Travaux sur le POPLMark Challenge

Stage de 5 mois à l'Université de Pennsylvanie
avec Benjamin Pierce et Stephanie Weirich

Arthur Charguéraud

ENS-Cachan 8 Sept. 2006

Plan

1) Contexte de recherche
2) Présentation de Coq
3) Théorèmes formalisés
4) Un peu de technique
5) Résultats
6) Conclusions

1) Contexte de recherche

Certification

Code source du programme → Compilation → Exécution sur matériel

Logiciels certifiés? un jour...

Compilateurs bientôt certifiés

Matériel déjà certifié

+ certifier le logiciel de certification

Preuves Formelles

Définitions

↓

Arguments

de la

preuve

↓

Théorèmes

Formalisation de Compilateurs

- Coq en Coq

  Bruno Barras et Benjamin Werner, 1996

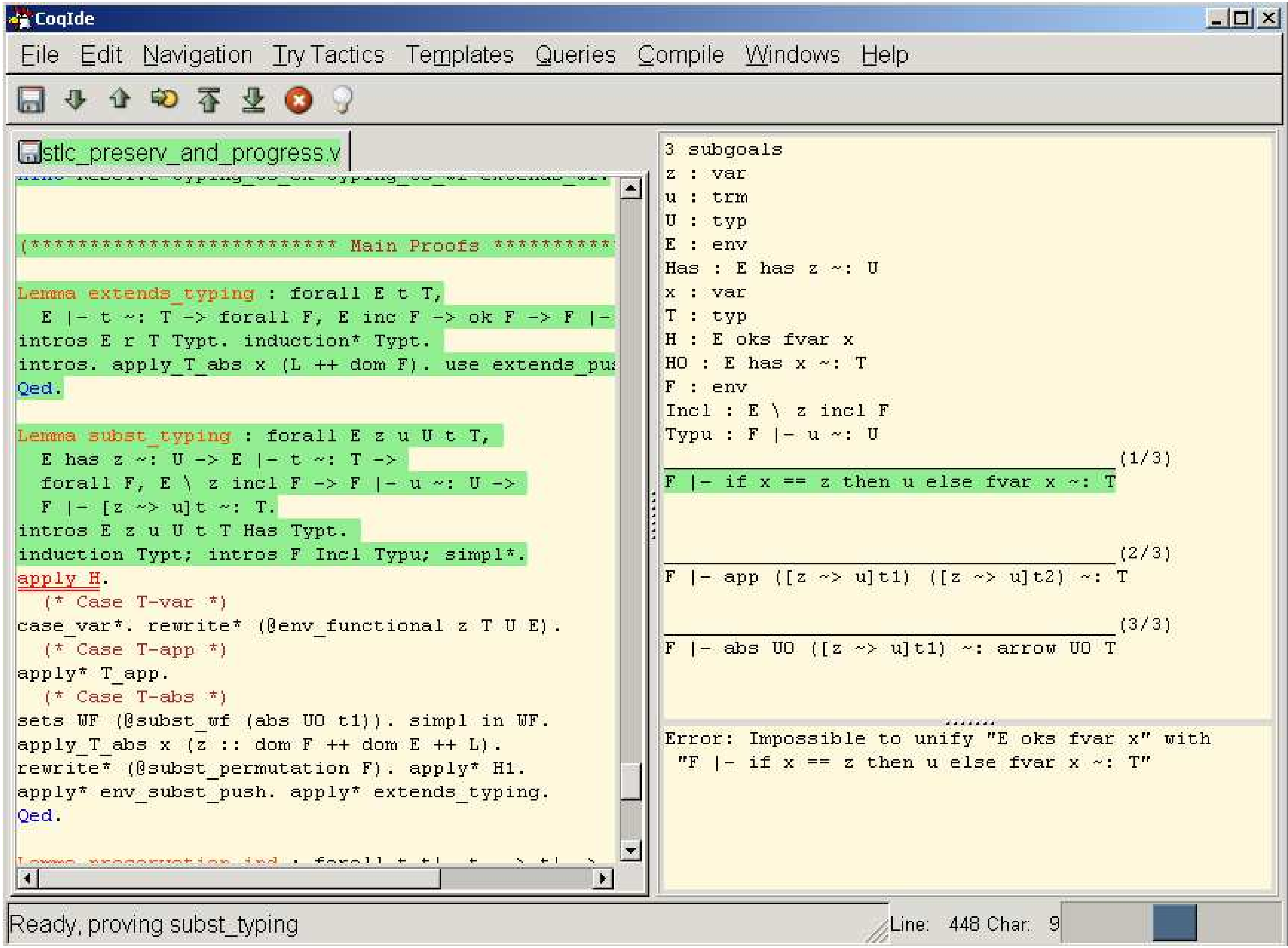
- SML en Twelf

  Karl Crary, Daniel Lee et Robert Harper, 2006

- Compilateur C-minor certifié

  Xavier Leroy, 2006

2) Présentation de Coq


CoqIde
File Edit Navigation Try Tactics Templates Queries Compile Windows Help
stlc_preserv_and_progress.v
(************************** Main Proofs ***********
Lemma extends_typing : forall E t T,
E |- t ~: T -> forall F, E inc F -> ok F -> F |-
intros E r T Typt. induction* Typt.
intros. apply_T_abs x (L ++ dom F). use extends_pus
Qed.
Lemma subst_typing : forall E z u U t T,
E has z ~: U -> E |- t ~: T ->
forall F, E \ z incl F -> F |- u ~: U ->
F |- [z ~> u]t ~: T.
intros E z u U t T Has Typt.
induction Typt; intros F Incl Typu; simpl*.
apply H.
(* Case T-var *)
case_var*. rewrite* (@env_functional z T U E).
(* Case T-app *)
apply* T_app.
(* Case T-abs *)
sets WF (@subst_wf (abs U0 t1)). simpl in WF.
apply_T_abs x (z :: dom F ++ dom E ++ L).
rewrite* (@subst_permutation F). apply* H1.
apply* env_subst_push. apply* extends_typing.
Qed.
3 subgoals
z : var
u : trm
U : typ
E : env
Has : E has z ~: U
x : var
T : typ
H : E oks fvar x
H0 : E has x ~: T
F : env
Incl : E \ z incl F
Typu : F |- u ~: U
(1/3)
F |- if x == z then u else fvar x ~: T
(2/3)
F |- app ([z ~> u]t1) ([z ~> u]t2) ~: T
(3/3)
F |- abs U0 ([z ~> u]t1) ~: arrow U0 T
Error: Impossible to unify "E oks fvar x" with
"F |- if x == z then u else fvar x ~: T"
Ready, proving subst_typing
Line: 448 Char: 9

