

# Preuve de programmes fonctionnels

David Delahaye

Faculté des Sciences  
[David.Delahaye@lirmm.fr](mailto:David.Delahaye@lirmm.fr)

Master M1 2017-2018

# Preuves formelles

## Plusieurs pré-requis

- Bien connaître la sémantique de son langage ;
- Être capable d'exprimer cette sémantique formellement ;
- Savoir spécifier le comportement de son programme ;
- Faire en sorte que la spécification soit totale.

## Plusieurs langages en jeu

- Le langage de programmation ;
- Le langage de spécification ;
- Le langage de preuve.

Si ces trois langages sont réunis au sein du même environnement, c'est plus pratique pour le développeur !

# Spécification

## Qu'est-ce que c'est ?

- C'est le « quoi » du programme, ce qu'il doit faire ;
- Peut-être exprimé dans le langage naturel (mais ambigu) :
  - ▶ Exemple : « ce programme calcule la racine carrée ».
- Plus formellement : spécification = type d'un programme.

## Plusieurs degrés de spécifications

- Spécifications partielles :
  - ▶ Exemple : `sqrt : float → float` ;
  - ▶ Donne de l'information mais pas assez ;
  - ▶ Beaucoup de fonctions ont ce type (pas que racine carrée).
- Spécifications totales :
  - ▶ Exemple :  $\forall x \in \mathbb{R}^+. f(x) \times f(x) = x$  ;
  - ▶ Seule racine carrée vérifie cette proposition ;
  - ▶ Nécessite un langage basé sur la logique.

# Preuves

## Objectifs

- Mettre en adéquation un programme et sa spécification ;
- Apporter une garantie sur l'exécution du programme.

## Remarques

- Plus simple sur des programmes fonctionnels ;
- Fonctionnel aussi utilisé pour encoder les preuves ;
- Outils basé sur du fonctionnel : Coq, HOL, PVS, etc. ;
- Outils basé sur de l'impératif : Atelier B.

## Peut-on automatiser ce processus ?

- Pas totalement (problème semi-décidable) ;
- Certains fragments sont décidables :
  - ▶ Propositionnel, arithmétique, réels, géométrie, etc.

# Une preuve triviale

## Spécification

- On cherche à écrire une fonction  $f$  telle que :  
$$\forall x \in \mathbb{N}. f(x) = x \times x$$

## Programme

- On considère le programme (fonction) suivant :  
$$g(x) = x \times x$$

## Preuve d'adéquation

- On doit démontrer que  $g$  vérifie la spécification :  
$$\forall x \in \mathbb{N}. g(x) = x \times x$$
- On « déplie » la définition de  $g$  :  
$$\forall x \in \mathbb{N}. x \times x = x \times x$$
- Ce qui est trivial.

# Une preuve plus difficile

## Spécification

- La même que précédemment, c'est-à-dire :

$$\forall x \in \mathbb{N}. f(x) = x \times x$$

## Programme

- On considère le programme (fonction) suivant :

$$h(x, i) = \begin{cases} x, & \text{si } i = 0, 1 \\ x + h(x, i - 1), & \text{sinon} \end{cases}$$
$$g(x) = h(x, x)$$

## Preuve ?

- Par induction... certes, mais laquelle ?
- Et il faut un bon support pour l'induction dans l'outil !

# Spécifications inductives et preuves par induction

## L'induction à la base de la formalisation

- On peut tout formaliser à base de types inductifs ;
- Types inductifs pour les types de données ;
- Relations inductives pour spécifier des comportements ;
- Fonctions récursives pour les programmes ;
- Preuves par induction pour l'adéquation prog./spéc. ;
- Moyen idiomatique de formalisation de beaucoup d'outils.

## Support pour l'induction

- Générer les schémas d'induction automatiquement ;
- Pouvoir en générer de nouveaux au besoin ;
- Gérer les lemmes d'inversion automatiquement.

