# Logika i teoria typów

Wykład 11

11 stycznia 2020

# Ko-rekursja i ko-indukcja

## Strumienie

Jaki sens ma równanie $\mathbf{Stream} = \mathbf{int} \times \mathbf{Stream}$?

Można je potraktować podobnie jak równanie $\mathit{int} = \mathbf{1} \oplus \mathbf{int}$, i rozważać typ rekurencyjny (indukcyjny)

$$\mathbf{Stream} = \mu p.\, \mathbf{int} \times p.$$

Ale to jest *najmniejsze* rozwiązanie równania: typ pusty.

**Najmniejszy punkt stały $\mu = \mu p.\, \sigma(p)$:**

- Kres górny (suma) ciągu aproksymacji skończonych.
- Najmniejszy „prefixpoint":

$$\frac{\sigma(R) \subseteq R}{\mu \subseteq R} \qquad \frac{f : \sigma(R) \to R}{\mathbf{Elim}\, f : \mu \to R}$$

**Największy punkt stały $\nu = \nu p.\, \sigma(p)$:**

- Kres dolny (przecięcie) ciągu aproksymacji nieskończonych.
- Największy „postfixpoint":

$$\frac{R \subseteq \sigma(R)}{R \subseteq \nu} \qquad \frac{f : R \to \sigma(R)}{\mathbf{Intro}\, f : R \to \nu}$$

## Największy „postfixpoint" $\nu = \nu p.\, \sigma(p)$

$$\frac{R \subseteq \sigma(R)}{R \subseteq \nu} \qquad \frac{f : R \to \sigma(R)}{\mathbf{Intro}\, f : R \to \nu}$$

**Na przykład $\mathbf{Stream} = \nu p.\, \mathit{nat} \times p$:**

$$\frac{R \subseteq \mathbf{nat} \times R}{R \subseteq \mathbf{Stream}} \qquad \frac{f : R \to \mathbf{nat} \times R}{\mathbf{Intro}\, f : R \to \mathbf{Stream}}$$

**Co to znaczy?**

- Jeśli $R \subseteq \mathbf{nat} \times R$, to każdy element $R$ jest strumieniem.
- Funkcja $R \to \mathbf{nat} \times R$ wyznacza funkcję $R \to \mathbf{Stream}$.

**Destruktory**

Destruktory nie są pierwotne.
Definiujemy je przez dopasowanie:

```
CoInductive Stream : Set
   := Cons : nat -> Stream -> Stream.

Definition hd (s : Stream)
   := match s with Cons a t => a end.

Definition tl (s : Stream)
   := match s with Cons a t => t end.
```

```
Coq < Check hd.
hd  : Stream -> nat

Coq < Check tl.
tl  : Stream -> Stream
```

## Przykład Coquanda

Może być więcej niż jeden konstruktor.
(Nie ma uniwersalnych destruktorów.)

Proces typu $P$ może wczytać liczbę i wypisać liczbę.

$\mathit{in} : (\mathbf{nat} \to P) \to P$;

$\mathit{out} : \mathbf{nat} \to P \to P$.

Na przykład coś takiego:

$\mathit{in}(\lambda n_1.\, \mathit{out}(0, \mathit{in}(\lambda n_2.\, \mathit{out}(n_1, \mathit{out}(1, \mathit{in}(\lambda n_3 \ldots$

Można definiować procesy ko-rekurencyjnie:

$p = \mathit{in}(\lambda n.\, \mathit{out}(n+1, p))$

## Przykład: definicje korekurencyjne:

```
CoFixpoint samezera : Stream := Cons 0 samezera.

CoFixpoint codrugieod : nat -> Stream :=
fun n :nat => Cons n (codrugieod (n+2)).

Definition wszystkieparzyste := codrugieod 0.

Coq < Eval simpl in hd (tl (wszystkieparzyste)).
     = 2
     : nat
```

## Predykaty koindukcyjne

Równość strumieni (ekstensjonalna):

$s = t$, wtedy i tylko wtedy, gdy:

– $hd(s) = hd(t)$ oraz
– $tl(s) = tl(t)$

To jest *bisymulacja*.

```
CoInductive EqSt (s1 s2: Stream) : Prop :=
 eqst :
 hd s1 = hd s2 -> EqSt (tl s1) (tl s2) -> EqSt s1 s2.
```

## Sens moralny

```
CoInductive Sameparzyste : Stream -> Prop :=
krok : forall x : Stream,
even (hd x) -> Sameparzyste (tl x) -> Sameparzyste x.

CoFixpoint Zeraparzyste : Sameparzyste samezera :=
krok samezera zeroparz Zeraparzyste.
```

Predykat `Sameparzyste` to w istocie największy punkt stały operacji $s \mapsto \mathtt{even} \times s$. W tym dowodzie korzystamy z reguły

$$\frac{R \subseteq \mathtt{even} \times R}{R \subseteq \mathtt{Sameparzyste}}$$

dla relacji $R = \{\mathtt{samezera}\}$

## Kolokwialny dowód przez ko-indukcję

Chcemy udowodnić, że strumień *samezera* składa się wyłącznie z liczb parzystych.

