Preuves par induction

David Delahaye
David.Delahaye@lirmm.fr

Université de Montpellier
Faculté des Sciences

Licence Informatique L3 2022-2023

Induction structurelle

Spécification

On définit la relation inductive is_sum de type $\mathbb{N} \times \mathbb{N} \to \mathrm{Prop}$ de la façon suivante :

1. On a : $is\_sum(0, 0)$;
2. 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 $\mathbb{N} \to \mathbb{N}$:

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

Induction structurelle

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 $\mathbb{N}$:

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

Dans notre cas :

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

Induction structurelle

Preuve

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

1. Cas de base :

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

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

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

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.

Induction structurelle

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 :

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

Induction structurelle

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.

Induction structurelle

Théorème de complétude

C'est le théorème inverse de la correction (les deux théorèmes sont nécessaires pour assurer l'adéquation entre la fonction et sa spécification) :

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

Preuve

La preuve se fait par induction sur la relation *is_sum*.
On utilise le schéma d'induction lié à la relation *is_sum* (ce schéma peut toujours être qualifié de structurel ; il suit la définition de la relation) :

$$\begin{array}{l} \forall P \in \mathbb{N} \times \mathbb{N} \rightarrow \mathrm{Prop}. \\ \quad P(0,0) \Rightarrow \\ \quad (\forall n, s \in \mathbb{N}. is\_sum(n, s) \Rightarrow P(n, s) \Rightarrow P(S(n), s + S(n))) \Rightarrow \\ \quad \forall n, s \in \mathbb{N}. is\_sum(n, s) \Rightarrow P(n, s) \end{array}$$

Induction structurelle

Preuve

On applique le schéma d'induction avec $P(n, s) = (f_{is\_sum}(n) = s)$.
On doit démontrer :

1. Cas de base :

$$f_{is\_sum}(0) = 0$$

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

Induction structurelle

Preuve

On applique le schéma d'induction avec $P(n, s) = (f_{is\_sum}(n) = s)$.
On doit démontrer :

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

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

sous les hypothèses d'induction :

$$is\_sum(n, s)$$

$$f_{is\_sum}(n) = s$$

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

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

Induction structurelle

Preuve

On applique le schéma d'induction avec $P(n, s) = (f_{is\_sum}(n) = s)$.
On doit démontrer :

2. Cas inductif :
   On remplace s par $f_{is\_sum}(n)$, ce qui donne (trivial) :

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

Induction structurelle en Coq

Définition de la relation inductive *is_sum*

```
Coq < Inductive is_sum : nat -> nat -> Prop :=
      | is_sum_O : is_sum 0 0
      | is_sum_S : forall n s : nat,
        is_sum n s -> is_sum (S n) (s + (S n)).
is_sum is defined
is_sum_ind is defined
```

Remarques

- La relation `is_sum` est définie.
- `is_sum_O` et `is_sum_S` sont ses constructeurs.
- Ils doivent être considérés comme des axiomes.
- Le schéma d'induction structurelle `is_sum_ind` est généré.

Induction structurelle en Coq

Schéma d'induction structurelle `is_sum_ind`

```
Coq < Print is_sum_ind.
is_sum_ind = ... :
  forall P : nat -> nat -> Prop,
  P 0 0 ->
  (forall n s : nat, is_sum n s -> P n s -> P (S n) (s + S n)) ->
  forall n n0 : nat, is_sum n n0 -> P n n0
```

Remarques

- Ce schéma permet de faire de l'induction sur la relation `is_sum`.
- Ce schéma reste complètement structurel.
- Il suit la définition de la relation inductive.

Induction structurelle en Coq

Définition de la fonction f_{is_sum}

```
Coq < Fixpoint sum (n : nat) : nat :=
        match n with
        | 0 => 0
        | (S n) => (sum n) + (S n)
        end.
sum is defined
sum is recursively defined (decreasing on 1st argument)
```

Remarques

- Toutes les fonctions en Coq doivent terminer.
- La terminaison est assurée par un ordre bien fondé.
- Généralement, on utilise l'ordre sous-terme.
- La récursion est alors dite structurelle (mot-clé `struct`).
- Quand on ne précise rien, elle est structurelle par défaut et se fait sur le premier argument de la fonction.

