

# Równość

Daria Walukiewicz-Chrząszcz

4 kwietnia 2017

# Typowanie jako relacja indukcyjna

```
Inductive exp : Set :=  
| Nat : nat → exp  
| Plus : exp → exp → exp  
| Bool : bool → exp  
| And : exp → exp → exp.
```

## Typowanie jako relacja indukcyjna

Inductive **type** : Set := TNat | TBool.

Inductive **hasType** : exp → type → Prop :=

| HtNat : ∀ n,

**hasType** (Nat n) TNat

| HtPlus : ∀ e1 e2,

**hasType** e1 TNat

  → **hasType** e2 TNat

  → **hasType** (Plus e1 e2) TNat

| HtBool : ∀ b,

**hasType** (Bool b) TBool

| HtAnd : ∀ e1 e2,

**hasType** e1 TBool

  → **hasType** e2 TBool

  → **hasType** (And e1 e2) TBool.

## Rozstrzygalność porównywania typów

Definition eq\_type\_dec :  $\forall t1 t2 : \mathbf{type}, \{t1 = t2\} + \{t1 \neq t2\}$ .  
*decide equality.*

Defined.

# Dowód przez indukcję po dowodzie

```
Lemma hasType_det :  $\forall e t1,$   
  hasType e t1  
   $\rightarrow \forall t2, \mathbf{hasType}$  e t2  
     $\rightarrow t1 = t2.$   
  induction 1; inversion 1; auto.  
Qed.
```

Patrz plik: `hastype.v`

## Konwersja — równość definicyjna, obliczeniowa

ang. *definitional equality*

reguła konwersji

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash A =_{\beta\eta\delta\zeta\iota} B \quad \Gamma \vdash B : s}{\Gamma \vdash M : B}$$

$$\Gamma \vdash A =_{\beta\eta\delta\zeta\iota} B$$

jeśli

- $\Gamma \vdash A \triangleright^* A'$
- $\Gamma \vdash B \triangleright^* B'$
- $A' = B'$  lub  
 $(A' = \lambda x : T. A'' \text{ i } \Gamma, x : T \vdash B' x =_{\beta\eta\delta\zeta\iota} A'')$  lub  
 $(B' = \lambda x : T. B'' \text{ i } \Gamma, x : T \vdash A' x =_{\beta\eta\delta\zeta\iota} B'')$

$$\Gamma \vdash A \triangleright B$$

domknięcie przechodnie redukcji beta, iota, delta i zeta.

## Konwersja — równość definicyjna, obliczeniowa

ang. *definitional equality*

reguła konwersji

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash A =_{\beta\eta\delta\zeta\iota} B \quad \Gamma \vdash B : s}{\Gamma \vdash M : B}$$

$$\Gamma \vdash A =_{\beta\eta\delta\zeta\iota} B$$

jeśli

- $\Gamma \vdash A \triangleright^* A'$
- $\Gamma \vdash B \triangleright^* B'$
- $A' = B'$  lub  
 $(A' = \lambda x : T. A'' \text{ i } \Gamma, x : T \vdash B'x =_{\beta\eta\delta\zeta\iota} A'')$  lub  
 $(B' = \lambda x : T. B'' \text{ i } \Gamma, x : T \vdash A'x =_{\beta\eta\delta\zeta\iota} B'')$

$$\Gamma \vdash A \triangleright B$$

domknięcie przechodnie redukcji beta, iota, delta i zeta.

# Reguły redukcji na przykładzie

```
Definition pred' (x : nat) :=  
  match x with  
  | 0 => 0  
  | S n' => let y := n' in y  
  end.
```

Theorem reduce\_me : pred' 1 = 0.

cbv delta.



## Reguły redukcji w taktyce cbv

```
=====
(fun x : nat ⇒ match x with
  | 0 ⇒ 0
  | S n' ⇒ let y := n' in y
end) 1 = 0
```

cbv beta.

## Reguły redukcji w taktyce cbv

```
=====
match 1 with
| 0  $\Rightarrow$  0
| S n'  $\Rightarrow$  let y := n' in y
end = 0
```

cbv iota.

## Reguły redukcji w taktyce cbv

=====  
(fun n' : nat ⇒ let y := n' in y) 0 = 0

cbv beta.

=====  
(let y := 0 in y) = 0

cbv zeta.

=====  
0 = 0

# eq — równość definiowalna

ang. *propositional equality*  
 definiowana jako relacja indukcyjna

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

```
@eq_refl: forall (A : Type) (x : A), eq A x x
```

eq to równość Leibniza:

```
eq_ind: forall (A : Type) (x : A) (P : A -> Prop),
          P x -> forall y : A, x = y -> P y
```

# eq — równość definiowalna

ang. *propositional equality*  
 definiowana jako relacja indukcyjna

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

```
@eq_refl: forall (A : Type) (x : A), eq A x x
```

eq to równość Leibniza:

```
eq_ind: forall (A : Type) (x : A) (P : A -> Prop),
          P x -> forall y : A, x = y -> P y
```

# eq — równość definiowalna

ang. *propositional equality*  
 definiowana jako relacja indukcyjna

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

```
@eq_refl: forall (A : Type) (x : A), eq A x x
```

eq to równość Leibniza:

```
eq_ind: forall (A : Type) (x : A) (P : A -> Prop),
          P x -> forall y : A, x = y -> P y
```

# eq — równość definiowalna

ang. *propositional equality*  
 definiowana jako relacja indukcyjna

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

```
@eq_refl: forall (A : Type) (x : A), eq A x x
```

eq to równość Leibniza:

```
eq_ind: forall (A : Type) (x : A) (P : A -> Prop),
          P x -> forall y : A, x = y -> P y
```

# Problemy z równością

lemmaUIP jest niedowodliwy:

Lemma lemmaUIP :  $\forall (x : A) (pf : x = x), pf = \text{eq\_refl } x$ .

dowodliwy jest lemma2:

Lemma lemma2 :  $\forall (x : A) (pf : x = x), 0 = \text{match } pf \text{ with eq\_refl} \Rightarrow$   
0 end.