Weźmy więc ten strumień.

Sprawdzamy, że $hd(samezera) = 0$ jest liczbą parzystą.

Sprawdzamy, że $tl(s)$ to też *samezera*.

Z założenia koindukcyjnego wnioskujemy, że strumień *samezera* składa się z samych liczb parzystych.

A zatem strumień *samezera* składa się tylko z liczb parzystych. Twierdzenie udowodnione.

## Definicje produktywne

Dowód przez koindukcję, to też obiekt koindukcyjny: nieskończony term dowodowy zbudowany z konstruktorów właściwych dla dowodzonego predykatu koindukcyjnego (w tym przypadku `krok`).

Definicja nieskończonego obiektu (np. strumienia) jest *produktywna*, jeśli definiowany obiekt redukuje się do *postaci kanonicznej* (z konstruktorem) i wszystkie jego składowe też.

Te definicje nie są produktywne:

```
CoFixpoint Bad :  Stream := Bad.
CoFixpoint Dab :  Stream := tl (Cons 0 Dab)
```

### Predykat koindukcyjny:
strumień składa się tylko z liczb parzystych.

```
CoInductive Sameparzyste : Stream -> Prop :=
krok : forall x : Stream,
even (hd x) -> Sameparzyste (tl x) -> Sameparzyste x.
```

### Dowód przez koindukcję:
(wiemy, że `zeroparz` : even(0))

```
CoFixpoint samezera : Stream := Cons 0 samezera.

CoFixpoint Zeraparzyste : Sameparzyste samezera :=
krok samezera zeroparz Zeraparzyste.

Coq < Zeraparzyste is corecursively defined
```

## Kolokwialny dowód przez ko-indukcję

Chcemy udowodnić, że każdy strumień z rodziny $R$ składa się wyłącznie z liczb parzystych.

Weźmy dowolny strumień $s \in R$.

Sprawdzamy, że $hd(s)$ jest liczbą parzystą.

Sprawdzamy, że $tl(s)$ też należy do rodziny $R$.

Z założenia koindukcyjnego wnioskujemy, że $tl(s)$ składa się z samych liczb parzystych.

A zatem $s$ też składa się tylko z liczb parzystych. Twierdzenie udowodnione.

## Bałamutny „dowód" przez ko-indukcję

Chcemy udowodnić, że strumień *samezera* składa się wyłącznie z liczb parzystych.

Korzystaliśmy z założenia koindukcyjnego, że strumień $tl(samezera) = samezera$ składa się z samych liczb parzystych.

Ale w takim razie teza wynika bezpośrednio z założenia koindukcyjnego!

Twierdzenie udowodnione!?

**Nie, ten „dowód" nie jest produktywny**

### Nieproduktywna definicja funkcji $F$ : **Stream** $\to$ **Stream**

```
CoFixpoint F : Stream -> Stream :=
fun s :  Stream =>
Cons ((hd s)+(hd(tl s))) (F(F(tl(tl(s)))))
```

Czyli po ludzku: $F(a :: b :: t) = (a + b :: F^2(t))$.

Co jest na drugim miejscu w strumieniu $F(0 :: 1 :: 2 :: 3...)$?

Liczymy:

$F(0 :: 1 :: 2 :: 3...) = 1 :: F(F(2 :: 3...)) =$
$1 :: F(2 + 3 :: F^2(4...)) = 1 :: F(2 + 3 :: F(4 + 5 :: F^2(6...))$

Nie można ewaluować $F(2 + 3 :: F^2(4...))$, bo argument nie jest postaci $a :: b :: t$.

## Jak zagwarantować produktywność

Definicje korekurencyjne w Coqu muszą być *strzeżone* (guarded). Wywołania korekurencyjne są dozwolone tylko bezpośrednio pod konstruktorem czołowym. Definicja:

```
CoFixpoint F : Stream -> Stream :=
fun s :  Stream =>
Cons ((hd s)+(hd(tl s))) (F(F(tl(tl(s)))))
```

jest niestrzeżona, bo F występuje pod F.

## Dependent types

## Gdzie drwa rąbią...

Ta definicja jest, owszem, produktywna:

$F(a :: t) = (a + 1 :: F^2(t))$.

