

# Logika i teoria typów

Wykład 10

21 grudnia 2020

# Siła wyrazu polimorfizmu

## Twierdzenie (Girard):

Funkcja jest definiowalna w systemie  $\mathbf{F}$  wtedy i tylko wtedy, gdy jest dowodliwie rekurencyjna w arytmetyce drugiego rzędu.

# Siła wyrazu polimorfizmu

## Twierdzenie (Girard):

Funkcja jest definiowalna w systemie  $\mathbf{F}$  wtedy i tylko wtedy, gdy jest dowodliwie rekurencyjna w arytmetyce drugiego rzędu.

## Idea dowodu (w uproszczeniu):

( $\Leftarrow$ ) Dowód formuły  $\forall n \exists m P(n, m)$  wyciera się do termu typu  $\omega \rightarrow \omega$ .

( $\Rightarrow$ ) Dowód silnej normalizacji dla ustalonej funkcji można przeprowadzić w arytmetyce drugiego rzędu.

# Siła wyrazu polimorfizmu

## Twierdzenie (Girard):

Funkcja jest definiowalna w systemie  $\mathbf{F}$  wtedy i tylko wtedy, gdy jest dowodliwie rekurencyjna w arytmetyce drugiego rzędu.

# Siła wyrazu polimorfizmu

## Twierdzenie (Girard):

Funkcja jest definiowalna w systemie  $\mathbf{F}$  wtedy i tylko wtedy, gdy jest dowodliwie rekurencyjna w arytmetyce drugiego rzędu.

**Wniosek:** Silna normalizacja dla systemu  $\mathbf{F}$  jest niezależna od arytmetyki drugiego rzędu.

# Siła wyrazu polimorfizmu

## Twierdzenie (Girard):

Funkcja jest definiowalna w systemie  $\mathbf{F}$  wtedy i tylko wtedy, gdy jest dowodliwie rekurencyjna w arytmetyce drugiego rzędu.

**Wniosek:** Silna normalizacja dla systemu  $\mathbf{F}$  jest niezależna od arytmetyki drugiego rzędu.

**Idea dowodu:** Silna normalizacja implikuje całkowitość funkcji uniwersalnej  $f(n, m) = f_n(m)$ , gdzie  $f_n$  to wszystkie funkcje definiowalne w  $\mathbf{F}$ . Ale funkcja  $f$  nie jest definiowalna, bo jeśli  $f(n, n) + 1 = f_k(n)$ , to  $f(k, k) + 1 = f(k, k)$ .

## Recursive and inductive types

# Recursive types (Curry style)

**Principal idea:** Type  $\mu p. \tau(p)$  is a solution of  $X = \tau(X)$ .  
For example, one identifies

$$\alpha = \mu p. q \rightarrow p \quad \text{with} \quad q \rightarrow \alpha.$$

Therefore

$$y : q, \quad x : \alpha \quad \vdash \quad xyy \dots y : \alpha$$



# Recursive types (Curry style)

**Principal idea:** Type  $\mu p. \tau(p)$  is a solution of  $X = \tau(X)$ .  
For example, one identifies

$$\alpha = \mu p. q \rightarrow p \quad \text{with} \quad q \rightarrow \alpha.$$

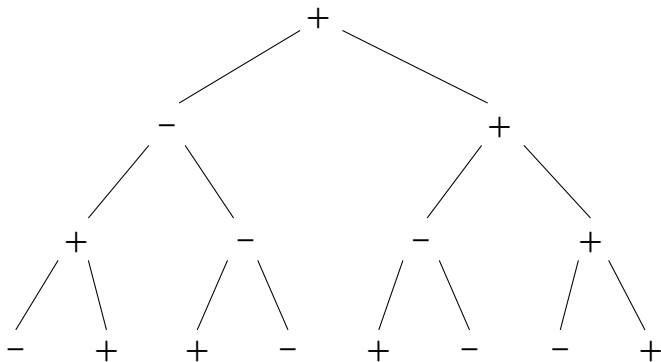
Therefore

$$y : q, \quad x : \alpha \quad \vdash \quad xyy \dots y : \alpha$$

**Dangerous trick:** Let  $\iota = \mu p. p \rightarrow p$ .

Then every closed term has type  $\iota$  (no more SN).

## Positive and Negative



**Monotoniczność:** Jeśli  $p$  występuje w  $\varphi$  tylko pozytywnie, to

$$p \rightarrow p' \vdash \varphi(p) \rightarrow \varphi(p').$$

# Positivity

- ▶  $Pos(p) = \{p\}$ ;
- ▶  $Neg(p) = \emptyset$ ;
- ▶  $Pos(\tau \rightarrow \sigma) = Neg(\tau) \cup Pos(\sigma)$ ;
- ▶  $Neg(\tau \rightarrow \sigma) = Pos(\tau) \cup Neg(\sigma)$ ;
- ▶  $Pos(\mu p \tau) = Pos(\tau) - \{p\}$ ;
- ▶  $Neg(\mu p \tau) = Neg(\tau) - \{p\}$

Type  $\mu p \tau$  is only permitted when  $p \notin Neg(\tau)$ .

## Positive recursive types (Curry style)

- ▶ Strong normalization holds (even in PA).

## Positive recursive types (Curry style)

- ▶ Strong normalization holds (even in PA).  
But add any negative fixpoint and SN is lost.

## Positive recursive types (Curry style)

- ▶ Strong normalization holds (even in PA).  
But add any negative fixpoint and SN is lost.
- ▶ Incomparable with **F** wrt typability:
  - ▶  $\lambda x. xx$  untypable.  
(Needs a negative fixpoint  $\tau = \tau \rightarrow \dots$ )

## Positive recursive types (Curry style)

- ▶ Strong normalization holds (even in PA).  
But add any negative fixpoint and SN is lost.
- ▶ Incomparable with **F** wrt typability:
  - ▶  $\lambda x. xx$  untypable.  
(Needs a negative fixpoint  $\tau = \tau \rightarrow \dots$ )
  - ▶ **22K** typable.  
(Take  $\alpha = \mu p. q \rightarrow p$ . Then  $\mathbf{K} : \alpha \rightarrow \alpha$ .)

## Positive recursive types

- ▶ Decidable inhabitation (reduction to simple types).



# Positive recursive types

- ▶ Decidable inhabitation  
(reduction to simple types).
- ▶ Decidable type assignment  
(unification with infinite regular trees).