# Spécifications inductives et preuves par induction

## Systèmes formels et outils

- Induction présente en théorie des ensembles ;
- Théories des types dédiées :
  - ▶ Système T de Gödel, théorie des types de Martin-Löf, calcul des constructions inductives de Coquand-Huet-Paulin.
- Outils dédiés : Coq, Lego, Alfa, etc.

## Historiquement

- Formulation explicite de l'induction au 17ème siècle ;
- Auparavant : utilisation de l'induction mathématique ;
- Pascal : « Traité du triangle arithmétique » ;
- Fermat : descente infinie.



# Preuve du théorème de Fermat pour $n = 4$

## Théorème

Il n'existe pas d'entiers non nuls  $x$ ,  $y$ , et  $z$ , tels que :

$$x^4 + y^4 = z^4$$

Le théorème se déduit aisément de la preuve du 20ème problème de Diophante : est-ce qu'un triangle rectangle dont les côtés sont mesurés par des entiers peut avoir une surface mesurée par un carré ?

Fermat a résolu la question par la négative et il a démontré qu'il n'existe pas d'entiers naturels non nuls tels que :

$$x^2 + y^2 = z^2 \text{ et } xy = 2t^2$$

# Preuve du théorème de Fermat pour $n = 4$

## Principe de la descente infinie

- Preuve par l'absurde : démontrer que résoudre le problème au rang  $n$  revient à le résoudre au rang  $n - 1$ , ce qui n'est pas possible car on est borné par 0 ;
- La descente infinie résout des propositions  $\nexists x.P(x)$  ;
- Mais équivalent au principe habituel (contraposée).

## Preuve de Fermat chargée d'histoire

- Première utilisation de l'induction ;
- Résolution d'un problème de Diophante (250 apr. J.-C.) ;
- Utilisation des triplets pythagoriciens (Euclide, 300 av. J.-C. ; Babyloniens, 1900-1600 av. J.-C.).

Voir preuve en Coq de Delahaye-Mayero

# Un premier petit exemple

## Spécification

On définit la relation inductive  $is\_sum$  de type  $\mathbb{N} \times \mathbb{N} \rightarrow \text{Prop}$  de la façon suivante :

- ❶ On a :  $is\_sum(0, 0)$  ;
- ❷ Pour  $n, s \in \mathbb{N}$ , si  $is\_sum(n, s)$ , alors on a :  $is\_sum(S(n), s + S(n))$ .

## Fonction

On définit la fonction suivante de type  $\mathbb{N} \rightarrow \mathbb{N}$  :

$$f_{is\_sum}(n) = \begin{cases} 0, & \text{si } n = 0 \\ f_{is\_sum}(p) + S(p), & \text{si } n = S(p), \text{ avec } p \in \mathbb{N} \end{cases}$$

# Un premier petit exemple

## Théorème de correction

L'adéquation entre la fonction et sa spécification se vérifie avec le théorème suivant :

$$\forall n, s \in \mathbb{N}. f_{is\_sum}(n) = s \Rightarrow is\_sum(n, s)$$

## Preuve

La preuve se fait par induction sur  $n$ .

On utilise le schéma d'induction structurelle sur  $\mathbb{N}$  :

$$\forall P \in \mathbb{N} \rightarrow \text{Prop}. P(0) \Rightarrow (\forall n \in \mathbb{N}. P(n) \Rightarrow P(S(n))) \Rightarrow \forall n \in \mathbb{N}. P(n)$$

Dans notre cas :

$$P(n) = \forall s \in \mathbb{N}. f_{is\_sum}(n) = s \Rightarrow is\_sum(n, s)$$

# Un premier petit exemple

## Preuve

On applique le schéma d'induction et on doit démontrer :

❶ Cas de base :

$$\forall x \in \mathbb{N}. f_{is\_sum}(0) = s \Rightarrow is\_sum(0, s)$$

On calcule  $f_{is\_sum}(0)$ , ce qui donne :

$$\forall x \in \mathbb{N}. 0 = s \Rightarrow is\_sum(0, s)$$

On remplace  $s$  par 0, et on doit démontrer  $is\_sum(0, 0)$ , qui est le cas de base de la spécification inductive de la relation  $is\_sum$ .