Ale Coq jej też nie lubi:

```
CoFixpoint F : Stream -> Stream :=
fun s : Stream =>
Cons ((hd s)+1) (F(F(tl(s)))).

Coq < Error:
Recursive definition of F is ill-formed.
In environment
F : Stream -> Stream
s : Stream
Nested recursive occurrences.
Recursive definition is:
"fun s : Stream => Cons (hd s + 1) (F (F (tl s)))".
```

## Dependent types

**Principal idea:** Type $array[n]$ depends on $n : int$.
It is created by a type constructor (type family)

$$array : int \Rightarrow *.$$

**Curry-Howard:** Constructor $P : \tau \Rightarrow *$ is a predicate on $\tau$.

Adding all numbers in an array:

$$sum_n : array[n] \to int$$

Parameterized by $n$:

$$sum : (n : int) \to array[n] \to int$$

$$sum : \forall n{:}int.\, array[n] \to int$$

## System $\lambda P$

Poziomy syntaktyczne:

- Termy, np. $\lambda x^{\forall z:q.\,\alpha(z)} \lambda y^{p\to q} v^{p}.\, x(yv)$;
- Typy i konstruktory, np. $\lambda x^{q}.\, \alpha x \to p$, $(\forall z : q.\, \alpha(z)) \to (y : p \to q) \to (v : p) \to \alpha(yv)$.
- Rodzaje, np. $\alpha$ jest rodzaju $q \Rightarrow *$.

Nie ma „absolutnej" definicji termu, konstruktora, ani rodzaju. Wszystko zależy od otoczenia.

Np. to, czy $\alpha x$ jest legalnym typem czy nie, zależy od rodzaju zmiennej $\alpha$ i typu zmiennej $x$.

## Surowa składnia („raw terms")

$$\kappa ::= * \mid (\Pi x{:}\phi\ \kappa);$$

$$\phi ::= \alpha \mid (\forall x{:}\phi\ \phi) \mid (\phi M) \mid (\lambda x{:}\phi\ \phi);$$

$$M ::= x \mid (MM) \mid (\lambda x{:}\phi\ M);$$

$$\Gamma ::= \varnothing \mid \Gamma, (x : \phi) \mid \Gamma, (\alpha : \kappa).$$

## Alternatywna notacja

- $(x : \tau) \Rightarrow \kappa$ zamiast $\Pi x{:}\tau.\, \kappa$;
- $\tau \Rightarrow \kappa$, gdy $x$ nie jest wolne w $\kappa$;
- $(x : \tau) \to \sigma$ zamiast $\forall x{:}\tau.\, \sigma$;
- $\tau \to \sigma$, gdy $x$ nie jest wolne w $\sigma$.

## Wiązanie zmiennych

$$\mathrm{FV}(*) = \varnothing,\ \mathrm{FV}(\alpha) = \{\alpha\},\ \mathrm{FV}(x) = \{x\},$$
$$\mathrm{FV}(EE') = \mathrm{FV}(E) \cup \mathrm{FV}(E'),$$
$$\mathrm{FV}(\nabla x{:}\phi\ E) = \mathrm{FV}(\phi) \cup (\mathrm{FV}(E) - \{x\}),$$

gdzie $\nabla \in \{\lambda, \Pi, \forall\}$.

(Zakładamy alfa-konwersję)

## Alfa-konwersja: ćwiczenie

Niech $x : p$ oraz $P : p \to *$.
Który z tych osądów jest poprawny?

- $\Gamma \vdash (\lambda x{:}Px.\, x) : \forall x{:}Px.\, Px$ ?
- $\Gamma \vdash (\lambda x{:}Px.\, x) : \forall y{:}Px.\, Px$ ?

Po przemianowaniu zmiennych:

- $\Gamma \vdash (\lambda y{:}Px.\, y) : \forall y{:}Px.\, Px$ (czyli $Px \to Px$).

Typ $\forall x{:}Px.\, Px$, czyli $\forall y{:}Px.\, Py$, jest nielegalny.

Bo wtedy $P : P\,x \Rightarrow *$.

## System $\lambda P$: reguły

są w pliku
https://www.mimuw.edu.pl/~urzy/Litt/wreczalt-lamP.pdf

dostępnym też z Moodla.

## Trzy rodzaje osądów

- Kind formation judgements of the form $\Gamma \vdash \boldsymbol{\kappa} : \Box$,
  ("$\boldsymbol{\kappa}$ is a kind in the environment $\Gamma$")
- Kinding judgements of the form $\Gamma \vdash \varphi : \boldsymbol{\kappa}$,
  ("$\varphi$ is a constructor of kind $\boldsymbol{\kappa}$ in $\Gamma$.")
- Typing judgements of the form $\Gamma \vdash M : \tau$,
  ("$M$ is a term of type $\tau$ in $\Gamma$.")

## Formowanie rodzajów

$$\vdash * : \Box \qquad \frac{\Gamma,\, x:\tau \vdash \boldsymbol{\kappa} : \Box}{\Gamma \vdash \Pi x{:}\tau\ \boldsymbol{\kappa} : \Box}$$

Wszystkie rodzaje są postaci $\Pi x_1 : \tau_1 \ldots x_n : \tau_n.\, *$,

czyli postaci $(x_1 : \tau_1) \Rightarrow \cdots \Rightarrow (x_n : \tau_n) \Rightarrow *$.

