Importing HOL-Light into Coq
Deep and shallow embeddings of the higher order logic into Coq
Work in progress

Chantal Keller
chantal.keller@ens-lyon.fr
ÉNS Lyon - INRIA Saclay - LIX

Bejamin Werner
benjamin.werner@inria.fr

2009 Types meeting
Introduction

What:

- long term: importing HOL-Light theorems and proofs into Coq
- short term:
  - encoding the Higher Order Logic into Coq
  - defining and exporting HOL-Light proof terms

Why:

- theoretical interest
- analysis libraries
- verification of HOL-Light into Coq
Double embedding

Deep embedding (data-type to represent types and terms):
- reasoning by induction over the structure

Shallow embedding (using Coq types and terms):
- using the Coq features
Double embedding

Deep embedding (data-type to represent types and terms):
- reasoning by induction over the structure

Shallow embedding (using Coq types and terms):
- using the Coq features
- obtaining Coq propositions
Double embedding

Deep embedding (data-type to represent types and terms):
- reasoning by induction over the structure
- simple
- compact

Shallow embedding (using Coq types and terms):
- using the Coq features
- obtaining Coq propositions
Double embedding

Deep embedding (data-type to represent types and terms):
- reasoning by induction over the structure
- simple
- compact

Shallow embedding (using Coq types and terms):
- using the Coq features
- obtaining Coq propositions

→ translation function from deep to shallow
Idea

HOL-Light + proof recording

HOL-Light theorem

encoded proof term

Coq + classical axioms

Coq proposition P

proof of P

translation function

export
1. A short presentation of HOL-Light

2. Embedding the higher order logic into Coq

3. Recording and exporting HOL-Light proof terms

4. Conclusion and perspectives
Part I

A short presentation of HOL-Light
HOL-Light:  
- proof assistant written by John Harrison et al.  
- in an OCaml top-level  
- higher order classical logic  
- automated tools and pre-proved theorems  
- programmable without compromising soundness  
- simpler logical kernel than HOL
Types and terms

Logical framework:

- simply-typed $\lambda$-calculus
- terms and type variables and constants
- polymorphism: type schemes
- all the types must be inhabited
- theorem: term of type $\text{bool}$ under the hypotheses of other terms of type $\text{bool}$
- no proof terms

Example:

\[
\vdash \lambda x:A. \ ?y:A. \ x = y
\]
Remarks

Example of an inference rule:

\[ \Gamma \vdash p \iff q \quad \Delta \vdash p \]
\[ \frac{}{\Gamma \cup \Delta \vdash q} \]

where \( \iff \) is \( =_{\text{bool}} \)

Constants:

- main type constants: \( \text{bool} \) and \( \to \)
- main term constants: \( = : A \to A \to \text{bool} \) and \( \varepsilon : (A \to \text{bool}) \to A \) (choice operator)
- possibility to define new constants
Part II

Embedding the higher order logic into Coq
Embedding the higher order logic into Coq

Presentation

What we want:

- deep and shallow embeddings
- translation function from deep to shallow
- HOL-Light inference rules
- proof of correctness of these inference rules with respect to semantics

Carrying out:

- inductive Coq data-types `type` and `term`
- a translation function `sem_term` that maps any `term` of `type` `Bool` onto a term of type `Prop` (in particular)
- inductive data-type `deriv : set term → term → Prop`
- a proof of: `forall G p, deriv G p → has_sem G → has_sem p`
**Types**

**Inductive data-type** \textbf{type}:

<table>
<thead>
<tr>
<th>\text{Type}</th>
<th>\text{Expression}</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{Bool} \in \text{type}</td>
<td>\text{Num} \in \text{type}</td>
</tr>
<tr>
<td>\text{TVar} X \in \text{type}</td>
<td>\text{X} \in \text{idT}</td>
</tr>
<tr>
<td>\text{C} \in \text{defT}</td>
<td>\text{T}_1, \ldots, \text{T}_n \in \text{type}</td>
</tr>
<tr>
<td>\text{TDef} \text{C} [\text{T}_1; \ldots; \text{T}_n] \in \text{type}</td>
<td>\text{A, B} \in \text{type}</td>
</tr>
<tr>
<td>\text{A} \rightarrow \text{B} \in \text{type}</td>
<td></td>
</tr>
</tbody>
</table>

idT, defT two sets.
Embedding the higher order logic into Coq

**Constants**

Inductive data-type `cst`:

- `Hand ∈ cst`
- `Hor ∈ cst`
- `Himp ∈ cst`
- `Hnot ∈ cst`
- `Htrue ∈ cst`
- `Hfalse ∈ cst`
- `A ∈ type`
- `A ∈ type`
- `A ∈ type`
- `A ∈ type`
Terms

