

Importation de preuves HOL-Light en Coq

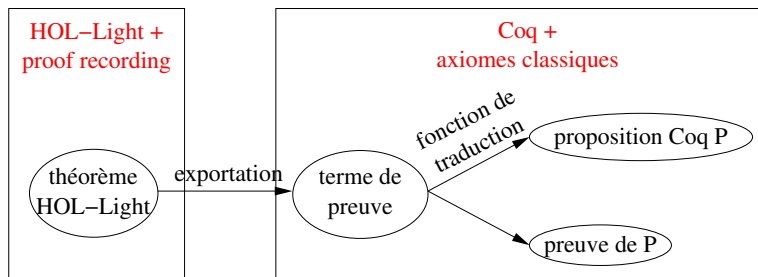
Stage de M2

Chantal Keller Benjamin Werner
chantal.keller@ens-lyon.fr benjamin.werner@inria.fr

ÉNS Lyon - INRIA Saclay - LIX

6 juillet 2009

Idée



Plan

- 1 Présentation de HOL-Light
 - Présentation
 - Règles d'inférence et connecteurs logiques
- 2 Modèle de HOL-Light en Coq
 - Plongements
 - Traduction
 - Dérivations
 - Termes de preuve
- 3 Enregistrement et exportation des termes de preuve de HOL-Light
- 4 Conclusion et perspectives



Première partie I

Présentation de HOL-Light



HOL-Light

HOL-Light :

- assistant de preuve écrit par John Harrison et al.
- dans un top-level OCaml
- logique classique d'ordre supérieur
- outils automatiques, bibliothèque étendue
- programmable sans compromettre la cohérence
- noyau logique simple et concis



Types et termes

Structure logique :

- λ -calcul simplement typé
- variables et constantes de termes et de types
- polymorphisme : schémas de types
- tous les types sont habités
- théorème : terme de type bool sous l'hypothèse d'autres termes de type bool
- pas de termes de preuve

Exemple : $\vdash !x:A. ?y:A. x = y$



Remarques

Exemple de règle d'inférence :

$$\frac{\Gamma \vdash p \Leftrightarrow q \quad \Delta \vdash p}{\Gamma \cup \Delta \vdash q}$$

où \Leftrightarrow représente $=_{\text{bool}}$.

Constantes de base :

- constantes de type : `bool` et `->`
- constante de terme : `= : A -> A -> bool`
- possibilité de définir de nouvelles constantes (connecteurs logiques, ε (opérateur de choix))



Règles du premier ordre

$$\frac{}{\vdash t = t} \text{ REFL}$$

$$\frac{}{\{p\} \vdash p} \text{ ASSUME}$$

$$\frac{}{\vdash (\lambda x. t) x = t} \text{ BETA}$$

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

$$\frac{\Gamma \vdash p \Leftrightarrow q \quad \Delta \vdash p}{\Gamma \cup \Delta \vdash q} \text{ EQ_MP}$$

$$\frac{\Gamma \vdash p \quad \Delta \vdash q}{(\Gamma - \{q\}) \cup (\Delta - \{p\}) \vdash p \Leftrightarrow q} \text{ DEDUCT_ANTISYM}$$



Règles du second ordre et de substitution

$$\frac{\Gamma \vdash s = t}{\Gamma \vdash (\lambda x. s) = (\lambda x. t)} \text{ ABS}$$

$$\frac{\Gamma \vdash s = t \quad \Delta \vdash u = v}{\Gamma \cup \Delta \vdash s \ u = t \ v} \text{ MK_COMB}$$

$$\frac{\Gamma(x_1, \dots, x_n) \vdash p[x_1, \dots, x_n]}{\Gamma(t_1, \dots, t_n) \vdash p[t_1, \dots, t_n]} \text{ INST}$$

$$\frac{\Gamma(\alpha_1, \dots, \alpha_n) \vdash p[\alpha_1, \dots, \alpha_n]}{\Gamma(\gamma_1, \dots, \gamma_n) \vdash p[\gamma_1, \dots, \gamma_n]} \text{ INST_TYPE}$$

Pour ABS : $x \notin FV(\Gamma)$.