Programmer en Coq

```
Fixpoint subst (z : var) (u : trm) (t : trm)
                                 {struct t} : trm :=
 match t with
 | Var x      => if (x == z) then u else (Var x)
 | App t1 t2 => App (subst z u t1) (subst z u t2)
 | Abs x t1  => Abs x (subst z u t1)
 end.
```

```
Lemme inf_à_son_double :

   forall x : nat,  x <= 2 * x.
```

```
Lemme inf_impl_inf_entre_doubles :

   forall x y : nat,  x <= y  ->  2*x <= 2*y.
```

Similarités entre Coq et Caml

– Basés sur le lambda-calcul et le typage

– Caml vient de ML, ML conçus pour les preuves

– Coq est implémenté en Caml

– Coq et Caml ont les mêmes origines

3) Résultats formalisés

λ-calcul simplement typé

$$
\begin{array}{lcl}
T & := & A \;\mid\; T_1 \rightarrow T_2 \\
t & := & x \;\mid\; (t_1\ t_2) \;\mid\; \lambda x{:}T.\ t_1
\end{array}
$$

$$
\frac{(x:T) \in E}{E \vdash x : T}\ \textsc{T-Var}
\qquad
\frac{E \vdash t_1 : S \rightarrow T \qquad E \vdash t_2 : S}{E \vdash (t_1\ t_2) : T}\ \textsc{T-App}
$$

$$
\frac{x \# E \qquad E, x : S \vdash t_1 : T}{E \vdash (\lambda x{:}S.\ t_1) : S \rightarrow T}\ \textsc{T-Abs}
$$

Préservation :

$$
t \longmapsto t' \quad \wedge \quad E \vdash t : T \quad \Rightarrow \quad E \vdash t' : T
$$

Sous-typage dans System-F$_{<:}$

$$T \quad ::= \quad \mathrm{Top} \quad | \quad X \quad | \quad T_1 \to T_2 \quad | \quad \forall X{<:}T_1 . T_2$$

$$\frac{}{E \vdash S <: \mathrm{Top}}\ \text{SA-TOP} \qquad \frac{E \vdash T_1 <: S_1 \qquad E \vdash S_2 <: T_2}{E \vdash (S_1 \to S_2) <: (T_1 \to T_2)}\ \text{SA-ARROW}$$

$$\frac{}{E \vdash X <: X}\ \text{SA-REFL-TVAR} \qquad \frac{(X{<:}U) \in E \qquad E \vdash U <: T}{E \vdash X <: T}\ \text{SA-TRANS-TVAR}$$

$$\frac{E \vdash T_1 <: S_1 \qquad X \# E \qquad (E, X{<:}T_1) \vdash S_2 <: T_2}{E \vdash (\forall X{<:}S_1 . S_2) <: (\forall X{<:}T_1 . T_2)}\ \text{SA-ALL}$$

Réflexivité : $E \vdash T <: T$

Transtitivité : $E \vdash S <: Q \quad \wedge \quad E \vdash Q <: T \quad \Rightarrow \quad E \vdash S <: T$

Préservation par substitution :

$$E, Z{<:}Q, F \vdash S <: T \quad \wedge \quad E \vdash P <: Q$$
$$\Rightarrow \quad E, [Z \to P]\, F \vdash [Z \to P]\, S <: [Z \to P]\, T$$

POPLMark: critères d’évaluation

1) Les définitions et l’énoncé des théorèmes doivent suivre les conventions usuelles.

2) L’argumentation dans les preuves doit apparaître et suivre le modèle donné.

3) Les techniques utilisées doivent être générales.

4) Le coût de la formalisation doit être raisonnable.

5) La technologie utilisée doit être transparente et avoir un coût d’entrée raisonnable.

Travail réalisé

1) Comprendre le Challenge, et apprendre Coq.

2) Analyser les techniques existantes.

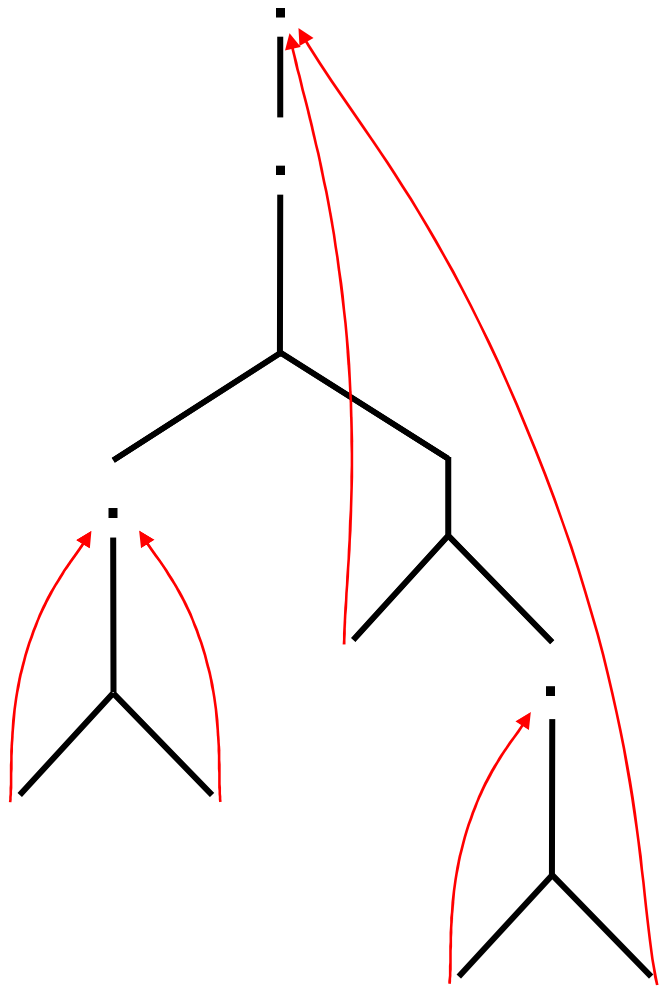
3) Expérimenter des nouvelles techniques.

