

Verification of Functional Programs

Induction

Andrés Sicard-Ramírez

EAFIT University

Semester 2014-1

Source Code

All the source code have been tested with [Agda](#) 2.3.2, [Coq](#) 8.4pl3 and [Isabelle](#) 2013-2.

The Principle of Mathematical Induction

The principle of mathematical induction

Let $A(x)$ be a propositional function. To prove $A(x)$ for all $x \in \mathbb{N}$, it suffices prove:

- the **basis** $A(0)$ and
- the **induction step**, that $A(n) \Rightarrow A(n+1)$, for all $n \in \mathbb{N}$
($A(n)$ is called the **induction hypothesis**).

The Principle of Mathematical Induction

First-order logic version

Let $A(x)$ be a formula with free variable x . For **each** formula $A(x)$:

$$[A(0) \wedge \forall x.A(x) \Rightarrow A(x+1)] \Rightarrow \forall x.A(x) \quad (\text{axiom schema of induction})$$

The Principle of Mathematical Induction

First-order logic version

Let $A(x)$ be a formula with free variable x . For **each** formula $A(x)$:

$$[A(0) \wedge \forall x.A(x) \Rightarrow A(x+1)] \Rightarrow \forall x.A(x) \quad (\text{axiom schema of induction})$$

Equivalent formulations

$$A(0) \Rightarrow [(\forall x.A(x) \Rightarrow A(x+1)) \Rightarrow \forall x.A(x)] \quad (\text{by exportation})$$

$$A(0) \Rightarrow (\forall x.A(x) \Rightarrow A(x+1)) \Rightarrow \forall x.A(x) \quad (\text{right-assoc. conditional})$$

The Principle of Mathematical Induction

First-order logic version

Let $A(x)$ be a formula with free variable x . For **each** formula $A(x)$:

$$[A(0) \wedge \forall x.A(x) \Rightarrow A(x+1)] \Rightarrow \forall x.A(x) \quad (\text{axiom schema of induction})$$

Equivalent formulations

$$A(0) \Rightarrow [(\forall x.A(x) \Rightarrow A(x+1)) \Rightarrow \forall x.A(x)] \quad (\text{by exportation})$$

$$A(0) \Rightarrow (\forall x.A(x) \Rightarrow A(x+1)) \Rightarrow \forall x.A(x) \quad (\text{right-assoc. conditional})$$

Inference rule style

$$\frac{A(0) \quad \forall x.A(x) \Rightarrow A(x+1)}{\forall x.A(x)}$$

The Principle of Mathematical Induction

Higher-order logic

'The adjective 'first-order' is used to distinguish the languages... from those in which are predicates having other predicates or functions as arguments, or quantification over functions or predicates, or both.' [Mendelson (1965) 1997, p. 56]

The Principle of Mathematical Induction

Higher-order logic

'The adjective 'first-order' is used to distinguish the languages... from those in which are predicates having other predicates or functions as arguments, or quantification over functions or predicates, or both.' [Mendelson (1965) 1997, p. 56]

Second-order logic version

Let X be a predicate variable.

$$\forall X.X(0) \Rightarrow (\forall x.X(x) \Rightarrow X(x+1)) \Rightarrow \forall x.X(x) \quad (\text{axiom of induction})$$

The Principle of Mathematical Induction

Historical remark

Dedekind [(1888) 2005] and Peano [(1889) 1967] axiom: $1 \in \mathbb{N}$.

The Principle of Mathematical Induction

Remark

Coq generates the induction principles associated to the inductively defined (data) types.

Example (**Coq**)

The inductive data type for natural numbers.

```
Require Import Unicode.Utf8.
```

```
Inductive nat : Set :=  
| 0 : nat  
| S : nat → nat.
```

Continued on next slide

The Principle of Mathematical Induction

Example (continuation)

The Check `nat_ind` command yields:

```
nat_ind : ∀ P : nat → Prop,  
          P 0 → (∀ n : nat, P n → P (S n)) → ∀ n : nat, P n
```

The Principle of Mathematical Induction

Example (continuation)

The Check `nat_ind` command yields:

$$\text{nat_ind} : \forall P : \text{nat} \rightarrow \mathbf{Prop}, \\ P\ 0 \rightarrow (\forall n : \text{nat}, P\ n \rightarrow P\ (S\ n)) \rightarrow \forall n : \text{nat}, P\ n$$

