Translating proofs from HOL to Coq

Theoretical and practical aspects

Chantal Keller and Benjamin Werner

Ecole Polytechnique & INRIA

What are mathematics ?

~~The Bodensee is beautiful~~

Everyone in Baden loves Dampfnudeln, Markus is in Baden, *thus* Markus loves Dampfnudeln.

syntax !

Proof-system :

- detail the proof up to primitive logical rules
- have it checked by the machine

Proof-system

- A formalism : language, logical rules.
- A software : for manipulating, checking, storing, building proofs.
- A proof language.
- A library : mathematical corpus.

Similar to a programming language + a compiler :

formalism = abstract syntax

proof language concrete syntax

Concrete syntax

Both systems use a proof language made of *tactics*.
They have a common ancestor : LCF

Thus, the proof languages bear some similarities, but are undoubtedly different (say like Java and C).

```
 Lemma subst_idt_lift_term : forall j u i,
    subst_idt (lift_term u i j) S = lift_term (subst_idt u S) i j.
  Proof.
    move => j; elim => [n|x X|[C|C||||||C|C]|c C|t IHt u IHu|A t
IHt] //=
      i.
    - by case: (_ <= _).
    - by rewrite IHt IHu.
    - by rewrite IHt.
  Qed.
```

Concrete syntax

Both systems use a proof language made of *tactics*.
They have a common ancestor : LCF

Thus, the proof languages bear some similarities, but are undoubtedly different (say like Java and C).

```
let EQ_MULT_LCANCEL = prove
 (`!m n p. (m * n = m * p) <=> (m = 0) \/ (n = p)`,
  INDUCT_TAC THEN REWRITE_TAC[MULT_CLAUSES; NOT_SUC] THEN
  REPEAT INDUCT_TAC THEN
  ASM_REWRITE_TAC[MULT_CLAUSES; ADD_CLAUSES; GSYM NOT_SUC; NOT_SUC]
THEN
  ASM_REWRITE_TAC[SUC_INJ; GSYM ADD_ASSOC; EQ_ADD_LCANCEL]);;
```

Diversity : for the worst or the best ?

- Many proof-systems; all incompatible. The common language of mathematics seems lost.
- Each proofs-system has its strengths :

  Coq : good for computations (four-color theorem, primality, but also specific design considerations for algebra...)

  HOL : good for classical analysis. Jordan curve theorem, prime number theorem...

HOL / HOL-light

Formalism : Church's Higher-Order logic

Objects : simply typed lambda-calculus (expressions with binders)

Proofs :

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B}$$

- No computations in the language (almost)
- The proofs are not stored

How can we trust them ?

Architecture of the HOL checker

HOL is implemented in ML; in the implementation :
$\Gamma \vdash A$: `thm`

All the functions allowing objects of type `thm` are simple and carefully checked : they correspond to logical steps.

If we trust these functions, we trust HOL.

Coq

Formalism : type theory

$$\frac{\Gamma \vdash p : A \quad \Gamma \vdash q : B}{\Gamma \vdash (p,q) : A \wedge B}$$

proofs are objects, proofs are kept - they can be re-checked

Objects are functional typed programs - with a very powerful type system.

Programs and functions

An example : addition

Coq	HOL
Define a function such that: $0+m \triangleright m$ $S(n)+m \triangleright S(n+m)$	Prove the existence of a function such that: $0+m = m$ $S(n)+m = S(n+m)$

Computational proofs

Predicate `P : nat -> Prop`

1. Prove `P n` in a standard way (tactics...)
2. Prove `P n` using computation

- Certificate:

```
certif:  nat -> Type
```

- Checker:

```
f:  forall n, certif n -> bool
```

such that

```
forall c n, f n (c n) = true -> P n
```

Certificate: HOL Light proof term...

Translation

- Translating the «concrete» syntax: unrealistic, unreliable, fragile.

  we have to translate the statements in the first place

- Translating the «abstract syntax» : Logical embedding

  HOL ⊂ Type Theory

  Two kinds of logical embedding : *deep and shallow*

Embedding HOL in type theory

These functions are defined outside of the formalisms

Shallow embedding

objects			$t \mapsto \|t\|$
propositions			$P \mapsto \|P\|$
proofs: if	$\Gamma \vdash P$	then	$\|\Gamma\| \vdash \|P\|$

Shallow embedding

- Example of the simply typed λ-calculus in Coq:

```
Definition type := Type.
```

- Representation of $\lambda x : bool.x$:

```
fun x:  bool => x
```

Embedding HOL in type theory

Deep embedding

Represent HOL in a datatype of type theory

«speak about» HOL in type theory

- Example of the simply typed λ-calculus in Coq:

```
Inductive type :  Type :=
   | B : type
   | A : type -> type -> type.
Inductive term :  Type :=
   | Var:  string -> term
   | Lam:  string -> type -> term -> term
   | App:  term -> term -> term.
```

