Importation de preuves HOL-Light en Coq

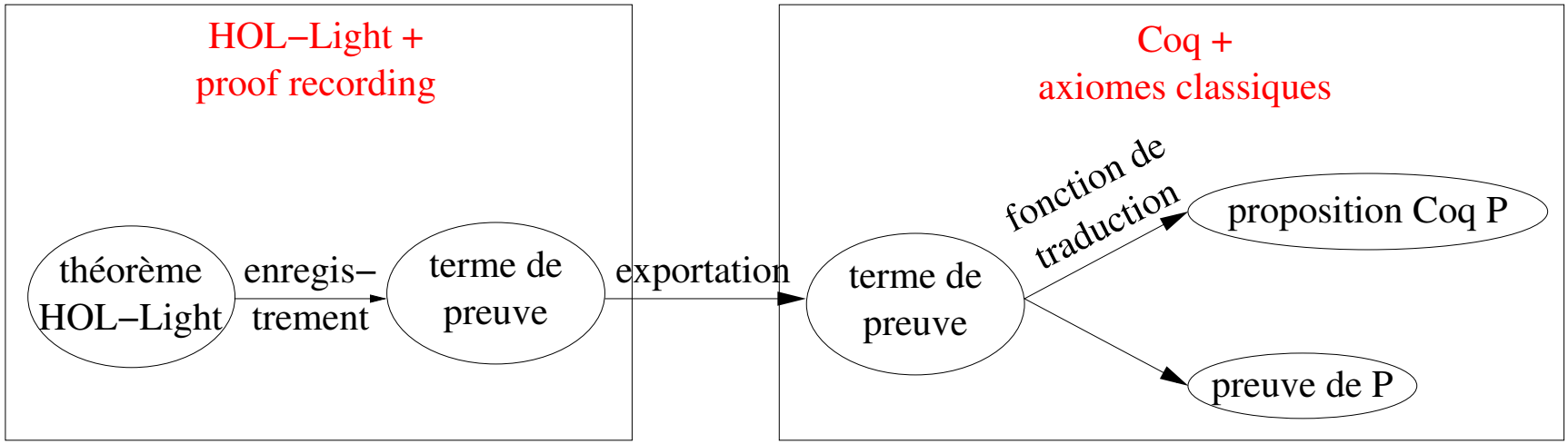
Stage de M2

Chantal Keller
chantal.keller@ens-lyon.fr
Sous la direction de Benjamin Werner

ÉNS Lyon - INRIA Saclay - LIX

7 septembre 2009

Idée


HOL–Light +
proof recording
théorème
HOL–Light
enregis–
trement
terme de
preuve
exportation
Coq +
axiomes classiques
terme de
preuve
fonction de
traduction
proposition Coq P
preuve de P

But

Plusieurs intérêts :

