorems are available in one system only and took several man-years to be formalized

Interest of proof system interoperability

- Avoid duplicating developments and losing time
- Facilitate development of new proofs and new systems
- Increase reliability of formal proofs (cross-checking)
- Facilitate validation by certification authorities
- Relativize the choice of a system (school, industry)
- Provide multi-system data to machine learning

Difficulties of proof system interoperability

- Each system is based on different axioms and deduction rules
- It is usually non trivial and sometimes impossible to translate a proof from one system to the other (e.g. a proof using impredicativity or proof irrelevance in a system not allowing these features)

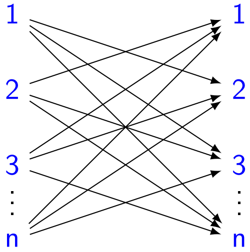
Some one-to-one translation tools

- HOL90 to NuPRL [Howe 1996, statements only]
- HOL98 to Coq [Denney 2000]
- HOL98 to NuPRL [Naumov et al 2001]

Flyspeck project with HOL-Light, Coq and Isabelle/HOL [2003]

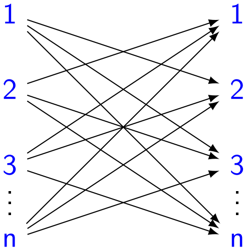
- HOL to Isabelle/HOL [Obua 2006]
- Isabelle/HOL to HOL-Light [McLaughlin 2006]
- HOL-Light to Coq [Wiedijk 2007, no implementation]
- HOL-Light to Coq [Keller & Werner 2010]
- HOL-Light to HOL4 [Kumar 2013]
- HOL-Light to Isabelle [Kaliszyk & Krauss 2013]
- HOL-Light to Metamath [Carneiro 2016]
- HOL4 to Isabelle/HOL [Immler et al 2019]
- Lean3 to Coq [Gilbert 2020]
- Lean3 to Lean4 [Lean community 2021]
- Maude to Lean [Rubio & Riesco 2022]

Interoperability between n systems



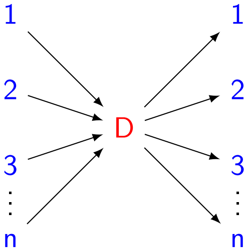
$n(n-1)$ translators

Interoperability between n systems



$n(n-1)$ translators

Can't we be more generic ?



$2n$ translators

The $\lambda\Pi/\mathcal{R}$ approach: encoding features

use **$\lambda\Pi$-calculus modulo rewriting** ($\lambda\Pi/\mathcal{R}$) as pivot language to represent the proofs of various systems **in a modular way**:

- functional pure type systems (Cousineau & Dowek, 2007)
- higher-order logic (Assaf & Burel, 2012)
- universe cumulativity (Thiré, 2015)
- predicate subtyping with proof irrelevance (Hondet, 2020)
- η-equivalence and universe polymorphism (Genestier, 2020)

- **Dedukti** is a type-checker for $\lambda\Pi/\mathcal{R}$
- **Lambdapi** is a Dedukti-compatible proof assistant with additional features (implicit arguments/coercions, tactics, ...)

What is the $\lambda\Pi$-calculus modulo rewriting ($\lambda\Pi/\mathcal{R}$)?

$\lambda\Pi/\mathcal{R}$	$= \lambda$	simply-typed λ-calculus
	$+ \Pi$	dependent types, e.g. Array n
	$+ \mathcal{R}$	identification of types modulo rewrite rules $l \hookrightarrow r$

a theory = a signature Σ + a set of rewrite rules $\mathcal{R}$

What is the $\lambda\Pi$-calculus modulo rewriting ($\lambda\Pi/\mathcal{R}$)?

$\lambda\Pi/\mathcal{R} = \lambda$		simply-typed λ-calculus
$+ \Pi$		dependent types, e.g. Array n
$+ \mathcal{R}$		identification of types modulo rewrite rules $l \hookrightarrow r$

a theory = a signature Σ + a set of rewrite rules $\mathcal{R}$

typing = typing rules of Edinburg's Logical Framework LF

$$+ \quad \frac{\Sigma \vdash t : A \quad A \equiv_{\beta\mathcal{R}} B}{\Sigma \vdash t : B}$$

