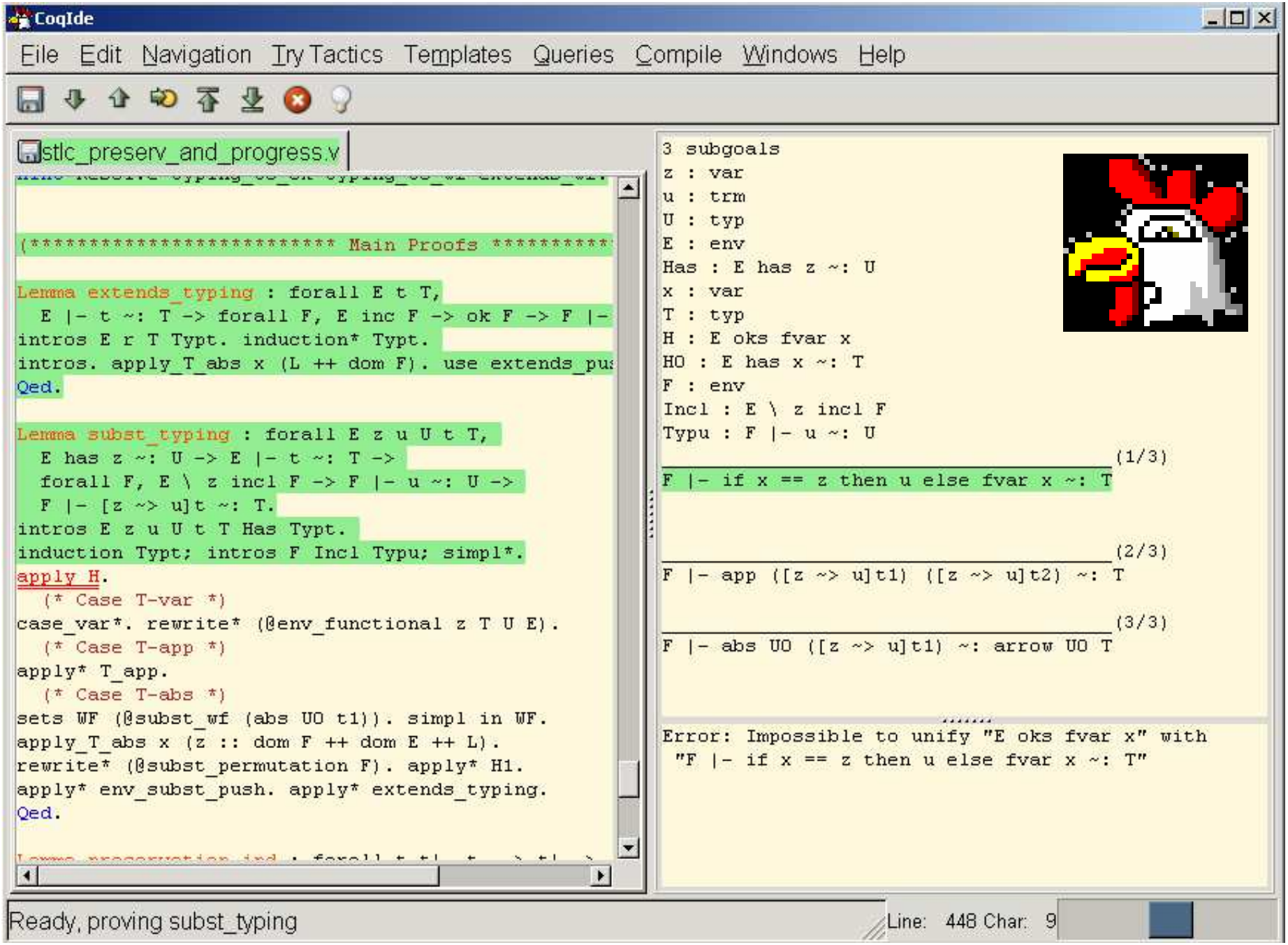
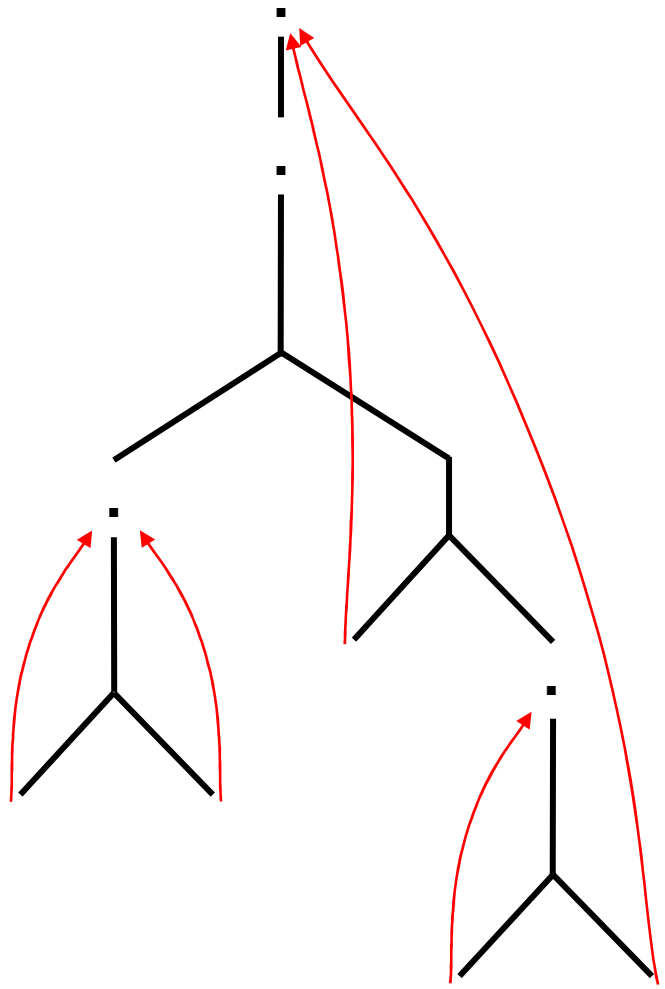
Preuves formelles sur des λ-termes

Arthur Charguéraud

Séminaire Informatique Des Élèves ENS, Paris, 6 Nov. 2006


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_pu
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

Un lambda-terme



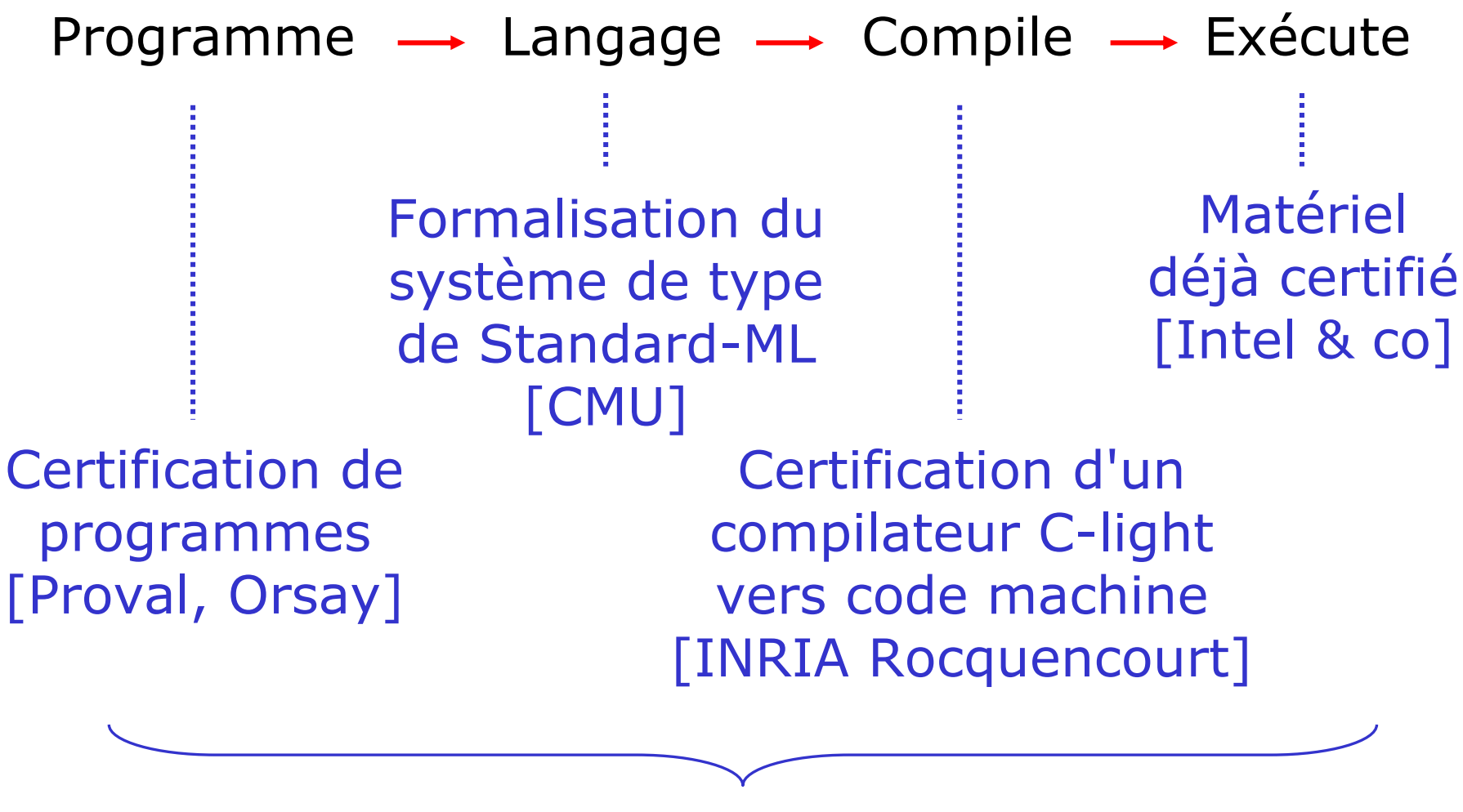
Applications :