4) Comparer les techniques entre elles.

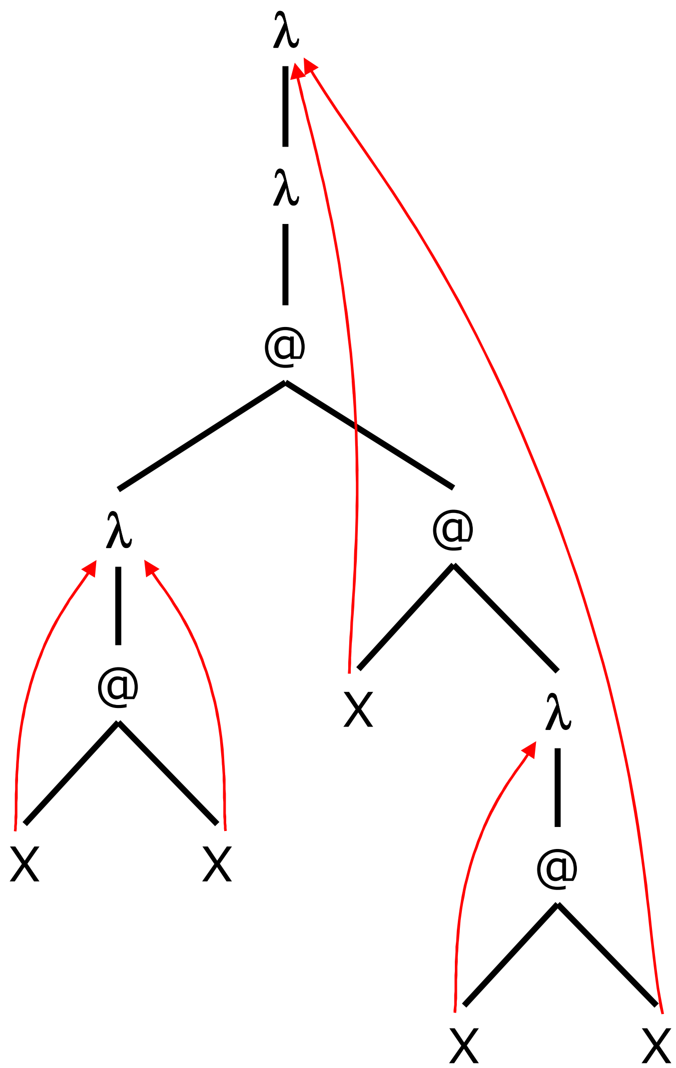
5) Synthétiser ces résultats dans une solution au POPLMark Challenge.

4) Un peu de technique

Qui suis-je ?

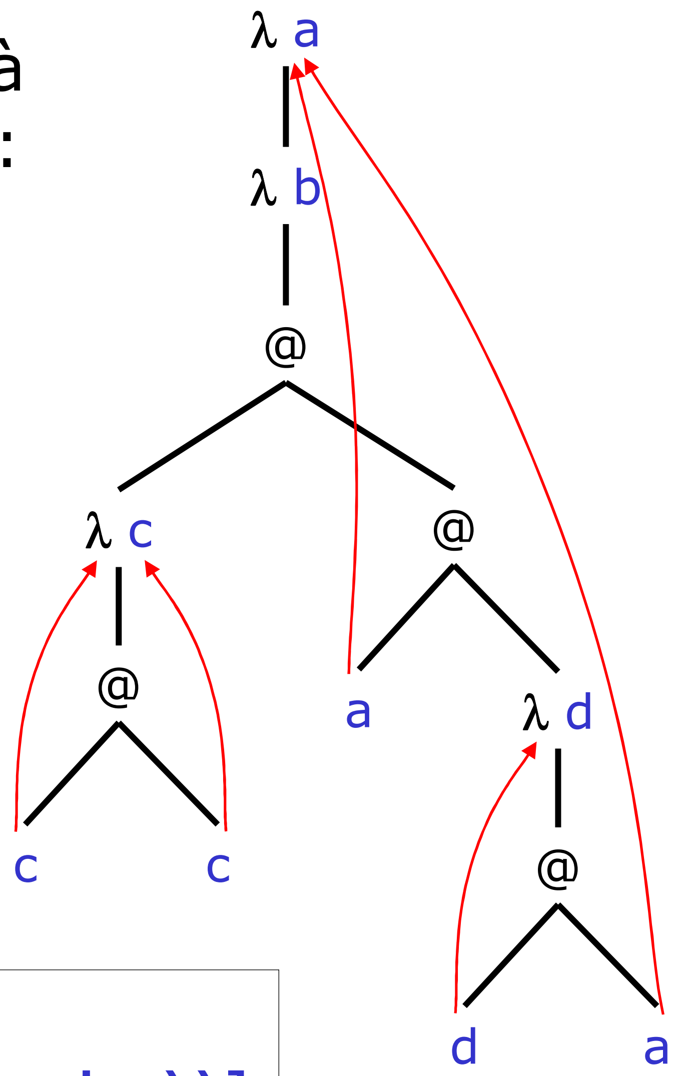


Qui suis-je ?