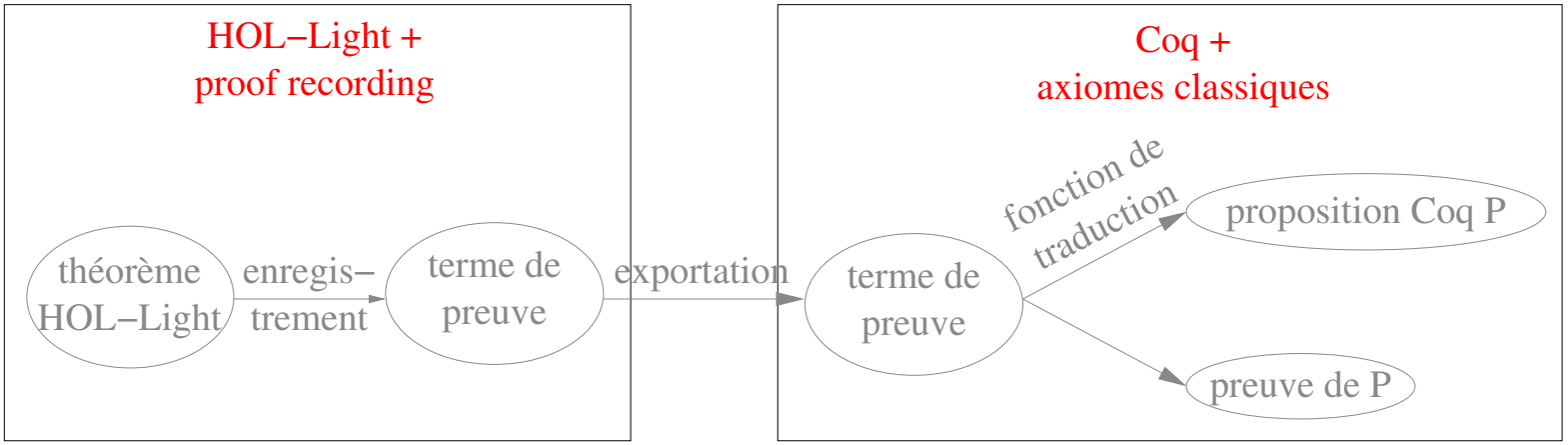
- utiliser les théorèmes de HOL-Light en Coq (analyse)
- établir un modèle de HOL-Light en Coq
- comparer deux systèmes de preuve

Plan

Première partie I

HOL-Light et Coq

Présentation



HOL-Light

HOL-Light :

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

Système logique

Système logique :

- λ-calcul simplement typé
- polymorphisme : schémas de types
- variables de types (et de termes)
- règles logiques traitant de l'égalité entre termes
- système de tactiques

Exemples de règle d'inférence :

$$\frac{}{\vdash (\lambda x.t)\ x = t} \qquad \frac{\Gamma \vdash p \Leftrightarrow q \qquad \Delta \vdash p}{\Gamma \cup \Delta \vdash q}$$

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

Correction

Correction :

- type thm **abstrait** (on ne peut construire des théorèmes qu'en utilisant les primitives fournies par HOL-Light)
- preuves non enregistrées, mais **interprétation** de HOL-Light à chaque ouverture de session

Coq

Coq :

- assistant de preuve
- compilé
- calcul des constructions inductives (intuitionniste, calculatoire)
- preuves réflexives
- noyau logique complexe

Système logique et correction

Système logique :

- extension de HOL-Light si on se donne des axiomes classiques

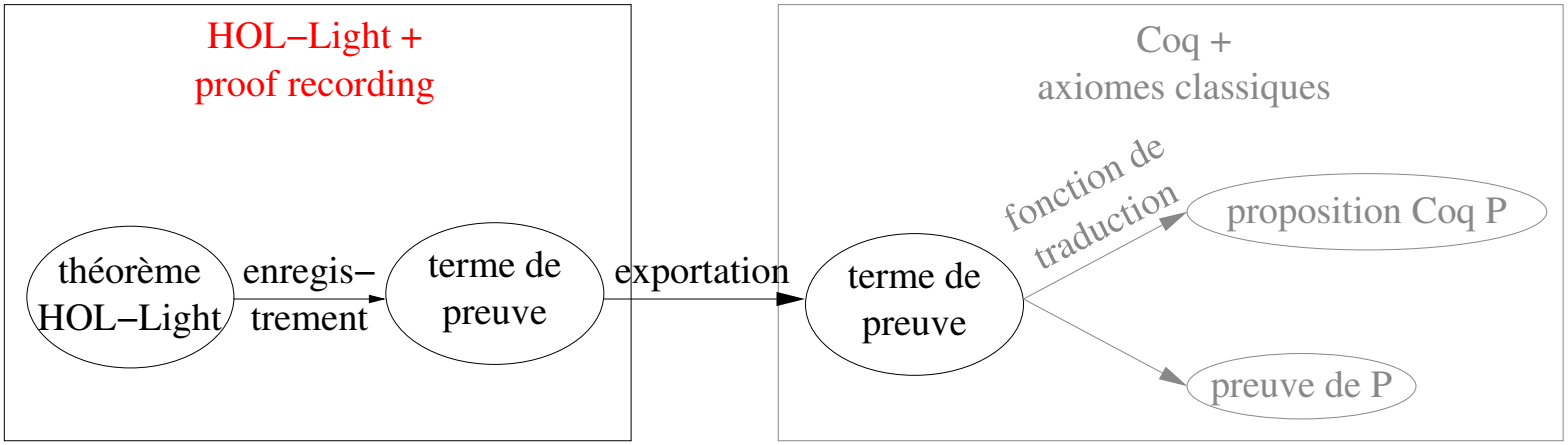
Correction :

