

# Extraction de programmes à partir de preuves classiques en Coq

Alexandre Miquel

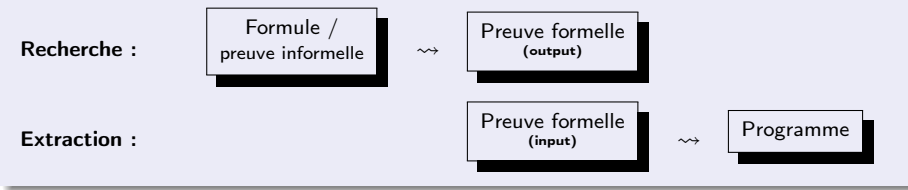
U. Paris 7 (PPS) & ENS Lyon (LIP/Plume)

23 avril 2009 – VERIMAG (Grenoble)

## La question existentielle (en logique)

- Quelle est la signification du quantificateur existentiel ?  
(dans une logique/théorie donnée)
  - 1 Qu'est ce qu'une preuve de  $(\exists x : D) P(x)$  ?
  - 2 Que peut on faire (i.e. calculer) avec ?
  - 3 Dans quelle mesure peut-on en extraire un **témoin** ?  
(en fait : extraire un **programme** qui calcule le témoin)

- Extraction de programme  $\neq$  Recherche (ou construction) de preuve



## Plan

- 1 Extraction de programmes en logique constructive
- 2 Traductions négatives et extraction classique
- 3 La réalisabilité classique
- 4 Le schéma d'extraction classique en Coq
- 5 Exemples

## Plan

- 1 Extraction de programmes en logique constructive
- 2 Traductions négatives et extraction classique
- 3 La réalisabilité classique
- 4 Le schéma d'extraction classique en Coq
- 5 Exemples

## Extraction constructive : le dictionnaire

- Sémantique de Brouwer-Heyting-Kolmogorov
- Correspondance de Curry-Howard

| Théorie de la démonstration                       | Programmation fonctionnelle                     |
|---|---|
| Proposition (formule)<br>Preuve (dérivation)      | Type de données<br>Programme (ou donnée)        |
| $A \wedge B, A \vee B, A \Rightarrow B$           | $A \times B, A + B, A \rightarrow B$            |
| Règle de déduction<br>Vérificateur de preuve      | Règle de typage<br>Vérificateur de type         |
| Élimination des coupures<br>Preuve sans coupure   | Évaluation (calcul)<br>Valeur                   |
| Preuve d'un lemme<br>Théorie (énoncés et preuves) | Sous-programme<br>Module (interface et implém.) |

## L'extraction de programme en logique constructive

- Basée sur la propriété du témoin en **logique intuitionniste** :

$$\begin{array}{c} \vdots p \\ \vdash \exists x A(x) \end{array} \rightsquigarrow \begin{array}{c} \vdots p' \\ t + \vdash A(t) \end{array}$$

- À travers Curry-Howard (intuitionniste) :

$$(\exists x : D) P(x) \equiv \Sigma x : D. P(x) \quad (\text{sigma-type})$$

$$p : \Sigma x : D. P(x) \rightsquigarrow (t, p') \quad (\text{avec } t : D \text{ et } p' : P(t))$$

- **Difficulté** : élimination du **code mort** (i.e. la justification  $p'$ )

- Extraction de programme :

$$p : (\forall x : A) (\exists y : B) P(x, y)$$

$$: \Pi x : A. \Sigma y : B. P(x, y) \rightsquigarrow f : A \rightarrow B$$

## Stratégies d'élimination du code mort

- **Dans HA** (MINLOG – Schwichtenberg *et al.*, Munich)

- ▶ Élimination des preuves des **formules de Harrop** (= code mort)

$$A ::= \perp \mid e = e \mid A \wedge A \mid A \Rightarrow A \mid \forall^{\sigma} x A \mid \exists^{\sigma} x A$$

$$H ::= \perp \mid e = e \mid H \wedge H \mid A \Rightarrow H \mid \forall^{\sigma} x H$$

(**Intuition** : formule de Harrop = formule à au plus une preuve)

- **Dans le calcul des constructions inductives** (Coq)

- ▶ Élimination de code mort basée sur la stratification :  
On garde ce qui est dans Set/Type, on jette ce qui est dans Prop.
- ▶ Nécessite de distinguer 3 quantificateurs existentiels !