λ-terme avec des noms

Donner un nom à chaque variable :

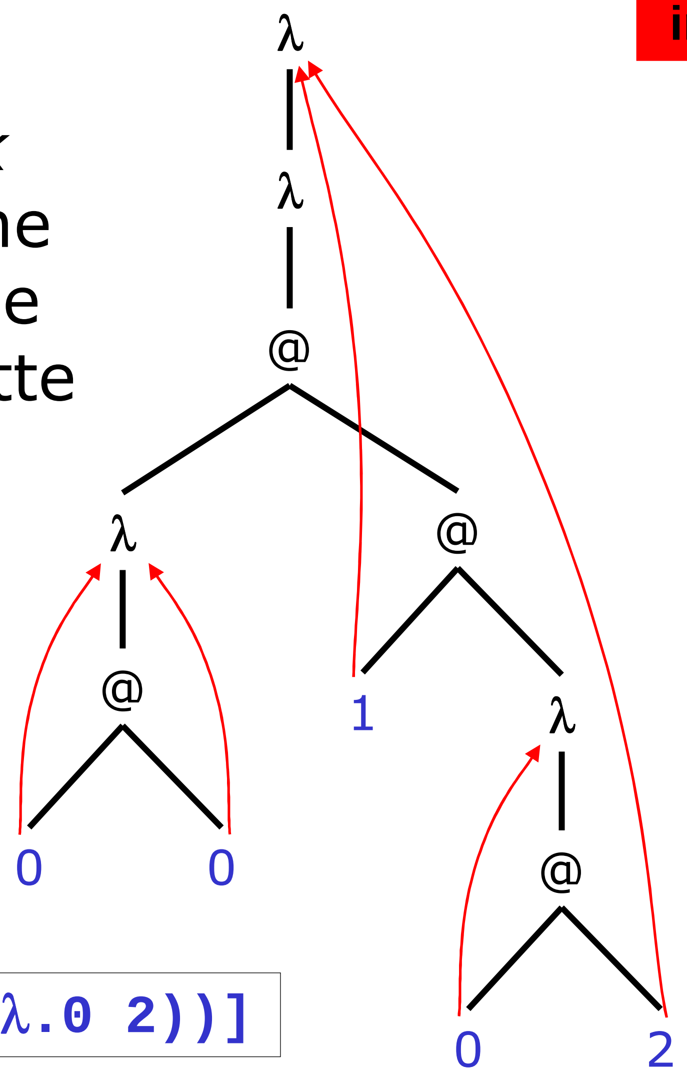


λa.λb.
[(λc. c c) (a (λd. d a))]

λ-terme avec des indices

indices de « de-Bruijn »

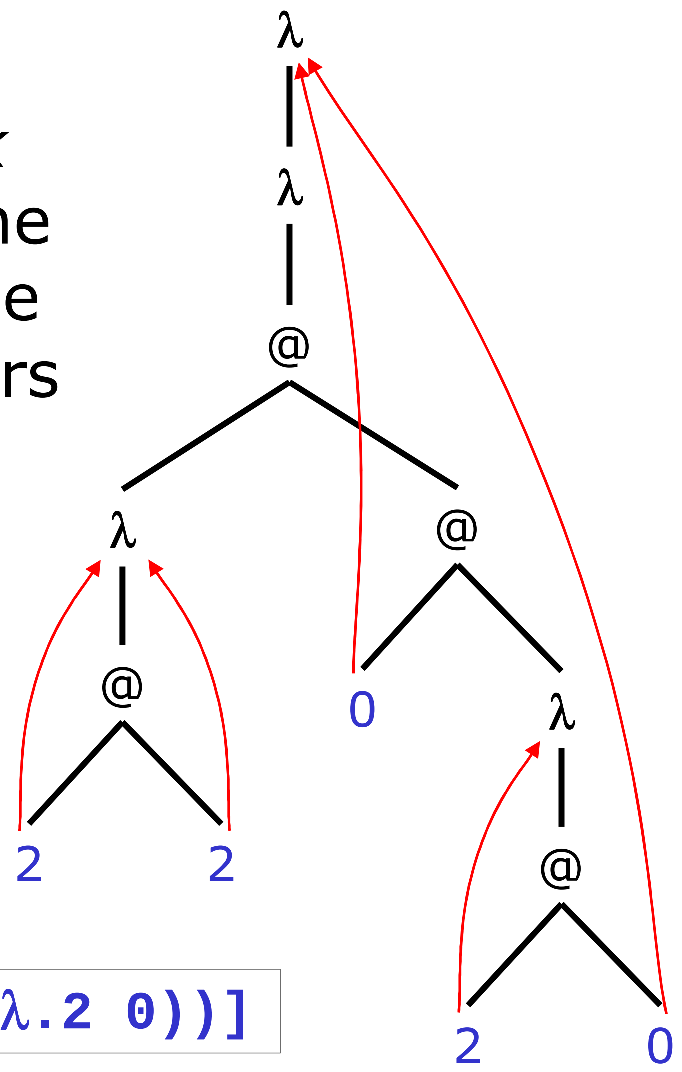
Une variable portant l'indice *k* pointe à la *k*-ième abstraction située au-dessus de cette variable :




```
λ.λ.[(λ.0 0) (1 (λ.0 2))]
```

λ-terme avec des niveaux

Une variable portant l'indice *k* pointe à la *k*-ième abstraction située sur le chemin vers cette variable en partant de la racine :



`λ.λ.[(λ.2 2) (0 (λ.2 0))]`

Exemple d'implémentation

Noms, en Caml

```
type term =
  | Var of string
  | App of term * term
  | Abs of string * term
```

Noms, en Coq

```
Inductive term : Set :=
  | Var : name -> term
  | App : term -> term -> term
  | Abs : name -> term -> term
```