# Un premier petit exemple

## Preuve

On applique le schéma d'induction et on doit démontrer :

② Cas inductif : pour  $n \in \mathbb{N}$ ,

$$\forall s \in \mathbb{N}. f_{is\_sum}(S(n)) = s \Rightarrow is\_sum(S(n), s)$$

sous l'hypothèse d'induction :

$$\forall s \in \mathbb{N}. f_{is\_sum}(n) = s \Rightarrow is\_sum(n, s)$$

On calcule  $f_{is\_sum}(S(n))$ , ce qui donne :

$$\forall s \in \mathbb{N}. f_{is\_sum}(n) + S(n) = s \Rightarrow is\_sum(S(n), s)$$

On remplace  $s$  par  $f_{is\_sum}(n) + S(n)$ , et on doit démontrer :

$$is\_sum(S(n), f_{is\_sum}(n) + S(n))$$

# Un premier petit exemple

## Preuve

On applique le schéma d'induction et on doit démontrer :

② Cas inductif :

On applique le cas inductif de la spécification de *is\_sum*, et on doit démontrer :

$$is\_sum(n, f_{is\_sum}(n))$$

On applique l'hypothèse d'induction avec  $s = f_{is\_sum}(n)$ , et il nous reste à démontrer que  $f_{is\_sum}(n) = f_{is\_sum}(n)$ , ce qui est trivial.

## Un deuxième exemple plus complexe ou quand l'induction structurelle ne suffit plus

### Spécification

On définit la relation inductive *is\_even* de type  $\mathbb{N} \rightarrow \text{Prop}$  de la façon suivante :

- 1 On a : *is\_even*(0) ;
- 2 Pour  $n \in \mathbb{N}$ , si *is\_even*( $n$ ), alors on a : *is\_even*( $S(S(n))$ ).

### Fonction

On définit la fonction suivante de type  $\mathbb{N} \rightarrow \mathbb{B}$  :

$$f_{is\_even}(n) = \begin{cases} \top, & \text{si } n = 0 \\ \perp, & \text{si } n = 1 \\ f_{is\_even}(p), & \text{si } n = S(S(p)), \text{ avec } p \in \mathbb{N} \end{cases}$$



## Un deuxième exemple plus complexe ou quand l'induction structurelle ne suffit plus

### Théorème de correction

L'adéquation entre la fonction et sa spécification se vérifie avec le théorème suivant :

$$\forall n \in \mathbb{N}. f_{is\_even}(n) = \top \Rightarrow is\_even(n)$$

### Preuve

Par induction structurelle sur  $n$  :

❶ Cas de base :

$$f_{is\_even}(0) = \top \Rightarrow is\_even(0)$$

On applique simplement le cas de base de  $is\_even$ .

## Un deuxième exemple plus complexe ou quand l'induction structurelle ne suffit plus

### Preuve

Par induction structurelle sur  $n$  :

② Cas inductif : pour  $n \in \mathbb{N}$ ,

$$f_{is\_even}(S(n)) = \top \Rightarrow is\_even(S(n))$$

sous l'hypothèse d'induction :

$$f_{is\_even}(n) = \top \Rightarrow is\_even(n)$$

## Un deuxième exemple plus complexe ou quand l'induction structurelle ne suffit plus

### Preuve

Par induction structurelle sur  $n$  :

#### ② Cas inductif :

On doit refaire une deuxième induction sur  $n$  :

#### ① Cas de base :

$$f_{is\_even}(1) = \top \Rightarrow is\_even(1)$$

On calcule  $f_{is\_even}(1)$ , ce qui donne :

$$\perp = \top \Rightarrow is\_even(1)$$

ce qui est trivial car  $\perp = \top$  est faux.