# Positive recursive types

- ▶ Decidable inhabitation (reduction to simple types).
- ▶ Decidable type assignment (unification with infinite regular trees).
- ▶ Uwaga: ostrożnie z drzewami: typy  $\mu p. p \rightarrow q$  i  $\mu p. (p \rightarrow q) \rightarrow q$  rozwijają się w to samo drzewo.

# Typy rekurencyjne „w sensie Churcha”?

Oznaczenie  $\mu = \mu p. \sigma(p)$ .

Potrzebujemy wprowadzania i eliminacji.

To pierwsze to operator  $in : \sigma(\mu) \rightarrow \mu$ :

$$\frac{\Gamma \vdash M : \sigma(\mu)}{\Gamma \vdash in(M) : \mu}$$

# Jaki eliminator jest odpowiedni?

**Pierwszy pomysł:**

Oznaczenie  $\mu = \mu p. \sigma(p)$ . Mamy  $in : \sigma(\mu) \rightarrow \mu$

Wprowadzamy operator

$$out : \mu \rightarrow \sigma(\mu)$$

i redukcję  $out(in M) \Rightarrow M$ .

To jest umiarkowanie ciekawe. (Niewiele można zdefiniować.)

# Jaki eliminator jest odpowiedni?

**Per Martin-Löf:** Reguły wprowadzania dla spójnika logicznego (typu danych) są pierwotne. Reguła eliminacji i zasada beta-redukcji są wtórne.

Reguły wprowadzania określają jakie są kanoniczne formy elementów danego typu. Na przykład, dla alternatywy czyli koproduktu mamy dwie postaci kanoniczne:

$$\frac{\Gamma \vdash M : \alpha}{\Gamma \vdash \text{inl}(M) : \alpha \vee \beta}$$

$$\frac{\Gamma \vdash M : \beta}{\Gamma \vdash \text{inr}(M) : \alpha \vee \beta}$$

# Alternatywa/koprodukt

Dwie postaci kanoniczne = dwie reguły wprowadzania:

$$\frac{\Gamma \vdash M : \alpha}{\Gamma \vdash \text{inl}(M) : \alpha \vee \beta}$$

$$\frac{\Gamma \vdash M : \beta}{\Gamma \vdash \text{inr}(M) : \alpha \vee \beta}$$

Reguła eliminacji podaje sposób użycia elementów kanonicznych. Ma po jednej przesłance niegłówniej dla każdej postaci kanonicznej. Dla alternatywy dwie.

$$\frac{\Gamma \vdash M : \alpha \vee \beta \quad \Gamma, u:\alpha \vdash R : \tau \quad \Gamma, v:\beta \vdash Q : \tau}{\Gamma \vdash \text{case } M \text{ of } [u]R \text{ or } [v]Q : \tau}$$

# Alternatywa/koprodukt

$$\frac{\Gamma \vdash M : \alpha}{\Gamma \vdash \text{inl}(M) : \alpha \vee \beta}$$

$$\frac{\Gamma \vdash M : \beta}{\Gamma \vdash \text{inr}(M) : \alpha \vee \beta}$$

$$\frac{\Gamma \vdash M : \alpha \vee \beta \quad \Gamma, u:\alpha \vdash R : \tau \quad \Gamma, v:\beta \vdash Q : \tau}{\Gamma \vdash \text{case } M \text{ of } [u]R \text{ or } [v]Q : \tau}$$

Każda przesłanka niegłówna to sposób użycia innej postaci kanonicznej. To determinuje też zasadę beta-redukcji:

$$\text{case inl}(P^\alpha) \text{ of } [x^\alpha]M^\rho, [y^\beta]N^\rho \implies M[x := P] : \rho$$

$$\text{case inr}(Q^\beta) \text{ of } [x^\alpha]M^\rho, [y^\beta]N^\rho \implies N[y := Q] : \rho$$

# Alternatywa/koprodukt

$$\frac{\Gamma \vdash M : \alpha}{\Gamma \vdash \text{inl}(M) : \alpha \oplus \beta}$$

$$\frac{\Gamma \vdash M : \beta}{\Gamma \vdash \text{inr}(M) : \alpha \oplus \beta}$$

$$\frac{\Gamma \vdash M : \alpha \oplus \beta \quad \Gamma, u:\alpha \vdash R : \tau \quad \Gamma, v:\beta \vdash Q : \tau}{\Gamma \vdash \text{case } M \text{ of } [u]R \text{ or } [v]Q : \tau}$$

Każda przesłanka niegłówna to sposób użycia innej postaci kanonicznej. To determinuje też zasadę beta-redukcji:

$$\text{case inl}(P^\alpha) \text{ of } [x^\alpha]M^\rho, [y^\beta]N^\rho \implies M[x := P] : \rho$$

$$\text{case inr}(Q^\beta) \text{ of } [x^\alpha]M^\rho, [y^\beta]N^\rho \implies N[y := Q] : \rho$$



## To są niewłaściwe eliminacje

Dla koniunkcji dostaniemy coś takiego:

$$\frac{\Gamma \vdash M : A \wedge B \quad \Gamma, x : A, y : B \vdash N : C}{\Gamma \vdash \text{let } M \text{ be } \langle x, y \rangle \text{ in } N : C}$$

## To są niewłaściwe eliminacje

Dla koniunkcji dostaniemy coś takiego:

$$\frac{\Gamma \vdash M : A \wedge B \quad \Gamma, x : A, y : B \vdash N : C}{\Gamma \vdash \text{let } M \text{ be } \langle x, y \rangle \text{ in } N : C}$$

Beta-redukcja:

$$\text{let } \langle P, Q \rangle \text{ be } \langle x, y \rangle \text{ in } N \Rightarrow N[x := P][y := Q]$$

## To są niewłaściwe eliminacje

Dla koniunkcji dostaniemy coś takiego:

$$\frac{\Gamma \vdash M : A \wedge B \quad \Gamma, x : A, y : B \vdash N : C}{\Gamma \vdash \text{let } M \text{ be } \langle x, y \rangle \text{ in } N : C}$$

Beta-redukcja:

$$\text{let } \langle P, Q \rangle \text{ be } \langle x, y \rangle \text{ in } N \Rightarrow N[x := P][y := Q]$$

**Ćwiczenie:** jaka koniunkcja jest w Coqu?

# Studium przypadku: liczby naturalne

Typ **1** ma jeden element  $\bullet : \mathbf{1}$ .

$$\mathbf{int} \approx \mathbf{1} \oplus \mathbf{1} \oplus \mathbf{1} \oplus \mathbf{1} \oplus \mathbf{1} \oplus \dots$$

czyli

$$\mathbf{int} \approx \mathbf{1} \oplus \mathbf{int},$$

czyli

$$\mathbf{int} = \mu p. \mathbf{1} \oplus p,$$

Zero to element  $\mathit{in}(\mathit{inl}(\bullet))$ .