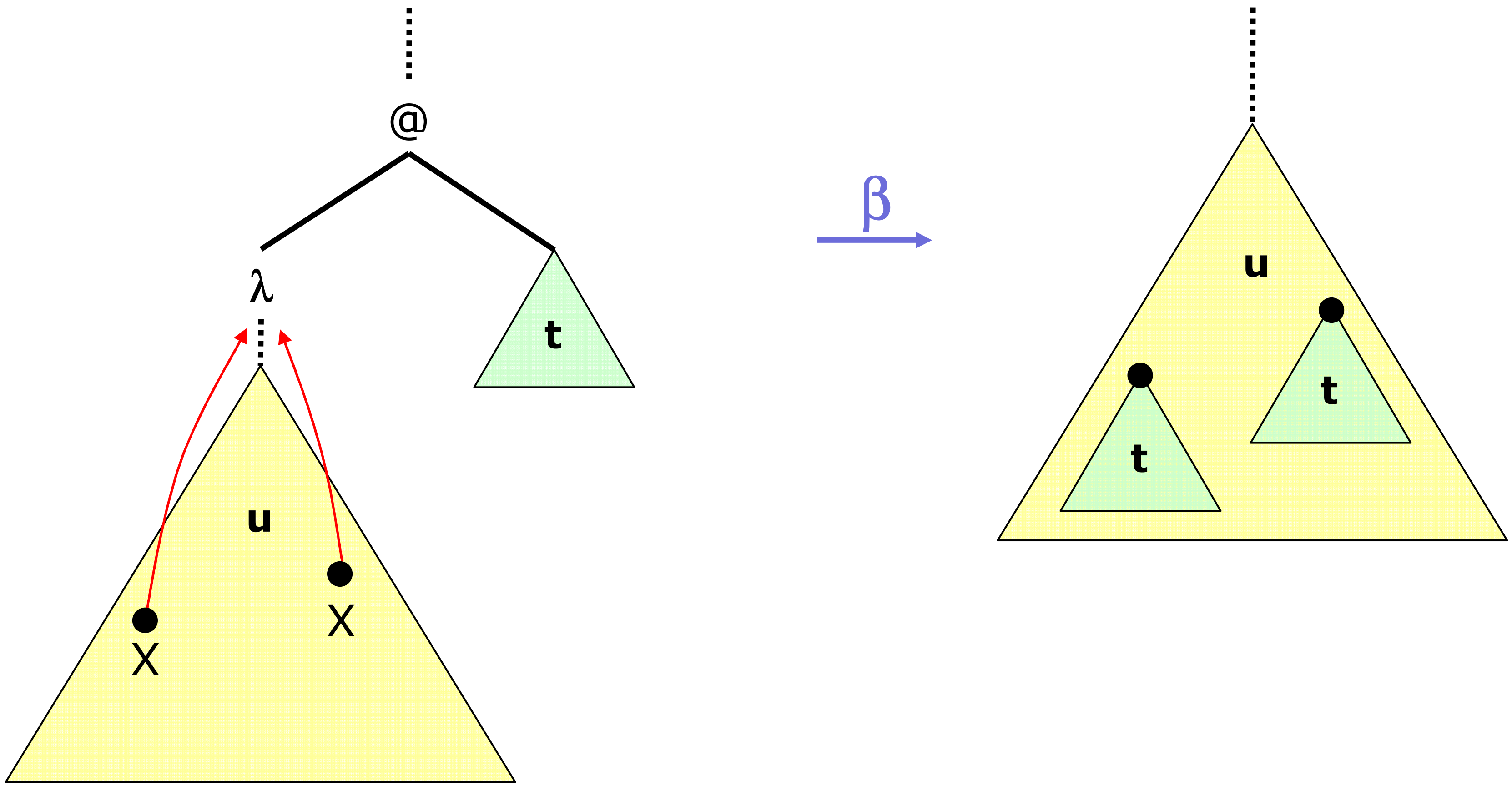
Indices, en Caml

```
type term =
  | Var of int
  | App of term * term
  | Abs of term
```

Indices, en Coq

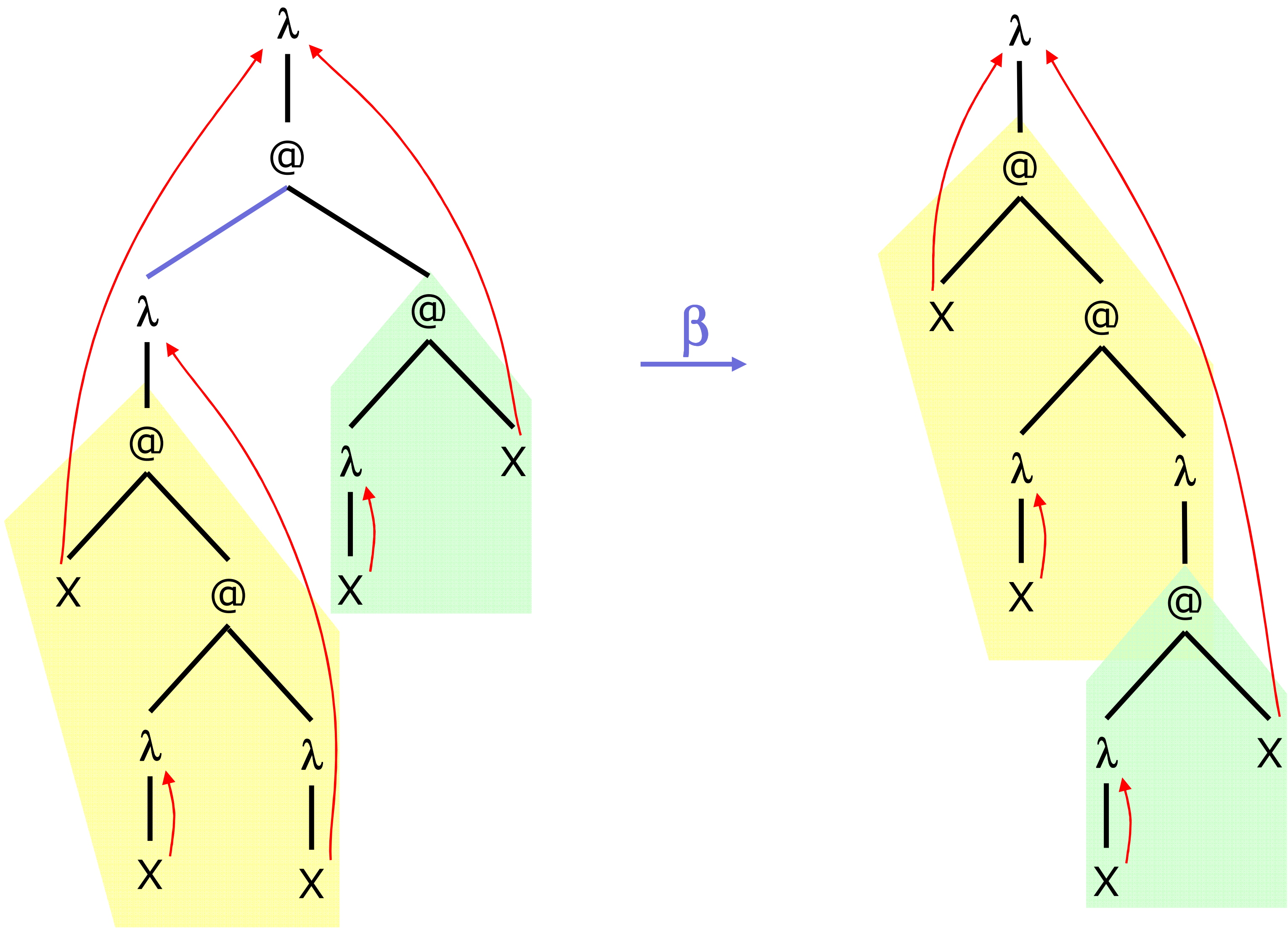
```
Inductive term : Set :=
  | Var : int -> term
  | App : term -> term -> term
  | Abs : term -> term
```