- preuves enregistrées et vérifiables

Deuxième partie II

Enregistrement et exportation des termes de preuve de HOL-Light

Présentation



Proof recording (S. Obua)

Enregistrement de l'arbre de dérivation :

- termes de preuve compacts
- courte durée d'enregistrement

$\hookrightarrow$ granularité

```
type proof =
    | Proof of (proof_info * proof_content * (unit -> unit))
and proof_content =
    | Pconj of proof * proof
    | Pconjunct1 of proof
    | Pconjunct2 of proof
    ...
```

Exportation

Exportation des arbres de dérivation :

- petits fichiers
- petit nombre de fichiers

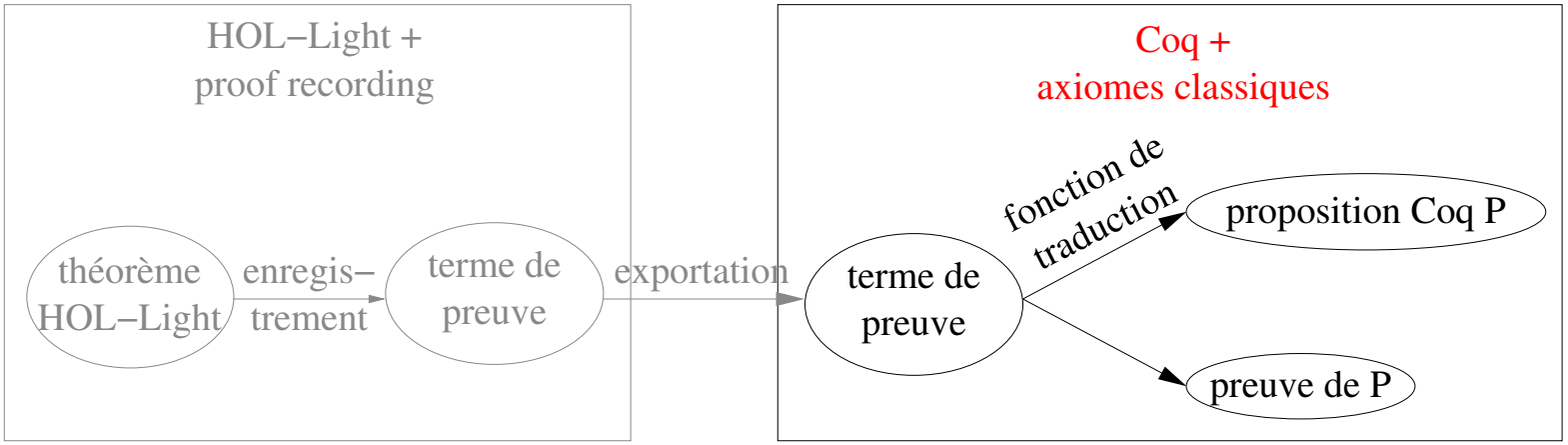
$\hookrightarrow$ partage

```
Inductive proof : Type :=
| Pconj : proof →proof →proof
| Pconjunct1 : proof →proof
| Pconjunct2 : proof →proof
...
| Poracle : forall h t, deriv h t →proof.
```