Liczba  $n + 1$  to element  $\mathit{in}(\mathit{inr}(n))$ .

## Przykład: liczby naturalne

Niech  $\mathbf{int} = \mu p. \mathbf{1} \oplus p$  oraz  $in : \mathbf{1} \oplus \mathbf{int} \rightarrow \mathbf{int}$ .

Zero to element  $in(inl(\bullet))$ .

Liczba  $n + 1$  to element  $in(inr(n))$ .

Funkcja następnika:  $succ = \lambda n^{\mathbf{int}}. in(inr(n))$

Uwaga: włożenie  $in : \mathbf{1} \oplus \mathbf{int} \rightarrow \mathbf{int}$ , to moralnie to samo, co para  $\langle 0, s \rangle : \mathbf{int} \times (\mathbf{int} \rightarrow \mathbf{int})$ .

Włózenie  $in : \mathbf{1} \oplus \mathbf{int} \rightarrow \mathbf{int}$ , to moralnie  
to samo, co para  $\langle 0, s \rangle : \mathbf{int} \times (\mathbf{int} \rightarrow \mathbf{int})$ .

Włożenie  $in : \mathbf{1} \oplus \mathbf{int} \rightarrow \mathbf{int}$ , to moralnie  
to samo, co para  $\langle 0, s \rangle : \mathbf{int} \times (\mathbf{int} \rightarrow \mathbf{int})$ .

Inductive nat : Set := 0 : nat | S : nat -> nat.

# Trochę inaczej to samo

Zamiast jednej reguły wprowadzania

$$\frac{\Gamma \vdash M : \mathbf{1} \oplus \mathbf{int}}{\Gamma \vdash \mathit{in}(M) : \mathbf{int}}$$

dwie reguły dla dwóch przypadków:

$$\frac{\Gamma \vdash \bullet : \mathbf{1}}{\Gamma \vdash \mathit{in}(\mathit{inl}(\bullet)) : \mathbf{int}}$$

$$\frac{\Gamma \vdash M : \mathbf{int}}{\Gamma \vdash \mathit{in}(\mathit{inr}(M)) : \mathbf{int}}$$

Albo trochę przyjaźniej:

$$\Gamma \vdash 0 : \mathbf{int}$$

$$\frac{\Gamma \vdash M : \mathbf{int}}{\Gamma \vdash \mathit{s}(M) : \mathbf{int}}$$



$$\text{int} = \mu p. 1 \oplus p$$

Jaki eliminator jest odpowiedni dla liczb naturalnych?

$$\mathbf{int} = \mu p. \mathbf{1} \oplus p$$

Jaki eliminator jest odpowiedni dla liczb naturalnych?

Reguła wprowadzania

$$\frac{\Gamma \vdash M : \mathbf{1} \oplus \mathbf{int}}{\Gamma \vdash \mathit{in}(M) : \mathbf{int}}$$

sugeruje taką (naiwną) eliminację:

$$\frac{\Gamma \vdash M : \mathbf{int} \quad \Gamma, x : \mathbf{1} \oplus \mathbf{int} \vdash R : \tau}{\Gamma \vdash \mathit{Elim} M \text{ with } [x]R : \tau}$$

# Naiwna eliminacja

$$\frac{\Gamma \vdash M : \mathbf{int} \quad \Gamma, x : \mathbf{1} \oplus \mathbf{int} \vdash R : \tau}{\Gamma \vdash \text{Elim } M \text{ with } [x]R : \tau}$$

Naiwna eliminacja rozbita na przypadki:

$$\frac{\Gamma \vdash M : \mathbf{int} \quad \Gamma \vdash P : \tau \quad \Gamma, v : \mathbf{int} \vdash Q : \tau}{\Gamma \vdash \text{Elim } M \text{ with } P \text{ or } [v]Q : \tau}$$

Z tego wychodzi taka beta-redukcja:

$$\text{Elim } 0 \text{ with } P \text{ or } [v]Q \Rightarrow P$$

$$\text{Elim } s(n) \text{ with } P \text{ or } [v]Q \Rightarrow Q[v := n]$$

$\text{int} = \mu p. \mathbf{1} \oplus p$

$$\frac{\Gamma \vdash M : \text{int} \quad \Gamma \vdash P : \tau \quad \Gamma, v:\text{int} \vdash Q : \tau}{\Gamma \vdash \text{Elim } M \text{ with } P \text{ or } [v]Q : \tau}$$

$$\text{Elim } s(n) \text{ with } P \text{ or } [v]Q \Rightarrow Q[v := n]$$

Sposób użycia postaci kanonicznej jest zdeterminowany przez sposób użycia jej „składowych” (cokolwiek to znaczy).

Liczba  $n$  jest składową swojego następnika  $s(n)$ . Zatem sposób użycia  $s(n)$  powinien zależeć od sposobu użycia  $n$ .

Ale jeśli  $n = s(k)$ , to sposób użycia  $n$  powinien zależeć od sposobu użycia  $k$ . Naiwna reguła na to nie pozwala.

Jaka eliminacja dla  $\mathbf{int} = \mu p. \mathbf{1} \oplus p$ ?

Do przesłanek reguły

$$\frac{\Gamma \vdash M : \mathbf{int} \quad \Gamma \vdash P : \tau \quad \Gamma, v : \mathbf{int} \vdash Q : \tau}{\Gamma \vdash \text{Elim } M \text{ with } P \text{ or } [v]Q : \tau}$$

trzeba dodać sposób eliminacji  $V : \tau$  dla  $v : \mathbf{int}$ :

$$\frac{\Gamma \vdash M : \mathbf{int} \quad \Gamma \vdash P : \tau \quad \Gamma, v : \mathbf{int}, V : \tau \vdash Q : \tau}{\Gamma \vdash \text{Elim } M \text{ with } P \text{ or } [v, V]Q : \tau}$$

Sposób użycia  $s(n)$  określamy znając sposób użycia  $V$  dla  $n$ :

$$\text{Elim } s(n) \text{ with } P \text{ or } [v, V]Q \Rightarrow Q[v := n, V := ?]$$

Jaka eliminacja dla  $\mathbf{int} = \mu p. \mathbf{1} \oplus p$ ?

