Verification of Functional Programs
Induction

Andrés Sicard-Ramírez

Universidad EAFIT

Semester 2026-1

Sets Defined by Induction

Sets Defined by Induction

Example

We inductively define the set of natural numbers Nat.

Usual definition.

(i) $\mathrm{zero} : \mathrm{Nat}$.

(ii) If $n \in \mathrm{Nat}$ then $\mathrm{succ}\ n : \mathrm{Nat}$.

Definition using inference rules.

$$\frac{}{\mathrm{zero} : \mathrm{Nat}} \qquad \frac{n : \mathrm{Nat}}{\mathrm{succ}\ n : \mathrm{Nat}}$$

Sets Defined by Induction

Example

We inductively define the set of natural numbers Nat.

Usual definition.

(i) $\mathrm{zero} : \mathrm{Nat}$.

(ii) If $n \in \mathrm{Nat}$ then $\mathrm{succ}\ n : \mathrm{Nat}$.

Definition using inference rules.

$$\frac{}{\mathrm{zero} : \mathrm{Nat}} \qquad \frac{n : \mathrm{Nat}}{\mathrm{succ}\ n : \mathrm{Nat}}$$

Remark: In both definitions the fact that Nat is the smallest set generated by the clauses/rules is not stated explicitly.

Structural Induction

Structural Induction

Example

Let P be a unary property on Nat. To prove that $P\ n$ for all $n : \mathrm{Nat}$, we can use **structural induction** for P.

$$\dfrac{\begin{array}{c}[P\ n] \\ \vdots \\ P\ \mathrm{zero} \qquad P\ (\mathrm{succ}\ n)\end{array}}{P\ n}\ \text{(structural induction)}$$

Structural Induction

Example

Let A be a set. We inductively define the lists of elements of A, denoted $\text{List } A$, by the following inference rules:

$$\frac{}{\text{nil} : \text{List } A} \qquad \frac{x : A \qquad xs : \text{List} A}{\text{cons } x \; xs : \text{List} A}$$

Structural Induction

Example

Let A be a set. We inductively define the lists of elements of A, denoted $\text{List } A$, by the following inference rules:

$$\frac{}{\text{nil} : \text{List } A} \qquad \frac{x : A \qquad xs : \text{List} A}{\text{cons } x \; xs : \text{List} A}$$

Let A be a set and let P be a unary property on $\text{List } A$. To prove that $P \; xs$ for all $xs : \text{List } A$, we can use **structural induction** for P and $\text{List } A$.

$$\begin{array}{c} [P \; xs] \\ \vdots \\ \dfrac{P \text{ nil} \qquad P \; (\text{cons } x \; xs)}{P \; xs} \end{array} \text{(structural induction)}$$

Computation Induction

Computation Induction

Example

We recursively define the division by 2 (rounded downwards) function.

$$
\begin{aligned}
&\lfloor\, \mathrm{div2} \,\rfloor : \mathrm{Nat} \to \mathrm{Nat} \\
&\lfloor\, \mathrm{div2} \,\rfloor\ \mathrm{zero} &&= \mathrm{zero} \\
&\lfloor\, \mathrm{div2} \,\rfloor\ (\mathrm{succ}\ \mathrm{zero}) &&= \mathrm{zero} \\
&\lfloor\, \mathrm{div2} \,\rfloor\ (\mathrm{succ}\ (\mathrm{succ}\ n)) &&= \mathrm{succ}\ (\lfloor\, \mathrm{div2} \,\rfloor\ n)
\end{aligned}
$$

Computation Induction

Example

We recursively define the division by 2 (rounded downwards) function.

$$\begin{aligned}
&\lfloor\,\mathrm{div2}\,\rfloor : \mathrm{Nat} \to \mathrm{Nat} \\
&\lfloor\,\mathrm{div2}\,\rfloor\ \mathrm{zero} &&= \mathrm{zero} \\
&\lfloor\,\mathrm{div2}\,\rfloor\ (\mathrm{succ}\ \mathrm{zero}) &&= \mathrm{zero} \\
&\lfloor\,\mathrm{div2}\,\rfloor\ (\mathrm{succ}\ (\mathrm{succ}\ n)) &&= \mathrm{succ}\ (\lfloor\,\mathrm{div2}\,\rfloor\ n)
\end{aligned}$$