– Langages de programmation

– Systèmes de preuves

λa.λb.[(λc. c c) (a (λd. d a))]

Pourquoi faire ?



Le POPLMark Challenge

Mechanized Metatheory for the Masses:

The POPLMark Challenge

[March 2005]

- Pour formaliser les résultats de théorie des types
- Trouver la meilleure manière de s'y prendre
- Donne un challenge représentatif des difficultés.

Approche suivie

POPLmark

Formaliser

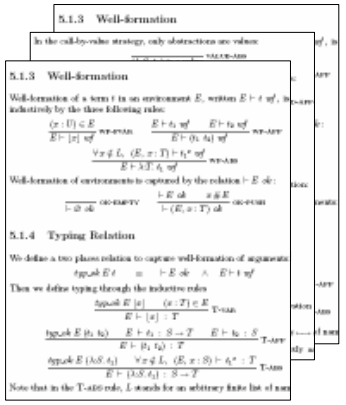
Coq

Preuves sur papier

Simple

Traduire sur machine

Reformuler



De quoi s'occuper

1) Comprendre le Challenge, et apprendre Coq.

2) Analyser les techniques existantes.

3) Expérimenter des nouvelles techniques.

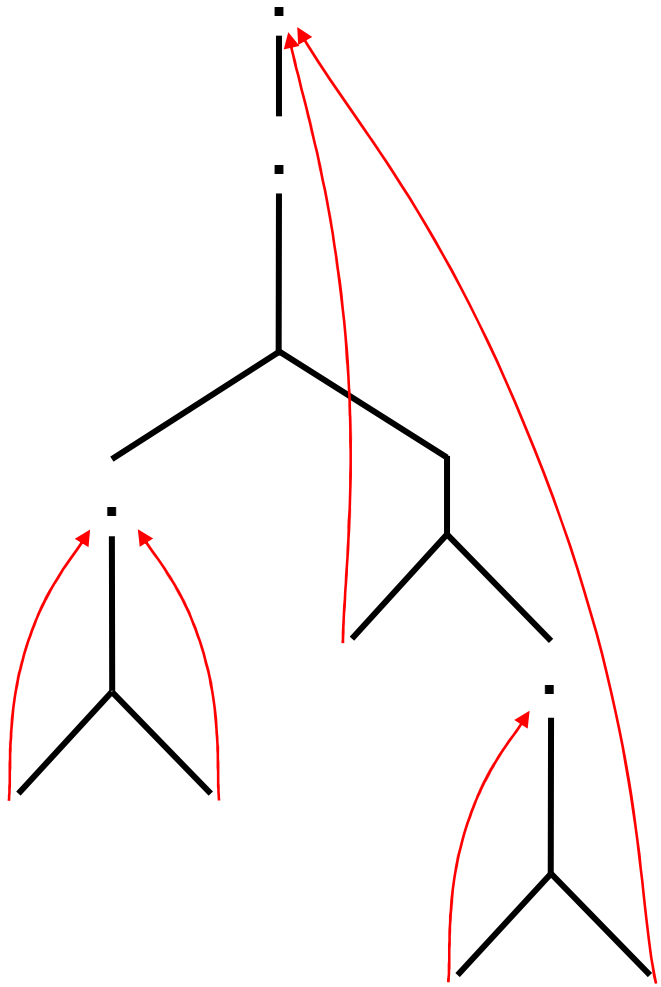
4) Comparer les techniques entre elles.

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

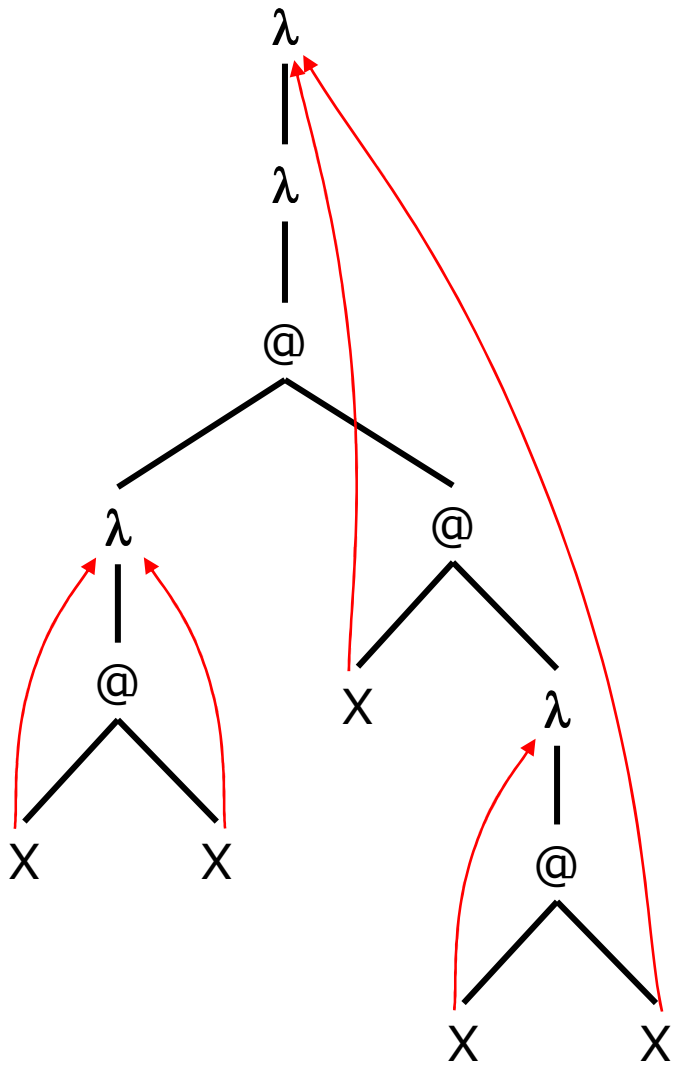
6) Présenter ce travail au Workshop on Mechanizing Metatheory.