$$\frac{\Gamma \vdash M : \mathbf{int} \quad \Gamma \vdash P : \tau \quad \Gamma, v:\mathbf{int}, V:\tau \vdash Q : \tau}{\Gamma \vdash \text{Elim } M \text{ with } P \text{ or } [v, V]Q : \tau}$$

Sposób użycia  $s(n)$  określamy znając sposób użycia  $V$  dla  $n$ :

$$\text{Elim } s(n) \text{ with } P \text{ or } [v, V]Q \Rightarrow Q[v := n, V := ?]$$

Jaka eliminacja dla  $\mathbf{int} = \mu p. \mathbf{1} \oplus p$ ?

$$\frac{\Gamma \vdash M : \mathbf{int} \quad \Gamma \vdash P : \tau \quad \Gamma, v:\mathbf{int}, V:\tau \vdash Q : \tau}{\Gamma \vdash \text{Elim } M \text{ with } P \text{ or } [v, V]Q : \tau}$$

Sposób użycia  $s(n)$  określamy znając sposób użycia  $V$  dla  $n$ :

Elim  $s(n)$  with  $P$  or  $[v, V]Q \Rightarrow$

$$Q[v := n, V := \text{Elim } n \text{ with } P \text{ or } [v, V]Q]$$

## User-friendly variant

$$\frac{\Gamma \vdash M : \mathbf{int} \quad \Gamma \vdash P : \tau \quad \Gamma, v:\mathbf{int}, V : \tau \vdash Q : \tau}{\Gamma \vdash \text{Elim } M \text{ with } P \text{ or } [v, V]Q : \tau}$$

Instead of  $\Gamma, v:\mathbf{int}, V : \tau \vdash Q : \tau$  and **Elim**  
take  $\Gamma \vdash Q : \mathbf{int} \rightarrow \tau \rightarrow \tau$  and *recursor*  $R_\tau$ :

$$\frac{\Gamma \vdash M : \mathbf{int} \quad \Gamma \vdash P : \tau \quad \Gamma \vdash Q : \mathbf{int} \rightarrow \tau \rightarrow \tau}{\Gamma \vdash R_\tau M P Q : \tau}$$



## Prostsza wersja rekursji: iteracja

$$\frac{\Gamma \vdash M : \text{int} \quad \Gamma \vdash P : \tau \quad \Gamma \vdash Q : \tau \rightarrow \tau}{\Gamma \vdash \text{It}_\tau MPQ : \tau}$$

$$\text{It}_\tau 0PQ \rightarrow P$$

$$\text{It}_\tau (sn)PQ \rightarrow Q(\text{It}_\tau nPQ)$$

Operator  $Q$  nie ma bezpośredniego dostępu do „zmiennnej sterującej”  $n$ .

# Ciekawostka historyczna

Typ rekursora

$$\mathbf{R}_\tau : \mathbf{int} \rightarrow \tau \rightarrow (\mathbf{int} \rightarrow \tau \rightarrow \tau) \rightarrow \tau$$

jest wytarciem schematu indukcji Peana w wersji z 1889 roku:

$$\forall x (\mathbf{int}(x) \rightarrow \tau(0) \rightarrow \forall y (\mathbf{int}(y) \rightarrow \tau(y) \rightarrow \tau(sy)) \rightarrow \tau(x)).$$

Typ iteratora

$$\mathbf{It}_\tau : \mathbf{int} \rightarrow \tau \rightarrow (\tau \rightarrow \tau) \rightarrow \tau$$

jest wytarciem schematu indukcji Peana w wersji z 1891 roku:

$$\forall x (\mathbf{int}(x) \rightarrow \tau(0) \rightarrow \forall y (\tau(y) \rightarrow \tau(sy)) \rightarrow \tau(x)).$$

# Jak to można uogólnić?

Reguła dla iteratora:

$$\frac{\Gamma \vdash M : \mathbf{int} \quad \Gamma \vdash P : \tau \quad \Gamma \vdash Q : \tau \rightarrow \tau}{\Gamma \vdash \mathbf{It}_\tau M P Q : \tau}$$

przepisana z użyciem koproduktu:

$$\frac{\Gamma \vdash M : \mathbf{int} \quad \Gamma \vdash D : \mathbf{1} \oplus \tau \rightarrow \tau}{\Gamma \vdash \mathbf{Elim}_\tau D M : \tau}$$

## Jak to można uogólnić?

Reguła dla iteratora:

$$\frac{\Gamma \vdash M : \mathbf{int} \quad \Gamma \vdash P : \tau \quad \Gamma \vdash Q : \tau \rightarrow \tau}{\Gamma \vdash \mathbf{It}_\tau M P Q : \tau}$$

przepisana z użyciem koproduktu:

$$\frac{\Gamma \vdash M : \mathbf{int} \quad \Gamma \vdash D : \mathbf{1} \oplus \tau \rightarrow \tau}{\Gamma \vdash \mathbf{Elim}_\tau D M : \tau}$$

Redukcja:

$$\mathbf{Elim}_\tau D (\mathbf{in} N^{\mathbf{1} \oplus \mathbf{int}}) \Rightarrow D(\mathbf{case} N \text{ of } [x^{\mathbf{1}}]\mathbf{inl}(x) \text{ or } [y^{\mathbf{int}}]\mathbf{inr}(\mathbf{Elim}_\tau D y))$$

# What is going on here?

$$\text{Elim}_\tau D (\text{in } N^{1 \oplus \text{int}}) \Rightarrow \\ D(\text{case } N \text{ of } [x^1]\text{inl}(x) \text{ or } [y^{\text{int}}]\text{inr}(\text{Elim}_\tau D y))$$

Given  $D : \mathbf{1} \oplus \tau \rightarrow \tau$ , we define by induction a function  
 $F = \text{Elim}_\tau D : \mathbf{int} \rightarrow \tau$ :

$$F(\text{in}(\text{inl}(\bullet))) = D(\text{inl}(\bullet)) \qquad F(\text{in}(\text{inr}(n))) = D(\text{inr}(F(n)))$$

We imagine that  $\mathbf{int} = \mathbf{1} \oplus \mathbf{1} \oplus \mathbf{1} \oplus \mathbf{1} \oplus \mathbf{1} \oplus \dots$

We want to define  $F(4)$  (number 4 is the red component)  
when  $F$  is defined for the first four components. Then:

$$F(4) = F(\text{in}(\text{inr } 3)) = D(\text{inr } F(3))$$

Mamy  $D : 1 \oplus \tau \rightarrow \tau$ , definiujemy  $F = \text{Elim}_\tau D : \mathbf{int} \rightarrow \tau$ .