Connecteurs logiques

$$\begin{aligned} \top &\triangleq (\lambda p.p) = (\lambda p.p) \\ \wedge &\triangleq \lambda p q.(\lambda f.f p q) = (\lambda f.f \top \top) \\ \Rightarrow &\triangleq \lambda p q.p \wedge q \Leftrightarrow p \\ &\vdots \end{aligned}$$

$$\forall, \exists, \vee, \perp, \neg$$



Deuxième partie II

Modèle de HOL-Light en Coq



Plongements

Plongement profond (types de données pour représenter les types et les termes) :

- raisonnement par induction sur la structure des objets

Plongement peu profond (en utilisant les types et les termes de Coq)

- facilités de Coq



Plongements

Plongement profond (types de données pour représenter les types et les termes) :

- raisonnement par induction sur la structure des objets

Plongement peu profond (en utilisant les types et les termes de Coq)

- facilités de Coq
- obtenir des propositions Coq



Plongements

Plongement profond (types de données pour représenter les types et les termes) :

- raisonnement par induction sur la structure des objets
- simple
- compact

Plongement peu profond (en utilisant les types et les termes de Coq)

- facilités de Coq
- obtenir des propositions Coq



Plongements

Plongement profond (types de données pour représenter les types et les termes) :

- raisonnement par induction sur la structure des objets
- simple
- compact

Plongement peu profond (en utilisant les types et les termes de Coq)

- facilités de Coq
- obtenir des propositions Coq

↔ fonction de traduction de profond vers peu profond

Types

Inductif **type** :

$$\begin{array}{c}
 \hline
 \text{Bool} \in \mathbf{type} \\
 \hline
 \\
 \hline
 C \in \text{defT} \quad T_1, \dots, T_n \in \mathbf{type} \\
 \hline
 \text{TDef } C [T_1; \dots; T_n] \in \mathbf{type}
 \end{array}
 \qquad
 \begin{array}{c}
 \hline
 \text{Num} \in \mathbf{type} \\
 \hline
 \\
 \hline
 A, B \in \mathbf{type} \\
 \hline
 A \longrightarrow B \in \mathbf{type}
 \end{array}
 \qquad
 \begin{array}{c}
 X \in \text{idT} \\
 \hline
 \text{TVar } X \in \mathbf{type}
 \end{array}$$

idT , defT deux ensembles (eqType).



Termes

Inductif **term** :

$$\frac{c \in \mathbf{cst}}{\mathbf{Cst } c \in \mathbf{term}}$$

$$\frac{n \in \mathbb{N}}{\mathbf{Dbr } n \in \mathbf{term}}$$

$$\frac{x \in \mathbf{idV} \quad A \in \mathbf{type}}{\mathbf{Var } x A \in \mathbf{term}}$$

$$\frac{c \in \mathbf{defV} \quad C \in \mathbf{type}}{\mathbf{Def } c C \in \mathbf{term}}$$

$$\frac{u, v \in \mathbf{term}}{\mathbf{App } u v \in \mathbf{term}}$$

$$\frac{A \in \mathbf{type} \quad u \in \mathbf{term}}{\mathbf{Abs } A u \in \mathbf{term}}$$

\mathbf{idV} , \mathbf{defV} deux ensembles (\mathbf{eqType}).



Constantes

Inductif **cst** :

$$\frac{}{\text{Hand} \in \mathbf{cst}}$$

$$\frac{}{\text{Hor} \in \mathbf{cst}}$$

$$\frac{}{\text{Himp} \in \mathbf{cst}}$$

$$\frac{}{\text{Hnot} \in \mathbf{cst}}$$

$$\frac{}{\text{Htrue} \in \mathbf{cst}}$$

$$\frac{}{\text{Hfalse} \in \mathbf{cst}}$$

$$\frac{A \in \mathbf{type}}{\text{Heq } A \in \mathbf{cst}}$$

$$\frac{A \in \mathbf{type}}{\text{Heps } A \in \mathbf{cst}}$$

$$\frac{A \in \mathbf{type}}{\text{Hforall } A \in \mathbf{cst}}$$

$$\frac{A \in \mathbf{type}}{\text{Hexists } A \in \mathbf{cst}}$$



Types

Types : interface entre syntaxe et sémantique.

Traduction d'un type : $|T|$

$$|\text{Arrow } A \ B| \quad \equiv \quad |A| \rightarrow |B|$$



Types

Types : interface entre syntaxe et sémantique.

Traduction d'un type : $|T|$

$$|\text{Arrow } A \ B| \quad \equiv \quad |A| \rightarrow |B|$$

Types habités ?