## Dowód lemma2

```
Definition lemma2 :=  
  fun (x : A) (pf : x = x) =>  
    match pf return (0 = match pf with  
      | eq_refl => 0  
    end) with  
    | eq_refl => eq_refl 0  
  end.
```

## Aksjomat UIP\_refl

Check UIP\_refl.

UIP\_refl

$$: \forall (U : \text{Type}) (x : U) (p : x = x), p = \text{eq\_refl } x$$

## UIP\_refl równoważny z aksjomatem Streicher\_K

Check `Streicher_K`.

`Streicher_K`

$$: \forall (U : \text{Type}) (x : U) (P : x = x \rightarrow \text{Prop}), \\ P \text{ eq\_refl} \rightarrow \forall p : x = x, P p$$

Streicher's axiom K jest niesprzeczny z CIC i niedowodliwy w CIC

## Dla typów rozstrzygalnych

czyli takich, że

Variable eq\_dec : forall x y:A, {x = y} + {x <> y}.

zachodzą prawa UIP\_dec, UIP\_refl i K\_dec

forall (x y:A) (p1 p2:x = y), p1 = p2

forall (x y:A) (p:x = x), p = eq\_refl x

forall (x:A) (P:x = x -> Prop), P (eq\_refl x)  
-> forall p:x = x, P p

bez dodatkowych aksjomatów (patrz: moduł Eqdep\_dec z biblioteki standardowej)

Lemat `UIP_refl_nat` jest dowodliwy w Coqu

`UIP_refl_nat`

`:  $\forall (x : nat) (p : x = x), p = eq\_refl\ x$`

(patrz: plik `UIP_refl_nat.v`)

# Problemy z równością cd.

Poniższego twierdzenia “nie da się zapisać”

Theorem vappend\_assoc :  $\forall a b c$   
 $(va : \text{vector } a) (vb : \text{vector } b) (vc : \text{vector } c),$   
 $\text{vappend (vappend } va \text{ } vb) \text{ } vc = \text{vappend } va \text{ (vappend } vb \text{ } vc).$

Error:

The term "vappend va (vappend vb vc)"  
 has type "vector (a + (b + c))"  
 while it is expected to have type "vector (a + b + c)".

## Trzeba wstawić "type-cast"

```
Theorem vappend_assoc :  $\forall a b c$   
  (va : vector a) (vb : vector b) (vc : vector c),  
  vappend (vappend va vb) vc =  
    match Plus.plus_assoc a b cin (_ = X)  
    return vector X with  
    | eq_refl  $\Rightarrow$  vappend va (vappend vb vc)  
  end.
```

# Równość heterogeniczna

```
Inductive JMeq (A : Type) (x : A) :  $\forall B : \text{Type}, B \rightarrow \text{Prop} :=$   
  JMeq_refl : JMeq x x
```

```
Infix "==" := JMeq (at level 70, no associativity).
```



## Związki eq i JMeq

```
JMeq_rect  
  : forall (A : Type) (x : A) (P : forall B : Type, B -> Type),  
    P A x -> forall (B : Type) (b : B), x == b -> P B b
```

```
eq_rect  
  : forall (A : Type) (x : A) (P : A -> Type),  
    P x -> forall y : A, x = y -> P y
```

## Związki eq i JMeq

Lemma eq\_JMeq :  $\forall (A : \text{Type}) (x y : A), x = y \rightarrow x == y$ .  
 intros; rewrite H; reflexivity.  
 Qed.

Ale odwrotna implikacja jest niedowodliwa:

Check *JMeq\_eq*.

*JMeq\_eq*

:  $\forall (A : \text{Type}) (x y : A), x == y \rightarrow x = y$

## Związki eq i JMeq

Lemma eq\_JMeq :  $\forall (A : \text{Type}) (x y : A), x = y \rightarrow x == y$ .  
 intros; rewrite H; reflexivity.  
 Qed.

Ale odwrotna implikacja jest niedowodliwa:

Check *JMeq\_eq*.

*JMeq\_eq*  
 :  $\forall (A : \text{Type}) (x y : A), x == y \rightarrow x = y$

# Aksjomat JMeq\_eq

- może być bezpiecznie dodany do CIC
- może być używany przez rewrite w postaci

JMeq\_ind\_r

```
: forall (A : Type) (x : A) (P : A -> Type),  
P x -> forall y : A, x == y -> P y
```

# Dowód pairC'

Dwa sposoby:

- używający aksjomatu JMeq\_eq
- używający naturalnej zasady indukcji dla JMeq

Użyte aksjomaty sprawdzamy za pomocą:

```
Print Assumptions pairC'.
```

Patrz plik JMeqRew.v

# Dowód pairC'

Dwa sposoby:

- używający aksjomatu JMeq\_eq
- używający naturalnej zasady indukcji dla JMeq

Użyte aksjomaty sprawdzamy za pomocą:

```
Print Assumptions pairC'.
```

Patrz plik JMeqRew.v

# Lemat vappend\_assoc

Patrz plik vappend\_assoc.v