To define  $F(4) = D(\text{inr } F(3))$  we use the defined part of  $F$ , applying it at the “right channel”:

$$\begin{array}{ccc}
 \mathbf{int} \approx 1 \oplus \mathbf{int} & & 1 \oplus 1 \oplus 1 \oplus 1 \oplus 1 \oplus \dots \\
 \downarrow \text{id} \quad \downarrow F & & \downarrow \text{id} \quad \quad \quad \downarrow F \\
 1 \oplus \tau & & 1 \quad \oplus \quad \tau \\
 \downarrow D & & \downarrow D \\
 \tau & & \tau
 \end{array}$$

$F : \mathbf{int} \rightarrow \tau$  is “lifted” to  $F^+ : 1 \oplus \mathbf{int} \rightarrow 1 \oplus \tau$

$$F^+(Z) = \text{case } Z \text{ of } [x^1]\text{inl}(x) \text{ or } [y^{\mathbf{int}}]\text{inr}(Fy)$$

$$F \circ \text{in} = D \circ F^+$$

# Podsumowanie

$$\frac{\Gamma \vdash M : \mathbf{int} \quad \Gamma \vdash D : \mathbf{1} \oplus \tau \rightarrow \tau}{\Gamma \vdash F(M) : \tau}$$

$$F(\mathit{in} N^{\mathbf{1} \oplus \mathbf{int}}) \Rightarrow D(\mathit{case} N \text{ of } [x^{\mathbf{1}}]\mathit{inl}(x) \text{ or } [y^{\mathbf{int}}]\mathit{inr}(Fy))$$

$$F^+(Z) = \mathit{case} Z \text{ of } [x^{\mathbf{1}}]\mathit{inl}(x) \text{ or } [y^{\mathbf{int}}]\mathit{inr}(Fy)$$

$$F(\mathit{in} N) \Rightarrow D(F^+(N))$$

Inny przykład: długość listy

$\text{lista} \approx \mathbf{1} \oplus (\text{int} \times \text{lista})$ , czyli:

$\text{lista} \approx T(\text{lista})$ , gdzie  $T(\alpha) = \mathbf{1} \oplus (\text{int} \times \alpha)$



## Inny przykład: długość listy

**lista**  $\approx \mathbf{1} \oplus (\mathbf{int} \times \mathbf{lista})$ , czyli:

**lista**  $\approx T(\mathbf{lista})$ , gdzie  $T(\alpha) = \mathbf{1} \oplus (\mathbf{int} \times \alpha)$

*suma* : **lista**  $\rightarrow$  **int**

*suma*(*nil*) = 0,      *suma*(*n* :: *l*) = *n* + *suma*(*l*)

## Inny przykład: długość listy

**lista**  $\approx \mathbf{1} \oplus (\mathbf{int} \times \mathbf{lista})$ , czyli:

**lista**  $\approx T(\mathbf{lista})$ , gdzie  $T(\alpha) = \mathbf{1} \oplus (\mathbf{int} \times \alpha)$

*suma* : **lista**  $\rightarrow$  **int**

$$\mathit{suma}(\mathit{nil}) = 0, \quad \mathit{suma}(n :: \ell) = n + \mathit{suma}(\ell)$$

Mamy tu operację  $D : T(\mathbf{int}) \rightarrow \mathbf{int}$ :

$$D(\mathit{inl}(\bullet)) = 0, \quad D(\mathit{inr}(\langle p, d \rangle)) = p + d.$$

## Inny przykład: długość listy

**lista**  $\approx \mathbf{1} \oplus (\mathbf{int} \times \mathbf{lista})$ , czyli:

**lista**  $\approx T(\mathbf{lista})$ , gdzie  $T(\alpha) = \mathbf{1} \oplus (\mathbf{int} \times \alpha)$

*suma* : **lista**  $\rightarrow$  **int**

$$\mathit{suma}(\mathit{nil}) = 0, \quad \mathit{suma}(n :: \ell) = n + \mathit{suma}(\ell)$$

Mamy tu operację  $D : T(\mathbf{int}) \rightarrow \mathbf{int}$ :

$$D(\mathit{inl}(\bullet)) = 0, \quad D(\mathit{inr}(\langle p, d \rangle)) = p + d.$$

Wtedy  $\mathit{suma}(\mathit{nil}) = D(\mathit{suma}^+(\mathit{inl}(\bullet)))$

oraz  $\mathit{suma}(n :: \ell) = D(\mathit{suma}^+(\mathit{inr}(\langle n, \ell \rangle)))$ ,

gdzie  $\mathit{suma}^+ : T(\mathbf{lista}) \rightarrow T(\mathbf{int})$ , to „podniesiona” *suma*:

$$\mathit{suma}^+(\mathit{inl}(\bullet)) = \mathit{inl}(\bullet),$$

$$\mathit{suma}^+(\mathit{inr}(\langle n, \ell \rangle)) = \mathit{inr}(\langle n, \mathit{suma}(\ell) \rangle)$$

Znowu schemat  $F(\mathit{in} N) \Rightarrow D(F^+(N))$

## A generalization

Assume  $p$  only positive in  $\sigma$ . Write  $\mu = \mu p. \sigma(p)$ .

$$\text{in} : \sigma(\mu) \rightarrow \mu \quad \text{Elim}_\tau : (\sigma(\tau) \rightarrow \tau) \rightarrow \mu \rightarrow \tau$$

Let  $D : \sigma(\tau) \rightarrow \tau$ . To define  $F = \text{Elim}_\tau D$  of type  $\mu \rightarrow \tau$ , we recursively apply  $F$  in the form of

$$F^+ : \sigma(\mu) \rightarrow \sigma(\tau).$$

Reduction rule:

$$F(\text{in } N) \Rightarrow D(F^+(N))$$

$$\text{Elim}_\tau D(\text{in } N) \Rightarrow D(\text{Lift}_\sigma(\text{Elim}_\tau D)N)$$

$$\text{Lift}_\sigma : (\mu \rightarrow \tau) \rightarrow \sigma(\mu) \rightarrow \sigma(\tau).$$

## Powrót do pierwszego pomysłu?

Mamy  $in : \sigma(\mu) \rightarrow \mu$ ,  $Elim_{\tau} : (\sigma(\tau) \rightarrow \tau) \rightarrow \mu \rightarrow \tau$   
i redukcję  $Elim_{\tau} D (in N) \Rightarrow D(Lift(Elim_{\tau} D)N)$

Czy umiemy zdefiniować operację  $out : \mu \rightarrow \sigma(\mu)$ ?

