Collaboration d'outils de preuve interactifs et automatiques

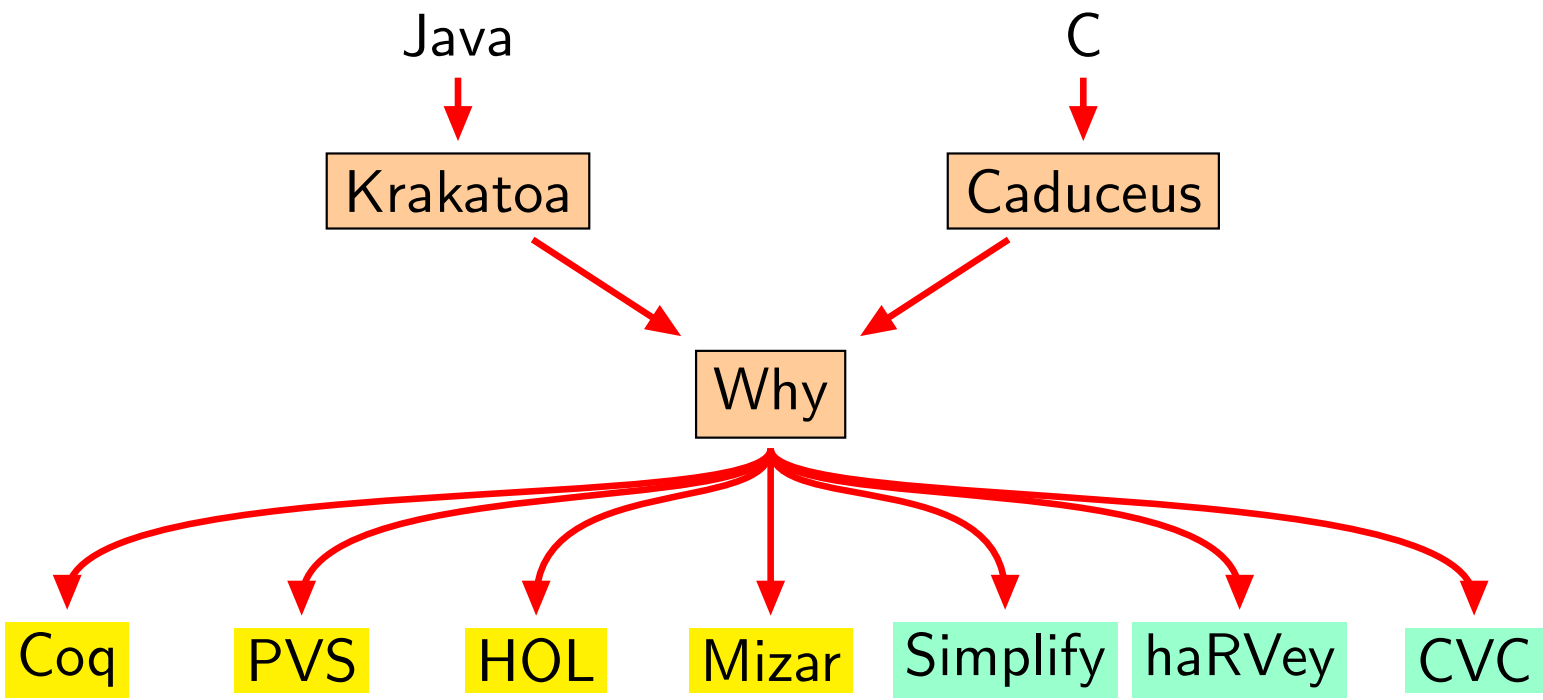
Nicolas Ayache

Stage de Master 2
Équipe Démons, LRI – Université Paris Sud
responsable : Jean-Christophe Filliâtre