- \*  $\exists x : T, P x$  (dans Prop, avec  $T : \text{Type}$  et  $P x : \text{Prop}$ )
- \*  $\{x : T \mid P x\}$  (dans **Type**, avec  $T : \text{Type}$  et  $P x : \text{Prop}$ )
- \*  $\{x : T \& P x\}$  (dans **Type**, avec  $T : \text{Type}$  et  $P x : \text{Type}$ )

## Recherche de preuve ou extraction : un exemple

- Pour tous  $x, y, z, n \in \mathbb{N}$ , soit :

$$F(x, y, z, n) = \begin{cases} 1 & \text{si } x > 0, y > 0, n > 2 \text{ et } x^n + y^n = z^n \\ 0 & \text{sinon} \end{cases}$$

**Théorème (Wiles, Taylor) :**  $(\forall x, y, z, n : \mathbb{N}) F(x, y, z, n) = 0$  ( $\Pi_1^0$ )

- **Recherche de preuve** : > 350 ans

- ▶ Preuve très complexe (utilise plusieurs branches des mathématiques)
- ▶ À quand le terme de preuve  $p$  en Coq ?

- **Extraction de programme** : aucun intérêt !

- ▶ Comportement calculatoire prédictible :  $p x y z n \succ^* \text{id}$
  - ▶ Calculatoirement, on peut remplacer  $p$  par  $p' \equiv \lambda x y z n. \text{id}$
- $\rightsquigarrow$  Point de vue de la **réalisabilité**

# Typage et réalisabilité

Pourquoi le terme  $\lambda x. x$  a pour type  $\text{nat} \rightarrow \text{nat}$  ?

| Typage ( $t : A$ )  | Réalisabilité ( $t \Vdash A$ )   |
|---|--|
| $\frac{x : \text{nat} \vdash x : \text{nat}}{\vdash \lambda x. x : \text{nat} \rightarrow \text{nat}}$  | pour tout $n \in \text{nat}$<br>$(\lambda x. x) n \succ n \in \text{nat}$  |
| <ul style="list-style-type: none"> <li>Analyse syntaxique des termes</li> <li>Au minimum semi-décidable</li> <li>Justification simple : dérivation</li> </ul> ↪ Recherche de preuve | <ul style="list-style-type: none"> <li>Analyse calculatoire des termes</li> <li>Fortement indécidable</li> <li>Justification externe (preuve)</li> </ul> ↪ Extraction de programme |

**Adéquation :** Correction pour le typage  $\Rightarrow$  Correction pour la réalisabilité

Historiquement, la réalisabilité a été introduite en **logique** (Kleene'45) comme une relation  $t \Vdash A$  entre des **programmes** et des **formules**

# Non constructivité de la logique classique

## Disjonction sans alternative :

- $A \vee \neg A$  ( $A \equiv$  conj. de Riemann, hyp. du continu, etc.)
- $e + \pi$  transcendant  $\vee e \times \pi$  transcendant

## Existentiel sans témoin :

- $(\exists x : \mathbb{N}) ((A \wedge x = 1) \vee (\neg A \wedge x = 0))$  (mêmes exemples pour  $A$ )

## Fonctions non calculables (oracles) :

- Fonction d'arrêt des machines de Turing :
 
$$(\forall x : \mathbb{N}) (\exists y : \mathbb{N}) ((\text{Halt}(x) \wedge y = 1) \vee (\neg \text{Halt}(x) \wedge y = 0))$$
- Principe du minimum ( $X \neq \emptyset$ ) :
 
$$(\forall f : X \rightarrow \mathbb{N}) (\exists x : X) (\forall y : X) f(x) \leq f(y)$$

# Plan

- Extraction de programmes en logique constructive
- Traductions négatives et extraction classique
- La réalisabilité classique
- Le schéma d'extraction classique en Coq
- Exemples

# R-traduction (historiquement : A-traduction)

- Problème :** Comment **rétracter** une théorie classique  $\mathcal{T}_c$  (PA, Z, ZF, ...) sur son (ou un) fragment intuitionniste  $\mathcal{T}_i \subset \mathcal{T}_c$ ? (HA, IZ, IZF, ...)