Types

Types : interface entre syntaxe et sémantique.

Traduction d'un type : $|T|$

$$|\text{Arrow } A \ B| \quad \equiv \quad |A| \rightarrow |B|$$

Types habités ?

`Record type_translation` : `Type` := { `ttrans` :> `Type`; `tinhab` : `ttrans` }. [Wie07]



Types

Types : interface entre syntaxe et sémantique.

Traduction d'un type : $|T|$

$$|\text{Arrow } A \ B| \quad \equiv \quad |A| \rightarrow |B|$$

$$|\text{TVar } X| \quad \equiv \quad ?$$

Types habités ?

`Record type_translation` : `Type` := { ttrans :> Type; tinhab : ttrans }. [Wie07]



Types

Types : interface entre syntaxe et sémantique.

Traduction d'un type : $|T|_{\mathcal{I}}$

$$|\text{Arrow } A \ B|_{\mathcal{I}} \equiv |A|_{\mathcal{I}} \rightarrow |B|_{\mathcal{I}}$$

$$|\text{TVar } X|_{\mathcal{I}} \equiv \mathcal{I}(X)$$

Types habités ?

`Record type_translation` : `Type` := { ttrans :> `Type`; tinhab : ttrans }. [Wie07]



Types

Types : interface entre syntaxe et sémantique.

Traduction d'un type : $|T|_{\mathcal{I}}$

$$|\text{Arrow } A \ B|_{\mathcal{I}} \equiv |A|_{\mathcal{I}} \rightarrow |B|_{\mathcal{I}}$$

$$|\text{TVar } X|_{\mathcal{I}} \equiv \mathcal{I}(X)$$

Types habités ?

Record `type_translation` : `Type` := { `ttrans` :> `Type`; `tinhab` : `ttrans` }. [Wie07]

Definition `type_context` := `idT` -> `type_translation`.

Definition `type_definition_tr` := `defT` -> `list Type` -> `type_translation`.



Types

Types : interface entre syntaxe et sémantique.

Traduction d'un type : $|T|_{\mathcal{I}}$

$$|\text{Arrow } A \ B|_{\mathcal{I}} \equiv |A|_{\mathcal{I}} \rightarrow |B|_{\mathcal{I}}$$

$$|\text{TVar } X|_{\mathcal{I}} \equiv \mathcal{I}(X)$$

Types habités ?

Record `type_translation` : `Type` := { `ttrans` :> `Type`; `tinhab` : `ttrans` }. [Wie07]

Definition `type_context` := `idT` -> `type_translation`.

Definition `type_definition_tr` := `defT` -> list `Type` -> `type_translation`.

Fixpoint `tr_type` (`tc`: `type_context`) (`tdt`: `type_definition_tr`) (`ty`: `type`) : `Type`

:= ...



Termes

Utilisation des types dépendants. [GW07]

Traduction d'un terme :

si $\Gamma \vdash t : T$, alors $|t|_{\mathcal{I}} : |T|_{\mathcal{I}}$

- si t non typable, pas de sémantique ;
- Γ : contexte de De Bruijn.



Termes

Utilisation des types dépendants. [GW07]

Traduction d'un terme :

si $\Gamma \vdash t : T$, alors $|t|_{\mathcal{I}} : |T|_{\mathcal{I}}$

- si t non typable, pas de sémantique ;
- Γ : contexte de De Bruijn.

```
Fixpoint sem_term (g: context) (t: term) : option {ty: type & forall [...],
tr_type [...] ty} := ...
```



Dérivations

Idée générale :

- inductif `deriv` : `set term` \rightarrow `term` \rightarrow `Prop`
- une preuve de : `forall` `G` `p`, `deriv` `G` `p` \rightarrow `has_sem` `G` \rightarrow `has_sem` `p`
- `has_sem` `p` :
 - $\emptyset \vdash p$: `Bool` (`p` est donc localement clos)
 - la traduction de `p` est une proposition Coq correcte



Règles d'inférence

Règles d'inférence :

- règles de base sauf INST et INST_TYPE
- règles de la déduction naturelle, contraposée classique
- η -expansion, affaiblissement



Règles d'inférence

Règles d'inférence :