Représentation des λ-termes

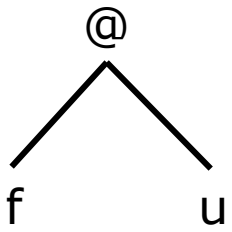
Un lambda-terme



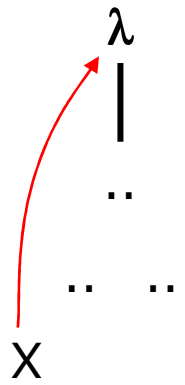
Un lambda-terme



Application

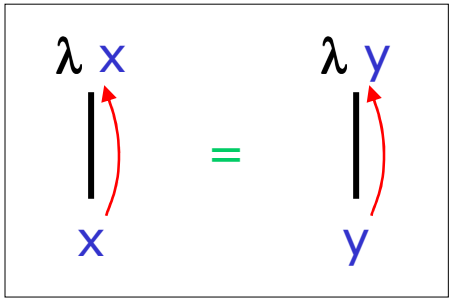


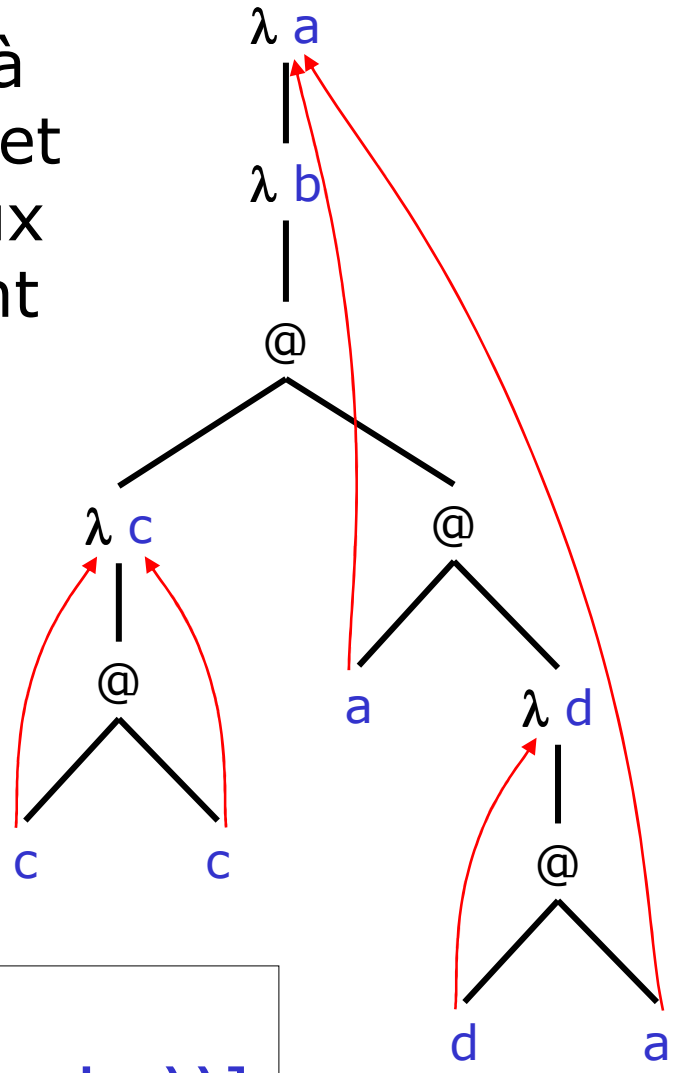
Fonction



λ-terme : avec des noms

Donner un nom à chaque lambda, et le même nom aux variables pointant vers ce lambda.






```
λa.λb.
[(λc. c c) (a (λd. d a))]
```

Pour :

– c'est ce qu'on fait à la main

Contre :

– quotient par α

– α-conversion

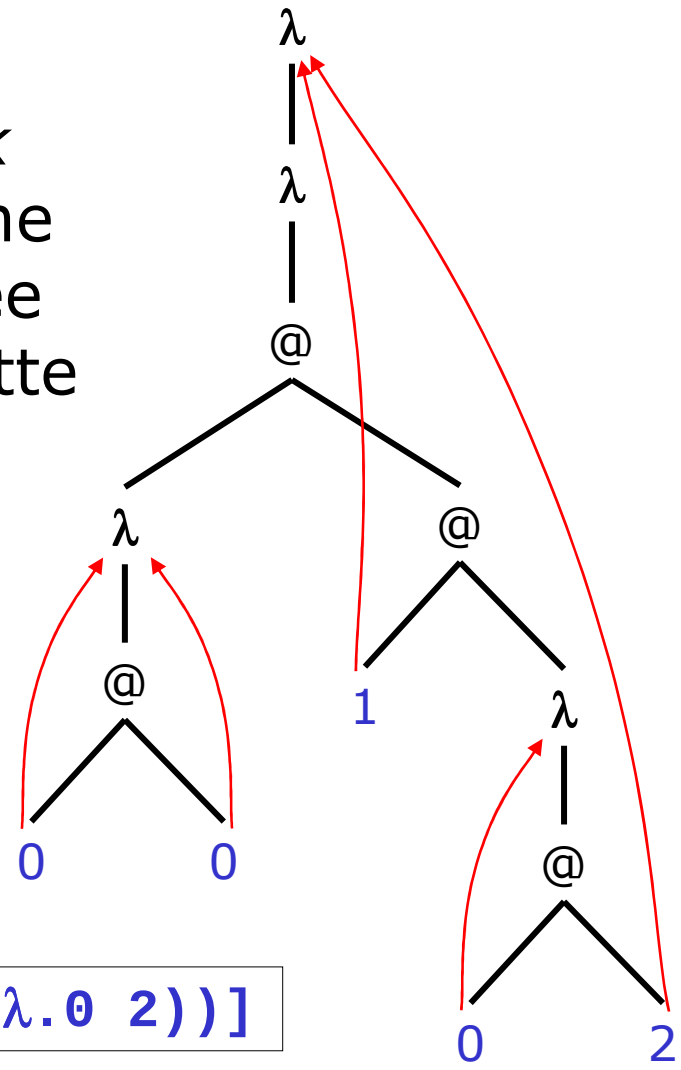
Utilisation :

– sur papier

– code source

λ-terme : 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))]
```

Pour :

– plus de soucis d'α-equivalence

Contre :

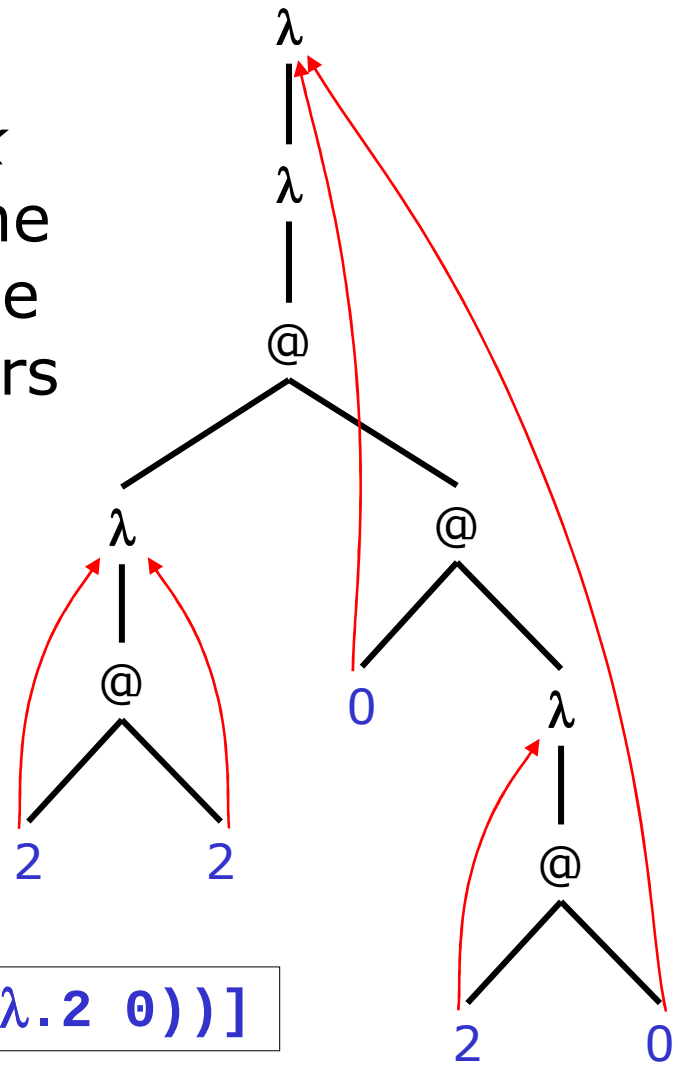
– décalage des indices des variables libres

Utilisation :

– interpréteur de λ-calcul

λ-terme : niveaux de *de Bruijn*

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))]
```

Pour :

– plus de soucis d'α-equivalence

Contre :

– décalage des indices des variables liées

Utilisation :