Ale tu mogą być zależności:

$$(x_1 : \tau_1) \Rightarrow (x_2 : \tau_2(x_1)) \Rightarrow (x_3 : \tau_3(x_1, x_2)) \Rightarrow \cdots \\ \cdots \Rightarrow (x_n : \tau_n(x_1, \ldots, x_{n-1})) \Rightarrow *.$$

## Beta-redukcja

Podstawienie $E[x := M]$, gdzie $M$ to surowy term, definiujemy jak zwykle.

Beta-redukcja:

$$\begin{aligned} (\lambda x{:}\tau.\, M)N &\to_\beta M[x := N]; \\ (\lambda x{:}\tau.\, \phi)N &\to_\beta \phi[x := N]. \end{aligned}$$

Metazmienne w regułach:

- $\kappa$ – rodzaj;
- $\tau, \sigma$ – typy;
- $\varphi$ – konstruktor;
- $M$ – term obiektowy;
- $x$ – zmienna obiektowa;
- $\alpha$ – zmienna konstruktorowa (typowa).

## Poprawne otoczenie

Otoczenie $\Gamma$ jest *ciągiem* deklaracji, a nie zbiorem.

*Poprawne otoczenie*: takie, które występuje w jakimś wyprowadzalnym osądzie.

Nie każdy ciąg deklaracji jest poprawnym otoczeniem, np. ciąg $\{\alpha : p \Rightarrow *, x : p\}$ nie jest, bo najpierw trzeba zadeklarować $p : *$.

Można udowodnić taki fakt:

$\Gamma$ jest poprawne wtedy i tylko wtedy, gdy $\Gamma \vdash * : \Box$.

## Rodzajowanie

$$\frac{\Gamma \vdash \boldsymbol{\kappa} : \Box}{\Gamma,\, \alpha : \boldsymbol{\kappa} \vdash \alpha : \boldsymbol{\kappa}}\ (\alpha \notin \mathrm{Dom}(\Gamma))$$

$$\frac{\Gamma,\, x:\tau \vdash \sigma : *}{\Gamma \vdash (\forall x{:}\tau\, \sigma) : *}$$

$$\frac{\Gamma \vdash \varphi : (\Pi x{:}\tau\ \boldsymbol{\kappa}) \quad \Gamma \vdash M : \tau}{\Gamma \vdash (\varphi M) : \boldsymbol{\kappa}[x := M]}$$

$$\frac{\Gamma,\, x:\tau \vdash \varphi : \boldsymbol{\kappa}}{\Gamma \vdash (\lambda x{:}\tau\, \varphi) : (\Pi x{:}\tau\ \boldsymbol{\kappa})}$$

## Typowanie

$$\frac{\Gamma \vdash \tau : *}{\Gamma, x:\tau \vdash x : \tau}\ (x \notin \mathrm{Dom}(\Gamma))$$

$$\frac{\Gamma \vdash M : (\forall x{:}\tau\, \sigma) \quad \Gamma \vdash N : \tau}{\Gamma \vdash (MN) : \sigma[x := N]}$$

$$\frac{\Gamma,\, x:\tau \vdash M : \sigma}{\Gamma \vdash (\lambda x{:}\tau\, M) : (\forall x{:}\tau\, \sigma)}$$

## Konwersja

$$\frac{\Gamma \vdash \varphi : \boldsymbol{\kappa} \quad \Gamma \vdash \boldsymbol{\kappa}' : \Box}{\Gamma \vdash \varphi : \boldsymbol{\kappa}'}\ (\boldsymbol{\kappa} =_\beta \boldsymbol{\kappa}')$$

$$\frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash \sigma' : *}{\Gamma \vdash M : \sigma'}\ (\sigma =_\beta \sigma')$$

**Uwaga:** Sprawdzanie typu jest nieelementarne!
(Bo taka jest konwersja $=_\beta$.)
Ale rozstrzygalne, bo to jest rachunek w stylu Churcha.

## Wzajemne uwikłanie

- Rodzaje nie występują w termach, konstruktorach i typach, ale:
- W termach i rodzajach mogą występować konstruktory i typy;
- W rodzajach, konstruktorach i typach mogą występować termy.

Na przykład w rodzaju $(y : \tau) \Rightarrow \alpha(\lambda z^{\beta(y)}\, z) \Rightarrow *$
jest term $\lambda z^{\beta(y)}\, z$, a w nim typ $\beta(y)$.

## Wycieranie zależności

Wycieranie zależności z konstruktorów i typów:

- $\overline{\alpha} = \alpha$;
- $\overline{(\forall x : \tau)\sigma} = \overline{\tau} \to \overline{\sigma}$;
- $\overline{\varphi M} = \overline{\varphi}$.

Wynikiem jest zawsze typ prosty.