8 septembre 2005

Motivation

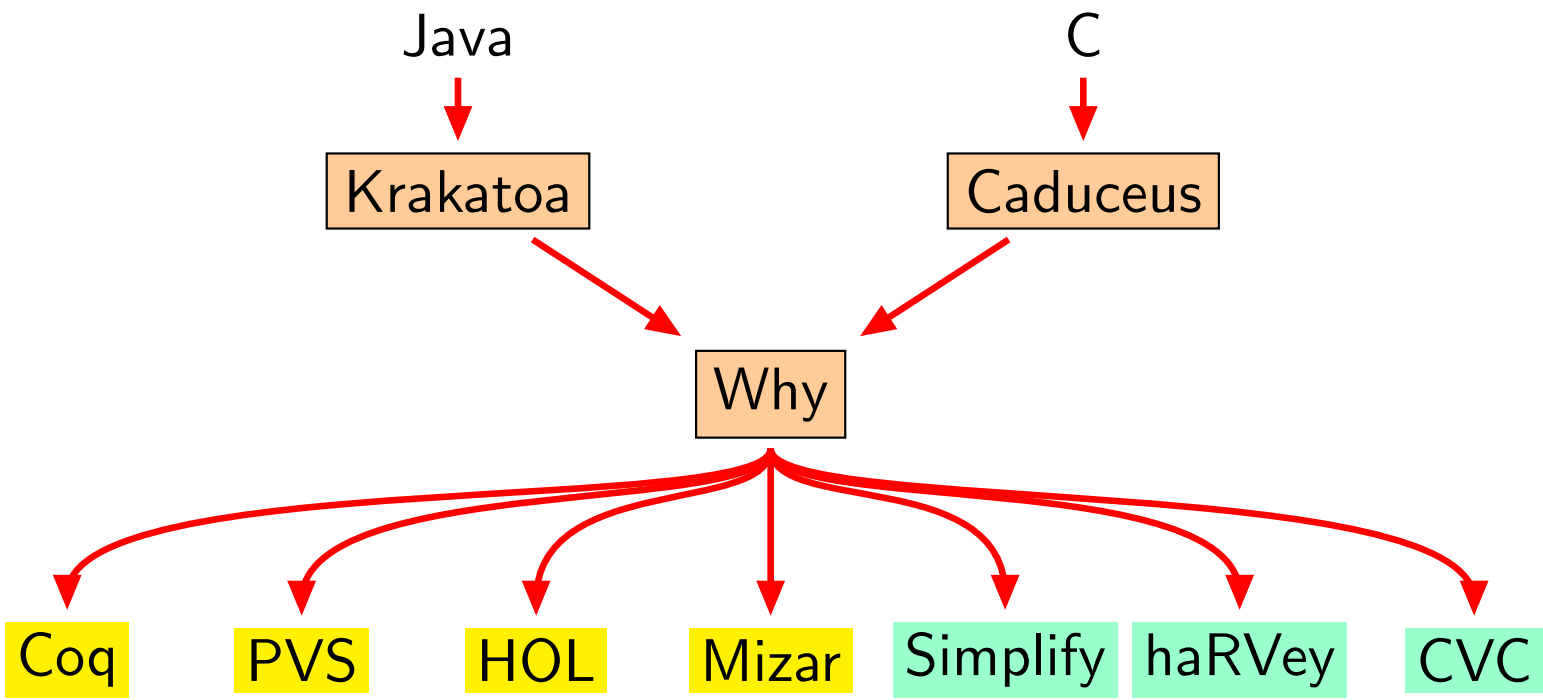
- Contexte : preuve de programmes



- Généricité : facilité pour interfacer de nouveaux prouveurs

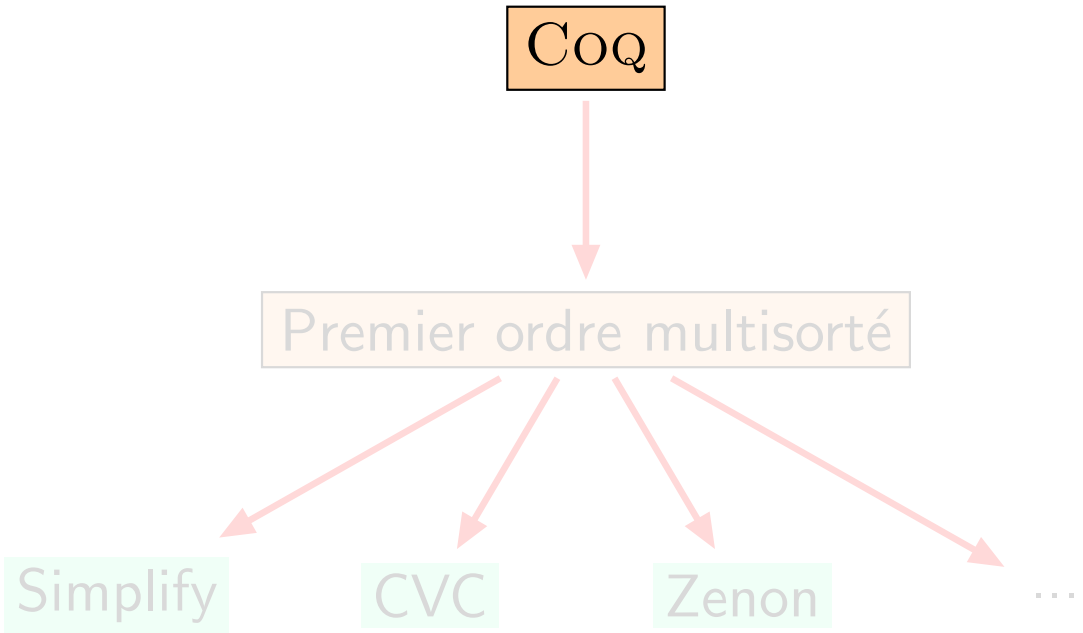
Motivation

- Contexte : preuve de programmes

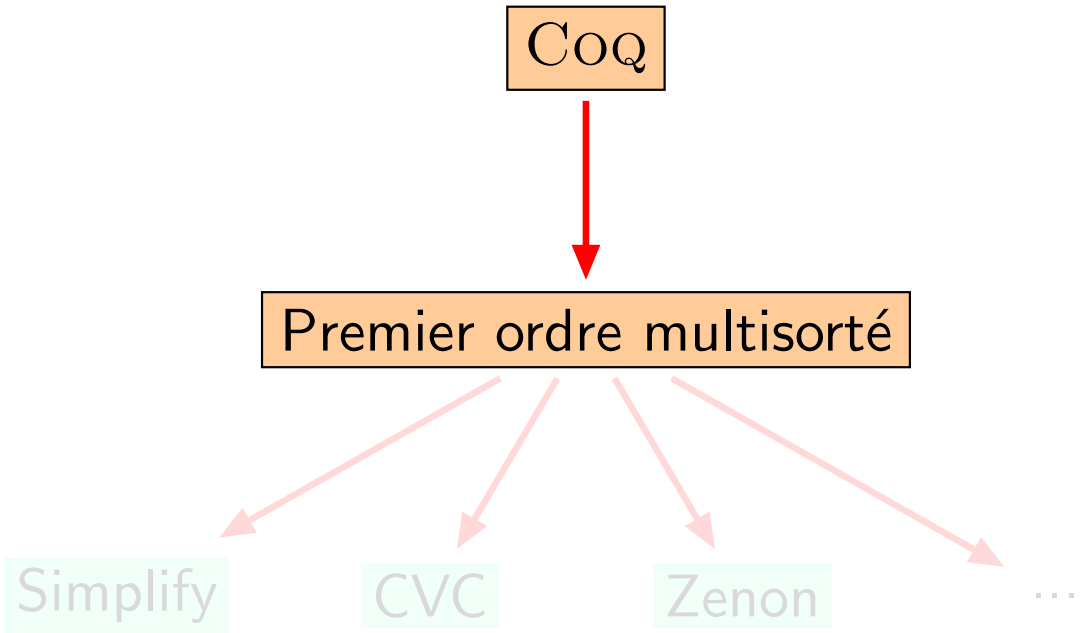


- Généricité : facilité pour interfacer de nouveaux prouveurs

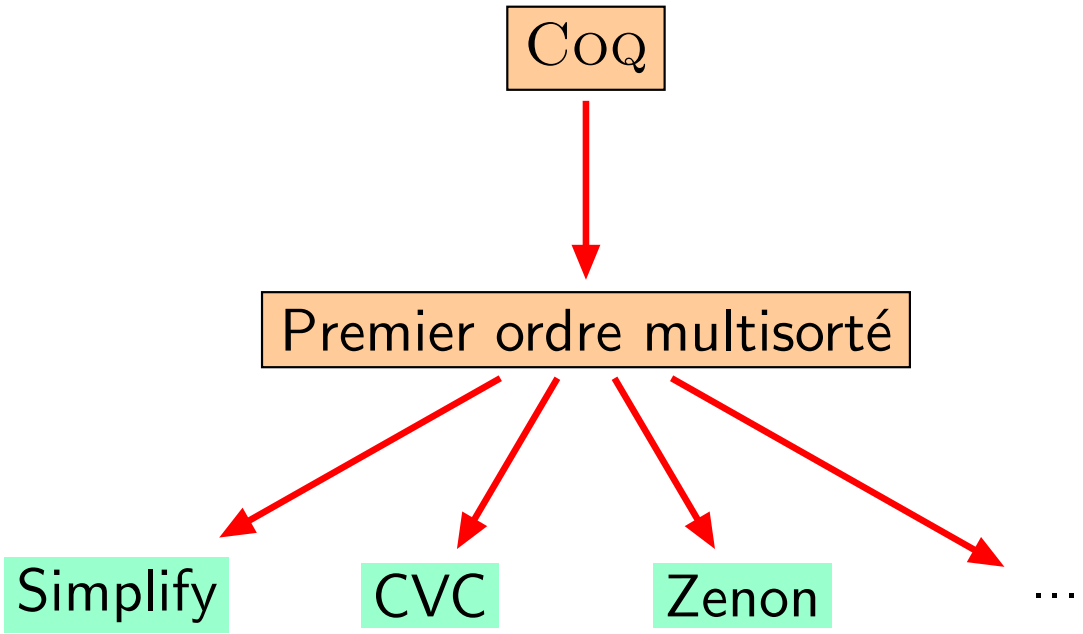
Factorisation



Factorisation



Factorisation



Travaux connexes

- Interface HOL / MESON par combinateurs (Joe Hurd)
- Interface Isabelle / CVC (Tom Harke)
- Interface Coq / JProver (Huang Guan-Shieng)

Plan

1. Traduction CCI vers logique du premier ordre multisortée
   - Logiques
   - Propriété de correction
2. Utilisation en pratique
   - Appel interactif
   - Exemple
3. Perspectives

Premier ordre multisorté

- Sortes
  - nat
- Signature et termes
  - $0 : \text{nat}$
  - $S : \text{nat} \longrightarrow \text{nat}$
  - $S(S(0)) : \text{nat}$
- Propositions
  - $pair : \text{nat}$
  - $\forall x : \text{nat},\ pair(x) \Rightarrow pair(S(S(x)))$

Typage, prouvabilité

Règles d'inférence, jugements

- $\Gamma \vdash_{FOL} t : S$
- $\Gamma \vdash_{FOL} P\ bf$

Système de déduction naturelle

- $\Gamma; \Delta \models P$

Calcul des Constructions Inductives

λ-calcul typé avec :