The Check `nat_rec` command yields:

$$\text{nat_rec} : \forall P : \text{nat} \rightarrow \mathbf{Set}, \\ P\ 0 \rightarrow (\forall n : \text{nat}, P\ n \rightarrow P\ (S\ n)) \rightarrow \forall n : \text{nat}, P\ n$$

The Principle of Mathematical Induction

Example (continuation)

The Check `nat_ind` command yields:

$$\text{nat_ind} : \forall P : \text{nat} \rightarrow \mathbf{Prop}, \\ P\ 0 \rightarrow (\forall n : \text{nat}, P\ n \rightarrow P\ (S\ n)) \rightarrow \forall n : \text{nat}, P\ n$$

The Check `nat_rec` command yields:

$$\text{nat_rec} : \forall P : \text{nat} \rightarrow \mathbf{Set}, \\ P\ 0 \rightarrow (\forall n : \text{nat}, P\ n \rightarrow P\ (S\ n)) \rightarrow \forall n : \text{nat}, P\ n$$

The Check `nat_rect` command yields:

$$\text{nat_rect} : \forall P : \text{nat} \rightarrow \mathbf{Type}, \\ P\ 0 \rightarrow (\forall n : \text{nat}, P\ n \rightarrow P\ (S\ n)) \rightarrow \forall n : \text{nat}, P\ n$$

The Principle of Mathematical Induction

Implementation remark

What happen if instead of using

Inductive nat : **Set** := 0 : nat | S : nat → nat

we renamed the data type nat by

Inductive P : **Set** := 0 : P | S : P → P

or we renamed the data constructor S by

Inductive nat : **Set** := 0 : nat | P : nat → nat

?

Source: McBride and McKinna [2004]

The Principle of Mathematical Induction

Remark

Isabelle also generates the induction principles associated to the inductively defined (data) types.

Example (**Isabelle**)

The inductive data type for natural numbers.

```
datatype nat = Z | S nat
```

The Principle of Mathematical Induction

Remark

Isabelle also generates the induction principles associated to the inductively defined (data) types.

Example (Isabelle)

The inductive data type for natural numbers.

```
datatype nat = Z | S nat
```

The `print_theorems` command yields (among others):

```
nat.induct: ?P Z  $\Rightarrow$   $\forall x.$  ?P x  $\Rightarrow$  ?P (S x)  $\Rightarrow$  ?P ?nat
```


The Principle of Mathematical Induction

Remark

Agda doesn't generate the induction principles, but the user can use pattern matching on the inductively defined (data) types.

Example (Agda)

The inductive data type for natural numbers.

```
data ℕ : Set where
```

```
  zero : ℕ
```

```
  succ : ℕ → ℕ
```

Continued on next slide

The Principle of Mathematical Induction

Example (continuation)

The principle of mathematical induction.

$$\begin{aligned} \mathbb{N}\text{-ind} : & (A : \mathbb{N} \rightarrow \mathbf{Set}) \rightarrow \\ & A \text{ zero} \rightarrow \\ & (\forall n \rightarrow A \ n \rightarrow A \ (\text{succ } n)) \rightarrow \\ & \forall n \rightarrow A \ n \end{aligned}$$
$$\mathbb{N}\text{-ind } A \ A0 \ h \ \text{zero} \quad = \ A0$$
$$\mathbb{N}\text{-ind } A \ A0 \ h \ (\text{succ } n) = h \ n \ (\mathbb{N}\text{-ind } A \ A0 \ h \ n)$$

The Principle of Mathematical Induction

Remark

In *Agda*, *Coq* and *Isabelle*, the ‘axiom of induction’ is not an axiom

The Principle of Mathematical Induction

Remark

In *Agda*, *Coq* and *Isabelle*, the ‘axiom of induction’ is not an axiom (the introduction rules induce the induction principles).

Course-of-Values Induction

Course-of-values induction (strong or complete induction)

Let $A(x)$ be a propositional function. To prove $A(x)$ for all $x \in \mathbb{N}$, it is enough to prove:

$$(\forall 0 \leq k < n)(A(k) \Rightarrow A(n)), \text{ for all } n \in \mathbb{N}.$$

Course-of-Values Induction

Example

The Fibonacci numbers are defined by $F_0 = 0$, $F_1 = 1$ and $F_{k+2} = F_k + F_{k+1}$, so $F = \{0, 1, 1, 2, 3, 5, 8, 13, 21, \dots\}$.

Course-of-Values Induction

Example

The Fibonacci numbers are defined by $F_0 = 0$, $F_1 = 1$ and $F_{k+2} = F_k + F_{k+1}$, so $F = \{0, 1, 1, 2, 3, 5, 8, 13, 21, \dots\}$.