Troisième partie III

Modèle de HOL-Light en Coq

Présentation



Plongements

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

- raisonnement par induction sur la structure des objets

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

- facilités de Coq

Plongements

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

- raisonnement par induction sur la structure des objets

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

- facilités de Coq
- obtenir des propositions Coq

Plongements

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

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

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

- facilités de Coq
- obtenir des propositions Coq

Plongements

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

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

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

- facilités de Coq
- obtenir des propositions Coq

$\hookrightarrow$ fonction de traduction de profond vers peu profond

Types

Inductif **type** :

$$\frac{}{\mathsf{Bool} \in \textbf{type}} \qquad \frac{X \in \mathsf{idT}}{\mathsf{TVar}\ X \in \textbf{type}} \qquad \frac{A, B \in \textbf{type}}{A \longrightarrow B \in \textbf{type}} \qquad \ldots$$

idT un ensemble infini dénombrable

Termes

Inductif **term** :

$$\frac{n \in \mathbb{N}}{\text{Dbr } n \in \textbf{term}} \qquad \frac{x \in \text{idV} \qquad A \in \textbf{type}}{\text{Var } x\ A \in \textbf{term}} \qquad \frac{u, v \in \textbf{term}}{\text{App } u\ v \in \textbf{term}}$$

$$\frac{A \in \textbf{type} \qquad u \in \textbf{term}}{\text{Abs } A\ u \in \textbf{term}} \qquad \ldots$$

idV un ensemble infini dénombrable

Typage

Typage :

- λ-calcul simplement typé
- contexte pour les indices de De Bruijn (liste de types)

Types

Types : interface entre syntaxe et sémantique.
Traduction d'un type : $|T|$

$$\begin{aligned} |\text{Bool}| &\equiv \textbf{Prop} \\ |A \longrightarrow B| &\equiv |A| \rightarrow |B| \end{aligned}$$

Types

Types : interface entre syntaxe et sémantique.
Traduction d'un type : $|T|$

$$
\begin{aligned}
|\text{Bool}| &\equiv \textbf{Prop} \\
|A \longrightarrow B| &\equiv |A| \rightarrow |B| \\
|\text{TVar } X| &\equiv ?
\end{aligned}
$$

Types

Types : interface entre syntaxe et sémantique.
Traduction d'un type : $|T|_{\mathcal{I}}$

$$
\begin{aligned}
|\mathsf{Bool}|_{\mathcal{I}} &\equiv \mathbf{Prop} \\
|A \longrightarrow B|_{\mathcal{I}} &\equiv |A|_{\mathcal{I}} \to |B|_{\mathcal{I}} \\
|\mathsf{TVar}\ X|_{\mathcal{I}} &\equiv \mathcal{I}(X)
\end{aligned}
$$

Termes

Utilisation des types dépendants. [GW07]
Traduction d'un terme :

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

$\hookrightarrow$ si t non typable, $|t|_{\mathcal{I}}$ n'est pas défini

Exemple : si $t \triangleq p \Leftrightarrow q$
t a le type Bool et on veut que $|t|_{\mathcal{I}}$ soit une proposition Coq

Deux types de données

Inductive proof : **Type** :=
| Pconj : proof →proof →proof
| Pconjunct1 : proof →proof
| Pconjunct2 : proof →proof
...

Inductive deriv : set term →term →**Prop** :=
| Dconj : **forall** h1 h2 t1 t2, deriv h1 t1 →deriv h2 t2 →deriv (union h1 h2) t1 ∧ t2
| Dconjunct1 : **forall** h t1 t2, deriv h t1 ∧ t2 →deriv h t1
| Dconjunct2 : **forall** h t1 t2, deriv h t1 ∧ t2 →deriv h t2
...