## Osłabianie (generyczne)

$$\frac{\Gamma \vdash \tau : * \quad \Gamma \vdash \mathsf{ls} : \mathsf{ps}}{\Gamma,\, x:\tau \vdash \mathsf{ls} : \mathsf{ps}}\ (x \notin \mathrm{Dom}(\Gamma))$$

$$\frac{\Gamma \vdash \boldsymbol{\kappa} : \Box \quad \Gamma \vdash \mathsf{ls} : \mathsf{ps}}{\Gamma,\, \alpha:\boldsymbol{\kappa} \vdash \mathsf{ls} : \mathsf{ps}}\ (\alpha \notin \mathrm{Dom}(\Gamma))$$

## Przykłady

Niech $\Gamma = \{\tau : *, \alpha : \tau \Rightarrow *, \beta : \tau \Rightarrow *\}$.

- Term $\lambda x^{\forall z:\tau.\, \alpha z \to \beta z} \lambda y^{\forall z:\tau.\, \alpha z} \lambda z^{\tau}.\, xz(yz)$ ma w $\Gamma$ typ $(\forall x{:}\tau.\, \alpha x \to \beta x) \to (\forall x{:}\tau.\, \alpha x) \to \forall x{:}\tau.\, \beta x$.
- Term $\lambda f^{\tau\to\tau} \lambda y^{\forall x:\tau.\, \alpha x \to \alpha(fx)} \lambda x^{\tau} \lambda z^{\alpha x}.\, y(fx)(yxz)$ ma typ $\forall f{:}\tau\to\tau.\, (\forall x{:}\tau.\, \alpha x \to \alpha(fx)) \to \forall x{:}\tau.\, \alpha x \to \alpha(f^2(x))$.
- Term $\lambda y^{\tau} \lambda f^{\Pi_{x:\tau}\alpha(x)} \lambda g^{\alpha(y)\to\tau}.\, f(g(fy))$ ma typ $\forall y^{\tau} \forall f^{\Pi_{x:\tau}\alpha(x)} \forall g^{\alpha(y)\to\tau}.\, \alpha(g(f(y)))$.

## Poprawność

### Lemma

*Jeśli* $\Gamma \vdash \varphi : \boldsymbol{\kappa}$, *to* $\Gamma \vdash \boldsymbol{\kappa} : \Box$, *a jeśli* $\Gamma \vdash M : \tau$ *to* $\Gamma \vdash \tau : *$.

### Theorem

- *Jeśli* $\Gamma \vdash M : \sigma$ *oraz* $M \to_\beta M'$, *to* $\Gamma \vdash M' : \sigma$.
- *Jeśli* $\Gamma \vdash \varphi : \boldsymbol{\kappa}$ *oraz* $\varphi \to_\beta \varphi'$, *to* $\Gamma \vdash \varphi' : \boldsymbol{\kappa}$.
- *Jeśli* $\Gamma \vdash \boldsymbol{\kappa} : \Box$ *oraz* $\boldsymbol{\kappa} \to_\beta \boldsymbol{\kappa}'$, *to* $\Gamma \vdash \boldsymbol{\kappa}' : \Box$.

## Wycieranie zależności

Wycieranie zależności z termów:

- $\overline{x} = x$;
- $\overline{MN} = \overline{M}\,\overline{N}$;
- $\overline{\lambda x{:}\tau.M} = \lambda x{:}\overline{\tau}.\overline{M}$.

Wycieranie zależności z otoczenia:

- $\overline{\Gamma} = \{(x : \overline{\tau}) \mid (x : \tau) \in \Gamma\}$.

## Wycieranie zależności

Wycieranie zależności z rodzajów (na razie niepotrzebne):

- $\overline{*} = 0$;
- $\overline{\Pi(x:\tau)\boldsymbol{\kappa}} = \overline{\tau} \to \overline{\boldsymbol{\kappa}}$;

Wynikiem jest typ prosty.

## Wycieranie zależności (term : typ)

**Fakt:** Jeśli $\Gamma \vdash M : \tau$, to $\overline{\Gamma} \vdash \overline{M} : \overline{\tau}$

Na przykład:

- Term $\lambda x^{\forall z:\tau.\,\alpha z \to \beta z} \lambda y^{\forall z:\tau.\,\alpha z} \lambda z^{\tau}.\, xz(yz)$ ma typ $(\forall x{:}\tau.\,\alpha x \to \beta x) \to (\forall x{:}\tau.\,\alpha x) \to \forall x{:}\tau.\,\beta x$.
- Jego wytarciem jest $\mathsf{S} = \lambda x^{\overline{\tau}\to\overline{\alpha}\to\overline{\beta}} \lambda y^{\overline{\tau}\to\overline{\alpha}} \lambda z^{\overline{\tau}}.\, xz(yz)$ typu $(\overline{\tau} \to \overline{\alpha} \to \overline{\beta}) \to (\overline{\tau} \to \overline{\alpha}) \to \overline{\tau} \to \overline{\beta}$

## Wycieranie typów

Wycieranie typów

- $|x| = x$;
- $|\lambda x{:}\tau.M| = \lambda x\,|M|$;
- $|MN| = |M||N|$.