Induction structurelle en Coq

Théorème de correction de f_{is_sum} vis-à-vis de *is_sum*

```
Coq < Lemma sum_sound :
      forall (n s : nat), (sum n) = s -> is_sum n s.
1 subgoal
  ============================
  forall n s : nat, sum n = s -> is_sum n s

Coq < induction n; intros.
2 subgoals
  s : nat
  H : sum 0 = s
  ============================
  is_sum 0 s
subgoal 2 is:
 is_sum (S n) s
```

Induction structurelle en Coq

Théorème de correction de f_{is_sum} vis-à-vis de *is_sum*

```
Coq < rewrite <- H.
2 subgoals
  s : nat
  H : sum 0 = s
  ============================
  is_sum 0 (sum 0)

Coq < simpl.
2 subgoals
  s : nat
  H : sum 0 = s
  ============================
  is_sum 0 0

Coq < apply is_sum_O.
```

Induction structurelle en Coq

Théorème de correction de f_{is_sum} vis-à-vis de *is_sum*

```
1 subgoal
  n : nat
  IHn : forall s : nat, sum n = s -> is_sum n s
  s : nat
  H : sum (S n) = s
  ============================
  is_sum (S n) s

Coq < rewrite <- H.
1 subgoal
  n : nat
  IHn : forall s : nat, sum n = s -> is_sum n s
  s : nat
  H : sum (S n) = s
  ============================
  is_sum (S n) (sum (S n))
```

Induction structurelle en Coq

Théorème de correction de f_{is_sum} vis-à-vis de *is_sum*

```
Coq < simpl.
1 subgoal
  n : nat
  IHn : forall s : nat, sum n = s -> is_sum n s
  s : nat
  H : sum (S n) = s
  ============================
  is_sum (S n) (sum n + S n)

Coq < apply is_sum_S.
1 subgoal
  n : nat
  IHn : forall s : nat, sum n = s -> is_sum n s
  s : nat
  H : sum (S n) = s
  ============================
  is_sum n (sum n)
```

Induction structurelle en Coq

Théorème de correction de f_{is_sum} vis-à-vis de is_sum

```
Coq < apply IHn.
1 subgoal
  n : nat
  IHn : forall s : nat, sum n = s -> is_sum n s
  s : nat
  H : sum (S n) = s
  ============================
  sum n = sum n

Coq < reflexivity.
No more subgoals.

Coq < Qed.
sum_sound is defined
```

Induction structurelle en Coq

Théorème de complétude de f_{is_sum} vis-à-vis de *is_sum*

```
Coq < Lemma sum_complete :
      forall (n s : nat), is_sum n s -> (sum n) = s.
1 subgoal
  ============================
  forall n s : nat, is_sum n s -> sum n = s

Coq < intros.
1 subgoal
  n, s : nat
  H : is_sum n s
  ============================
  sum n = s
```

Induction structurelle en Coq

Théorème de complétude de f_{is_sum} vis-à-vis de is_sum

```
Coq < elim H; intros.
2 subgoals
  n, s : nat
  H : is_sum n s
  ============================
  sum 0 = 0
subgoal 2 is:
 sum (S n0) = s0 + S n0

Coq < simpl.
2 subgoals
  n, s : nat
  H : is_sum n s
  ============================
  0 = 0

Coq < reflexivity.
```

Induction structurelle en Coq

Théorème de complétude de f_{is_sum} vis-à-vis de *is_sum*

```
1 subgoal
  n, s : nat
  H : is_sum n s
  n0, s0 : nat
  H0 : is_sum n0 s0
  H1 : sum n0 = s0
  ============================
  sum (S n0) = s0 + S n0

Coq < simpl.
1 subgoal
  [...]
  H0 : is_sum n0 s0
  H1 : sum n0 = s0
  ============================
  sum n0 + S n0 = s0 + S n0
```