Inductive data-type term:

\[
\begin{align*}
\text{Cst} & \quad c \in \text{cst} & \text{Cst} & \quad c \in \text{term} \\
\text{Dbr} & \quad n \in \mathbb{N} & \text{Dbr} & \quad n \in \text{term} \\
\text{Var} & \quad x \in \text{idV} & \text{Var} & \quad A \in \text{type} \\
\text{Def} & \quad c \in \text{defV} & \text{Def} & \quad C \in \text{type} \\
\text{App} & \quad u, v \in \text{term} & \text{App} & \quad u, v \in \text{term} \\
\text{Abs} & \quad A \in \text{type} & \text{Abs} & \quad u \in \text{term} \\
\end{align*}
\]

idV, defV two sets
Translation

General idea:

- types: interface between syntax and semantics
- translation of a type: $|T|$

|A → B| ≡ |A| → |B|

- translation of a term (using dependent types):

$$\forall t, T, \ t : T \rightarrow |T|$$
Translation

General idea:

- **types**: interface between syntax and semantics
- **translation of a type**: \(| T | \)

\[ |A \rightarrow B| \equiv |A| \rightarrow |B| \]

- **translation of a term (using dependent types)**:

\[ \forall t, T, \ t : T \rightarrow |T| \]
Translation

General idea:

- types: interface between syntax and semantics
- translation of a type: $|T|$?

$$|A \rightarrow B|? \equiv |A|? \rightarrow |B|?$$

- translation of a term (using dependent types):

$$\forall t, T, \; t : T \rightarrow |T|?$$

- a De Bruijn context
- interpretation functions for variables and definitions
Translation

General idea:

- types: interface between syntax and semantics
  
- translation of a type: $|T| ?$

\[ |A \rightarrow B| ? \equiv |A| ? \rightarrow |B| ? \]

- translation of a term (using dependent types):

\[ \forall t, T, \ t : T \rightarrow |T| ? \]

- a De Bruijn context
- interpretation functions for variables and definitions
- Code
Inference rules

General idea:

- inductive data-type \( \text{deriv} : \text{set} \ \text{term} \rightarrow \text{term} \rightarrow \text{Prop} \)
- a proof of: \( \text{forall} \ G \ p, \ \text{deriv} \ G \ p \rightarrow \text{has}_\text{sem} \ G \rightarrow \text{has}_\text{sem} \ p \)
- \( \text{has}_\text{sem} \ p \):
  - \( p \) is locally closed
  - \( p : \text{Bool} \)
  - the translation of \( p \) is a correct proposition
Inference rules

General idea:

- inductive data-type \( \text{deriv} : \text{set} \; \text{term} \rightarrow \text{term} \rightarrow \text{Prop} \)
- a proof of: \( \forall G \; p, \; \text{deriv} \; G \; p \rightarrow \text{has}_\text{sem} \; G \rightarrow \text{has}_\text{sem} \; p \)
- \( \text{has}_\text{sem} \; p \):
  - \( p \) is locally closed
  - \( p : \text{Bool} \)
  - the translation of \( p \) is a correct proposition

- Code
Example

\[ \vdash \forall x : A. \exists y : A. x = A y \]

\[
\begin{align*}
\vdash x &= A x \\
\vdash \exists y : A. x &= A y \\
\vdash \forall x : A. \exists y : A. x &= A y
\end{align*}
\]

REFL \(x\)

EXISTS \(\exists y : A. x = A y\) \(x\)

GEN \(x\)
Example

|− !x:A. ?y:A. x = y

\[
\begin{align*}
\Gamma & : \vdash x =_A x \\
\vdash \exists y : A. x =_A y & \quad \text{EXISTS } \exists y : A. x =_A y \quad x' \\
\vdash \forall x : A. \exists y : A. x =_A y & \quad \text{GEN } \forall x : A. \exists y : A. x =_A y
\end{align*}
\]

Code
Part III

Recording and exporting HOL-Light proof terms
Proof-recording system by S. Obua

Challenge:
- compact proofs
- short recording time

Solution:
- granularity

Statistics:
- recording the basic HOL-Light proofs (1694 theorems): 3 min
Exporting

Challenge:

- small files
- small number of files

Solution:

- sharing (proofs, types and terms...)

Statistics:

- exporting the basic HOL-Light proofs (1694 theorems):
  - 14 min
  - 191652 '.v' files
  - 2.2 Gb
Part IV

Conclusion and perspectives
Conclusion

HOL-Light:

- recording proof terms
- export proofs

Coq:

- Coq representation of HOL-Light data-types
- standard lemmas (substitution...)
- translation function
- Coq representation of HOL-Light inference rules
- proof of correctness
Perspectives:

- finish the interface and the proofs
- deal with inhabited types, definitions, axioms
- more efficient Coq data-types
- more efficient exportation and smaller proof terms
- user interface
- scaling up
Thank you for your attention!

Any questions?