β-réduction



(λx.u) t   se réduit en   [x->t]u

Exemple de β-réduction

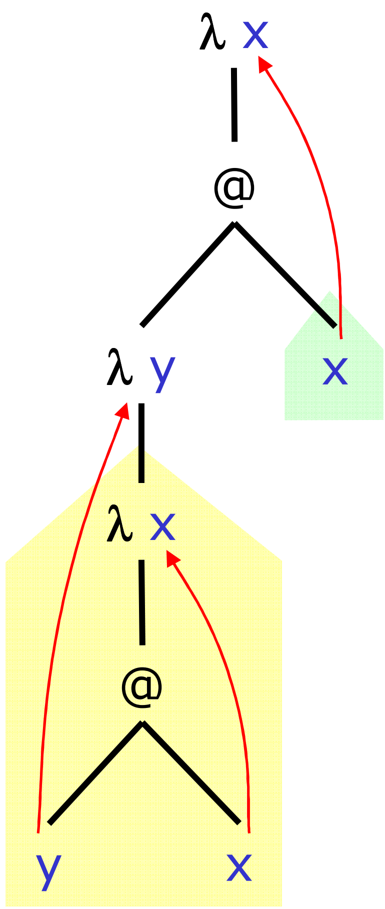


β-réduction avec des noms

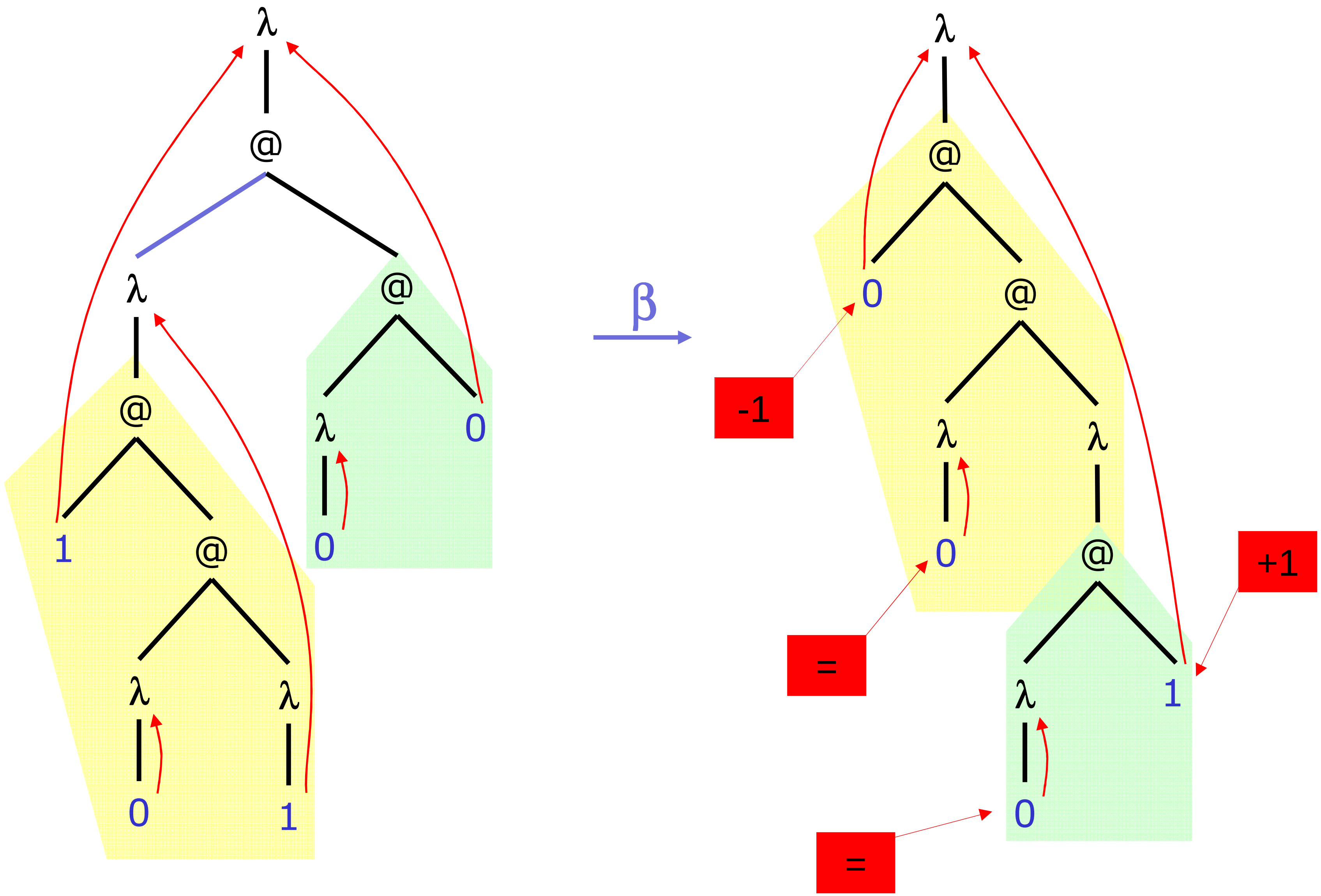
```
   (λz. z z) (λy. λx. y x)
-> (λy. λx. y x) (λy. λx. y x)
-> λx. [(λy. λx. y x) x]
```