- Representation of $\lambda x : bool.x$:

```
Lam "x" B (Var "x")
```

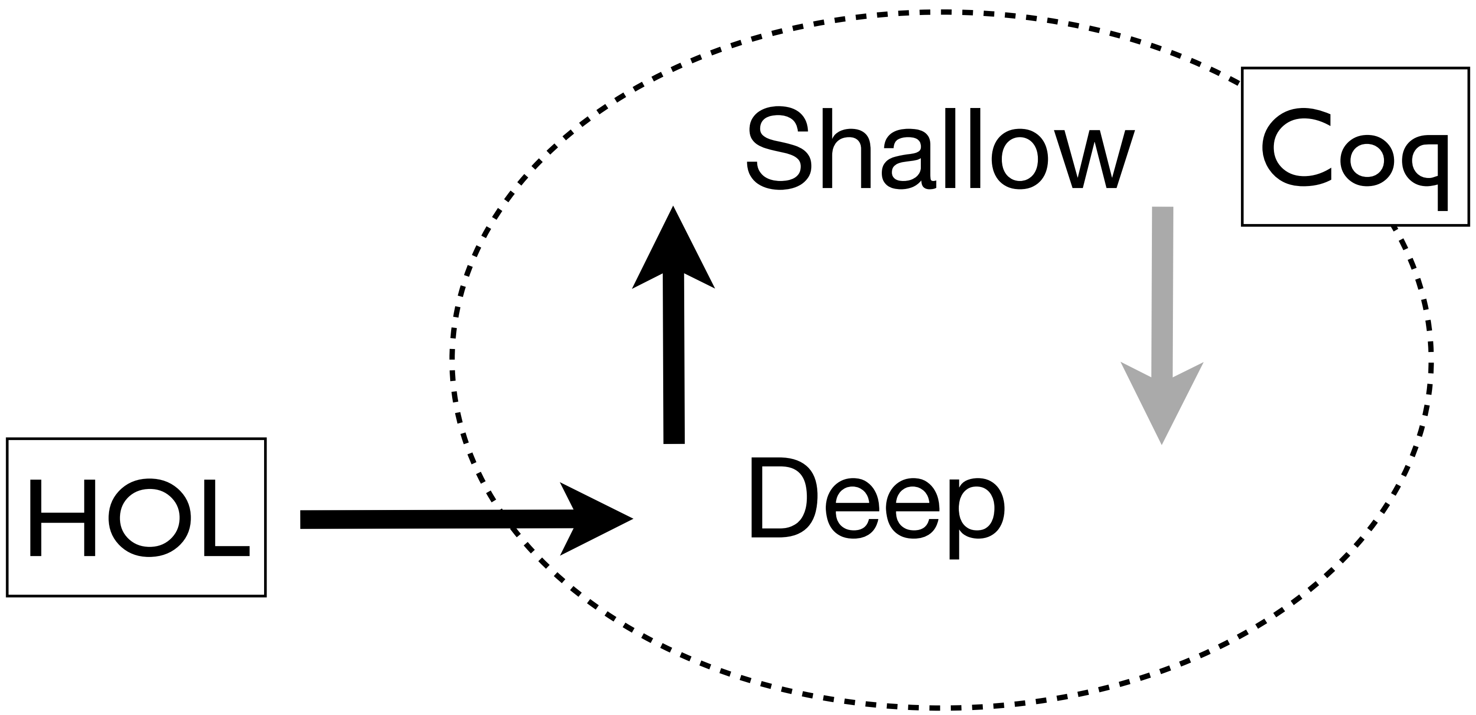
The trick

Shallow

Deep

Type theory allows lifting deep from shallow encoding (various work, from Martin-Löf to Garrillot & Werner, 2007)

The trick



The encoding is the interface between the two systems

Encoding : types

```
Inductive type : Type :=
| TVar : idT → type | Bool : type
| Arrow : type → type → type
| TDef : defT → list_type → type
with list_type : Type :=
| Tnil : list_type
| Tcons : type → list_type → list_type.
```

Encoding : terms

```
Inductive term : Type :=
| Dbr : nat → term | Var : idV → type → term
| Cst : cst → term | Def : defV → type → term
| App : term → term → term
| Abs : type → term → term.
```

Lifting to Coq

$$\text{if } \Gamma \vdash \underbrace{t}_{\texttt{term}} : \underbrace{T}_{\texttt{type}} \text{ then } |t|_{\mathcal{I}} \in [T]_{\mathcal{I}}$$

```
Record type_translation : Type :=
  mkTT {ttrans :> Type; tinhab : ttrans}.
```

```
tr_type: forall ℐ, type → type_translation
```

```
sem_term : context → term →
  option {ty: type & forall ℐ, tr_type ℐ ty}
```

Modelling the proofs

```
Inductive proof : Type :=
| Prefl : term → proof
| Pconj : proof → proof → proof
| Pconjunct1 : proof → proof
| Pconjunct2 : proof → proof
| ...
```

A function

`check: term → proof → bool`

such that if `(check t p)=true` then :