Let Φ and $\hat{\Phi}$ be the roots of the equation $x^2 - x - 1$:

$$\Phi = \frac{1 + \sqrt{5}}{2} \text{ and } \hat{\Phi} = \frac{1 - \sqrt{5}}{2},$$

so $\Phi^2 = \Phi + 1$ and $\hat{\Phi}^2 = \hat{\Phi} + 1$. Then [Bird and Wadler 1988, p. 107.]

$$F_k = \frac{1}{\sqrt{5}}(\Phi^k - \hat{\Phi}^k), \text{ for all } k \in \mathbb{N}.$$

Mathematical and Course-of-Values Induction

Theorem

Mathematical induction and course-of-values induction are equivalent [Winskel 2010].

Structural Induction

Structural induction

Let $A(X)$ be a propositional function about the structures X that are defined by some **recursive/inductive** definition.

Structural Induction

Structural induction

Let $A(X)$ be a propositional function about the structures X that are defined by some **recursive/inductive** definition.

To prove $A(X)$ for all the structures X , it suffices prove [Hopcroft, Motwani and Ullman (1979) 2007]:

- $A(X)$ for the basis structure(s) of X and

Structural Induction

Structural induction

Let $A(X)$ be a propositional function about the structures X that are defined by some **recursive/inductive** definition.

To prove $A(X)$ for all the structures X , it suffices prove [Hopcroft, Motwani and Ullman (1979) 2007]:

- $A(X)$ for the basis structure(s) of X and
- given a structure X whose recursive/inductive definition says is formed from $Y_1, \dots, Y_k$, that $A(X)$ assuming that the properties $A(Y_1), \dots, A(Y_k)$ hold.

Structural Induction for Lists

Example (Coq)

The parametric inductive data type.

```
Require Import Unicode.Utf8.
```

```
Inductive list (A : Type) : Type :=  
| nil   : list A  
| cons  : A → list A → list A.
```

Structural Induction for Lists

Example (Coq)

The parametric inductive data type.

```
Require Import Unicode.Utf8.
```

```
Inductive list (A : Type) : Type :=  
| nil   : list A  
| cons  : A → list A → list A.
```

The induction principle.

```
list_ind : ∀ (A : Type) (P : list A → Prop),  
  P (nil A) →  
  (∀ (a : A) (l : list A), P l → P (cons A a l)) →  
  ∀ l : list A, P l
```

Structural Induction for Lists

Example (Isabelle)

The polymorphic inductive data type.

```
datatype 'a list = Nil | Cons 'a "'a list"
```

Structural Induction for Lists

Example (Isabelle)

The polymorphic inductive data type.

```
datatype 'a list = Nil | Cons 'a "'a list"
```

The induction principle.

```
list.induct: ?P Nil  $\Rightarrow$   $\forall x1\ x2.$  ?P x2  $\Rightarrow$  ?P (Cons x1 x2))  $\Rightarrow$   
             ?P ?list
```

Structural Induction for Lists

Example (Agda)

The parametric inductive data type.

```
data List (A : Set) : Set where  
  []   : List A  
  _::_ : A → List A → List A
```


Structural Induction for Lists

Example (Agda)

The parametric inductive data type.

```
data List (A : Set) : Set where  
  [] : List A  
  _::_ : A → List A → List A
```

The induction principle.

```
List-ind : {A : Set} (B : List A → Set) →  
  B [] →  
  ((x : A) (xs : List A) → B xs → B (x :: xs)) →  
  ∀ xs → B xs  
List-ind B B[] h []      = B[]  
List-ind B B[] h (x :: xs) = h x xs (List-ind B B[] h xs)
```

Well-Founded Induction

Definition

Let $\prec$ be a binary relation on a set A . The relation $\prec$ is a **well-founded** relation iff every non-empty subset $S \subseteq A$ has a minimal element, that is,

$$(\forall S \subseteq A)[S \neq \emptyset \Rightarrow (\exists m \in S)(\forall s \in S)(s \not\prec m)].$$

Well-Founded Induction

Definition

Let $\prec$ be a binary relation on a set A . The relation $\prec$ is a **well-founded** relation iff every non-empty subset $S \subseteq A$ has a minimal element, that is,

$$(\forall S \subseteq A)[S \neq \emptyset \Rightarrow (\exists m \in S)(\forall s \in S)(s \not\prec m)].$$

Definition (Well-founded induction)

Let $\prec$ be a well-founded relation on a set A and $A(x)$ a propositional function. To prove $A(x)$ for all $a \in A$, it suffices prove:

$$(\forall b \prec a)(A(b) \Rightarrow A(a)), \text{ for all } a \in A.$$

Well-Founded Induction

Example

Let $\prec$ be the well-founded relation on $\mathbb{N}$ given by the graph of the successor function $n \mapsto n + 1$.