## Powrót do pierwszego pomysłu?

Mamy  $in : \sigma(\mu) \rightarrow \mu$ ,  $Elim_\tau : (\sigma(\tau) \rightarrow \tau) \rightarrow \mu \rightarrow \tau$   
i redukcję  $Elim_\tau D (in N) \Rightarrow D(Lift(Elim_\tau D)N)$

Czy umiemy zdefiniować operację  $out : \mu \rightarrow \sigma(\mu)$ ?

Można tak:  $out = Elim_{\sigma(\mu)}(Lift(in))$ ,  
gdzie  $Lift(in) : \sigma(\sigma(\mu)) \rightarrow \sigma(\mu)$ .

## Powrót do pierwszego pomysłu?

Mamy  $in : \sigma(\mu) \rightarrow \mu$ ,  $Elim_\tau : (\sigma(\tau) \rightarrow \tau) \rightarrow \mu \rightarrow \tau$   
i redukcję  $Elim_\tau D (in N) \Rightarrow D(Lift(Elim_\tau D)N)$

Czy umiemy zdefiniować operację  $out : \mu \rightarrow \sigma(\mu)$ ?

Można tak:  $out = Elim_{\sigma(\mu)}(Lift(in))$ ,  
gdzie  $Lift(in) : \sigma(\sigma(\mu)) \rightarrow \sigma(\mu)$ . Ale  $out \circ in \neq id$  :

## Powrót do pierwszego pomysłu?

Mamy  $in : \sigma(\mu) \rightarrow \mu$ ,  $Elim_\tau : (\sigma(\tau) \rightarrow \tau) \rightarrow \mu \rightarrow \tau$   
i redukcję  $Elim_\tau D (in N) \Rightarrow D(Lift(Elim_\tau D)N)$

Czy umiemy zdefiniować operację  $out : \mu \rightarrow \sigma(\mu)$ ?

Można tak:  $out = Elim_{\sigma(\mu)}(Lift(in))$ ,  
gdzie  $Lift(in) : \sigma(\sigma(\mu)) \rightarrow \sigma(\mu)$ . Ale  $out \circ in \neq id$  :

$$out(in N) \rightarrow Lift(in)(Lift(out)(N))$$

$$out \circ in = In \circ Out$$



## Twierdzenie

*Jeśli  $\langle A, \leq \rangle$  jest kratą zupełną, to każda monotoniczna funkcja  $f : A \rightarrow A$  ma najmniejszy punkt stały.*

## Twierdzenie

*Jeśli  $\langle A, \leq \rangle$  jest kratą zupełną, to każda monotoniczna funkcja  $f : A \rightarrow A$  ma najmniejszy punkt stały.*

**Dowód:** Niech  $B = \{x \in A \mid f(x) \leq x\}$ ; niech  $a = \inf B$ . Pokażemy, że  $a$  jest najmniejszym punktem stałym funkcji  $f$ .

Dla dowolnego  $x \in B$  mamy  $a \leq x$ , więc  $f(a) \leq f(x) \leq x$ . Zatem  $f(a)$  jest ograniczeniem dolnym zbioru  $B$ , skąd  $f(a) \leq a$ , bo  $a$  jest kresem dolnym.

Ale skoro  $f(a) \leq a$ , to także  $f(f(a)) \leq f(a)$ , więc  $f(a) \in B$ . Zatem  $a \leq f(a)$  i mamy równość.

Ponieważ wszystkie punkty stałe funkcji  $f$  muszą należeć do  $B$ , więc  $a$  jest najmniejszym punktem stałym. □

# Reprezentacja w systemie $\mathbf{F}$

W logice drugiego rzędu można definiować punkty stałe.

$$\text{LFP}_\phi(a) := \forall R((\Phi(R) \subseteq R) \rightarrow Ra)$$

# Reprezentacja w systemie $\mathbf{F}$

W logice drugiego rzędu można definiować punkty stałe.

$$\text{LFP}_\phi(a) := \forall R((\Phi(R) \subseteq R) \rightarrow Ra)$$

Po wytarciu to wygląda tak:

$$\mu r. \sigma(r) = \forall r((\sigma(r) \rightarrow r) \rightarrow r).$$

Okazuje się, że to jest dobra definicja  $\mu r. \sigma(r)$ .

$$\mu r. \sigma(r) = \forall r((\sigma(r) \rightarrow r) \rightarrow r)$$

Włożenie  $in : \sigma(\mu) \rightarrow \mu$

Iterator  $Elim_\tau : (\sigma(\tau) \rightarrow \tau) \rightarrow \mu \rightarrow \tau$

Redukcja  $Elim_\tau D (in N) \Rightarrow D(Lift(Elim_\tau D)N)$

Definiujemy to w systemie **F**:

$Elim = \Lambda r \lambda y^{\sigma(r) \rightarrow r} \lambda z^\mu. zry$

$in = \lambda x^{\sigma(\mu)} \Lambda r \lambda y^{\sigma(r) \rightarrow r}. y(Lift(Elim r y)x),$

gdzie  $Lift : (\mu \rightarrow q) \rightarrow \sigma(\mu) \rightarrow \sigma(q)$

**Ćwiczenie:** policzyć, że przy tej definicji

$$Elim_\tau D (in N) \rightarrow_\beta D(Lift(Elim_\tau D)N)$$

$$\mu r. \sigma(r) = \forall r ((\sigma(r) \rightarrow r) \rightarrow r)$$

Niech  $\sigma(r) = 1 \oplus r$ . Co to jest  $\mu r. \sigma(r)$ ?

$$\mu r. \sigma(r) = \forall r((\sigma(r) \rightarrow r) \rightarrow r)$$

Niech  $\sigma(r) = 1 \oplus r$ . Co to jest  $\mu r. \sigma(r)$ ?

Wychodzi takie coś:  $\mu r. 1 \oplus r = \forall r((1 \oplus r \rightarrow r) \rightarrow r)$

Typ  $\forall r((1 \oplus r \rightarrow r) \rightarrow r)$  to z grubsza to samo, co

$$\forall r(((1 \rightarrow r) \times (r \rightarrow r)) \rightarrow r),$$

$$\mu r. \sigma(r) = \forall r((\sigma(r) \rightarrow r) \rightarrow r)$$

Niech  $\sigma(r) = 1 \oplus r$ . Co to jest  $\mu r. \sigma(r)$ ?