– aucune

Exemples d'implémentations

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
```

Représentation en ordre supérieur

First-Order Abstract Syntax

exemple avec les noms

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

App (Abs x t) u $\xrightarrow{\beta}$ [x->u]t

Higher-Order Abstract Syntax

exemple en Twelf

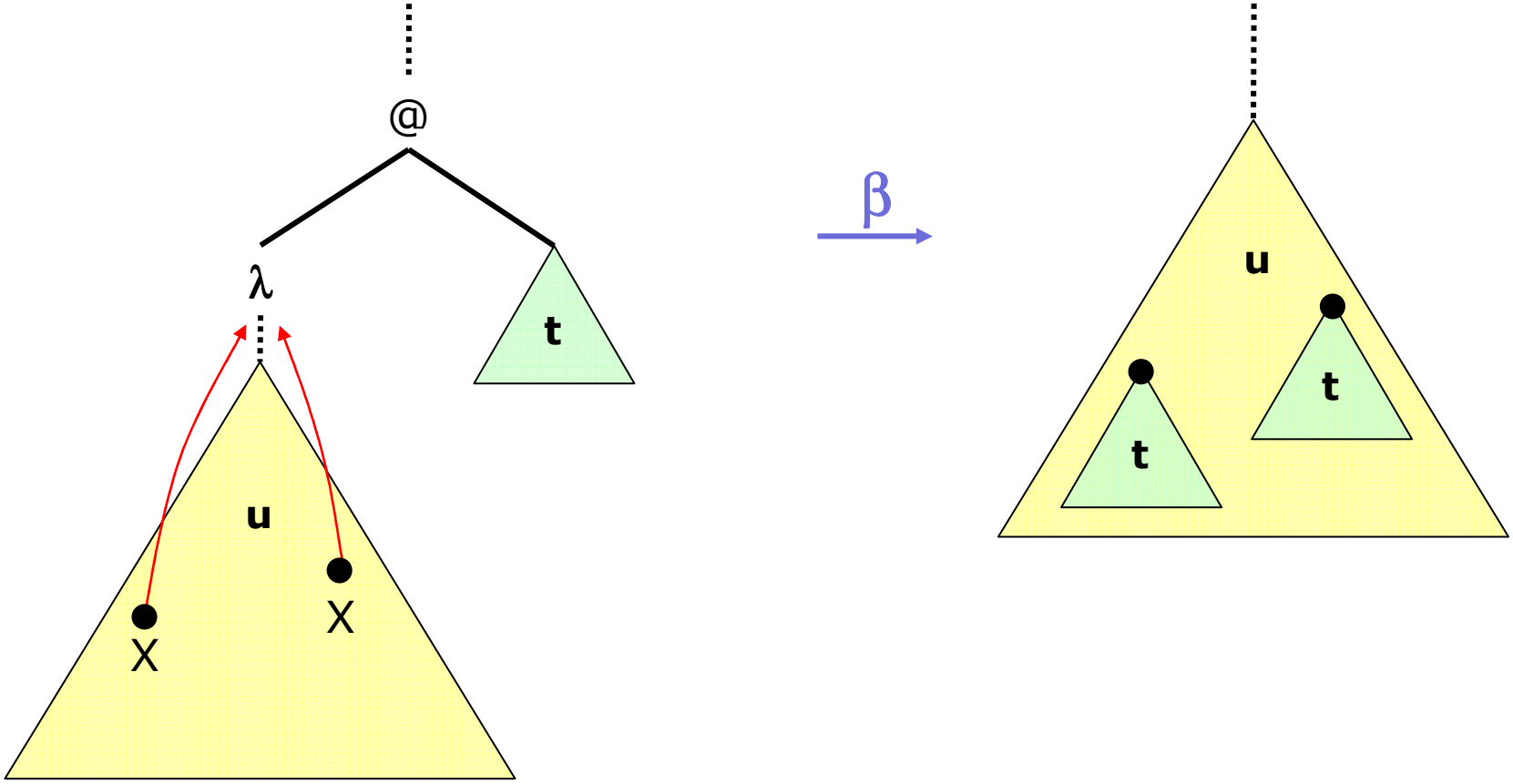
```
term : type