Let P be a unary property on Nat and $\lfloor\,\mathrm{div2}\,\rfloor$. To prove that $P\ n$ for all $n : \mathrm{Nat}$, we can use **computation induction** for P and $\lfloor\,\mathrm{div2}\,\rfloor$.

$$\frac{\begin{array}{c}[P\ n] \\ \vdots \\ P\ \mathrm{zero} \qquad P\ (\mathrm{succ}\ \mathrm{zero}) \qquad P\ (\mathrm{succ}\ (\mathrm{succ}\ n))\end{array}}{P\ n}\ \text{(computation induction)}$$

Computation Induction

Description

Let $f : \sigma \to \tau$ be a terminating function on the inductive sets σ and τ and let P be a unary property on σ and f. To prove that $P\ x$ for all $x : \sigma$ by induction on the computation of f it is necessary (p. 6):

(i) proving that $P\ e$ for every non-recursive defining equation $f\ e = e'$, and

(ii) proving that

$$P\ \sigma_1 \wedge \cdots \wedge P\ \sigma_n \to P\ e$$

for every recursive defining equation

$$f\ e = \ldots f\ \sigma_1 \ldots f\ \sigma_n$$

where $f\ \sigma_1$, ..., $f\ \sigma_n$ are all the recursive calls.

Computation Induction[†]

Description

Let $f : \sigma \to \tau$ be a terminating function on the inductive sets σ and τ and let P be a unary property on σ and f. To prove that $P\ x$ for all $x : \sigma$ by induction on the computation of f it is necessary (p. 6):

(i) proving that $P\ e$ for every non-recursive defining equation $f\ e = e'$, and

(ii) proving that

$$P\ \sigma_1 \wedge \cdots \wedge P\ \sigma_n \to P\ e$$

for every recursive defining equation

$$f\ e = \ldots f\ \sigma_1 \ldots f\ \sigma_n$$

where $f\ \sigma_1$, ..., $f\ \sigma_n$ are all the recursive calls.

[†]This induction schema is also called "recursion induction". See, e.g. (Nipkow, Paulson and Wenzel [2002] 2026).

Rule Induction

Rule Induction

Example

We inductively define the inductive relation Even on natural numbers by the following rules:

$$\frac{}{\mathrm{zero} : \mathrm{Even}} \qquad \frac{n : \mathrm{Even}}{\mathrm{succ}\ (\mathrm{succ}\ n) : \mathrm{Even}}$$

Rule Induction

Example

We inductively define the inductive relation Even on natural numbers by the following rules:

$$\frac{}{\mathrm{zero} : \mathrm{Even}} \qquad \frac{n : \mathrm{Even}}{\mathrm{succ}\ (\mathrm{succ}\ n) : \mathrm{Even}}$$

Let P be a unary property on Nat. To prove that $\mathrm{Even}\ n \Rightarrow P\ n$, we can use **rule induction** for P and Even.

$$\frac{\mathrm{Even}\ n \qquad P\ \mathrm{zero} \qquad \begin{array}{c}[\mathrm{Even}\ n, P\ n] \\ \vdots \\ P\ (\mathrm{succ}\ (\mathrm{succ}\ n))\end{array}}{P\ n}\ \text{(rule induction)}$$

Rule Inversion

Example

Rule inversion goes in the opposite direction to rule induction. For example, given $\text{Even}\ n$ we can use rule inversion for asking which rules could have been used to derive $\text{Even}\ n$ and what constraints these rules impose on n.

References

References

Tobias Nipkow, Lawrence C. Paulson and Markus Wenzel (2002). Isabelle/HOL. A Proof Assistant for Higher-Order Logic. Vol. 2283. Lecture Notes in Computer Science. Springer. DOI: 10.1007/3-540-45949-9 (cit. on p. 19).

Tobias Nipkow, Lawrence C. Paulson and Markus Wenzel [2002] (18th Jan. 2026). Isabelle/HOL. A Proof Assistant for Higher-Order Logic. Updated version of (Nipkow, Paulson and Wenzel 2002) matching changes in Isabelle/HOL. URL: https://isabelle.in.tum.de/dist/Isabelle2025-2/doc/tutorial.pdf (cit. on p. 13).