- polymorphisme : $\lambda\ A : \mathrm{Set}.\ \lambda\ x : A.\ x$
- ordre supérieur : $\forall\ f : \mathrm{nat} \to \mathrm{nat},\ f\ 0 = 0$
- types dépendants :
  $\mathrm{append} : \forall n\ m : \mathrm{nat},\ \mathrm{list}\ n \to \mathrm{list}\ m \to \mathrm{list}\ (n + m)$
- inductifs : $\mathrm{nat} := 0 : \mathrm{nat} \mid S : \mathrm{nat} \to \mathrm{nat}$

Typage, prouvabilité :

- $E; G \vdash_{CCI} t : T$
- isomorphisme de Curry-Howard

$$\lambda\ A : \mathrm{Set}.\ \lambda\ x : A.\ x \quad : \quad \forall A : \mathrm{Set},\ A \to A$$

Calcul des Constructions Inductives

λ-calcul typé avec :

- polymorphisme : $\lambda\ A : \mathrm{Set}.\ \lambda\ x : A.\ x$
- ordre supérieur : $\forall\ f : \mathrm{nat} \to \mathrm{nat},\ f\ 0 = 0$
- types dépendants :
  $\mathrm{append} : \forall n\ m : \mathrm{nat},\ \mathrm{list}\ n \to \mathrm{list}\ m \to \mathrm{list}\ (n + m)$
- inductifs : $\mathrm{nat} := 0 : \mathrm{nat} \mid S : \mathrm{nat} \to \mathrm{nat}$

Typage, prouvabilité :

- $E; G \vdash_{CCI} t : T$
- isomorphisme de Curry-Howard

$$\lambda\ A : \mathrm{Set}.\ \lambda\ x : A.\ x \quad : \quad \forall A : \mathrm{Set},\ A \to A$$

Calcul des Constructions Inductives

λ-calcul typé avec :

- polymorphisme : $\lambda\ A : \mathrm{Set}.\ \lambda\ x : A.\ x$
- ordre supérieur : $\forall\ f : \mathrm{nat} \to \mathrm{nat},\ f\ 0 = 0$
- types dépendants :
  $\mathrm{append} : \forall n\ m : \mathrm{nat},\ \mathrm{list}\ n \to \mathrm{list}\ m \to \mathrm{list}\ (n + m)$
- inductifs : $\mathrm{nat} := 0 : \mathrm{nat} \mid S : \mathrm{nat} \to \mathrm{nat}$

Typage, prouvabilité :

- $E; G \vdash_{CCI} t : T$
- isomorphisme de Curry-Howard

$$\lambda\ A : \mathrm{Set}.\ \lambda\ x : A.\ x \quad : \quad \forall A : \mathrm{Set},\ A \to A$$

Éléments traduits

- égalité
- arithmétique
- définitions

$$\forall x : s,\ f(x) = t$$

- définitions récursives et filtrage

$$plus : \text{nat} \longrightarrow \text{nat} \longrightarrow \text{nat}$$
$$\forall x : \text{nat},\ plus(0, x) = x$$
$$\forall x, y : \text{nat},\ plus(S(y), x) = S(plus(y, x))$$

- types inductifs

$$\forall x : \text{nat},\ 0 \neq S(x)$$
$$\forall x : \text{nat},\ x = 0 \ \vee\ \exists y : \text{nat},\ x = S(y)$$
$$\forall x, y : \text{nat},\ S(x) = S(y) \Rightarrow x = y$$

Éléments traduits

- égalité
- arithmétique
- définitions

$$\forall x : s,\ f(x) = t$$

- définitions récursives et filtrage

$$plus : \text{nat} \to \text{nat} \to \text{nat}$$
$$\forall x : \text{nat},\ plus(0, x) = x$$
$$\forall x, y : \text{nat},\ plus(S(y), x) = S(plus(y, x))$$

- types inductifs

$$\forall x : \text{nat},\ 0 \neq S(x)$$
$$\forall x : \text{nat},\ x = 0 \ \vee\ \exists y : \text{nat},\ x = S(y)$$
$$\forall x, y : \text{nat},\ S(x) = S(y) \Rightarrow x = y$$