Wychodzi takie coś:  $\mu r. 1 \oplus r = \forall r((1 \oplus r \rightarrow r) \rightarrow r)$

Typ  $\forall r((1 \oplus r \rightarrow r) \rightarrow r)$  to z grubsza to samo, co

$$\forall r(((1 \rightarrow r) \times (r \rightarrow r)) \rightarrow r), \text{ czyli}$$

$$\forall r(r \times (r \rightarrow r) \rightarrow r),$$



$$\mu r. \sigma(r) = \forall r((\sigma(r) \rightarrow r) \rightarrow r)$$

Niech  $\sigma(r) = 1 \oplus r$ . Co to jest  $\mu r. \sigma(r)$ ?

Wychodzi takie coś:  $\mu r. 1 \oplus r = \forall r((1 \oplus r \rightarrow r) \rightarrow r)$

Typ  $\forall r((1 \oplus r \rightarrow r) \rightarrow r)$  to z grubsza to samo, co

$$\forall r(((1 \rightarrow r) \times (r \rightarrow r)) \rightarrow r), \text{ czyli}$$

$$\forall r(r \times (r \rightarrow r) \rightarrow r), \text{ czyli}$$

$$\forall r(r \rightarrow (r \rightarrow r) \rightarrow r),$$

$$\mu r. \sigma(r) = \forall r((\sigma(r) \rightarrow r) \rightarrow r)$$

Niech  $\sigma(r) = 1 \oplus r$ . Co to jest  $\mu r. \sigma(r)$ ?

Wychodzi takie coś:  $\mu r. 1 \oplus r = \forall r((1 \oplus r \rightarrow r) \rightarrow r)$

Typ  $\forall r((1 \oplus r \rightarrow r) \rightarrow r)$  to z grubsza to samo, co

$\forall r(((1 \rightarrow r) \times (r \rightarrow r)) \rightarrow r)$ , czyli

$\forall r(r \times (r \rightarrow r) \rightarrow r)$ , czyli

$\forall r(r \rightarrow (r \rightarrow r) \rightarrow r)$ , czyli

$\forall r((r \rightarrow r) \rightarrow r \rightarrow r) = \omega$ .

## Problem otwarty

Ta konstrukcja nie przenosi się na (uogólnione) rekursory.

Wiadomo, że w systemie  $\mathbf{F}$  nie można definiować **rekursorów** używając beta-redukcji.

Nie wiadomo, czy w systemie  $\mathbf{F}$  można zdefiniować dowolny rekursor używając **beta-eta-redukcji**.

## Paradoks Reynoldsa, czyli

„Polymorphism is not set-theoretic”

Niech  $T(\alpha) = (\alpha \rightarrow \mathbf{bool}) \rightarrow \mathbf{bool}$ .

Można zdefiniować typ  $\mu = \mu p. (T(p) \rightarrow p) \rightarrow p$   
i operację  $\mathbf{in} : T(\mu) \rightarrow \mu$ .

## Paradoks Reynoldsa, czyli

„Polymorphism is not set-theoretic”

Niech  $T(\alpha) = (\alpha \rightarrow \mathbf{bool}) \rightarrow \mathbf{bool}$ .

Można zdefiniować typ  $\mu = \mu p. (T(p) \rightarrow p) \rightarrow p$   
i operację  $\mathbf{in} : T(\mu) \rightarrow \mu$ .

Część trudna: w modelu teoriomnogościowym:

- typ  $\alpha \rightarrow \mathbf{bool}$  musi być interpretowany jako  $\mathbf{P}(\alpha)$ ;
- typ  $T(\alpha)$  jako  $\mathbf{P}(\mathbf{P}(\alpha))$ ;
- funkcja  $\mathbf{in} : T(\mu) \rightarrow \mu$  musi być różnowartościowa.

## Paradoks Reynoldsa, czyli

„Polymorphism is not set-theoretic”

Niech  $T(\alpha) = (\alpha \rightarrow \mathbf{bool}) \rightarrow \mathbf{bool}$ .

Można zdefiniować typ  $\mu = \mu p. (T(p) \rightarrow p) \rightarrow p$   
i operację  $\mathbf{in} : T(\mu) \rightarrow \mu$ .

Część trudna: w modelu teoriomnogościowym:

- typ  $\alpha \rightarrow \mathbf{bool}$  musi być interpretowany jako  $\mathbf{P}(\alpha)$ ;
- typ  $T(\alpha)$  jako  $\mathbf{P}(\mathbf{P}(\alpha))$ ;
- funkcja  $\mathbf{in} : T(\mu) \rightarrow \mu$  musi być różnowartościowa.

Zatem nie istnieje model teoriomnogościowy.

# Wada kodowania impredykatywnego

Typy kodowane w systemie **F** mają za dużo elementów.

# Wada kodowania impredykatywnego

Typy kodowane w systemie **F** mają za dużo elementów.

Najprostszy przykład:  $\alpha \times \beta = \forall p((\alpha \rightarrow \beta \rightarrow p) \rightarrow p)$ .



# Wada kodowania impredykatywnego

Typy kodowane w systemie **F** mają za dużo elementów.

Najprostszy przykład:  $\alpha \times \beta = \forall p((\alpha \rightarrow \beta \rightarrow p) \rightarrow p)$ .

Niech  $\alpha = \forall r(r \rightarrow \forall p(p \rightarrow r) \rightarrow r)$     $\beta = \forall r(r \rightarrow r)$ .

Wtedy **K** :  $\alpha$  oraz **I** :  $\beta$ .

# Wada kodowania impredykatywnego

Typy kodowane w systemie **F** mają za dużo elementów.

Najprostszy przykład:  $\alpha \times \beta = \forall p((\alpha \rightarrow \beta \rightarrow p) \rightarrow p)$ .

Niech  $\alpha = \forall r(r \rightarrow \forall p(p \rightarrow r) \rightarrow r)$   $\beta = \forall r(r \rightarrow r)$ .

Wtedy **K** :  $\alpha$  oraz **I** :  $\beta$ .

Dla  $x : \alpha \rightarrow \beta \rightarrow p$  wyrażenie **xKI** ma typ  $p$ .

Zatem taki inhabitant typu  $\alpha \times \beta$

$$\Lambda p \lambda x^{\alpha \rightarrow \beta \rightarrow p}. x(\Lambda r \lambda y^r \lambda z^{\forall p(p \rightarrow r)}. zp(xKI))I$$

nie jest parą  $\langle M, N \rangle = \Lambda p \lambda x MN$ , bo  $x \in FV(M)$

# Wada kodowania impredykatywnego

Nie można udowodnić zasady indukcji, nawet dla typu bool.