# Un deuxième exemple plus complexe ou quand l'induction structurelle ne suffit plus

## Preuve

Par induction structurelle sur  $n$  :

### ② Cas inductif :

On doit refaire une deuxième induction sur  $n$  :

### ② Cas inductif :

$$f_{is\_even}(S(S(n))) = \top \Rightarrow is\_even(S(S(n)))$$

sous les hypothèses :

$$f_{is\_even}(S(n)) = \top \Rightarrow is\_even(S(n))$$

$$(f_{is\_even}(n) = \top \Rightarrow is\_even(n)) \Rightarrow \\ f_{is\_even}(S(n)) = \top \Rightarrow is\_even(S(n))$$

# Un deuxième exemple plus complexe ou quand l'induction structurelle ne suffit plus

## Preuve

Par induction structurelle sur  $n$  :

### ② Cas inductif :

On doit refaire une deuxième induction sur  $n$  :

### ② Cas inductif :

On calcule  $f_{is\_even}(S(S(n)))$  et on applique le cas inductif de la relation  $is\_even$ , et on doit démontrer  $is\_even(n)$  sous les hypothèses :

$$f_{is\_even}(n) = \top$$

$$f_{is\_even}(S(n)) = \top \Rightarrow is\_even(S(n))$$

$$(f_{is\_even}(n) = \top \Rightarrow is\_even(n)) \Rightarrow \\ f_{is\_even}(S(n)) = \top \Rightarrow is\_even(S(n))$$

## Un deuxième exemple plus complexe ou quand on a besoin d'induction fonctionnelle

### Induction fonctionnelle

- Nouveau schéma d'induction qui « suit » la fonction ;
- Le schéma sera propre à la fonction ;
- N'introduit pas un axiome (démontrable).

### Dans le cas de $f_{is\_even}$

$$\forall P \in \mathbb{N} \times \mathbb{B} \rightarrow Prop.$$

$$P(0, \top) \Rightarrow P(1, \perp) \Rightarrow$$

$$(\forall p \in \mathbb{N}. P(p, f_{is\_even}(p)) \Rightarrow P(S(S(p)), f_{is\_even}(p))) \Rightarrow$$

$$\forall n \in \mathbb{N}. P(n, f_{is\_even}(n))$$

## Un deuxième exemple plus complexe ou quand on a besoin d'induction fonctionnelle

### Preuve

$$\forall n \in \mathbb{N}. f_{is\_even}(n) = \top \Rightarrow is\_even(n)$$

Ici, le prédicat  $P$  du schéma d'induction est :

$$P(n, b) = b = \top \Rightarrow is\_even(n)$$

❶ Cas de base (1) :

$$\top = \top \Rightarrow is\_even(0)$$

On applique le cas de base de la relation  $is\_even$ .

## Un deuxième exemple plus complexe ou quand on a besoin d'induction fonctionnelle

### Preuve

- ② Case de base (2) :

$$\perp = \top \Rightarrow is\_even(1)$$

ce qui est trivial car  $\perp = \top$  est faux.

- ③ Cas inductif : pour  $p \in \mathbb{N}$ ,

$$f_{is\_even}(p) = \top \Rightarrow is\_even(S(S(p)))$$

sous l'hypothèse d'induction :

$$f_{is\_even}(p) = \top \Rightarrow is\_even(p)$$



## Un deuxième exemple plus complexe ou quand on a besoin d'induction fonctionnelle

### Preuve

- ③ Cas inductif : pour  $p \in \mathbb{N}$ ,  
On suppose  $f_{is\_even}(p) = \top$ , puis on applique le cas inductif de la relation  $is\_even$ , et on doit démontrer :

$$is\_even(p)$$

sous les hypothèses :

$$f_{is\_even}(p) = \top$$

$$f_{is\_even}(p) = \top \Rightarrow is\_even(p)$$

ce qui se démontre en appliquant l'hypothèse d'induction à l'hypothèse introduite précédemment.