app : term -> term -> term
abs : (term -> term) -> term
```

app (abs t) u $\xrightarrow{\beta}$ t u

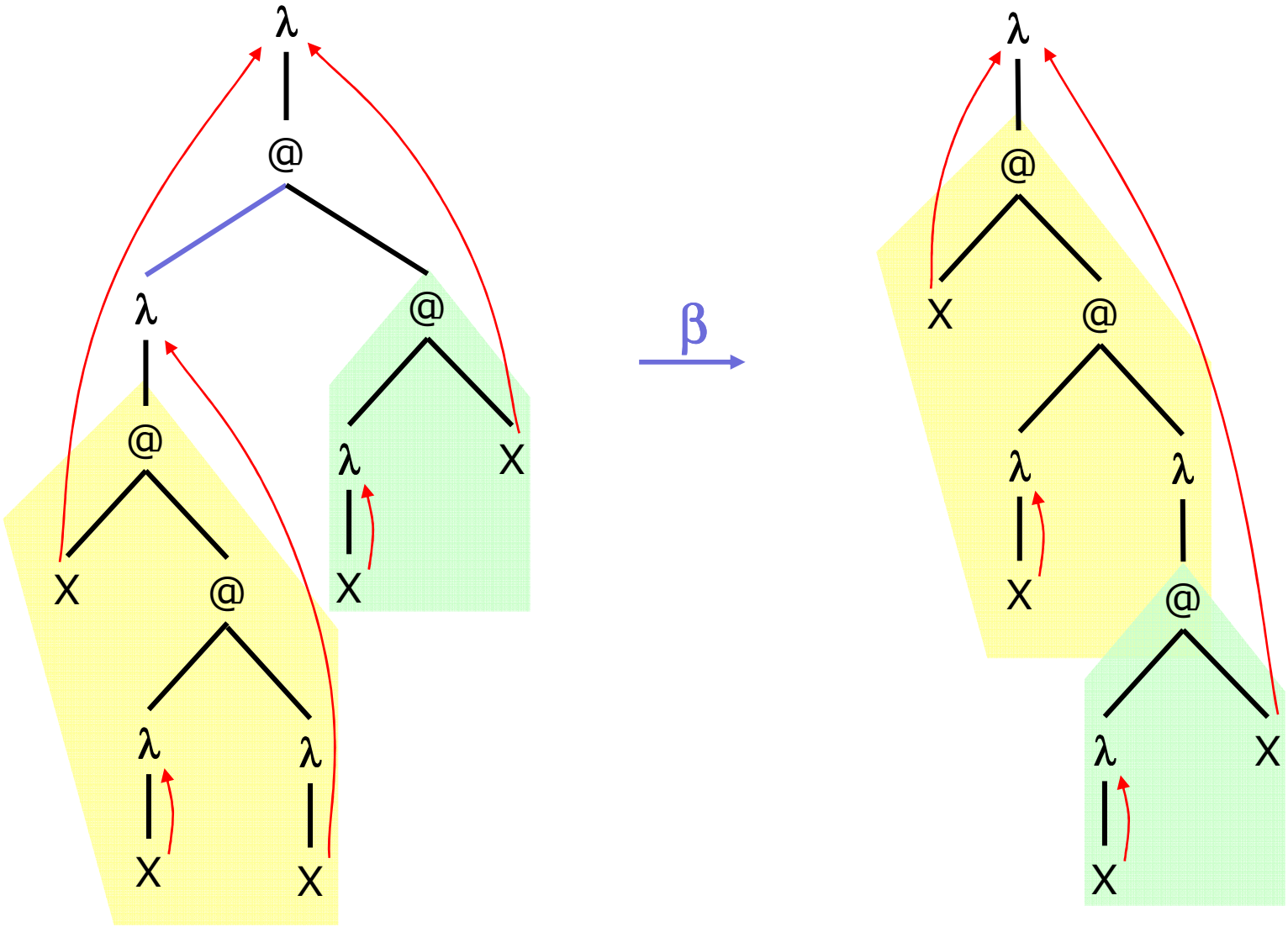
Réduction des λ-termes

β-réduction



`(λx.u) t  se réduit en  [x->t]u`

Exemple de β-réduction

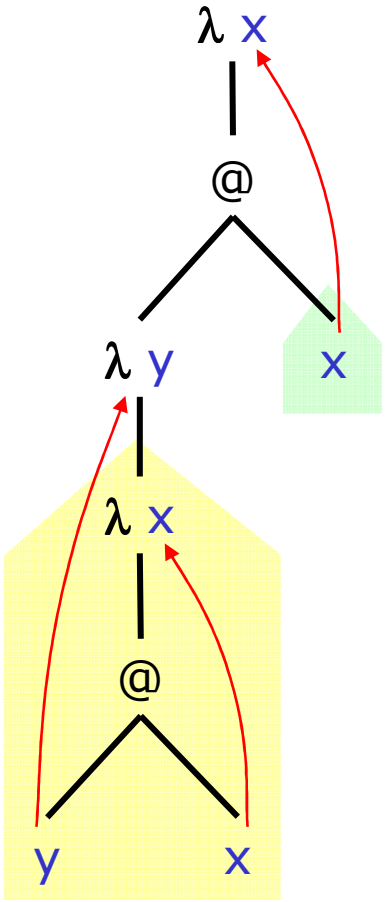


β-réduction avec des noms

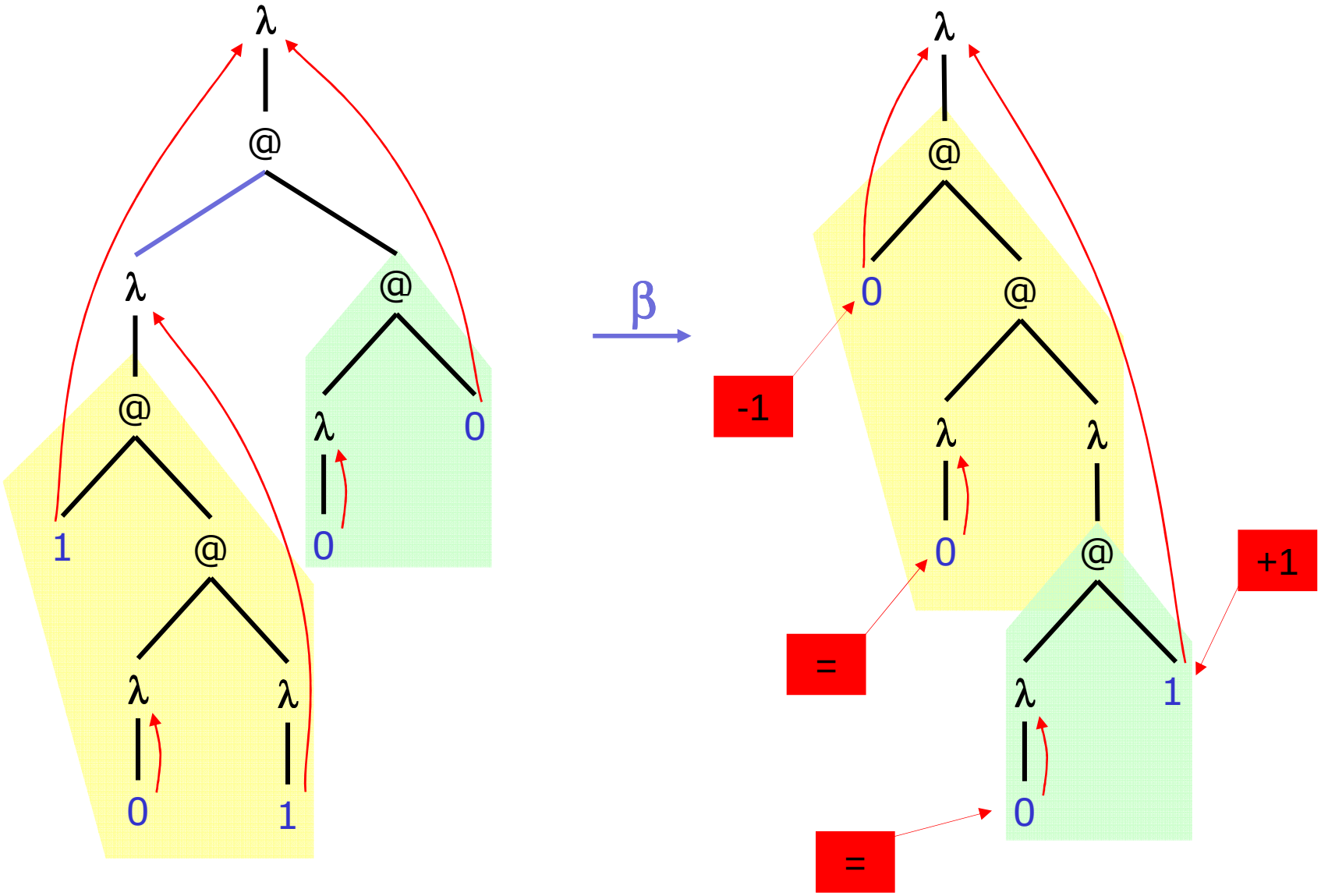
```
   (λz. z z) (λy. λx. y x)
-> (λy. λx. y x) (λy. λx. y x)
-> λx. [(λy. λx. y x) x]
```

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

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



β-réduction avec des indices



Décalages d'indices : *aïe aïe aïe...*

Propriétés du shifting et de la substitution :

$$
\begin{array}{l}
i \leq j \Longrightarrow j \leq i + m \Longrightarrow \uparrow_\tau\ n\ j\ (\uparrow_\tau\ m\ i\ T) = \uparrow_\tau\ (m + n)\ i\ T \\
i + m \leq j \Longrightarrow \uparrow_\tau\ n\ j\ (\uparrow_\tau\ m\ i\ T) = \uparrow_\tau\ m\ i\ (\uparrow_\tau\ n\ (j - m)\ T) \\
k \leq k' \Longrightarrow k' < k + n \Longrightarrow \uparrow_\tau\ n\ k\ T[k' \mapsto_\tau\ U]_\tau = \uparrow_\tau\ (n - 1)\ k\ T \\
k \leq k' \Longrightarrow \uparrow_\tau\ n\ k\ (T[k' \mapsto_\tau\ U]_\tau) = \uparrow_\tau\ n\ k\ T[k' + n \mapsto_\tau\ U]_\tau \\
k' < k \Longrightarrow \uparrow_\tau\ n\ k\ (T[k' \mapsto_\tau\ U]_\tau) = \uparrow_\tau\ n\ (k + 1)\ T[k' \mapsto_\tau \uparrow_\tau\ n\ (k - k')\ U]_\tau \\
k \leq k' \Longrightarrow \uparrow_\tau\ n\ k'\ (T[k \mapsto_\tau\ Top]_\tau) = \uparrow_\tau\ n\ (Suc\ k')\ T[k \mapsto_\tau\ Top]_\tau \\
k \leq k' \Longrightarrow k' \leq k + n \Longrightarrow \uparrow\ n'\ k'\ (\uparrow\ n\ k\ t) = \uparrow\ (n + n')\ k\ t \\
i \leq j \Longrightarrow T[Suc\ j \mapsto_\tau\ V]_\tau[i \mapsto_\tau\ U[j - i \mapsto_\tau\ V]_\tau]_\tau = T[i \mapsto_\tau\ U]_\tau[j \mapsto_\tau\ V]_\tau
\end{array}
$$

source: Berghofer 2005

Autre problème : exemple du weakening

$$E \vdash t : T \quad \Rightarrow \quad (E, F) \vdash t : T$$

$$\Gamma \vdash t : T \Longrightarrow \Delta\ @\ \Gamma \vdash \uparrow\ \|\Delta\|\ 0\ t : \uparrow_\tau\ \|\Delta\|\ 0\ T$$

Noms ou Indices ?

Un peu d'histoire

Yr	Author	Formalization	Prover	Encoding
85	Natarajan Shankar	Church-Rosser in λ	Boyer-Moore	de-Bruijn
93	Thorsten Altenkirch	SN of System-F	LEGO	de-Bruijn
93	J.McKinna, R.Pollack	Pure Type Systems	LEGO	names
94	Gérard Huet	Residual Theory in λ	Coq	de-Bruijn
95	Ole Rasmussen	Church-Rosser in λ	Isabelle/ZF	de-Bruijn
96	Tobias Nipkow	Church-Rosser in λ	Isabelle/HOL	de-Bruijn
96	B.Barras, B.Werner	Kernel of Coq	Coq	de-Bruijn
97	J.McKinna, R.Pollack	λ-calculus & types	LEGO	names
01	Vestergaard, Brotherston	Church-Rosser in λ	Isabelle/HOL	names
01	J.Ford, I.Mason	Church-Rosser in λ	PVS	names
01	Peter Homeier	Church-Rosser in λ	HOL	names

Soumissions sur le POPLMark

Author	Part 1	Part 2	Prover	Encoding
Stephan Berghofer	Y	Y	Isabelle	de-Bruijn indices
Ashley, Crary, Harper	Y	Y	Twelf	higher-order
Jérome Vouillon	Y	Y	Coq	de-Bruijn indices
Hongwei Xi		Y	ATS/LF	higher-order
Jevgenijs Sallinens	Y		Coq	de-Bruijn indices
Xavier Leroy	Y		Coq	locally nameless
Aaron Stump	Y		Coq	names / levels
Christian Urban	Y		Isabelle	nominal
Hirschowitz, Maggesi	Y		Coq	de-Bruijn (nested)
Adam Chlipala	Y		Coq	locally nameless
Arthur Charguéraud	Y		Coq	locally nameless

Distinguer variables libres et variables liées

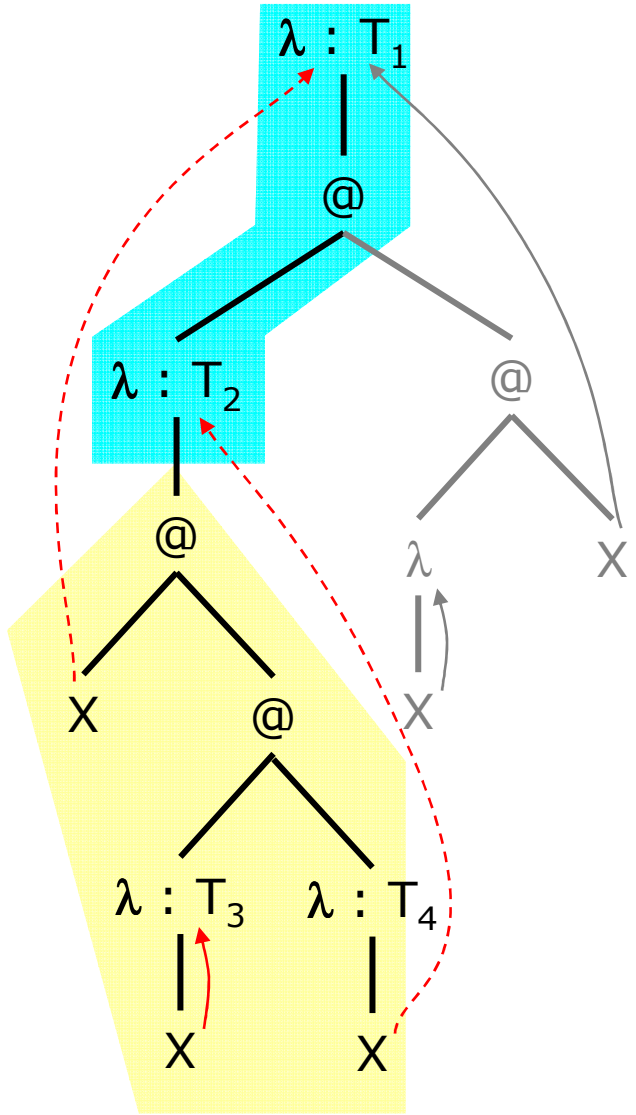
Variables liées et libres

environement *E*

jugement de typage

E **|-** ***t*** **:** ***T***

terme *t*



Locally nameless

– *variables liées representées avec des indices de de-Bruijn,*
– *variables libres representées à l'aide de noms.*

$$t := \lceil i \rceil \mid \lfloor x \rfloor \mid (t_1\ t_2) \mid \lambda{:}S.\ t_1$$

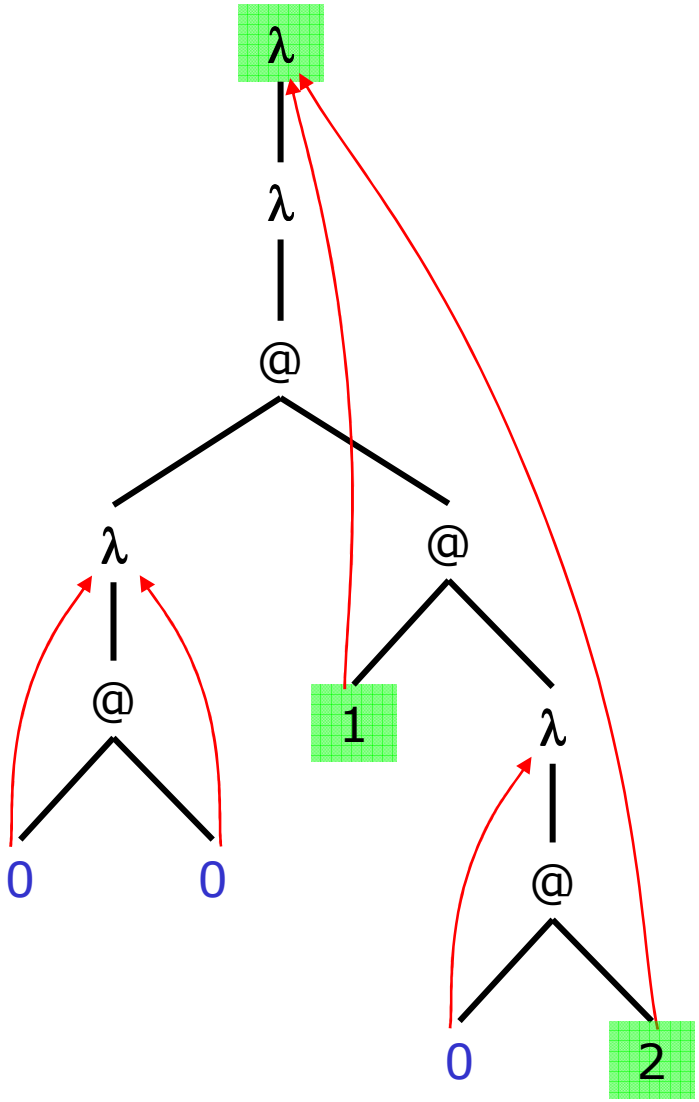
en Caml