- `t` is a well-formed proposition / boolean
- `p` is a proof of `t`

A function

`check: term → proof → bool`

such that if `(check t p)=true` then :

- `t` is a well-formed proposition / boolean
- `p` is a proof of |`t`|
- this entails that |`t`| is true - *in Coq*

Nice point :

|`t`| is a "real" Coq theorem : it is intelligible

```
Theorem hollight_MOD_EQ_0_thm :
  forall x x0 : N, x0 <> 0 →
    x0 | x = (exists a : N, x = a * x0).

Notation "a|b" := (Nmod b a = 0).
```

Status of definitions in the two systems

`Definition four := 4.`

In HOL :

new object : four : N

new lemma : four = 4

In Coq :

new object : four : nat

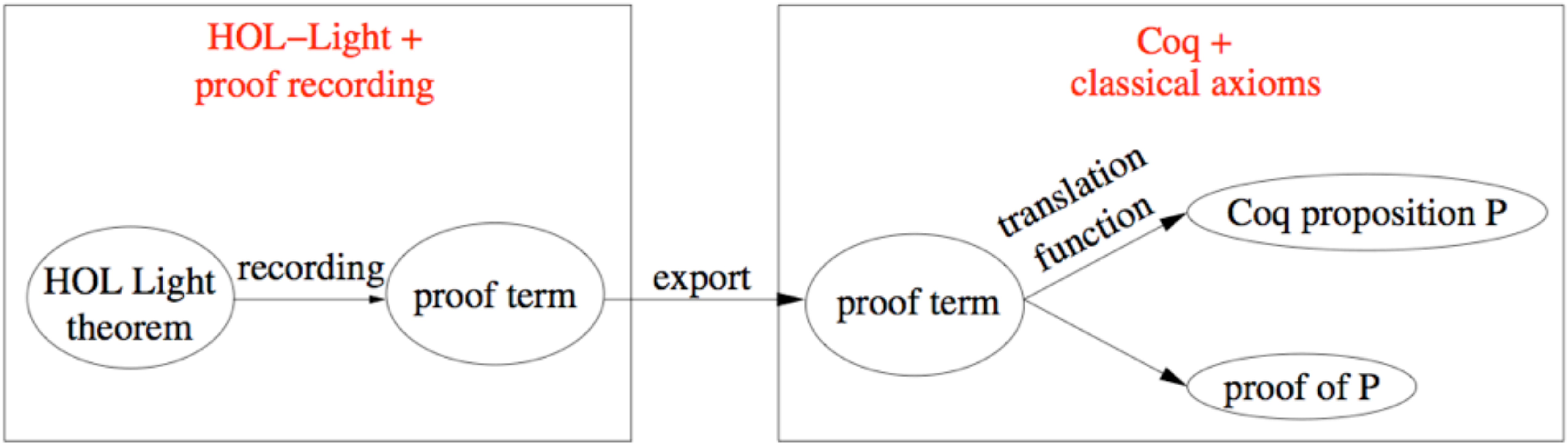
new rule : four $\rhd$ 4

Recording HOL-light proofs

The type proof is a pure data-type; we can :

- define its twin in ML, in the HOL-light implementation
- instrument the basic tactics so that they construct the proof-tree on the fly (reuse code of S. Obua and now from the OpenTheory projet)
- export these proof-trees to Coq by straightforward pretty-printing

The bottleneck becomes the size of these proof-trees (as expected)
We introduce new lemmas for sharing.



$$\text{CONJ\_SYM} \quad : \quad \forall\ t_1\ t_2.\ t_1 \wedge t_2 \Leftrightarrow t_2 \wedge t_1$$

$$\textbf{forall}\ \ x\ x0\ :\ \textbf{Prop},\ (x\ \wedge x0)\ =\ (x0\ \wedge x)$$

The bottleneck becomes the size of these proof-trees (as expected)

We introduce new lemmas for sharing.

Bench.	Number		Time		
	Theorems	Lemmas	Rec.	Exp.	Comp.
Stdlib	1,726	195,317	2 min 30	6 min 30	10h
Model	2,121	322,428	6 min 30	29 min	44h
Vectors	2,606	338,087	6 min 30	21 min	39h

Bench.	Memory		
	H.D.D.	Virt. OCaml	Virt. Coq
Stdlib	218 Mb	1.8 Gb	4.5 Gb
Model	372 Mb	5.0 Gb	7.6 Gb
Vectors	329 Mb	3.0 Gb	7.5 Gb

Substantial gains expected in a reasonable close future

What about classical logic ?

HOL is inherently classical :

- excluded middle
- Hilbert's ε choice operator

We have no choice : we need to add classical axioms to Coq

Conclusion

- Translation and cooperation between proof-systems can work, sometimes.
- Allows re-using but also re-checking of HOL proofs in Coq
- Relies on work specific to the two involved formalisms.
- Nice point : the translated theorems are intelligible and reusable.
- Efficiency and memory consumptation remains an issue; currently some further progress by using Coq arrays and switching to OpenTheory
- Mathematical proofs as massive date; a flavour of the future ?