# Induction fonctionnelle en Coq

## Relation *is\_even*

```
Require Import FunInd.
```

```
Inductive is_even : nat → Prop :=  
| is_even_O : is_even 0  
| is_even_S : forall n : nat, is_even n → is_even (S (S n)).
```

```
Fixpoint even (n : nat) : Prop :=  
  match n with  
  | 0 ⇒ True  
  | 1 ⇒ False  
  | (S (S n)) ⇒ even n  
  end.
```

# Induction fonctionnelle en Coq

## Relation *is\_even*

Functional **Scheme** even\_ind := Induction for even Sort **Prop**.

**Theorem** even\_sound :

**forall** (n : nat) (v : **Prop**), (even n) = True → is\_even n.

**Proof**.

do 2 **intro**.

**functional induction** (even n) **using** even\_ind; **intros**.

**apply** is\_even\_O.

**elimtype** False; **rewrite** H; **auto**.

**apply** is\_even\_S; **apply** IHP; **assumption**.

**Qed**.

## Factorielle

- Spécifier (inductivement) cette relation ;
- Écrire la fonction factorielle ;
- Générer le schéma d'induction fonctionnelle ;
- Démontrer la correction de la fonction.

# Spécifications inductives non calculatoires

## Spécifications quelconques

- Les spécifications sont parfois plus abstraites ;
- Elles ne contiennent pas forcément un algorithme ;
- C'est même mieux si elles n'en contiennent pas ;
- Si elles contiennent un algorithme, elles n'en imposent pas forcément un au niveau de l'implantation (il faudra en démontrer l'équivalence).

## Exemple

- Le pgcd  $d$  de deux entiers relatifs  $a$  et  $b$  peut être spécifié par :
  - ▶  $d$  divise  $a$  et  $b$ , et il existe deux entiers relatifs  $x$  et  $y$  tels que  $ax + by = d$  (théorème de Bachet-Bézout) ;
- La spécification précédente n'offre aucun schéma de calcul ;
- Il existe plusieurs algorithmes (algorithme d'Euclide, méthode des soustractions, etc.).

## Tri d'une liste d'entiers

- Spécifier la relation « être une permutation de » pour deux listes ;
- Démontrer que la liste  $[1; 2; 3]$  est une permutation de  $[3; 2; 1]$  ;
- Spécifier la relation « être triée » pour une liste ;
- Démontrer que la liste  $[1; 2; 3]$  est triée ;
- Écrire la fonction de tri par insertion (deux fonctions à écrire) ;
- Démontrer que la fonction est correcte.

# Exercice

## Relation « être une permutation de »

- ❶ Pour  $a \in \mathbb{N}$  et  $l_0, l_1$  deux listes, si  $is\_perm(l_0, l_1)$  alors on a :  $is\_perm(a :: l_0, a :: l_1)$ .
- ❷ Pour  $a \in \mathbb{N}$  et  $l$  une liste, on a :  $is\_perm(a :: l, l ++ [a])$ ;

## Preuve de $is\_perm([1; 2; 3], [3; 2; 1])$

- ❶ Règle de transitivité sur  $[1; 2; 3]$ ,  $[2; 3] ++ [1]$ , et  $[3; 2; 1]$  ;
- ❷ Règle de transitivité sur  $[2; 3; 1]$ ,  $[3; 1] ++ [2]$ , et  $[3; 2; 1]$  ;
- ❸ Règle sur le cons (1) ;
- ❹ Règle de transitivité sur  $[1; 2]$ ,  $[1] ++ [2]$ , et  $[2; 1]$ .

# Exercice

## Relation « être une permutation de »

- ❶ Pour  $a \in \mathbb{N}$  et  $l_0, l_1$  deux listes, si  $is\_perm(l_0, l_1)$  alors on a :  $is\_perm(a :: l_0, a :: l_1)$  ;
- ❷ Pour  $a \in \mathbb{N}$  et  $l$  une liste, on a :  $is\_perm(a :: l, l++[a])$  ;
- ❸ + règles de clôture réflexive, transitive, et symétrique.

