

Definicje indukcyjne w Coqu

17 marca 2009

Plan

- Zasady konstrukcji definicji indukcyjnych
- Destrukcja - operator `match`
- Rekursja - operator `fix`
- Sprawy praktyczne

Składnia formalna

Assuming Γ_I is $[I_1 : A_1; \dots; I_k : A_k]$, and Γ_C is $[c_1 : C_1; \dots; c_n : C_n]$, the general typing rules are, for $1 \leq j \leq k$ and $1 \leq i \leq n$:

$$\frac{\text{Ind}(\Gamma)(\Gamma_I := \Gamma_C) \in E}{(I_j : A_j) \in E}$$

$$\frac{\text{Ind}(\Gamma)(\Gamma_I := \Gamma_C) \in E}{(c_i : C_i) \in E}$$

Parametry i zwykłe argumenty

An inductive definition $\text{Ind}(\Gamma)(\Gamma_I := \Gamma_C)$ admits r inductive parameters if each type of constructors $(c : C)$ in Γ_C is such that

$$C \equiv \forall p_1 : P_1, \dots, \forall p_r : P_r, \forall a_1 : A_1, \dots, \forall a_n : A_n, (I p_1 \dots p_r t_1 \dots t_q)$$

with I one of the inductive definitions in Γ_I . We say that q is the number of real arguments of the constructor c .

We use the following notation:

$$\text{Ind}(\Gamma)[r](\Gamma_I := \Gamma_C)$$

and

$$\Gamma_P = [p_1 : P_1; \dots; p_r : P_r]$$

The latter will be called the *context of parameters*.

The List definition has 1 parameter:

$$\text{Ind()}(\text{List} : \text{Set} \rightarrow \text{Set} := \text{nil} : (\forall A : \text{Set}, \text{List } A), \\ \text{cons} : (\forall A : \text{Set}, A \rightarrow \text{List } A \rightarrow \text{List } A))$$

This is also the case for this more complex definition where there is a recursive argument on a different instance of List:

$$\text{Ind()}(\text{List} : \text{Set} \rightarrow \text{Set} := \text{nil} : (\forall A : \text{Set}, \text{List } A), \\ \text{cons} : (\forall A : \text{Set}, A \rightarrow \text{List } (A \rightarrow A) \rightarrow \text{List } A))$$

But the following definition has 0 parameters:

$$\text{Ind()}(\text{List} : \text{Set} \rightarrow \text{Set} := \text{nil} : (\forall A : \text{Set}, \text{List } A), \\ \text{cons} : (\forall A : \text{Set}, A \rightarrow \text{List } A \rightarrow \text{List } (A * A)))$$

Zasady budowy definicji indukcyjnych

A type is an *arity of sort* s if it converts to the sort s or to a product $\forall x : T, U$ with U an arity of sort s .

To put it simply: $\forall \overrightarrow{x : T}, s$.

For instance $A \rightarrow \text{Set}$ or $\forall A : \text{Prop}, A \rightarrow \text{Prop}$ are arities of sort respectively Set and Prop .

A *type of constructor of* I is either a term $(I t_1 \dots t_n)$ or $\forall x : T, C$ with C recursively a *type of constructor of* I .

To put it simply: $\forall \overrightarrow{x : T}, (I t_1 \dots t_n)$

Zasady budowy definicji indukcyjnych (c.d.)

The type of constructor C will be said to *satisfy the positivity condition* for a constant I in the following cases:

- $C = (I t_1 \dots t_n)$ and I does not occur free in any t_i
- $C = \forall x : T, V$ and I occurs only strictly positively in T and the type V satisfies the positivity condition for I

To put it simply: $C = \forall x : \overrightarrow{T}, (I t_1 \dots t_n)$, where I occurs only strictly positively in every T

Zasady budowy definicji indukcyjnych (c.d.)

The constant I occurs strictly positively in T in the following cases:

- I does not occur in T
- T converts to $(I t_1 \dots t_n)$ and I does not occur in any of t_i
- T converts to $\forall x : U, V$ and I does not occur in type U but occurs strictly positively in type V

- T converts to $(I' a_1 \dots a_m t_1 \dots t_p)$ where I' is the name of an inductive declaration of the form

$$\text{Ind}(\Gamma)[m](I : A := c_1 : \forall p_1 : P_1, \dots \forall p_m : P_m, C_1; \dots ; \\ c_n : \forall p_1 : P_1, \dots \forall p_m : P_m, C_n)$$

(in particular, it is not mutually defined and it has m parameters) and I does not occur in any of the t_i , and the (instantiated) types of constructor $C_i\{p_j/a_j\}_{j=1\dots m}$ of I' satisfy the nested positivity condition for I

Zasady budowy definicji indukcyjnych (c.d.)