α-conversion obligatoire !
(renommage du x en z)

```
-> λx. [(λy. λz. y z) x]
-> λx. [λz. x z]
```



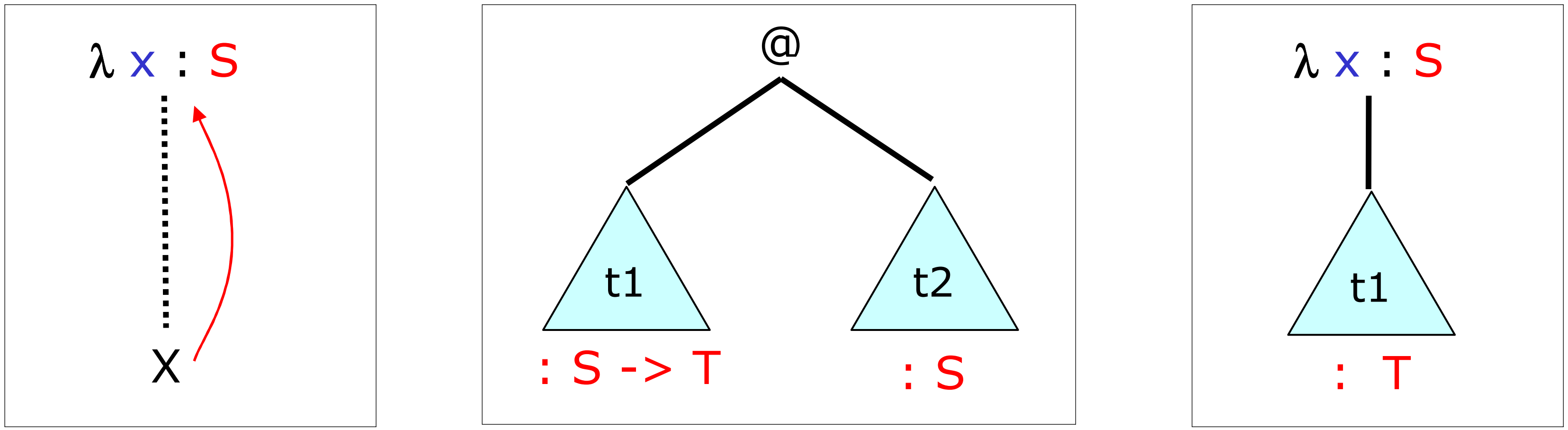
β-réduction avec des indices



Typage des λ-termes

$$\frac{(x : T) \in E}{E \vdash x \ : \ T}\ \textsc{T-var} \qquad \frac{E \vdash t_1 \ : \ S \rightarrow T \qquad E \vdash t_2 \ : \ S}{E \vdash (t_1 \ t_2) \ : \ T}\ \textsc{T-app}$$

$$\frac{x \# E \qquad E, x : S \vdash t_1 \ : \ T}{E \vdash (\lambda x{:}S.\, t_1) \ : \ S \rightarrow T}\ \textsc{T-abs}$$



S

T

S -> T

Exemple de choix : T-var et T-abs

Standard	$\dfrac{(x : T) \in E}{E \vdash x \;:\; T}$	$\dfrac{\mathrm{Q}(x) \qquad (E, x : S) \vdash t_1 \;:\; T}{E \vdash (\lambda x{:}S.\, t_1) \;:\; S \to T}$
Mixed names	$\dfrac{(x : T) \in E}{E \vdash x \;:\; T}$	$\dfrac{\mathrm{Q}(x) \qquad (E, x : S) \vdash [y \to x]\, t_1 \;:\; T}{E \vdash (\lambda y{:}S.\, t_1) \;:\; S \to T}$
Distinct names	$\dfrac{(x : T) \in E}{E \vdash \lfloor x \rfloor \;:\; T}$	$\dfrac{\mathrm{Q}(x) \qquad (E, x : S) \vdash [y \to \lfloor x \rfloor]\, t_1 \;:\; T}{E \vdash (\lambda y{:}S.\, t_1) \;:\; S \to T}$
Locally nameless	$\dfrac{(x : T) \in E}{E \vdash \lfloor x \rfloor \;:\; T}$	$\dfrac{\mathrm{Q}(x) \qquad (E, x : S) \vdash {t_1}^{x} \;:\; T}{E \vdash (\lambda{:}S.\, t_1) \;:\; S \to T}$
Mixed indices	$\dfrac{\mathit{lookup}\ i\ E = \mathrm{Some}\ T}{E \vdash i \;:\; T}$	$\dfrac{(E, : S) \vdash t_1 \;:\; T}{E \vdash (\lambda{:}S.\, t_1) \;:\; S \to T}$
Indices + levels	$\dfrac{\mathit{lookup}\ k\ E = \mathrm{Some}\ T}{E \vdash \lfloor k \rfloor \;:\; T}$	$\dfrac{(E, : S) \vdash {t_1}^{\lvert E \rvert} \;:\; T}{E \vdash (\lambda{:}S.\, t_1) \;:\; S \to T}$

où Q(x) est à choisir parmi: $\exists x \# E$ ou $\forall x \# E$ ou $\forall x \notin L$.