Éléments traduits

- égalité
- arithmétique
- définitions

$$\forall x : s,\ f(x) = t$$

- définitions récursives et filtrage

$$\begin{aligned} &\mathit{plus} : \mathsf{nat} \to \mathsf{nat} \to \mathsf{nat} \\ &\forall x : \mathsf{nat},\ \mathit{plus}(0, x) = x \\ &\forall x, y : \mathsf{nat},\ \mathit{plus}(S(y), x) = S(\mathit{plus}(y, x)) \end{aligned}$$

- types inductifs

$$\begin{aligned} &\forall x : \mathsf{nat},\ 0 \neq S(x) \\ &\forall x : \mathsf{nat},\ x = 0 \ \vee\ \exists y : \mathsf{nat},\ x = S(y) \\ &\forall x, y : \mathsf{nat},\ S(x) = S(y) \Rightarrow x = y \end{aligned}$$

Éléments traduits

- égalité
- arithmétique
- définitions

$$\forall x : s,\ f(x) = t$$

- définitions récursives et filtrage

$$\begin{aligned} &\mathit{plus} : \mathsf{nat} \to \mathsf{nat} \to \mathsf{nat} \\ &\forall x : \mathsf{nat},\ \mathit{plus}(0, x) = x \\ &\forall x, y : \mathsf{nat},\ \mathit{plus}(S(y), x) = S(\mathit{plus}(y, x)) \end{aligned}$$

- types inductifs

$$\begin{aligned} &\forall x : \mathsf{nat},\ 0 \neq S(x) \\ &\forall x : \mathsf{nat},\ x = 0 \ \vee\ \exists y : \mathsf{nat},\ x = S(y) \\ &\forall x, y : \mathsf{nat},\ S(x) = S(y) \Rightarrow x = y \end{aligned}$$

Problèmes liés à la convertibilité

```
Parameter A : Set.                  | A : TYPE
Definition B : Set := A.            | B : TYPE
Parameter f : A -> A.               | f : A -> A

forall (b : B), f b = b             | forall (b:B), f(b) = b
```

Solution : utiliser la forme normale des types

Problèmes liés à la convertibilité

```
Parameter A : Set.                  | A : TYPE
Definition B : Set := A.            | B : TYPE
Parameter f : A -> A.               | f : A -> A
                                    |
forall (b : B), f b = b             | forall (b:B), f(b) = b
```

Solution : utiliser la forme normale des types

Problèmes liés à la convertibilité

```
Parameter A : Set.                 | A : TYPE
Definition B : Set := A.           | B : TYPE
Parameter f : A -> A.              | f : A -> A
                                   |
forall (b : B), f b = b            | forall (b:B), f(b) = b
```

Solution : utiliser la forme normale des types

Jugements de traduction

- $E; G \Rightarrow_{Env} S; \Gamma; \Delta$
- $E; G \vdash T \Rightarrow_{Sort} s$
- $E; G \vdash t \Rightarrow_{Term} t'$
- $E; G \vdash P \Rightarrow_{Prop} P'$

Notation : $[A]_{E;G}$ pour la traduction de A dans $E; G$ lorsque celle-ci existe.

Jugements de traduction (suite)

Exemple :

$$\frac{E; G \vdash T \Rightarrow_{Sort} s \qquad E; (x : T) :: G \vdash P \Rightarrow_{Prop} P'}{E; G \vdash \forall x : T, P \Rightarrow_{Prop} \forall x : s, P'} \text{ forall-trad}$$

Résultats

Théorème : Si P bien typée dans $E; G$,
alors $[P]_{E;G}$ est bien formée dans $[E; G]$.

Théorème : Si $[E; G] \models [P]_{E;G}$ alors il existe h tel que $E; G \vdash_{CCI} h : P$.