```
type term =
  | Bvar of int
  | Fvar of string
  | App of term * term
  | Abs of string * term
```

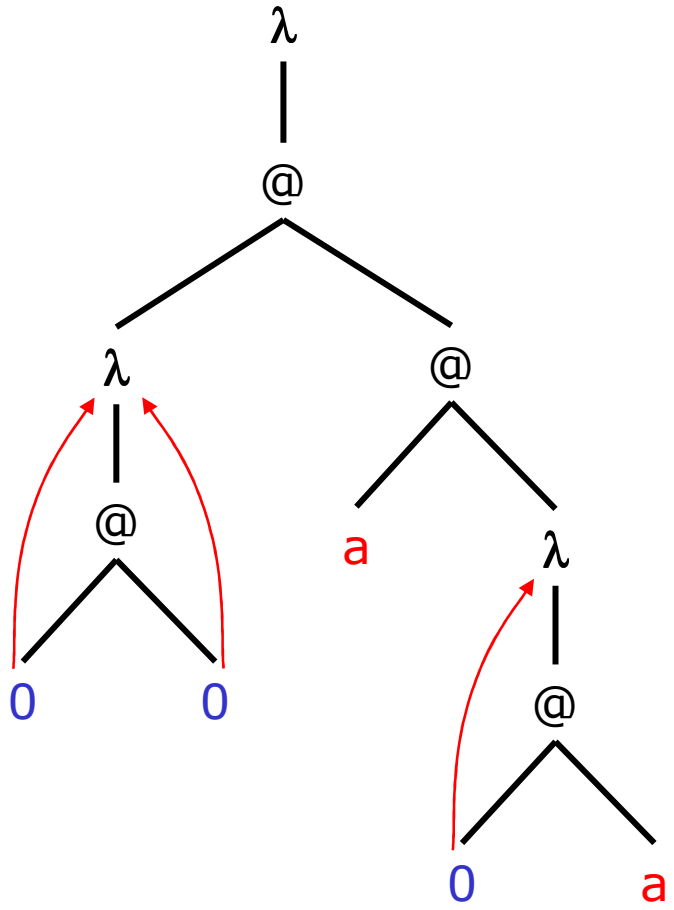
en Coq