$\equiv$  Traduction  $A \mapsto A^*$  t.q.  $\mathcal{T}_c \vdash_{(LK)} A$  entraîne  $\mathcal{T}_i \vdash_{(LJ)} A^*$

- R-traduction** = généralisation de la  $\neg\neg$ -traduction où l'on remplace la négation ordinaire  $\neg A \equiv A \Rightarrow \perp$  par une **négation relative**

$$\overset{R}{\neg} A \equiv A \Rightarrow R$$

où  $R$  est un paramètre de la traduction (donc :  $\neg\neg$ -traduction = cas où  $R \equiv \perp$ )

Exemple de R-traduction au 1er ordre

$$\begin{aligned} \perp^* &\equiv R \\ p(\vec{t})^* &\equiv \overset{R}{\neg} \overset{R}{\neg} p(\vec{t}) & (A \Rightarrow B)^* &\equiv A^* \Rightarrow B^* \\ (A \vee B)^* &\equiv \overset{R}{\neg} (\overset{R}{\neg} A^* \wedge \overset{R}{\neg} B^*) & (A \wedge B)^* &\equiv A^* \wedge B^* \\ (\exists x A)^* &\equiv \overset{R}{\neg} \forall x \overset{R}{\neg} A^* & (\forall x A)^* &\equiv \forall x A^* \end{aligned}$$

## Exemples et contre-exemples

- Exemples de  $(\mathcal{T}_c, \mathcal{T}_i)$  pour lesquels une  $R$ -traduction existe :

- Les arithmétiques :  $(\text{PAN}, \text{HAN})$   $(1 \leq n \leq \omega)$
- Théories des ensembles :  $(\mathbb{Z}, \text{IZ}), (\mathbb{ZF}, \text{IZF}_C)$  [Friedman]
- PTS logiques non dépendants [Coquand-Herbelin]

- Contre-exemple :  $(\text{MLTT} + \text{tiers-exclu}, \text{MLTT})$   
où MLTT = Théorie des types de Martin-Löf (et ses variantes)

- Pour chacun des exemples où une  $R$ -traduction existe :

- $\mathcal{T}_c$  et  $\mathcal{T}_i$  sont **équiconsistantes** (cas :  $R \equiv \perp$ )
- $\mathcal{T}_c$  est **conservative** sur  $\mathcal{T}_i$  pour les formules  $\Sigma_1^0, \Pi_2^0$  (astuce de Friedman)
- $\mathcal{T}_c$  et  $\mathcal{T}_i$  sont **équi-1-consistantes**

## Logique classique et continuations

- La découverte fondamentale** [Felleisen & Griffin 90]

- L'opérateur de contrôle **call/cc** [Felleisen] a pour type :

$$((A \rightarrow B) \rightarrow A) \rightarrow A$$

- Côté logique [Griffin 90], il s'agit de la **loi de Peirce...**  
... qui entraîne (constructivement) le tiers-exclu
- Logique classique = programmation par **continuations**  
= méthode **essai/erreur**
- Relié aux **traductions négatives** par les **transformations CPS**

- Exemple** : comment prouver

$$A \vee (A \Rightarrow \perp) ?$$

**Réponse** : `call-cc  $\lambda k$ . Right ( $\lambda x$ .  $k$  (Left( $x$ )))`

## $R$ -traduction et extraction de programmes

- $R$ -traduction + Curry-Howard (ou réalis.) intuitionniste  
 $\rightsquigarrow$  **extraction de programme** pour les formules  $\Sigma_1^0$  et  $\Pi_2^0$