Well-Founded Induction

Example

Let $\prec$ be the well-founded relation on $\mathbb{N}$ given by the graph of the successor function $n \mapsto n + 1$.

Then mathematical induction is a special case of well-founded induction.

Well-Founded Induction

Example

Let $\prec$ be the well-founded relation on $\mathbb{N}$ given by the graph of the successor function $n \mapsto n + 1$.

Then mathematical induction is a special case of well-founded induction.

Example

Let $\prec$ be the well-founded relation 'less than' on $\mathbb{N}$.

Well-Founded Induction

Example

Let $\prec$ be the well-founded relation on $\mathbb{N}$ given by the graph of the successor function $n \mapsto n + 1$.

Then mathematical induction is a special case of well-founded induction.

Example

Let $\prec$ be the well-founded relation 'less than' on $\mathbb{N}$.

Then course-of-values induction is a special case of well-founded induction.

Well-Founded Induction

Example

'If we take $\prec$ to be the relation between expressions such that $a \prec b$ holds iff a is an immediate sub-expression of b we obtain the principle of structural induction as a special case of well-founded induction.' [Winskel 2010, p. 93]

Empty Type

In type theory $a : A$ denotes that a is a term (or proof term) of type A .

Empty Type

In type theory $a : A$ denotes that a is a term (or proof term) of type A .

Under the proposition-as-types principle, the **empty type** represents the false (absurdity or contradiction) proposition [Sørensen and Urzyczyn 2006].

Empty Type

In type theory $a : A$ denotes that a is a term (or proof term) of type A .

Under the proposition-as-types principle, the **empty type** represents the false (absurdity or contradiction) proposition [Sørensen and Urzyczyn 2006].

Therefore $e : \text{EmptyType}$ represents a contradiction in our formalisation.

Empty Type

Example (Agda)

```
data  $\perp$  : Set where
```

```
 $\perp$ -elim : {A : Set}  $\rightarrow$   $\perp$   $\rightarrow$  A
```

```
 $\perp$ -elim ()  -- The absurd pattern.
```

Empty Type

Example (Coq)

(From the standard library)

```
Inductive Empty_set : Set :=.
```

```
Empty_set_rect :  $\forall$  (P : Empty_set  $\rightarrow$  Type) (e : Empty_set), P e
```

Empty Type

Example (Coq)

(From the standard library)

```
Inductive Empty_set : Set :=.
```

```
Empty_set_rect :  $\forall$  (P : Empty_set  $\rightarrow$  Type) (e : Empty_set), P e
```

```
Theorem emptySetElim {A : Set}(e : Empty_set) : A.
```

```
  apply (Empty_set_rect (fun _ => A) e).
```

```
Qed.
```

Empty Type

Example (Coq)

(From the standard library)

```
Inductive Empty_set : Set :=.
```

```
Empty_set_rect :  $\forall$  (P : Empty_set  $\rightarrow$  Type) (e : Empty_set), P e
```

```
Theorem emptySetElim {A : Set}(e : Empty_set) : A.
```

```
  apply (Empty_set_rect (fun _ => A) e).
```

```
Qed.
```

```
Theorem emptySetElim' {A : Set}(e : Empty_set) : A.
```

```
  elim e.
```

```
Qed.
```

Strictly Positive Inductive Types

Remark

The inductive types can be defined/represented as least fixed-points of appropriated functions (functors).

Strictly Positive Inductive Types

Remark

The inductive types can be defined/represented as least fixed-points of appropriated functions (functors).

Example

Let 1 be the unity type, and $+$ and $\times$ be the operators for disjoint union and Cartesian product, respectively. Then

$$\begin{aligned}\text{Nat} &:= \mu X. 1 + X, \\ \text{List } A &:= \mu X. 1 + (A \times X).\end{aligned}$$

Strictly Positive Inductive Types

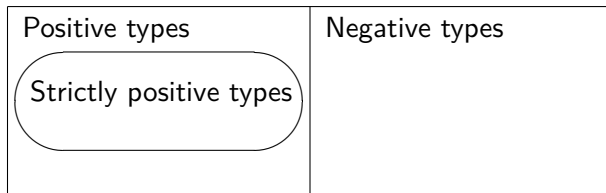
Definition

'The occurrence of a type variable is **positive** iff it occurs within an even number of left hand sides of $\rightarrow$ -types, it is **strictly positive** iff it never occurs on the left hand side of a $\rightarrow$ -type.' [Abel and Altenkirch 2000, p. 21].