- règles de base sauf INST et INST_TYPE
- règles de la déduction naturelle, contraposée classique
- η -expansion, affaiblissement

$$\frac{\Gamma(x_1, \dots, x_n) \vdash p[x_1, \dots, x_n]}{\Gamma(t_1, \dots, t_n) \vdash p[t_1, \dots, t_n]} \text{ INST}$$

$$\frac{\Gamma(\alpha_1, \dots, \alpha_n) \vdash p[\alpha_1, \dots, \alpha_n]}{\Gamma(\gamma_1, \dots, \gamma_n) \vdash p[\gamma_1, \dots, \gamma_n]} \text{ INST_TYPE}$$



Règles d'inférence

Règles d'inférence :

- règles de base sauf INST et INST_TYPE
- règles de la déduction naturelle, contraposée classique
- η -expansion, affaiblissement

$$\frac{\Gamma \vdash s = t}{\Gamma \vdash (\lambda x.s) = (\lambda x.t)} \text{ ABS}$$

$$\frac{\forall x \notin L, \Gamma \vdash s^x = t^x}{\Gamma \vdash (\lambda s) = (\lambda t)} \text{ ABS'}$$

Pour ABS : $x \notin FV(\Gamma)$.



Correction

Correction :

```

Definition has_sem (t: term) : Prop :=
  match sem_term nil t with
  | Some (existT Bool evT) => evT [...]
  | _ => False
  end.

```

Definition sem_hyp (h: hyp_set):= set.For_all has_sem h.

Lemma sem_deriv : forall (h: hyp_set) (t: term), deriv h t -> forall [...],
 sem_hyp h -> has_sem t.



Type des termes de preuve

```
Inductive proof : Type :=  
| Prefl : term -> proof  
| Pbeta : idV -> type -> term -> proof  
| Pinstt : proof -> list (idT * type) -> proof  
| :  
| Pfact : proof -> proof.  
| Poracle : forall h t, deriv h t -> proof.
```



Traduction en dérivation

```

Fixpoint proof2deriv (p: proof) : option (hyp_set * term) :=
  match p with
  | Prefl t =>
    match infer nil t with
    | Some A => Some (hyp_empty, heq A t t)
    | None => None
    end
  :
  end.

```



Lemme important

```
Lemma proof2deriv_correct : forall p,  
  match proof2deriv p with  
  | Some (h, t) => deriv h t  
  | None => True  
end.
```



Exemple

$$\vdash !x:A. ?y:A. x = y$$

$$\frac{\frac{\frac{}{\vdash x =_A x} \text{REFL } 'x'}{\vdash \exists y : A. x =_A y} \text{EXISTS } '\exists y : A. x =_A y' 'x'}{\vdash \forall x : A. \exists y : A. x =_A y} \text{GEN } 'x'$$



Exemple

$$\vdash !x:A. \exists y:A. x = y$$

$$\frac{\frac{\frac{}{\vdash x =_A x} \text{REFL 'x'}}{\vdash \exists y : A. x =_A y} \text{EXISTS '}\exists y : A. x =_A y \text{' 'x'}}{\vdash \forall x : A. \exists y : A. x =_A y} \text{GEN 'x'}}$$

Code

Troisième partie III

Enregistrement et exportation des termes de preuve de HOL-Light

Proof-recording de S. Obua

But :

- termes de preuve compacts
- courte durée d'enregistrement

Solution :

- granularité

Statistiques :

- enregistrement de la bibliothèque standard de HOL-Light (1694 théorèmes) : 3 min

Exportation

But :

- petits fichiers
- petit nombre de fichiers

Solution :

- partage (preuves, types et termes...)

Statistiques :

- exportation de la bibliothèque standard de HOL-Light (1694 théorèmes) :
 - 14 min
 - 191652 termes de preuve
 - 2.2 Go

Exemple

Début de la bibliothèque...

Quatrième partie IV

Conclusion et perspectives

Conclusion

HOL-Light :

- enregistrer des termes de preuve
- exporter des termes de preuve

Coq :

- modèle de HOL-Light
- prouvé correct

Interface :

- début de la bibliothèque
- partage de preuves

Perspectives

Théoriques :

- gérer les axiomes et les définitions de types
- minimaliser le nombre d'axiomes en Coq

Pratiques :

- utiliser des types Coq plus efficaces (entiers machine, ensembles finis)
- exportation plus efficace, termes de preuve plus petits
- interface utilisateur
- passage à l'échelle

Merci de votre attention !

Des questions ?