Transformation de preuve d'une formule  $\Sigma_1^0$  (de  $\mathcal{T}_c$  dans  $\mathcal{T}_i$ )

- $\pi$  :  $\exists x f(x) = 0$  (dans  $\mathcal{T}_c$ )
- $\pi^*$  :  $\overset{R}{\neg} \forall x \overset{R}{\neg} f(x) = 0$  (dans  $\mathcal{T}_i$ ,  $\overset{R}{\neg} A \equiv A \Rightarrow R$ )  
:  $\forall x (f(x) = 0 \Rightarrow R) \Rightarrow R$
- Astuce de Friedman :  $R \equiv \exists y f(y) = 0$
- $\pi^*$  :  $\forall x (f(x) = 0 \Rightarrow \underbrace{\exists y f(y) = 0}_{\exists\text{-intro}}) \Rightarrow \exists y f(y) = 0$
- $\pi^*(\exists\text{-intro})$  :  $\exists y f(y) = 0$  (dans  $\mathcal{T}_i$ )


- Berardi, Berger, Coquand, Kohlenbach, Schwichtenberg

## Plan

- Extraction de programmes en logique constructive
- Traductions négatives et extraction classique
- La réalisabilité classique
- Le schéma d'extraction classique en Coq
- Exemples

## La réalisabilité classique (Krivine)

- **Réalisabilité classique** = réalisabilité à la Kleene à travers une  $R$ -traduction<sup>1</sup>, le tout reformulé en style direct :  

$$\text{Krivine} = (\text{CPS})^{-1} \circ \text{Kleene} \circ (R\text{-traduction})$$
- Un langage de réalisateurs :  $\lambda_c = \lambda + \text{call-cc} + \dots$  (extensible)  
 + évaluation en **appel par nom** (exit les problèmes de confluence !)
- La réalisabilité classique est définie :
  - ▶ dans PA2 + DC (DC = choix dépendant)
  - ▶ dans ZF (+ DC)
  - ▶ dans  $\text{CC}^\omega$  ( $\text{CC}^\omega$  = calcul des constructions avec univers) [CSL'07]
  - ▶ dans pCIC  (pCIC = CIC – Set imprédictif)

<sup>1</sup>R-traduction de Lafont-Reus-Streicher-Oliva

## Principes de la réalisabilité classique

- **Intuitions :**
  - ▶ terme = “**preuve**” / pile = “**contre-preuve**”
  - ▶ processus = “**contradiction**” (ne jamais faire confiance à un réalisateur !)
- À chaque formule  $A$  on associe :
  - ▶ Un ensemble de piles  $\|A\|$  (valeur de fausseté)
  - ▶ Un ensemble de termes  $|A|$  (valeur de vérité)
- **Exemple :**  $\|A \Rightarrow B\| = |A| \cdot \|B\| = \{t \cdot \pi : t \in |A|, \pi \in \|B\|\}$
- Définition de la valeur de vérité  $|A|$  par **orthogonalité** :  

$$|A| = A^\perp = \{t \in \Lambda : \forall \pi \in \|A\| \ t \star \pi \in \perp\}$$
 où  $\perp$  est un ensemble de processus, paramètre de la construction ( $\perp \approx R$ )

### Théorème d'adéquation (dans PA2, ZF, pCIC, ...)

À toute preuve classique de  $A$  on sait associer un réalisateur  $t \Vdash A$  ( $\equiv t \in |A|$ )

## Le $\lambda_c$ -calcul (Krivine)

### Syntaxe

|                  |  |
|------------------|--|
| <b>Termes</b>    | $t, u ::= x \mid \lambda x. t \mid \underbrace{tu \mid \mathfrak{c} \mid \dots}_{\text{quasi-preuves}} \mid k_\pi$ |
| <b>Piles</b>     | $\pi ::= \alpha \mid u \cdot \pi \quad (u, \pi \text{ clos})$  |
| <b>Processus</b> | $p, q ::= t \star \pi \quad (t, \pi \text{ clos})$   |