Strictly Positive Inductive Types

Definition

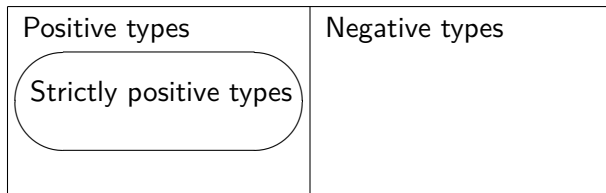
Let $\mu X.F(X)$ be an inductive type. The type $\mu X.F(X)$ is a **strictly positive type** if X occurs strictly positive in $F(X)$.



Strictly Positive Inductive Types

Definition

Let $\mu X.F(X)$ be an inductive type. The type $\mu X.F(X)$ is a **strictly positive type** if X occurs strictly positive in $F(X)$.



Proof assistants

Agda, Coq and Isabelle accept only strictly positive inductive types.

Strictly Positive Inductive Types

Some issues with non-strictly positive inductive types

- Infinite unfolding
See source code in the course web page.

Strictly Positive Inductive Types

Some issues with non-strictly positive inductive types

- Infinite unfolding
See source code in the course web page.
- Proving absurdity
See source code in the course web page.

Strictly Positive Inductive Types

The following examples of inductive types* are rejected by [Agda](#) ([Coq](#) and [Isabelle](#)) because they are not strictly positive inductive types.

Example (negative type)

$$D := \mu X. X \rightarrow X$$

```
data D : Set where
```

```
  lam : (D → D) → D
```

```
-- D is not strictly positive, because it occurs to the left  
-- of an arrow in the type of the constructor lam in the  
-- definition of D.
```

*Adapted from the Coq'Art, Matthes' PhD thesis and Agda's source code.

Strictly Positive Inductive Types

Example (positive, non-strictly positive type)

$$P := \mu X. (X \rightarrow 2) \rightarrow 2$$

data P : Set where

 p : ((P → Bool) → Bool) → P

-- P is not strictly positive, because it occurs to the left
-- of an arrow in the type of the constructor p in the
-- definition of P.

References

- Abel, Andreas and Altenkirch, Thorsten (2000). A Predicative Strong Normalisation Proof for a λ -Calculus with Interleaving Inductive Types. In: Types for Proofs and Programs (TYPES 1999). Ed. by Coquand, Thierry et al. Vol. 1956. Lecture Notes in Computer Science. Springer, pp. 21–40 (cit. on p. 50).
- Bird, Richard and Wadler, Philip (1988). Introduction to Functional Programming. Series in Computer Sciences. Prentice Hall International (cit. on pp. 22, 23).
- Dedekind, Richard [1888] (2005). Was sind und was sollen die Zahlen? In: From Kant to Hilbert: A Source Book in the Foundations of Mathematics. Vol. II. Clarendon Press, pp. 787–833 (cit. on p. 9).
- Hopcroft, John E., Motwani, Rajeev and Ullman, Jefferey D. [1979] (2007). Introduction to Automata Theory, Languages, and Computation. 3rd ed. Pearson Education (cit. on pp. 25–27).
- McBride, Conor and McKinna, James (2004). Functional Pearl: I am not a Number—I am a Free Variable. In: Proceedings of the ACM SIGPLAN 2004 Haskell Workshop, pp. 1–9 (cit. on p. 14).
- Mendelson, Elliott [1965] (1997). Introduction to Mathematical Logic. 4th ed. Chapman & Hall (cit. on pp. 7, 8).

References

- Peano, Giuseppe [1889] (1967). The Principles of Arithmetic, Presented by a New Method. In: From Frege to Gödel: A Source Book in Mathematical Logic, 1879–1931. Ed. by van Heijenoort, Jean. Source Books in the History of the Sciences. Translation of '*Arithmetices principia, nova methodo exposita*' by the editor. Harvard University Press, pp. 83–97 (cit. on p. 9).
- Sørensen, Morten-Heine and Urzyczyn, Paul (2006). Lectures on the Curry-Howard Isomorphism. Vol. 149. Studies in Logic and the Foundations of Mathematics. Elsevier (cit. on pp. 41–43).
- Winskel, Glynn (2010). Set Theory for Computer Science. (Cit. on pp. 24, 40).