## Typy ściśle pozytywne

Nieściła definicja ścisłej pozytywności: typ  $\mu p. \sigma(p)$  jest dozwolony tylko wtedy, gdy  $p$  nie występuje w  $\sigma(p)$  w argumencie implikacji.

## Typy ściśle pozytywne

Nieściła definicja ścisłej pozytywności: typ  $\mu p. \sigma(p)$  jest dozwolony tylko wtedy, gdy  $p$  nie występuje w  $\sigma(p)$  w argumencie implikacji.

**Przykład 1:** ten typ nie jest ściśle pozytywny:

$$\mu p. 0 \rightarrow (p \rightarrow 0) \rightarrow 0$$

## Typy ściśle pozytywne

Nieściła definicja ścisłej pozytywności: typ  $\mu p. \sigma(p)$  jest dozwolony tylko wtedy, gdy  $p$  nie występuje w  $\sigma(p)$  w argumencie implikacji.

**Przykład 1:** ten typ nie jest ściśle pozytywny:

$$\mu p. 0 \rightarrow (p \rightarrow 0) \rightarrow 0$$

Liczby Parigota:

$$\bar{0} = \lambda x f. x, \quad \bar{1} = \lambda x f. f \bar{0}, \quad \bar{2} = \lambda x f. f \bar{1}, \quad \dots$$

## Typy ściśle pozytywne

Nieściła definicja ścisłej pozytywności: typ  $\mu p. \sigma(p)$  jest dozwolony tylko wtedy, gdy  $p$  nie występuje w  $\sigma(p)$  w argumencie implikacji.

**Przykład 1:** ten typ nie jest ściśle pozytywny:

$$\mu p. 0 \rightarrow (p \rightarrow 0) \rightarrow 0$$

Liczby Parigota:

$$\bar{0} = \lambda x f. x, \quad \bar{1} = \lambda x f. f \bar{0}, \quad \bar{2} = \lambda x f. f \bar{1}, \quad \dots$$

**Przykład 2:** ten typ jest ściśle pozytywny:

$$\omega\text{-tree} = \mu p. \mathbf{1} \oplus (\text{int} \rightarrow p).$$

# Zalety ścisłej pozytywności

- ▶ Takie typy najczęściej występują w praktyce;
- ▶ Łatwiejsza metateoria (np. dowodzenie SN);
- ▶ Mniejsze ryzyko paradoksu:  
typ  $\mu p ((p \rightarrow \mathbf{bool}) \rightarrow \mathbf{bool})$   
jest niebezpiecznie podobny do  $A \approx \mathbf{P}(\mathbf{P}(A))$ .
- ▶ Naturalne uogólnienie na typy zależne.



# Typy indukcyjne

Typowa sytuacja: typ rekurencyjny jest zadany przez kilka konstruktorów (kilka schematów postaci kanonicznych), np:

- ▶ Zero  $0 : \mathbf{int}$  i następnik  $s : \mathbf{int} \rightarrow \mathbf{int}$ ,  
dla  $\mathbf{int} \approx \mathbf{1} \oplus \mathbf{int}$ ;
- ▶  $nil : list$  i  $cons : \mathbf{int} \times list \rightarrow list$ ,  
dla  $list \approx \mathbf{1} \oplus \mathbf{int} \times list$ ;
- ▶  $leaf : \omega\text{-tree}$  oraz  $node : (\mathbf{int} \rightarrow \omega\text{-tree}) \rightarrow \omega\text{-tree}$ ,  
dla  $\omega\text{-tree} \approx \mathbf{1} \oplus (\mathbf{int} \rightarrow \omega\text{-tree})$ .

## Przykłady

```
Inductive nat : Set := 0 : nat | S : nat -> nat.
```

## Przykłady

```
Inductive nat : Set := 0 : nat | S : nat -> nat.
```

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

## Przykłady

```
Inductive nat : Set := 0 : nat | S : nat -> nat.
```

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

Typy indukcyjne nie muszą być rekurencyjne:

## Przykłady

```
Inductive nat : Set := 0 : nat | S : nat -> nat.
```

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

Typy indukcyjne nie muszą być rekurencyjne:

```
Inductive bool : Set := true : bool | false : bool.
```

## Przykłady

```
Inductive nat : Set := 0 : nat | S : nat -> nat.
```

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

Typy indukcyjne nie muszą być rekurencyjne:

```
Inductive bool : Set := true : bool | false : bool.
```

```
Inductive unit : Set := tt : unit.
```

## Przykłady

```
Inductive nat : Set := 0 : nat | S : nat -> nat.
```

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

Typy indukcyjne nie muszą być rekurencyjne:

```
Inductive bool : Set := true : bool | false : bool.
```

```
Inductive unit : Set := tt : unit.
```

```
Inductive empty_set : Set := .
```

## Spójniki zdaniowe:

Inductive and (A B : Prop) : Prop :=

conj : A -> B -> A /\ B.

Inductive or (A B : Prop) : Prop :=

or\_introl : A -> A \/ B | or\_intror : B -> A \/ B.

Inductive False : Prop := .



## Spójniki zdaniowe:

Inductive and (A B : Prop) : Prop :=

conj : A -> B -> A /\ B.

Inductive or (A B : Prop) : Prop :=

or\_introl : A -> A \/ B | or\_intror : B -> A \/ B.

Inductive False : Prop := .

## Kwantyfikator szczegółowy:

Inductive ex (A : Type) (P : A -> Prop) : Prop :=

ex\_intro : forall x : A, P x -> ex P.

Piszemy „exists x:A,Px” zamiast „ex(fun x:A => P x)”.

# Preedykaty indukcyjne

## Preedykaty indukcyjne

```
Inductive even : nat -> Prop :=  
  zeroparz : even 0 |  
  nastparz : forall n : nat, even n -> even (n + 2)
```

## Przydykaty indukcyjne

```
Inductive even : nat -> Prop :=  
  zeroparz : even 0 |  
  nastparz : forall n : nat, even n -> even (n + 2)
```

```
Inductive eq (A : Type) (x : A) : A -> Prop :=  
  eq_refl : x = x
```

Przyjęta notacja:  $x =_A y$ .

## Proof by induction

```
Coq < Goal forall n : nat, even (n+n).
```

```
1 subgoal
```

```
=====
```

```
forall n : nat, even (n + n)
```

```
Unnamed_thm < induction n.
```

```
2 subgoals
```

```
=====
```

```
even (0 + 0)
```

```
subgoal 2 is:
```

```
even (S n + S n)
```