Passage de l'un à l'autre

$$\text{p : proof} \xrightarrow{\text{proof2deriv}} \begin{cases} \text{Some (h,t) tels que deriv h t} \\ \text{None} \end{cases}$$

$$\text{deriv h t} \xrightarrow{\text{sem\_deriv}} |\text{h}| \text{ correct} \Rightarrow |\text{t}| \text{ correct}$$

t correct :

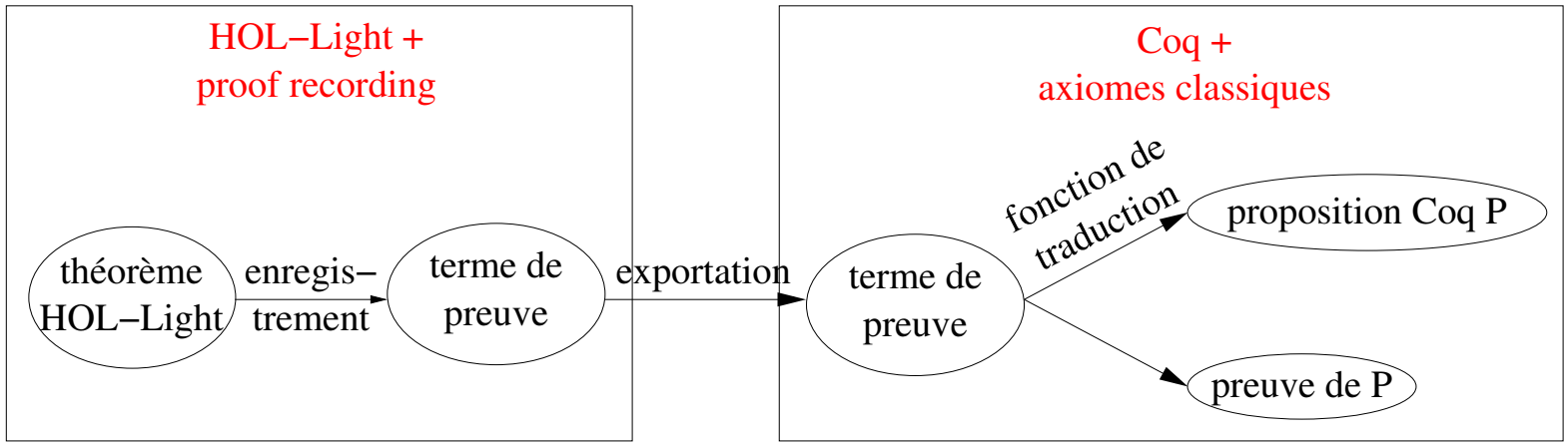
- $\emptyset \vdash t$: Bool (t est donc localement clos)
- la traduction de t est une proposition Coq correcte

Quatrième partie IV

Résultats

Résultats

Qualitatifs



$\hookrightarrow$ théorèmes utilisables en Coq

CONJ_SYM : $\forall\ t_1\ t_2.\ t_1 \wedge t_2 \Leftrightarrow t_2 \wedge t_1$

forall x x0 : **Prop**, (x ∧ x0) = (x0 ∧ x)

Quantitatifs

Échelle	Nombre théorèmes	Nombre lemmes	Taille
Petite	69	1213	0,7 Mo
Grande	1694	191620	921 Mo

Échelle	D. enregistrement	D. exportation	D. compilation
Petite	4 s	1 s	21 s
Grande	3 min	7 min 30	X

Environ 6900 lignes de code

Cinquième partie V

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

Améliorer l'efficacité :

- utiliser des types Coq efficaces (ensembles finis)
- termes de preuve plus petits
- exportation plus rapide

Passer à l'échelle :

- grands développements
- interface utilisateur

Merci de votre attention !

Des questions ?

La fonction de traduction pour les contextes de De Bruijn

Γ contexte de De Bruijn (liste de types)

$$|\Gamma|_{\mathcal{I}} \equiv \forall n, \quad \begin{array}{l} \textbf{match } \text{nth } n\ \Gamma \textbf{ return } \text{Type } \textbf{with} \\ \quad \mid \text{Some } T \Rightarrow |T|_{\mathcal{I}} \\ \quad \mid \text{None} \Rightarrow \text{unit} \\ \textbf{end} \end{array}$$

Si $\Gamma \vdash n : T$, alors $|n|_{\mathcal{I}, f \in |\Gamma|_{\mathcal{I}}} \equiv f(n) \in |T|_{\mathcal{I}}$.