The type of constructor T of I' satisfies the nested positivity condition for a constant I in the following cases:

- $T = (I' b_1 \dots b_m u_1 \dots u_p)$, I' is an inductive definition with m parameters and I does not occur in any u_i
- $T = \forall x : U, V$ and I occurs only strictly positively in U and the type V satisfies the nested positivity condition for I

Zasady budowy definicji indukcyjnych (c.d.)

Let E be an environment and $\Gamma, \Gamma_P, \Gamma_I, \Gamma_C$ are contexts such that Γ_I is $[I_1 : \forall \Gamma_P, A_1; \dots; I_k : \forall \Gamma_P, A_k]$ and Γ_C is $[c_1 : \forall \Gamma_P, C_1; \dots; c_n : \forall \Gamma_P, C_n]$.

$$\frac{(E[\Gamma; \Gamma_P] \vdash A_j : s'_j)_{j=1 \dots k} \quad (E[\Gamma; \Gamma_I; \Gamma_P] \vdash C_i : s_{q_i})_{i=1 \dots n}}{\mathcal{WF}(E; \text{Ind}(\Gamma)[p](\Gamma_I := \Gamma_C))[\Gamma]}$$

provided that the following side conditions hold:

- $k > 0$ and all of I_j and c_i are distinct names for $j = 1 \dots k$ and $i = 1 \dots n$,
- p is the number of parameters of $\text{Ind}(\Gamma)(\Gamma_I := \Gamma_C)$ and Γ_P is the context of parameters,
- for $j = 1 \dots k$ we have that A_j is an arity of sort s_j and $I_j \notin \Gamma \cup E$,
- for $i = 1 \dots n$ we have that C_i is a type of constructor of I_{q_i} which satisfies the positivity condition for $I_1 \dots I_k$ and $c_i \notin \Gamma \cup E$.

Destrukcja - match

Simple form:

$$\text{match } m \text{ with } (c_1 \ x_{11} \ \dots \ x_{1p_1}) \Rightarrow f_1 \mid \dots \mid (c_n \ x_{n1} \ \dots \ x_{np_n}) \Rightarrow f_n \text{ end}$$

Full form:

$$\text{match } m \text{ as } x \text{ in } I _ a \text{ return } (P \ a \ x) \text{ with} \\ (c_1 \ x_{11} \ \dots \ x_{1p_1}) \Rightarrow f_1 \mid \dots \mid (c_n \ x_{n1} \ \dots \ x_{np_n}) \Rightarrow f_n \text{ end}$$

For the purpose of presenting the inference rules, we use a more compact notation :

$$\text{case}(m, (\lambda a x, P), \lambda x_{11} \ \dots \ x_{1p_1}, f_1 \mid \dots \mid \lambda x_{n1} \ \dots \ x_{np_n}, f_n)$$

Destrukcja - match (c.d)

Type of branches. Let c be a term of type C , we assume C is a type of constructor for an inductive definition I . Let P be a term that represents the property to be proved. We assume r is the number of parameters. We define a new type $\{c : C\}^P$ which represents the type of the branch corresponding to the $c : C$ constructor.

$$\begin{aligned}\{c : (I_i p_1 \dots p_r t_1 \dots t_p)\}^P &\equiv (P t_1 \dots t_p c) \\ \{c : \forall x : T, C\}^P &\equiv \forall x : T, \{(c x) : C\}^P\end{aligned}$$

We write $\{c\}^P$ for $\{c : C\}^P$ with C the type of c .

Examples. For `List_A` the type of P will be `List_A → s` for $s \in \mathcal{S}$.
 $\{(\text{cons } A)\}^P \equiv \forall a : A, \forall l : \text{List_A}, (P (\text{cons } A a l)).$

Destrukcja - match (c.d)

For Length_A , the type of P will be

$\forall l : \text{List_A}, \forall n : \text{nat}, (\text{Length_A } l \ n) \rightarrow \text{Prop}$ and the expression $\{(\text{Lcons } A)\}^P$ is defined as:

$\forall a : A, \forall l : \text{List_A}, \forall n : \text{nat}, \forall h :$

$(\text{Length_A } l \ n), (P (\text{cons } A \ a \ l) (\text{S } n) (\text{Lcons } A \ a \ l \ n \ l)).$

If P does not depend on its third argument, we find the more natural expression:

$\forall a : A, \forall l : \text{List_A}, \forall n : \text{nat}, (\text{Length_A } l \ n) \rightarrow (P (\text{cons } A \ a \ l) (\text{S } n)).$

Typing rule.