Induction structurelle en Coq

Théorème de complétude de f_{is_sum} vis-à-vis de is_sum

```
Coq < rewrite H1.
1 subgoal
  n, s : nat
  H : is_sum n s
  n0, s0 : nat
  H0 : is_sum n0 s0
  H1 : sum n0 = s0
  ============================
  s0 + S n0 = s0 + S n0

Coq < reflexivity.
No more subgoals.

Coq < Qed.
sum_complete is defined
```

Induction structurelle en Coq

Une autre version de la preuve de complétude

```
Coq < Lemma sum_complete :
      forall (n s : nat), is_sum n s -> (sum n) = s.
1 subgoal
  ============================
  forall n s : nat, is_sum n s -> sum n = s

Coq < induction n; intros.
2 subgoals
  s : nat
  H : is_sum 0 s
  ============================
  sum 0 = s
subgoal 2 is:
 sum (S n) = s
```

Induction structurelle en Coq

Une autre version de la preuve de complétude

```
Coq < simpl.
2 subgoals
  s : nat
  H : is_sum 0 s
  ============================
  0 = s

Coq < inversion H.
2 subgoals
  s : nat
  H : is_sum 0 s
  H1 : 0 = s
  ============================
  0 = 0

Coq < reflexivity.
```

Induction structurelle en Coq

Une autre version de la preuve de complétude

```
1 subgoal
  n : nat
  IHn : forall s : nat, is_sum n s -> sum n = s
  s : nat
  H : is_sum (S n) s
  ============================
  sum (S n) = s

Coq < simpl.
1 subgoal
  n : nat
  IHn : forall s : nat, is_sum n s -> sum n = s
  s : nat
  H : is_sum (S n) s
  ============================
  sum n + S n = s
```

Induction structurelle en Coq

Une autre version de la preuve de complétude

```
Coq < inversion H.
1 subgoal
  n : nat
  IHn : forall s : nat, is_sum n s -> sum n = s
  s : nat
  H : is_sum (S n) s
  n0, s0 : nat
  H1 : is_sum n s0
  H0 : n0 = n
  H2 : s0 + S n = s
  ============================
  sum n + S n = s0 + S n
```

Induction structurelle en Coq

Une autre version de la preuve de complétude

```
Coq < rewrite (IHn s0 H1).
1 subgoal
  [...]
  IHn : forall s : nat, is_sum n s -> sum n = s
  H1 : is_sum n s0
  ============================
  s0 + S n = s0 + S n

Coq < reflexivity.
No more subgoals.

Coq < Qed.
sum_complete_bis is defined
```

Induction fonctionnelle

Spécification

On définit la relation inductive *is_even* de type $\mathbb{N} \rightarrow \mathrm{Prop}$ 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 $\mathbb{N} \rightarrow \mathbb{B}$:

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

Induction fonctionnelle

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 :

1. Cas de base :

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

   On applique simplement le cas de base de is_even.

Induction fonctionnelle

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 :

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

Induction fonctionnelle

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 :

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

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

Induction fonctionnelle

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 :
   $$f_{is\_even}(S(n)) = \top \Rightarrow is\_even(S(n))$$
   $$\begin{array}{l}(f_{is\_even}(n) = \top \Rightarrow is\_even(n)) \Rightarrow \\ f_{is\_even}(S(n)) = \top \Rightarrow is\_even(S(n))\end{array}$$

Induction fonctionnelle

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 :

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

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

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

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}

$$
\begin{array}{l}
\forall P \in \mathbb{N} \times \mathbb{B} \to Prop. \\
P(0, \top) \Rightarrow P(1, \bot) \Rightarrow \\
(\forall p \in \mathbb{N}. P(p, f_{is\_even}(p)) \Rightarrow P(S(S(p)), f_{is\_even}(p))) \Rightarrow \\
\forall n \in \mathbb{N}. P(n, f_{is\_even}(n))
\end{array}
$$

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 :

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

1. Cas de base (1) :

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

   On applique le cas de base de la relation *is_even*.