$k_\pi$  = (constante de) continuation associée à  $\pi$ ,  $\alpha$  = (constante de) fond de pile

### Règles d'évaluation : $p \succ p'$

|                |  |
|----------------|--|
| <b>Push</b>    | $tu \star \pi \succ t \star u \cdot \pi$                       |
| <b>Grab</b>    | $\lambda x. t \star u \cdot \pi \succ t\{x := u\} \star \pi$   |
| <b>Call/cc</b> | $\mathfrak{c} \star t \cdot \pi \succ t \star k_\pi \cdot \pi$ |
| <b>Resume</b>  | $k_\pi \star t \cdot \pi' \succ t \star \pi$                   |
|                | $\dots \quad \dots$  |

## Structure du quantificateur existentiel

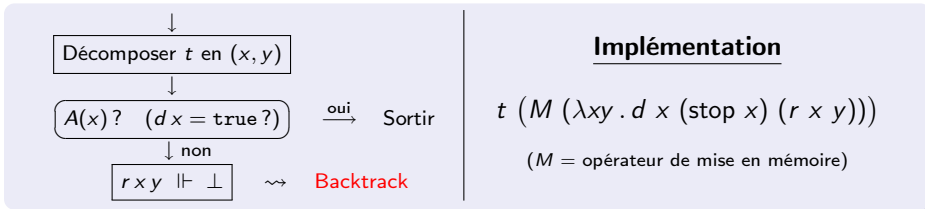
En réalisabilité intuitionniste et classique, un réalisateur de  $\exists^{\mathbb{N}} x A(x)$  contient un couple **témoin/justification**  $(n, t)$ , avec  $n \in \mathbb{N}$  et  $t \Vdash A(n)$

- En réalisabilité intuitionniste
  - ▶ On a la garantie que  $n$  vérifie  $A(n)$
  - ▶ La justification  $t \Vdash A(n)$  est donc superflue (code mort)
- En réalisabilité classique
  - ▶ Aucune garantie que  $n$  vérifie  $A(n)$  ! (possibilité de **faux-témoin**)
  - ▶ La justification  $t \Vdash A(n)$  n'offre aucune garantie... (**parapreuve**)  
 ... mais contient le matériel nécessaire pour backtrack
- Comment obtenir des témoins fiables ?

## Recherche de témoin par essai/erreur

- Les ingrédients :

- ▶ Une preuve (classique)  $t : (\exists x : \mathbb{N}) A(x)$
- ▶ Une fonction de décision  $d : \mathbb{N} \rightarrow \mathbb{B}$  du prédicat  $A$
- ▶ Une fonction de réfutation  $r : r\ n \Vdash \neg A(n)$  dès que  $\neg A(n)$



### Proposition

- 1 Cette procédure converge en temps fini [Krivine, Raffalli, Guillermo]
- 2 À une CPS près : même algorithme que Friedman [TLCA'09]

## Plan

- 1 Extraction de programmes en logique constructive
- 2 Traductions négatives et extraction classique
- 3 La réalisabilité classique
- 4 Le schéma d'extraction classique en Coq
- 5 Exemples

## Extraction sur le PTS sous-jacent

- Une projection  $M \mapsto M^*$  des termes de pICC sur le  $\lambda$ -calcul, avec effondrement des types sur une instruction inerte :

$$\begin{aligned}
 x^* &\equiv x \\
 (\text{fun } x : T \Rightarrow M)^* &\equiv \lambda x . M^* \\
 (MN)^* &\equiv M^* N^* \\
 (\text{forall } x : T, U)^* &\equiv .\text{type} \\
 \text{Prop}^* &\equiv .\text{type} \\
 \text{Set}^* &\equiv .\text{type} \\
 \text{Type}^* &\equiv .\text{type}
 \end{aligned}$$

En fait on pose  $M^* \equiv .\text{type}$  dès que  $M$  est un type (ou un prédicat).