## Preuve de $is\_perm([1; 2; 3], [3; 2; 1])$

- ❶ Règle de transitivité sur  $[1; 2; 3]$ ,  $[2; 3]++[1]$ , et  $[3; 2; 1]$  ;
- ❷ Règle de transitivité sur  $[2; 3; 1]$ ,  $[3; 1]++[2]$ , et  $[3; 2; 1]$  ;
- ❸ Règle sur le cons (1) ;
- ❹ Règle de transitivité sur  $[1; 2]$ ,  $[1]++[2]$ , et  $[2; 1]$ .



# Exercice

## Relation « être triée »

- 1 On a :  $is\_sort([])$  ;
- 2 Pour  $n \in \mathbb{N}$ , on a :  $is\_sort([n])$  ;
- 3 Pour  $n, m \in \mathbb{N}$  et  $l$  une liste, si  $n \leq$  et  $is\_sort(m :: l)$ , alors on a :  $is\_sort(n :: m :: l)$ .

## Preuve de $is\_sort([1; 2; 3])$

- 1 Applications successives de la règle cons (3) ;
- 2 Puis application de la règle de la liste singleton (2).

# Exercice

## Fonction de tri

① Fonction d'insertion :

$$\text{insert}(x, l) = \begin{cases} [x], & \text{si } l = [] \\ x :: h :: t, & \text{si } l = h :: t \text{ et } x \leq h \\ h :: (\text{insert}(x, t)), & \text{si } l = h :: t \text{ et } x > h \end{cases}$$

② Fonction de tri :

$$\text{sort}(l) = \begin{cases} [], & \text{si } l = [] \\ \text{insert}(h, \text{sort}(t)), & \text{si } l = h :: t \end{cases}$$

# Exercice

## Théorème de correction

- Pour  $l, l'$  deux listes, si  $sort(l) = l'$ , alors on a :  $is\_perm(l, l')$  et  $is\_sort(l')$ .

## Preuve

- Par induction sur  $l$  ;
- Cas de base : trivial (remplacer  $l$  par  $[]$  partout) ;
- Cas inductif : on doit démontrer

$$sort(a :: l) = l' \Rightarrow is\_perm(a :: l, l') \wedge is\_sort(l')$$

avec l'hypothèse d'induction :

$$\forall l'. sort(l) = l' \Rightarrow is\_perm(l, l') \wedge is\_sort(l')$$

# Exercice

## Preuve

- On remplace  $l'$  par  $sort(a :: l)$ , que l'on calcule, et on élimine l'hypothèse d'induction avec  $l' = sort(l)$ . On doit donc montrer :

- 1  $is\_perm(a :: l, insert(a, sort(l)))$  ;
- 2  $is\_sort(insert(a, sort(l)))$ .

sous les hypothèses :

$$\begin{aligned} &is\_perm(l, sort(l)) \\ &is\_sort(sort(l)) \end{aligned}$$

- Pour (1), appliquer la transitivité avec  $a :: (sort(l))$  :
  - ▶ Pour  $is\_perm(a :: l, a :: (sort(l)))$ , règle du cons et hypothèse ;
  - ▶ Pour  $is\_perm(a :: (sort(l)), insert(a, sort(l)))$ , on démontre le théorème :

$$\forall a, l. is\_perm(a :: l, insert(a, l))$$

# Exercice

## Preuve

- Pour (2), on démontre le théorème :

$$\forall a, l. is\_sort(l) \Rightarrow is\_sort(insert(a, l))$$

- Preuve du théorème (1), par induction sur  $l$  :
  - ▶ Cas de base : trivial (remplacer  $l$  par  $[]$  partout) ;
  - ▶ Cas inductif : remplacer  $l$  par  $h :: t$  et raisonner par cas suivant que  $a \leq h$  ou non ; on doit démontrer un théorème intermédiaire :