```
Inductive term : Set :=
  | Bvar : nat -> term
  | Fvar : name -> term
  | App : term -> term -> term
  | Abs : name -> term -> term
```

Ouvrir les abstractions



a



Deux substitutions

Substitution de la variable liée *k* par un terme *u* dans *t* :

{k –> u}t

$$
\begin{array}{lcl}
\{k \to u\}\,\lceil i \rceil & \equiv & \text{if } i = k \text{ then } u \text{ else } \lceil i \rceil \\
\{k \to u\}\,\lfloor x \rfloor & \equiv & \lfloor x \rfloor \\
\{k \to u\}\,(t_1\ t_2) & \equiv & ((\{k \to u\}\,t_1)\ (\{k \to u\}\,t_2)) \\
\{k \to u\}\,(\lambda{:}T.\ t_1) & \equiv & \lambda{:}T.\ (\{(k+1) \to u\}\,t_1)
\end{array}
$$

Substitution de la variable libre *z* par un terme *u* dans *t* :

[k –> u]t

$$
\begin{array}{lcl}
[z \to u]\,\lceil i \rceil & \equiv & \lceil i \rceil \\
[z \to u]\,\lfloor x \rfloor & \equiv & \text{if } x = z \text{ then } u \text{ else } \lfloor x \rfloor \\
[z \to u]\,(t_1\ t_2) & \equiv & (([z \to u]\,t_1)\ ([z \to u]\,t_2)) \\
[z \to u]\,(\lambda{:}T.\ t_1) & \equiv & \lambda{:}T.\ ([z \to u]\,t_1)
\end{array}
$$

β-réduction en locally nameless

$$\frac{}{((\lambda{:}S.\ t_1)\ \ t_2) \longmapsto {t_1}^{t_2}}\ \text{RED-BETA}$$

$$\frac{t_1 \longmapsto t_1'}{(t_1\ \ t_2) \longmapsto (t_1'\ \ t_2)}\ \text{RED-APP-1} \qquad \frac{t_2 \longmapsto t_2'}{(t_1\ \ t_2) \longmapsto (t_1\ \ t_2')}\ \text{RED-APP-2}$$

$$\frac{{t_1}^{x} \longmapsto {t_1'}^{x}}{(\lambda{:}S.\ t_1) \longmapsto (\lambda{:}S.\ t_1')}\ \text{RED-ABS} \qquad x \# t_1,\ x \# t_1'$$

Où: $\quad {t_1}^{t_2} \equiv \{0 \to t_2\}\, t_1$ $\quad$ *et* $\quad {t_1}^{x} \equiv \{0 \to \lfloor x \rfloor\}\, t_1$

Résumé des représentations

	variables libres	variables liées
noms	nécessite de traiter l'α-equivalence	ok
indices de Bruijn	ok	nécessite de gérer du shifting
niveaux de Bruijn	nécessite de gérer du shifting	nécessite de gérer du shifting

Le gagnant est : locally nameless

Résultats formalisés

Le POPLMark Challenge

Préservation et Progression pour System-F<:

POPLMark Partie 1:

Propriétés du sous-typage

POPLMark Partie 2:

Reste de la formalisation

[étendue]

[simplifiée]

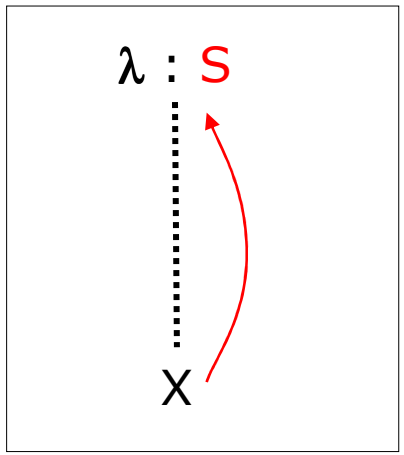
Mon Challenge B

Propriétés du sous-typage incluant sa préservation par subsitution de type

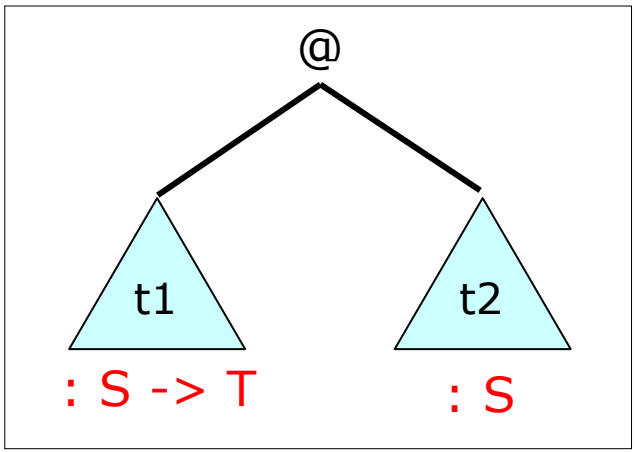
Mon Challenge A

Préservation et progression pour le λ-calcul simplement typé

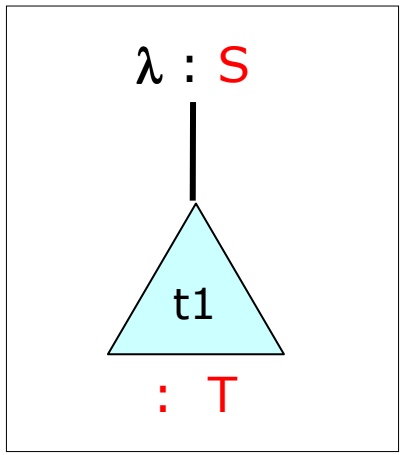
Typage simple des λ-termes



S



T



S -> T

$$\frac{(x : T) \in E}{E \vdash x : T}\ \text{T-VAR} \qquad \frac{E \vdash t_1 : S \to T \qquad E \vdash t_2 : S}{E \vdash (t_1\ t_2) : T}\ \text{T-APP}$$

$$\frac{x \# E \qquad E, x : S \vdash t_1 : T}{E \vdash (\lambda x{:}S.\ t_1) : S \to T}\ \text{T-ABS}$$

A) λ-calcul simplement typé

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

$$\frac{(x:T) \in E}{E \vdash x : T}\ \text{T-VAR} \qquad \frac{E \vdash t_1 : S \to T \qquad E \vdash t_2 : S}{E \vdash (t_1\ t_2) : T}\ \text{T-APP}$$

$$\frac{x \# E \qquad E, x:S \vdash t_1 : T}{E \vdash (\lambda x{:}S.\ t_1) : S \to T}\ \text{T-ABS}$$

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

Progress: $\varnothing \vdash t : T \ \Rightarrow\ [\, t \text{ is a value} \ \vee\ \exists t',\ t \longmapsto t' \,]$

B) Sous-typage dans System-$F_{<:}$

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

$$\frac{}{E \vdash S <: \mathrm{Top}}\ \textsc{sa-top} \qquad \frac{E \vdash T_1 <: S_1 \qquad E \vdash S_2 <: T_2}{E \vdash (S_1 \to S_2) <: (T_1 \to T_2)}\ \textsc{sa-arrow}$$

$$\frac{}{E \vdash X <: X}\ \textsc{sa-refl-tvar} \qquad \frac{(X{<:}U) \in E \qquad E \vdash U <: T}{E \vdash X <: T}\ \textsc{sa-trans-tvar}$$

$$\frac{E \vdash T_1 <: S_1 \qquad X \# E \qquad (E,\, X{<:}T_1) \vdash S_2 <: T_2}{E \vdash (\forall X{<:}S_1.\, S_2) <: (\forall X{<:}T_1.\, T_2)}\ \textsc{sa-all}$$

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 :

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

Preuves en Coq

ex: affaiblissement du sous-typage

Informel:

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

Preuve par induction sur la dérivation de sous-typage, utilisant le lemme de permutation pour le cas SA-all.
α-equivalence, convention de Barendregt, etc...

Formalisable:

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

Preuve par induction sur la dérivation de sous-typage, facile sauf pour le cas SA-all: choisit une variable X en dehors de dom(F) et utilise le lemme "extends_push".

Formel:

```
Lemma sub_extension : forall E S T, E |- S <: T
    -> forall F, E inc F -> ok F -> F |- S <: T.
intros E S T H. induction H; intros; auto**.
apply_SA_all X (L ++ dom F). use extends_push.
```

ex: transitivité du sous-typage

```
Theorem subtyping_transitivity : forall E S Q T,
  E |- S <: Q  ->  E |- Q <: T  ->  E |- S <: T.

intros. apply* (@sub_transitivity E Q). Qed.