- Remarque** : pas de distinction suivant la sorte du terme source  $M : T : \text{Prop/Set/Type}$  (contrairement à l'extraction constructive)

## Autres constructions

- Par défaut, les structures de données (co)inductives sont traduites par les codages habituels en  $\lambda$ -calcul
- Le filtrage est traduit à l'aide d'un **combinateur de filtrage**  $!/\text{case}$  associé à chaque type (co)inductif  $I$  :

$!/\text{case}$  (terme à déstructurer) (branches de filtrage...)

- Les (co)points-fixes sont traduits en introduisant des combinateurs de point fixe  $.\text{fix}_{n_i}$  ( $1 \leq i \leq n$ ) :

$$\begin{aligned}
 .\text{fix}_{n_i} F_1 \dots F_n &:= \\
 &F_i (. \text{fix}_{n_1} F_1 \dots F_n) \dots (. \text{fix}_{n_n} F_1 \dots F_n)
 \end{aligned}$$

- Par défaut, les axiomes sont traduits par des constantes inertes

## Exemple

- Les entiers naturels (codage par défaut) :

```
Coq.Init.Datatypes.O      := λe1e2. e1
Coq.Init.Datatypes.S      := λxe1e2. e2 x
Coq.Init.Datatypes.nat%case := λz.z
```

- ▶ Représentation (encore) plus verbeuse que les entiers de Church
- ▶ Intuition : même codage que le type option

`nat = option nat, 0 = None, S = Some`

- ▶ Prédécesseur en temps constant ( $\neq$  entiers de Church)
- ▶ Récursion à coup de `.fix_n_i` (pas possible autrement!)

## Le mécanisme d'override

- À chaque extraction de constante, l'extracteur consulte un fichier de définitions "override.lco" pour voir si cette constante y est définie.
  - ▶ Si oui, la définition correspondante est placée dans le code extrait
  - ▶ Sinon, l'extracteur produit une définition suivant le schéma par défaut
- En pratique le mécanisme d'override sert à :
  - ▶ Changer la représentation de certains inductifs (nat)
  - ▶ Réaliser certains axiomes (par défaut réalisés par des constantes inertes)  
↪ réalisation des axiomes de **logique classique**
  - ▶ Attacher à certains objets de la librairie des **réalisateurs optimisés** (par rapport à ceux produits suivant le schéma par défaut)

## Utilisation des entiers machine

- Redéfinition des entiers naturels dans "override.lc" :

```
nat n k := k n
Coq.Init.Datatypes.O := nat 0
Coq.Init.Datatypes.S n := n (λn.int_succ n nat)
Coq.Init.Datatypes.nat%case z e0 e1 :=
  z (λn.int_null n e0 (int_pred n (λp.e1 (nat p))))
```

- Dans le langage cible ( $\lambda_c$  étendu avec entiers primitifs) :

- ▶ `42 = entier-donnée = instruction sans règle d'évaluation`
- ▶ `nat 42 (=β λk.k 42) = entier-programme = entier paresseux`
- ▶ Primitives sur les entiers-données :

```
int_succ n k > k (n + 1)
int_pred n k > k (n - 1)
int_null n k0 k1 > k0|1 (suivant que n est nul ou non)
```

## Autres overrides possibles

- Le changement de représentation des naturels induit un changement de **complexité spatiale**, pas de **complexité temporelle**...  
... car les fonctions écrites en Coq continuent à compter sur les doigts
- Extension de l'override aux fonctions usuelles :

```
Coq.Init.Datatypes.plus n m :=
  n (λn.m (λm.int_plus n m nat))
Coq.Init.Datatypes.mult n m :=
  n (λn.m (λm.int_mult n m nat))
Coq.Init.Datatypes.pred n :=
  n (λn.int_null n (nat 0) (int_pred n nat))
```
- Redéfinition de certains réalisateurs extraits de la librairie standard :  
par ex. la loi de Peirce (prouvée en Coq à l'aide du tiers-exclu)
- Redéfinition des ordres  $\leq, <, \geq, >$  et de leurs propriétés