**Lemat:** $|\overline{M}| = |M|$.

## System $\lambda P$ à la Curry

**Def:** Beztypowy term $M$ jest *typowalny* w $\lambda P$, gdy istnieją takie $\Gamma$, $N$, $\tau$, że $\Gamma \vdash_{\lambda P} N : \tau$ oraz $|N| = M$.

**Fakt:** Term $M$ jest typowalny w $\lambda P$, wtedy i tylko wtedy, gdy jest typowalny w typach prostych.

**Dowód:** Jeśli $\Gamma \vdash_{\lambda P} N : \tau$, to $\overline{\Gamma} \vdash \overline{N} : \overline{\tau}$ w typach prostych w stylu Churcha. Zatem $\overline{\Gamma} \vdash |\overline{N}| : \overline{\tau}$ czyli $\overline{\Gamma} \vdash |N| : \overline{\tau}$.

## Sprawdzanie typu à la Curry

**Fakt 1:** Term $M$ jest typowalny w $\lambda P$, wtedy i tylko wtedy, gdy jest typowalny w typach prostych.

**Fakt 2:** Następujący problem jest nierozstrzygalny:

Dany beztypowy term $M$ oraz $\Gamma$ i $\tau$.
Czy istnieje taki term $N$, że $\Gamma \vdash_{\lambda P} N : \tau$ oraz $|N| = M$?

## Problem inhabitacji dla $\lambda P$

**Fakt:** Następujący problem jest nierozstrzygalny:

*Dane $\Gamma$ i $\tau$. Czy istnieje taki term $N$, że $\Gamma \vdash_{\lambda P} N : \tau$?*

**Dowód:** W systemie $\lambda P$ mieści się intuicjonistyczna logika pierwszego rzędu (dla $\to$ i $\forall$, ale to wystarczy).

## Silna normalizacja

**Twierdzenie:** *System $\lambda P$ ma własność silnej normalizacji.*

**Dowód:** Zrobimy redukcję do typów prostych.
Na razie mamy wycieranie zależności.

Jeśli $\Gamma \vdash M : \tau$, to $\overline{\Gamma} \vdash \overline{M} : \overline{\tau}$

ale to za mało. Zamiast $\overline{M}$ zrobimy $M^\bullet$.

## Translacja $^\bullet$

Termy $\lambda P$ przerabiamy na termy Churcha w typach prostych.

- $x^\bullet = x$;
- $(MN)^\bullet = M^\bullet N^\bullet$;
- $(\lambda x{:}\tau.\,M)^\bullet = (\lambda z^0 \lambda x^{\overline{\tau}}.\,M^\bullet)\tau^\bullet$.

**O co tu chodzi?**

Przy tej translacji nie gubią się redeksy wewnątrz typu $\tau$.

## Translacja $^\bullet$

Konstruktory i typy też przerabiamy na termy:

- $\alpha^\bullet = x_\alpha$, gdzie $x_\alpha$ jest nową zmienną;
- $(\varphi M)^\bullet = \varphi^\bullet M^\bullet$;
- $(\lambda x{:}\tau.\,\varphi)^\bullet = (\lambda z^0 \lambda x^{\overline{\tau}}.\,\varphi^\bullet)\tau^\bullet$;
- $(\forall x{:}\tau.\,\sigma)^\bullet = y\tau^\bullet(\lambda x^{\overline{\tau}}.\,\sigma^\bullet)$,
  gdzie $y : 0 \to (\overline{\tau} \to 0) \to 0$ jest nową zmienną.

## Translacja $^\bullet$

Dla dowolnego $\Gamma$ definiujemy:

$$\begin{aligned}\overline{\Gamma}^+ = \{&x : \overline{\tau} \mid (x : \tau) \in \Gamma\} \cup \\ &\cup \{x_\alpha : \overline{\boldsymbol{\kappa}} \mid (\alpha : \boldsymbol{\kappa}) \in \Gamma\} \cup \\ &\cup \{y_\rho : 0 \to (\rho \to 0) \to 0 \mid \rho \text{ typ prosty}\}.\end{aligned}$$

**Fakt 1:** Jeśli $\Gamma \vdash \varphi : \boldsymbol{\kappa}$, to $\overline{\Gamma}^+ \vdash \varphi^\bullet : \overline{\boldsymbol{\kappa}}$ w typach prostych.

**Fakt 2:** Jeśli $\Gamma \vdash M : \tau$, to $\overline{\Gamma}^+ \vdash M^\bullet : \overline{\tau}$ w typach prostych.