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 :

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

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

Définition de la relation inductive *is_even*

```
Coq < 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)).
is_even is defined
is_even_ind is defined
```

Définition de la fonction f_{is_even}

```
Coq < Fixpoint even (n : nat) : Prop :=
        match n with
        | 0 => True
        | 1 => False
        | (S (S n)) => even n
        end.
even is defined
even is recursively defined (decreasing on 1st argument)
```

Induction fonctionnelle en Coq

Génération du schéma d'induction fonctionnelle de f_{is_even}

```
Coq < Require Import FunInd.
[Loading ML file extraction_plugin.cmxs ... done]
[Loading ML file recdef_plugin.cmxs ... done]

Coq < Functional Scheme even_ind := Induction for even Sort Prop.
even_equation is defined
even_ind is defined
```

Induction fonctionnelle en Coq

Schéma d'induction fonctionnelle de la fonction f_{is_even}

```
Coq < Print even_ind.
even_ind = ... :
  forall P : nat -> Prop -> Prop,
  (forall n : nat, n = 0 -> P 0 True) ->
  (forall n n0 : nat, n = S n0 -> n0 = 0 -> P 1 False) ->
  (forall n n0 : nat,
   n = S n0 ->
   forall n1 : nat,
   n0 = S n1 -> P n1 (even n1) -> P (S (S n1)) (even n1)) ->
  forall n : nat, P n (even n)
```

Induction fonctionnelle en Coq

Théorème de correction de f_{is_even} vis-à-vis de *is_even*

```
Coq < Lemma even_sound :
      forall (n : nat), (even n) = True -> is_even n.
1 subgoal
  ============================
  forall n : nat, even n = True -> is_even n

Coq < intro.
1 subgoal
  n : nat
  ============================
  even n = True -> is_even n
```

Induction fonctionnelle en Coq

Théorème de correction de f_{is_even} vis-à-vis de *is_even*

```
Coq < functional induction (even n) using even_ind; intros.
3 subgoals
  H : True = True
  ============================
  is_even 0
subgoal 2 is:
 is_even 1
subgoal 3 is:
 is_even (S (S n1))

Coq < apply is_even_O.
```

Induction fonctionnelle en Coq

Théorème de correction de f_{is_even} vis-à-vis de *is_even*

```
2 subgoals
  H : False = True
  ============================
  is_even 1
subgoal 2 is:
 is_even (S (S n1))

Coq < elimtype False.
2 subgoals
  H : False = True
  ============================
  False
```

Induction fonctionnelle en Coq

Théorème de correction de f_{is_even} vis-à-vis de *is_even*

```
Coq < rewrite H.
2 subgoals
  H : False = True
  ============================
  True

Coq < auto.
1 subgoal
  n1 : nat
  IHP : even n1 = True -> is_even n1
  H : even n1 = True
  ============================
  is_even (S (S n1))
```

Induction fonctionnelle en Coq

Théorème de correction de f_{is_even} vis-à-vis de *is_even*

```
Coq < apply is_even_S.
1 subgoal
  n1 : nat
  IHP : even n1 = True -> is_even n1
  H : even n1 = True
  ============================
  is_even n1

Coq < apply (IHP H).
No more subgoals.

Coq < Qed.
even_sound is defined
```

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 (co-facteurs) 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.).

Induction bien fondée

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 Nœtherienne, qui est une induction plus générale.

Induction bien fondée

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 :

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

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

Induction bien fondée

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 ?

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

Induction bien fondée

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 : $x \mathcal{R} y \equiv x < y$.