## Exemple : commutativité de $+_{\text{nat}}$ (réalisateur extrait)

```
Coq.Init.Datatypes.nat_rect =
  \P\F\X0 .fix_1_1 (\F\N Coq.Init.Datatypes.nat%case n f (\n f0 n (F n))

Coq.Init.Datatypes.nat_ind =
  \P Coq.Init.Datatypes.nat_rect P

Coq.Init.Peano.plus_n_0 =
  \n
  Coq.Init.Datatypes.nat_ind
  .type (Coq.Init.Logic.refl_equal .type (nat 0))
  (\n\IHn
  Coq.Init.Logic.f_equal
  .type .type Coq.Init.Datatypes.S n (Coq.Init.Peano.plus n (nat 0))
  IHn) n

Coq.Init.Peano.plus_n_Sm =
  \n\m
  Coq.Init.Datatypes.nat_ind
  .type (Coq.Init.Logic.refl_equal .type (Coq.Init.Datatypes.S m))
  (\n\IHn
  Coq.Init.Logic.f_equal
  .type .type Coq.Init.Datatypes.S
  (Coq.Init.Datatypes.S (Coq.Init.Peano.plus n m))
  (Coq.Init.Peano.plus n (Coq.Init.Datatypes.S m)) IHn) n

Coq.Arith.Plus.plus_comm =
  \n\m
  Coq.Init.Datatypes.nat_ind
  .type (Coq.Init.Peano.plus_n_0 m)
  (\y\H
  Coq.Init.Logic.eq_ind
  .type (Coq.Init.Datatypes.S (Coq.Init.Peano.plus m y)) .type
  (Coq.Init.Logic.f_equal
  .type .type Coq.Init.Datatypes.S (Coq.Init.Peano.plus y m)
  (Coq.Init.Peano.plus m y) H)
  (Coq.Init.Peano.plus m (Coq.Init.Datatypes.S y))
  (Coq.Init.Peano.plus_n_Sm m y)) n
```

## Exemple : commutativité de $+_{\text{nat}}$ (réalisateur optimisé)

```
Coq.Arith.Plus.plus_comm =
  \n\m\z z
```

## Les commandes d'extraction classique

L'extracteur produit incrémentalement une **liste de  $\lambda_c$ -définitions**, de la forme  $instruction\ ident_1 \dots ident_n = \lambda_c-term$

- Classical Extraction Reset.
- Classical Extraction Save "file.lco".
- Classical Extract  $qualid_1 \dots qualid_n$ .
- Classical Extract Witness  $ex\_proof$  using  $dec\_proof$ .

Extrait une fonction de calcul de témoins existentiels à partir d'une preuve d'existence et d'une preuve de décidabilité

$$ex\_proof : \text{forall } \vec{x} : \vec{T}, \text{exists } y : U, P \vec{x} y$$
$$dec\_proof : \text{forall } \vec{x} : \vec{T}, \text{forall } y : U, \{P \vec{x} y\} + \{\sim P \vec{x} y\}$$

sous la forme d'une instruction `ex_proof%witness`

## Plan

- 1 Extraction de programmes en logique constructive
- 2 Traductions négatives et extraction classique
- 3 La réalisabilité classique
- 4 Le schéma d'extraction classique en Coq
- 5 Exemples

### Principe du minimum

Si  $X$  est un type de données habité :

$$(\forall f : X \rightarrow \mathbb{N}) (\exists x : X) (\forall y : X) f(x) \leq f(y)$$

**Preuve.** Par l'absurde, avec une récurrence forte.

- **Remarque :** Pas de preuve intuitionniste (oracle)

### Corollaire

$$(\forall f : \mathbb{N} \rightarrow \mathbb{N}) (\exists x : \mathbb{N}) f(x) \leq f(2x + 1)$$

Plus généralement :  $(\forall f, g : \mathbb{N} \rightarrow \mathbb{N}) (\exists x : \mathbb{N}) f(x) \leq f(g(x))$

**Preuve.** On prend le point  $x$  donné par le principe du minimum.

- **Remarque :** Il y a aussi une preuve intuitionniste (car  $\Pi_2^0$ )

Démo