## Silna normalizacja

**Fakt 3:** If $M_1 \to_\beta M_2$ then $M_1^\bullet \to_\beta^+ M_2^\bullet$.

**Fakt 4:** Jeśli $\varphi_1 \to_\beta \varphi_2$ to $\varphi_1^\bullet \to_\beta^+ \varphi_2^\bullet$.

**Twierdzenie:** *System $\lambda P$ ma własność silnej normalizacji.*

**Dowód:** Nieskończona redukcja w $\lambda P$

$$M_0 \to_\beta M_1 \to_\beta M_2 \to_\beta \cdots$$

tłumaczy się na nieskończoną redukcję w $\lambda_\to$:

$$M_0^\bullet \to_\beta^+ M_1^\bullet \to_\beta^+ M_2^\bullet \to_\beta^+ \cdots$$

## Rachunek konstrukcji

## Dependencies in lambda-calculi

**The origin:** Objects (terms) depend on terms:

$$\lambda x{:}\tau\, M^\sigma : \tau \to \sigma$$

Pattern: $(x : \tau^*) \to \sigma^* : *$ and rule: $(*, *, *)$.

(Arrow is a *constructor* of *kind* $* \Rightarrow * \Rightarrow *$.)

**Polymorphism:** Objects depend on types (system **F**):

$$\Lambda\alpha{:}*\, M^\sigma : \forall\alpha\, \sigma$$

Pattern: $(\alpha : *^\Box) \Rightarrow \sigma^* : *$ and rule: $(\Box, *, *)$.

## Dependencies in lambda-calculi

**Dependent types:** Types depend on objects (system $\lambda\mathbf{P}$):

$$\lambda x{:}\tau\, \sigma(x)^* : \tau \Rightarrow *$$

Pattern: $(x : \tau^*) \Rightarrow *^\Box : \Box$ and rule: $(*, \Box, \Box)$

**What is missing?** Types depend on types (system $\lambda\underline{\omega}$)

$$\lambda\alpha{:}*\, \sigma(\alpha) : * \Rightarrow *$$

Pattern: $(\alpha : *^\Box) \Rightarrow *^\Box : \Box$ and rule: $(\Box, \Box, \Box)$.

## Polimorfizm wyższego rzędu: system $F_\omega$

**Example:** Why is $(\lambda zy.\, y(z\mathsf{I})(z\mathsf{K}))(\lambda x.\, xx)$ untypable in **F**?

Because types we can assign to **I** and **K** differ too much:

$$\mathsf{I} : \forall p.\, p \to p \qquad \mathsf{K} : \forall p.\, p \to (q \to p).$$

There is no common pattern for these two types . . .
unless we write it this way:

$$\mathsf{I}, \mathsf{K} : \forall p.\, p \to \alpha(p).$$

Here, $\alpha$ is a constructor of kind $* \Rightarrow *$.

## Curry-style $\mathbf{F}_\omega$

Let $\alpha : * \Rightarrow *$, $\varphi : (* \Rightarrow *) \Rightarrow (* \Rightarrow *)$. Then:

$\mathsf{K} : \forall p\,(p \to q \to p)$

$\mathsf{2} : \forall\alpha\,(\forall p\,(p \to \alpha(p)) \to \forall p\,(p \to \alpha(\alpha(p))))$

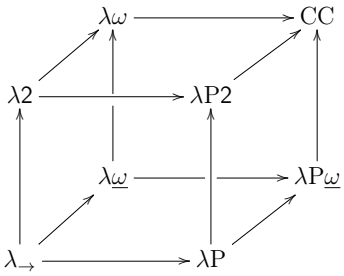
$\mathsf{2} : [\forall\alpha\,(\forall p\,(p \to \alpha(p)) \to \forall p\,(p \to (\alpha(\alpha(p)))))] \to$
$[\forall\alpha\,(\forall p\,(p \to \alpha(p)) \to \forall p\,(p \to (\alpha(\alpha(\alpha(\alpha(p))))))))]$

Therefore $\mathsf{22K} : \forall p\,(p \to q \to q \to q \to q \to p)$

$\mathsf{2} : \forall\varphi([\forall\alpha\,(\forall p\,(p \to \alpha(p)) \to \forall p\,(p \to \varphi(\alpha)(p))] \to$
$[\forall\alpha\,(\forall p\,(p \to \alpha(p)) \to \forall p\,(p \to (\varphi(\varphi(\alpha))(p)))))]$

Therefore $\mathsf{222K} : \forall p\,(p \to q \to q \to \cdots \to q \to q \to p)$

## Barendregt's Cube: combine all dependencies




$\lambda_{\to}$ to typy proste, $\lambda 2$ to system **F**, $\lambda\omega$ to system $\mathbf{F}_{\omega}$.
**CC** to rachunek konstrukcji