Our very general destructor for inductive definition enjoys the following typing rule

$$\frac{\begin{array}{c} E[\Gamma] \vdash c : (I \ q_1 \dots q_r \ t_1 \dots t_s) \\ E[\Gamma] \vdash P : B \\ \text{compat}(I \ q_1 \dots q_r)B \\ (E[\Gamma] \vdash f_i : \{(c_{p_i} \ q_1 \dots q_r)\}^P)_{i=1\dots l} \end{array}}{E[\Gamma] \vdash \text{case}(c, P, f_1 | \dots | f_l) : (P \ t_1 \dots t_s \ c)}$$

provided I is an inductive type in a declaration $\text{Ind}(\Delta)[r](\Gamma_I := \Gamma_C)$ with $\Gamma_C = [c_1 : C_1; \dots; c_n : C_n]$ and $c_{p_1} \dots c_{p_l}$ are the only constructors of I .

Destrukcja - match (redukcja)

A ι -reduction has the following form

$$\text{case}((c_{p_i} \ q_1 \dots q_r \ a_1 \dots a_m), P, f_1 | \dots | f_n) \triangleright_{\iota} (f_i \ a_1 \dots a_m)$$

with c_{p_i} the i -th constructor of the inductive type I with r parameters.

The basic concrete syntax for a recursive set of mutually recursive declarations is (with Γ_i contexts) :

$$\text{fix } f_1(\Gamma_1) : A_1 := t_1 \text{ with } \dots \text{ with } f_n(\Gamma_n) : A_n := t_n \text{ for } f_i$$

In the inference rules, we represent such a term by

$$\text{Fix } f_i \{ f_1 : A'_1 := t'_1 \dots f_n : A'_n := t'_n \}$$

with t'_i (resp. A'_i) representing the term t_i abstracted (resp. generalized) with respect to the bindings in the context Γ_i , namely $t'_i = \lambda\Gamma_i, t_i$ and $A'_i = \forall\Gamma_i, A_i$.

Typing rule:

$$\frac{(E[\Gamma] \vdash A_i : s_i)_{i=1\dots n} \quad (E[\Gamma, f_1 : A_1, \dots, f_n : A_n] \vdash t_i : A_i)_{i=1\dots n}}{E[\Gamma] \vdash \text{Fix } f_i \{f_1 : A_1 := t_1 \dots f_n : A_n := t_n\} : A_i}$$

But we need to specify the argument on which we recurse.

For doing this the syntax of fixpoints is extended and becomes

$$\text{Fix } f_i \{f_1/k_1 : A_1 := t_1 \dots f_n/k_n : A_n := t_n\}$$

where k_i are positive integers. Each A_i should be a type starting with at least k_i products $\forall y_1 : B_1, \dots \forall y_{k_i} : B_{k_i}, A'_i$ and B_{k_i} being an instance of an inductive definition.

Now in the definition t_i , if f_j occurs then it should be applied to at least k_j arguments and the k_j -th argument should be syntactically recognized as *structurally smaller* than y_{k_i}

Rekursja - fix (c.d)

Given a variable y of an inductive type from a declaration $\text{Ind}(\Gamma)[r](\Gamma_I := \Gamma_C)$ where Γ_I is $[I_1 : A_1; \dots; I_k : A_k]$, and Γ_C is $[c_1 : C_1; \dots; c_n : C_n]$. The terms structurally smaller than y are:

- $(t \ u), \lambda x : u, t$ when t is structurally smaller than y .
- $\text{case}(c, P, f_1 \dots f_n)$ when each f_i is structurally smaller than y .
If c is y or is structurally smaller than y , its type is an inductive definition I_p part of the inductive declaration corresponding to y .

Each f_i corresponds to a type of constructor

$C_q \equiv \forall p_1 : P_1, \dots, \forall p_r : P_r, \forall y_1 : B_1, \dots, \forall y_k : B_k, (I \ a_1 \dots a_k)$ and can consequently be written $\lambda y_1 : B'_1, \dots, \lambda y_k : B'_k, g_i$. (B'_i is obtained from B_i by substituting parameters variables) the variables y_j occurring in g_i corresponding to recursive arguments B_i (the ones in which one of the I_l occurs) are structurally smaller than y .

Rekursja - fix (redukcja)

Let F be the set of declarations: $f_1/k_1 : A_1 := t_1 \dots f_n/k_n : A_n := t_n$.

The reduction for fixpoints is:

$$(\text{Fix } f_i\{F\} a_1 \dots a_{k_i}) \triangleright_{\iota} t_i\{(f_k/\text{Fix } f_k\{F\})_{k=1\dots n}\} a_1 \dots a_{k_i}$$

when a_{k_i} starts with a constructor.

This last restriction is needed in order to keep strong normalization and corresponds to the reduction for primitive recursive operators.

- generowanie lematów indukcyjnych
- taktyki wprowadzania - `constructor`, `left`, `right`, `split`, `exists`
- taktyki destrukcji - `destruct`, `case`
- taktyki indukcyjne - `induction`, `elim`
- równość i typy indukcyjne - `discriminate`, `injection`, `simplify_eq`
- zaawansowana destrukcja - `inversion`