5) Résultats

Beaucoup de choix possibles

1) Représentation

- Variables libres 3
- Variables liées 3
- Mélange ou non 2
- Environnement 2

2) Opérations

- Substitution 3
- Lookup 3
- Concaténation 2
- Substitution dans l'environnement 3

3) Propriétés

- Freshness 3
- Bonne-formation 3
- BF dans les relations 4
- Termes clos 3
- Quantification des noms 4

4) Preuves

- Preuves par induction 3
- Méthode de preuve 2

De l'ordre de 5 millions de possibilités !

Exemple de formalisation

- Présentation standard du lemme

▶ SUBTYPING-WEAKENING:

$$E \vdash S <: T \quad \Rightarrow \quad E, F \vdash S <: T$$

- Présentation prête à être formalisée

▶ SUBTYPING-WEAKENING:

$$E \vdash S <: T \quad \wedge \quad E \subset F \quad \wedge \quad \vdash F \ ok \quad \Rightarrow \quad F \vdash S <: T$$

- Implémentation en Coq.

```
Lemma subtyping_extension : forall E F S T,
  E |- S <: T  ->  E inc F  ->  ok F  ->
  F |- S <: T.
```

Exemple de preuve : préservation

$$(z : U) \in E \quad \wedge \quad E \vdash t : T \quad \Rightarrow \quad \forall F,\ E \setminus z \subset F$$
$$\wedge \quad F \vdash u : U \quad \Rightarrow \quad F \vdash [z \to u]t : T$$

```
Lemma subst_typing : forall E z u U t T, E has z ~: U ->
  E |- t ~: T  ->  forall F,  E \ z incl F  ->
  F |- u ~: U  ->  F |- [z ~> u]t ~: T.
intros E z u U t T Has Typt.
induction Typt; intros F Incl Typu; simpl*.
  (* Case T-var *)
case_var*. rewrite* (@env_functional z T U E).
  (* Case T-app *)
apply* T_app.
  (* Case T-abs *)
sets WF (@subst_wf (abs U0 t1)). simpl in WF.
apply_T_abs x (z :: dom F ++ dom E ++ L).
rewrite* (@subst_permutation F). apply* H1.
apply* env_subst_push. apply* extends_typing.
```

Statistiques sur les sources Coq

	λ-calcul simplement typé	Propriétés du sous-typage
Définitions	8	9
Axiomes	0	0
Lemmes	26	34
Théorèmes	2	5
Lignes de preuves	63	104
Nombre d'étapes	209	280
...dont étapes clés	25	45
Lignes non vides	289	397

Complexité des solutions Coq

Nombre d'étapes (sans les tactiques du type trivial, assumption, et auto) des solutions en Coq sur la partie 1A du POPLMark Challenge (qui couvre les propriétés de base du sous-typage), par ordre chronologique.

Auteur	Étapes	Représentation
Jérome Vouillon	360	de-Bruijn indices
Aaron Stump	628	mixed names
Xavier Leroy	380	locally nameless
Hirschowitz, Maggesi	1427	de-Bruijn (nested)
Adam Chlipala	500	locally nameless
Arthur Charguéraud	250	locally nameless

6) Conclusion

Une expérience positive

- Un projet avec un objectif clair et précis.

- 5 mois de stage : donne le temps nécessaire.

- Un très bon cadre de travail.

- Des résultats motivants.

- Un stage très formateur.

Partage du savoir

- Discussions avec mes maîtres de stages et les thésards du même labo.

- Présentations lors des réunions de groupes.

- Annonces des résultats principaux sur la mailing list du POPLMark Challenge.

- Documentations des solutions soumises.

- Présentation au 1st Workshop on Mechanizing Metatheory (Portland, 21 Sept 2006).

- Papier de recherche en cours de rédaction.

Bibliographie annotée

1993, James McKinna and Robert Pollack - **Pure Type Systems Formalized**
LEGO, names. Inductive definition of well-formation (not using the set of free variables). Discusses the problem of adequacy of a representation in the conclusion. Discusses the quantification of the variable introduced when passing binders. Some slides.

1994, Gérard Huet - **Residual Theory in Lambda-Calculus** (dvi inside the archive)
Coq, de-Bruijn indices. Early version of Coq: writing recursive functions as proof term was easier than using the tactics. Section 7.2 discusses several representation for free variables. Comparison of Coq (version 5) versus Mizar and Boyer-Moore proof assistants.

1995, Ole Rasmussen - **The Church-Rosser Theorem in Isabelle: A Proof Porting Experiment**
(via citeseer)
Isabelle/ZF, de-Bruijn indices. porting from Huet's proof in Coq (listed above). interesting comparison between Coq and Isabelle, plus some technical comments. conclusion: porting the definitions saves a lot of time, but the proof script does not really help in general.

1996, Tobias Nipkow - **More Church-Rosser Proof in Isabelle/HOL**
Isabelle/HOL, de-Bruijn indices. point by point comparison on several issues. simpler than previous proofs thanks to past experiences and also to improvements of the proof assistant.

1996, Bruno Barras, Benjamin Werner - **Coq in Coq**
Coq, de-Bruijn indices. See also: Bruno Barras, Coq en Coq. Includes confluence and strong normalization for the calculus of construction. -- the link on Barras' webpage to the document is broken.

Travaux futurs

- Compléter la solution proposée sur la dernière partie du Challenge.

- Étendre ces résultats à un vrai langage de programmation, avec plus de constructions.

- Étendre ces résultats à un système de type plus avancé : le Calcul des Constructions.

Merci !