Preuve par récurrence sur $[E; G] \models [P]_{E;G}$.

Résultats

Théorème : Si P bien typée dans $E; G$,
alors $[P]_{E;G}$ est bien formée dans $[E; G]$.

Théorème : Si $[E; G] \models [P]_{E;G}$ alors il existe h tel que $E; G \vdash_{CCI} h : P$.

Preuve par récurrence sur $[E; G] \models [P]_{E;G}$.

Utilisation en pratique

Exemple :

```
Theorem exemple1 :
  forall f : nat -> nat,
  forall x : nat,
  f x = x ->
  f (f x) = x.
Proof.
  simplify.
Qed.
```

Nouvelle « tactique » admit ajoutée à Coq

```
Check exemple1_admitted.
exemple1_admitted
     : forall (f : nat -> nat) (x : nat),
       f x = x -> f (f x) = x
```

Utilisation en pratique

Exemple :

```
Theorem exemple1 :
  forall f : nat -> nat,
  forall x : nat,
  f x = x ->
  f (f x) = x.
Proof.
  simplify.
Qed.
```

Nouvelle « tactique » admit ajoutée à Coq

```
Check exemple1_admitted.
exemple1_admitted
     : forall (f : nat -> nat) (x : nat),
       f x = x -> f (f x) = x
```

Utilisation en pratique

Exemple :

```
Theorem exemple1 :
  forall f : nat -> nat,
  forall x : nat,
  f x = x ->
  f (f x) = x.
Proof.
  simplify.
Qed.
```

Nouvelle « tactique » admit ajoutée à Coq

```
Check exemple1_admitted.
exemple1_admitted
     : forall (f : nat -> nat) (x : nat),
       f x = x -> f (f x) = x
```

Utilisation en pratique

Exemple :

```
Theorem exemple1 :
  forall f : nat -> nat,
  forall x : nat,
  f x = x ->
  f (f x) = x.
Proof.
  simplify.
Qed.
```

Nouvelle « tactique » admit ajoutée à Coq

```
Check exemple1_admitted.
exemple1_admitted
     : forall (f : nat -> nat) (x : nat),
       f x = x -> f (f x) = x
```

Exemple

```
Inductive nat : Set := O : nat | S : nat -> nat.

Fixpoint plus (n m : nat) {struct n} : nat :=
  match n with
  | O => m
  | S n' => S (plus n' m)
end.

Goal forall n : nat, plus n O = n.
  induction n; zenon.
Qed.
```

```
"sort_ax_4" (%sort_nat O)
"sort_ax_5" (A. ((x1 "nat") (=> (%sort_nat x1) (%sort_nat (S x1)))))
"injection_S" (A. ((x1 "nat")
              (=> (%sort_nat x1)
              (A. ((y1 "nat")
              (=> (%sort_nat y1) (=> (= (S x1) (S y1)) (/\ (= x1 y1) True))))))
"sort_ax_6" (A. ((x1 "nat")
            (A. ((x2 "nat")
            (=> (%sort_nat x1) (=> (%sort_nat x2) (%sort_nat (plus x1 x2)))))))
"plus_O" (A. ((m "nat") (=> (%sort_nat m) (= (plus O m) m))))
"plus_S" (A. ((m "nat")
         (=> (%sort_nat m)
         (A. ((n' "nat")
         (=> (%sort_nat n') (= (plus (S n') m) (S (plus n' m))))))))))
"sort_ax_7" (%sort_nat n)
"IHn" (= (plus n O) n)
$goal (= (plus (S n) O) (S n))
```

Conclusion

- Théorie
  1. Traduction CCI $\Rightarrow$ premier ordre
  2. Correction (typabilité, prouvabilité)
- Pratique
  1. Implanté dans la version CVS de Coq
  2. Tactiques `simplify`, `cvcl` et `zenon`

Perspectives

- Études de cas
- Termes de preuves (CVC Lite et Zenon)