## Rachunek konstrukcji: składnia uniwersalna

**Sorty:** $s ::= * \mid \Box$

**Pseudo-termy:**

$$A ::= x \mid s \mid (AA) \mid (\lambda x{:}A\,A) \mid (\Pi x{:}A\,A)$$

**Redukcja:**

$$(\lambda x{:}A.\,B)C \quad \to_\beta \quad B[x := C].$$

## Calculus of Constructions: Raw syntax

**Sorts:**
$s ::= * \mid \Box$;

**Kinds:**
$\boldsymbol{\kappa} ::= * \mid (\Pi x{:}\phi\ \boldsymbol{\kappa}) \mid (\Pi x{:}\boldsymbol{\kappa}\ \boldsymbol{\kappa})$;

**Constructors:**
$\phi ::= \alpha \mid (\forall x{:}\phi\ \phi) \mid (\forall \alpha{:}\boldsymbol{\kappa}\ \phi) \mid (\phi M) \mid (\phi\phi) \mid (\lambda x{:}\phi\ \phi) \mid (\lambda\alpha{:}\kappa\ \phi)$;

**Terms:**
$M ::= x \mid (MM) \mid (M\phi) \mid (\lambda x{:}\phi\ M) \mid (\lambda\alpha{:}\kappa\ M)$;

**Environments:**
$\Gamma ::= \varnothing \mid \Gamma, (x : \phi) \mid \Gamma, (\alpha : \boldsymbol{\kappa})$.

## Rachunek konstrukcji: reguły

(Ax) $\quad \varnothing \vdash * : \Box$

(Var) $\quad \dfrac{\Gamma \vdash A : s}{\Gamma,\, x : A \vdash x : A} \quad (x \notin \mathrm{Dom}(\Gamma))$

(Prod) $\quad \dfrac{\Gamma \vdash A : s_1 \qquad \Gamma,\, x : A \vdash B : s_2}{\Gamma \vdash (\Pi x{:}A.\,B) : s_2} \quad (s_1, s_2 \in \{*, \Box\})$

## Rachunek konstrukcji: reguły

(Prod) $\quad \dfrac{\Gamma \vdash A : s_1 \qquad \Gamma,\, x : A \vdash B : s_2}{\Gamma \vdash (\Pi x{:}A.\,B) : s_2} \quad (s_1, s_2 \in \{*, \Box\})$

(Abs) $\quad \dfrac{\Gamma,\, x : A \vdash B : C \qquad \Gamma \vdash (\Pi x{:}A.\,C) : s}{\Gamma \vdash (\lambda x{:}A.\,B) : (\Pi x{:}A.\,C)}$

(App) $\quad \dfrac{\Gamma \vdash A : (\Pi x{:}B.\,C) \qquad \Gamma \vdash D : B}{\Gamma \vdash (AD) : C[x := D]}$

## Rachunek konstrukcji: reguły

(Weak) $\quad \dfrac{\Gamma \vdash A : B \qquad \Gamma \vdash C : s}{\Gamma,\, x : C \vdash A : B} \quad (x \notin \mathrm{Dom}(\Gamma))$

(Conv) $\quad \dfrac{\Gamma \vdash A : B \qquad \Gamma \vdash B' : s}{\Gamma \vdash A : B'} \quad (B =_\beta B')$

## Rachunek konstrukcji: własności

- Poprawność redukcji:
  Jeśli $\Gamma \vdash A : B$ i $A \twoheadrightarrow A'$ to $\Gamma \vdash A' : B$.
- "Wzmacnianie":
  Jeśli $\Gamma, x : A, \Delta \vdash A : B$,
  oraz $x$ nie jest wolne w $\Delta$, $A$ ani w $B$,
  to $\Gamma, \Delta \vdash A : B$.
- Jednoznaczność typu:
  Jeśli $\Gamma \vdash A : B$ i $\Gamma \vdash A : B'$, to $B =_\beta B'$.
- Silna normalizacja: Jeśli $\Gamma \vdash A : B$, to $A \in SN$.

## Klasyfikacja w rachunku konstrukcji

**Fakt:**
Jeśli $\Gamma \vdash A : B$ to zachodzi dokładnie jeden przypadek:

– albo $\Gamma \vdash B : *$ (czyli $A$ jest obiektem typu $B$),
– albo $\Gamma \vdash B : \Box$ (czyli $A$ jest konstruktorem rodzaju $B$),
– albo $B = \Box$ (czyli $A$ jest rodzajem).

# Rachunek konstrukcji: podsumowanie

- Dwa *sorty*: $*$ oraz $\Box$;
- Aksjomat $* : \Box$;
- Dozwolone są produkty postaci

  $$(x : A^s) \to B(x)^t : t,$$

  gdzie $s$ i $t$ są dowolnymi sortami;
- Te produkty są typami abstrakcji $\lambda x{:}A^s.\, M^{B(x)}$;
- Wierzchołki kostki: ograniczenie w tworzeniu produktów.