Lemma sub_transitivity :
 forall E Q (WQ : E wf Q), sub_trans_prop WQ.

intros. unfold sub_trans_prop. generalize_equality Q Q'.
induction WQ; intros Q' EQ F S T EincF SsubQ QsubT;
induction SsubQ; try discriminate; try injection EQ; intros;
inversion QsubT; subst; intuition eauto.
  (* Case SA-arrow *)
apply SA_arrow. auto. apply* IHWQ1. apply* IHWQ2.
  (* Case SA-all *)
apply_SA_all X ((dom E0) ++ L ++ L0 ++ L1). apply* H0.
asserts* WQ1 (E0 wf T1). apply* (sub_narrowing (WQ := WQ1)).
Qed.
```

Statistiques sur les scripts Coq

	λ-calculus simplement typé	Propriétés du sous-typage
Définitions	8	9
Axiomes	0	0
Lemmes	26	34
Théoremes	2	5
Lignes de preuves	63	104
Nombre de tactiques	202	279
Tactiques non triviales dans les résultats clés :	36	67

Complexité des solutions en Coq

Nombre de tactiques utilisées, sans compter les appels à *auto*, sur POPLMark Challenge partie 1A (propriétés du sous-typage).

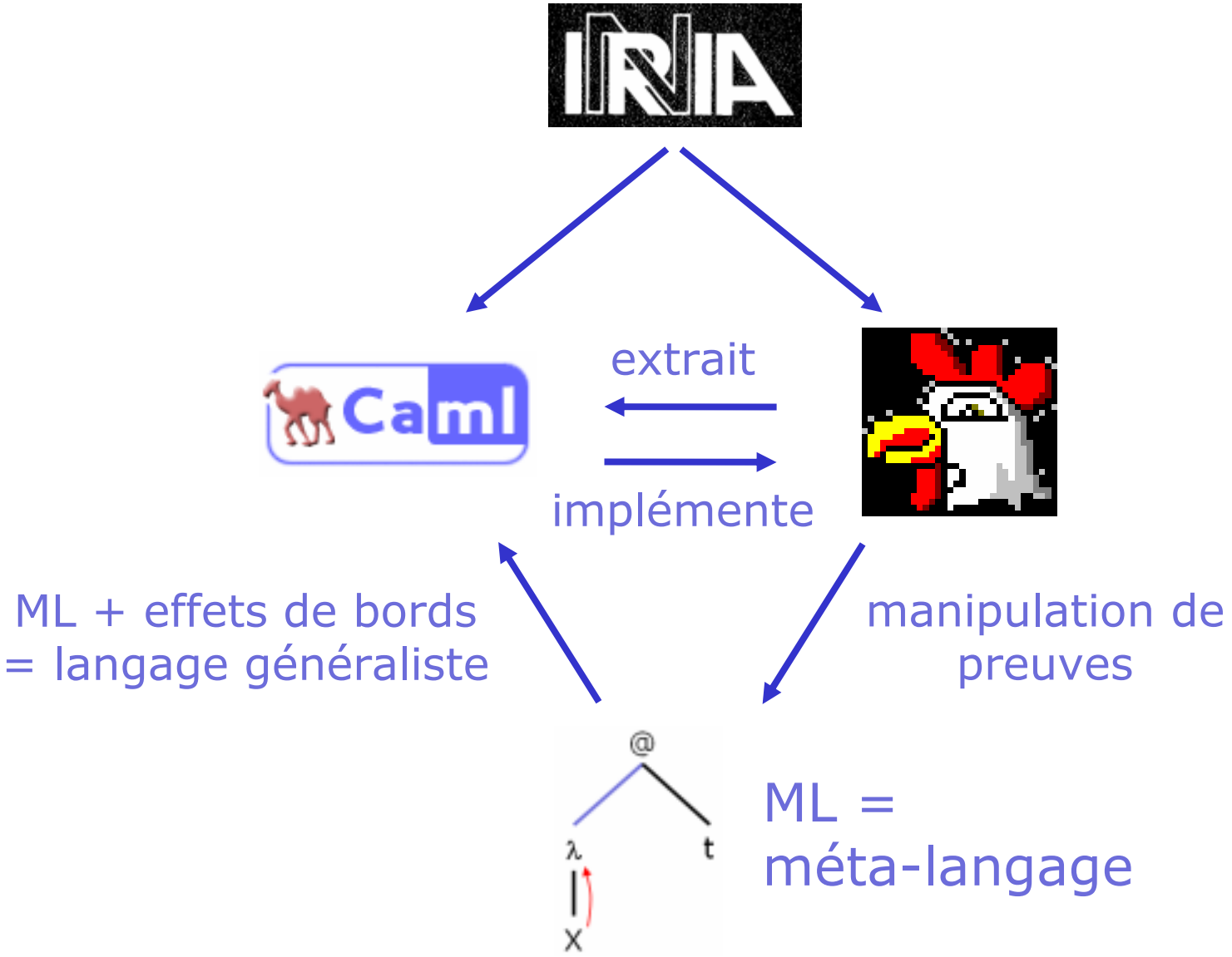
Auteur	Tactiques	Représentation
Jérome Vouillon	431	de-Bruijn indices
Aaron Stump	1147	names / levels
Xavier Leroy	630	locally nameless
Hirschowitz, Maggesi	1615	de-Bruijn (nested)
Adam Chlipala	342	locally nameless
Arthur Charguéraud	233	locally nameless

Conclusions

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 visibles.
- Un stage très formateur.

Coq c'est aussi cool que Caml



Locally nameless : c'est puissant

– Locally nameless a été utilisé pour implémenter des type checkers (Coq, LEGO, HOL4, Epigram) de manière simple et efficace.

– Locally nameless nous a permis de réaliser des preuves courtes et intuitives, relativement fidèles aux preuves sur papier.

– Le travail de McKinna et Pollack (variables libres et liées distinguées, mais les deux avec des noms) pourrait être simplifié en locally nameless.

Locally nameless : pas nouveau

1972: N.G. de-Bruijn

A Lambda Calculus Notation with Nameless Dummies, a Tool for Automatic Formula Manipulation, with Application to the Church-Rosser Theorem.

1989: G. Huet

The Constructive Engine

1994: A. Gordon

A Mechanisation of Name-carrying Syntax up to Alpha-conversion

2005-2006: X. Leroy, A. Charguéraud, A. Chlipala,

Solutions to the POPLMark Challenge

Prolongement de ces travaux

– Étendre la solution à la totalité du POPLMark.

– Formaliser des propriétés du λ-calcul.

– S'attaquer à des systèmes de types plus complexes (ex : Coq).

– S'attaquer à des langages de programmation avec des constructions évoluées (ex : Caml).

[contacter Benjamin Pierce]

Merci !

Environments as Lists or Sets?

Weakening Preserves Typing

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

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

where: $$E \subset F \quad \equiv \quad \forall x\, T, \quad (x : T) \in E \quad \Rightarrow \quad (x : T) \in F$$

Substitution Preserves Typing

Paper: $$E, z : U, F \vdash t : T \quad \wedge \quad E \vdash u : U \quad \Rightarrow \quad E, F \vdash [z \rightarrow u]t : T$$

Formal: $$\begin{array}{l} E \vdash t : T \quad \wedge \quad F \vdash u : U \quad \wedge \\ (z : U) \in E \quad \wedge \quad E \setminus z \subset F \quad \Rightarrow \quad F \vdash [z \rightarrow u]t : T \end{array}$$

where: $$E \setminus z \subset F \quad \equiv \quad \forall x\, T, \quad (x : T) \in E \quad \wedge \quad x \neq z \quad \Rightarrow \quad (x : T) \in F$$

Names pushed in the Environment

$$\frac{Quantify(x) \qquad (E,\, x : S) \vdash (t_1{}^x) \;:\; T}{E \vdash (\lambda{:}S.\; t_1) \;:\; S \to T}\ \text{T-ABS}$$

Quantify(x) =	∃ **x # E**	∀ **x # E**	∀ **x ∉ L**
Weakening	**swapping required**	**ok**	**ok**
Substitution	**ok**	**swapping required**	**ok**
Transitivity	**swapping required**	**ok**	**ok**

Weakening $E \vdash t : T \quad \Rightarrow \quad E, F \vdash t : T$

Substitution $E, z : U, F \vdash t : T \quad \wedge \quad E \vdash u : U \quad \Rightarrow \quad E, F \vdash [z \to u]t : T$

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

Well-formedness of Terms

- With recursive functions
  - all the free variables of *t* belong to the domain *D*
  - indices are smaller than the number of lambda above them

FV(t) ⊂ D ∧ wf_indices(0,t)

- With an inductive relation

t : term ∧ D |- t wf

$$\frac{x \in D}{D \vdash \lfloor x \rfloor\ wf}\ \text{WF-FVAR} \qquad \frac{D \vdash t_1\ wf \qquad D \vdash t_2\ wf}{D \vdash (t_1\ \ t_2)\ wf}\ \text{WF-APP}$$

$$\frac{\forall x \notin L,\ \ (D \cup \{x\}) \vdash ({t_1}^x)\ wf}{D \vdash (\lambda{:}T.\ t_1)\ wf}\ \text{WF-ABS}$$

- With dependent types

t : term D