Induction bien fondée en Coq

Définition de la fonction *gcd*

```
Coq < Require Export Arith.
[Loading ML file quote_plugin.cmxs ... done]
[Loading ML file newring_plugin.cmxs ... done]

Coq < Require Export Recdef.
[Loading ML file extraction_plugin.cmxs ... done]
[Loading ML file recdef_plugin.cmxs ... done]

Coq < Require Export Omega.
[Loading ML file omega_plugin.cmxs ... done]

Coq < Definition f_gcd (a b : nat * nat) :=
      (fst a) + (snd a) < (fst b) + (snd b).
f_gcd is defined
```

Induction bien fondée en Coq

Définition de la fonction *gcd*

```
Coq < Function gcd (c : nat * nat) {wf f_gcd c} : option nat :=
        match c with
        | (0, 0) => None
        | (0, _) => None
        | (_, 0) => None
        | _ =>
          let n := fst c in
          let m := snd c in
          match (lt_eq_lt_dec n m) with
          | inleft a =>
            match a with
            | left _ => gcd (n, (m - n))
            | right _ => Some n
            end
          | inright a => gcd ((n - m), m)
          end
        end.
```

Induction bien fondée en Coq

Définition de la fonction *gcd*

```
Coq < Print lt_eq_lt_dec.
lt_eq_lt_dec = ... :
  forall n m : nat, {n < m} + {n = m} + {m < n}

Coq < Print sumor.
Inductive sumor (A : Type) (B : Prop) : Type :=
    inleft : A -> A + {B} | inright : B -> A + {B}
```

Induction bien fondée en Coq

Définition de la fonction *gcd*

```
3 subgoals
  ==============================
  forall (c : nat * nat) (n n0 n1 : nat),
  n = S n1 ->
  forall n2 : nat,
  n0 = S n2 ->
  c = (S n1, S n2) ->
  forall
    (a : {fst (S n1, S n2) < snd (S n1, S n2)} +
         {fst (S n1, S n2) = snd (S n1, S n2)})
    (l : fst (S n1, S n2) < snd (S n1, S n2)),
  a = left l ->
  lt_eq_lt_dec (fst (S n1, S n2)) (snd (S n1, S n2)) =
    inleft (left l) ->
  f_gcd (fst (S n1, S n2), snd (S n1, S n2) - fst (S n1, S n2))
    (S n1, S n2)
```

Induction bien fondée en Coq

Définition de la fonction *gcd*

```
Coq < intros.
3 subgoals
  [...]
  ============================
  f_gcd (fst (S n1, S n2), snd (S n1, S n2) - fst (S n1, S n2))
    (S n1, S n2)

Coq < unfold f_gcd.
3 subgoals
  [...]
  ============================
  fst (fst (S n1, S n2), snd (S n1, S n2) - fst (S n1, S n2)) +
  snd (fst (S n1, S n2), snd (S n1, S n2) - fst (S n1, S n2)) <
  fst (S n1, S n2) + snd (S n1, S n2)
```

Induction bien fondée en Coq

Définition de la fonction *gcd*

```
Coq < simpl.
3 subgoals
  [...]
  ============================
  S (n1 + (n2 - n1)) < S (n1 + S n2)

Coq < omega.
2 subgoals
  ============================
  [...] ->
  f_gcd (fst (S n1, S n2) - snd (S n1, S n2), snd (S n1, S n2))
    (S n1, S n2)

Coq < [même preuve].
```

Induction bien fondée en Coq

Définition de la fonction *gcd*

```
1 subgoal
  ============================
  well_founded f_gcd

Coq < apply well_founded_ltof.
No more subgoals.

Coq < Defined.
gcd_tcc is defined
gcd_terminate is defined
gcd_ind is defined
gcd_rec is defined
gcd_rect is defined
R_gcd_correct is defined
R_gcd_complete is defined
```

Induction bien fondée en Coq

Définition de la fonction *gcd*

```
Coq < Print well_founded_ltof.
well_founded_ltof = ... :
  forall (A : Type) (f : A -> nat), well_founded (ltof A f)
Coq < Print ltof.

ltof =
fun (A : Type) (f : A -> nat) (a b : A) => f a < f b
     : forall A : Type, (A -> nat) -> A -> A -> Prop
```