$$\forall a_1, a_2, l. is\_perm(a_1 :: a_2 :: l, a_2 :: a_1 :: l)$$

- Preuve du théorème (2), par induction sur  $is\_sort(l)$  (schéma) :

$$\begin{aligned} & \forall P. P([]) \Rightarrow (\forall n. P([n])) \Rightarrow \\ & (\forall n, m, l. n \leq m \Rightarrow is\_sort(m :: l) \Rightarrow P(m :: l) \Rightarrow P(n :: m :: l)) \Rightarrow \\ & \forall l. is\_sort(l) \Rightarrow P(l) \end{aligned}$$

## Exercice (2)

### Pgcd de deux entiers naturels non nuls

- Spécifier la relation « être le pgcd de deux entiers naturels non nuls » ;
- Écrire la fonction de pgcd en utilisant l'algorithme des soustractions ;
- Démontrer que la fonction est correcte.

## Exercice (2)

### Relation de pgcd

Si  $r = \text{pgcd}(a, b)$ , avec  $a$  et  $b$  deux entiers naturels non nuls, alors on a :

- $r$  divise  $a$  et  $b$  ;
- il existe  $x$  et  $y$  (co-facteurs) tels que  $r = ax + by$  (Bachet-Bézout).

## Exercice (2)

### Fonction de pgcd

On définit le pgcd par soustractions successives par la fonction suivante de type  $\mathbb{N}^* \times \mathbb{N}^* \rightarrow \mathbb{N}^*$  :

$$\gcd(a, b) = \begin{cases} a, & \text{si } a = b \\ \gcd(a, b - a), & \text{si } b > a \\ \gcd(a - b, b), & \text{sinon} \end{cases}$$

- En l'état, cette fonction est mathématiquement mal définie, car on ne sait pas si elle termine ;
- On a besoin de se convaincre qu'elle termine en utilisant une relation bien fondée ;
- On a donc besoin d'induction bien fondée, appelée aussi induction Noëtherienne, qui est une induction plus générale.



## Exercice (2)

### Relation bien fondée

Soit une relation binaire  $\mathcal{R}$  sur un ensemble  $A$ , c'est-à-dire que  $\mathcal{R} \subseteq A \times A$ .

La relation  $\mathcal{R}$  sera bien fondée dans  $A$  s'il n'existe pas de chaînes descendantes infinies, c'est-à-dire de suite  $(u_i)$  dans  $A$  telle que  $u_{i+1} \mathcal{R} u_i$  pour tout  $i$ .

Une fonction  $f$  sur  $A$  sera définie par induction bien fondée si elle est de la forme suivante :

$$f(x) = g(x, f_{|\text{inf}(x)})$$

où  $f_{|\text{inf}(x)} = \{f(y) \mid y \mathcal{R} x\}$ .

## Exercice (2)

Retour à l'exemple

$$\gcd(a, b) = \begin{cases} a, & \text{si } a = b \\ \gcd(a, b - a), & \text{si } b > a \\ \gcd(a - b, b), & \text{sinon} \end{cases}$$

Quelle est la relation bien fondée ?

$$(x, y) \mathcal{R} (x', y') = x + y < x' + y'$$

## Exercice (2)

### Schéma d'induction bien fondée

Pour faire des preuves sur le pgcd, on a besoin du schéma d'induction bien fondée correspondant.

Le schéma général d'induction bien fondée est le suivant :

$$\forall P \in A \rightarrow Prop. (\forall x \in A. \forall y \in \inf(x). P(y) \Rightarrow P(x)) \Rightarrow \forall x \in A. P(x)$$

où  $\inf(x) = \{y \mid y \mathcal{R} x\}$ .

Sur  $\mathbb{N}$ , on retrouve les schémas d'induction habituels :

- Schéma d'induction structurelle :  $x \mathcal{R} y \equiv y = x + 1$  ;
- Schéma d'induction généralisée :  $x \mathcal{R} y \equiv x < y$ .