$\equiv_{\beta\mathcal{R}}$: equational theory generated by β and $\mathcal{R}$

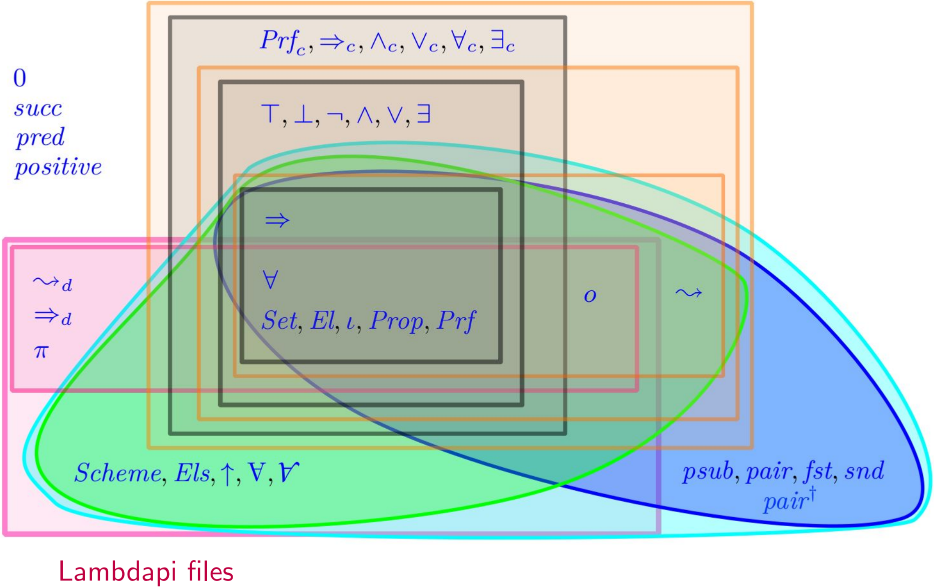
example:

concat : $\Pi p : \mathbb{N}, \text{Array } p \to \Pi q : \mathbb{N}, \text{Array } q \to \text{Array}(p + q)$

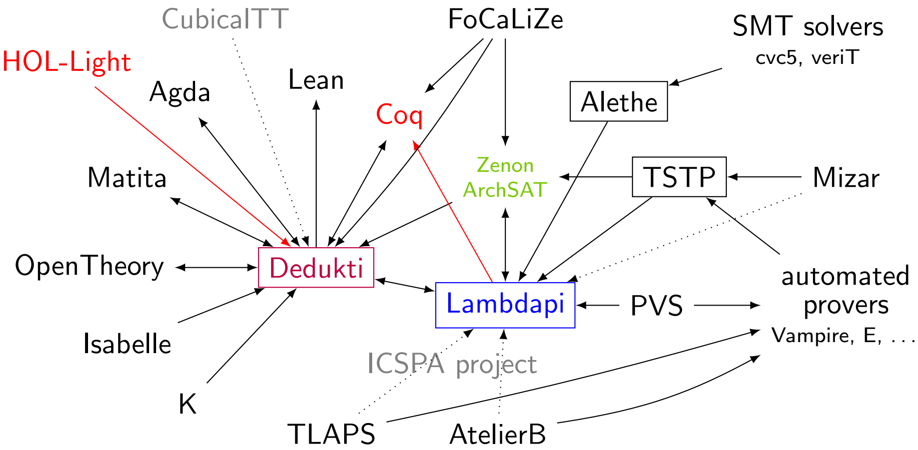
concat 2 a 3 b : $\text{Array}(2 + 3) \equiv_{\beta\mathcal{R}} \text{Array}(5)$

The modular $\lambda\Pi/\mathcal{R}$ theory U and its sub-theories

(43 symbols, 31 rules)



Dedukti, an assembly language for proof systems



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

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

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)

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

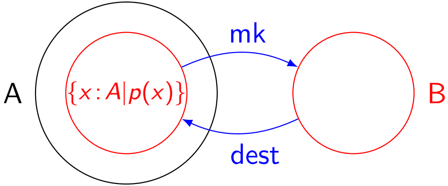
Term and type definitions in HOL-Light

- 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
==>	$\Rightarrow$	->
/\	$\wedge$	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 $\exists$`a.p a`
  - a typed constant `mk:A->B`
  - a typed constant `dest:B->A`
  - an axiom $\forall$`b:B.mk(dest b) = b`
  - an axiom $\forall$`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 $\forall$`b:B', mk(dest b) = b`
  - prove $\forall$`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