Programowanie Funkcyjne

lato 2013/2014

Jakub Kozik

Informatyka Analityczna
tcs@jagiellonian

Zasady

Punkty

- zadania programistyczne submitowane przez Satori (zadania różnie punktowane, w sumie 80 punktów)
- aby uzyskać pozytywną ocenę należy zaliczyć wszystkie zadania oznaczone jako obowiązkowe
- egzamin (20 punktów)
- aby uzyskać pozytywną ocenę należy z egzaminu uzyskać przynajmniej 10 punktów

Ostateczna ocena

0-50	ndst
50-60	dst
60-70	+dst
70-80	db
80-90	+db
90-100	bdb

Program Wykładu

Część 1: Lambda rachunek i podstawy SML

- podstawowe konstrukcje SML'a i ich odpowiedniki w lambda rachunku
- strategie ewaluacji termów/programów
- system typów (Hindley-Milner)
- podstawowe techniki programowania funkcyjnego

Część 2: Funkcyjne struktury danych

- ocena wydajności programów funkcyjnych
- amortyzowana analiza persystentnych struktur danych
- eliminacja amortyzacji
- implementacje z uleniwianiem/wymuszaniem obliczeń

Część 3: Haskell

- leniwa ewaluacja
- monady
- ...

Literatura

Paweł Urzyczyn, Rachunek Lambda, skrypt dostępny na stronie autora

Robert Harper, Programming in Standard ML, Working Draft dostępny na stronie autora

Chris Okasaki, Purely Functional Data Structures, Cambridge University Press 1999 (wczesna wersja książki (rozprawa doktorska) dostępna na stronie autora)

Simon L. Peyton Jones, The Implementation of Functional Programming Languages, Prentice Hall 1987 (książka w wersji elektronicznej jest udostępniana przez autora)

Bryan O'Sullivan, Don Stewart, John Goerzen, Real World Haskell, O'Reilly Media, 2008 (książka w wersji elektronicznej jest udostępniana przez autorów)

Lambda rachunek - przypomnienie

Składnia

- zmienne są termami (przeliczalny zbiór zmiennych Var)
- jeśli T jest termem a x jest zmienną to $\lambda x.T$ jest termem (abstrakcja)
- jeśli T oraz S są termami to $(T \cdot S)$ jest termem (aplikacja)

Konwencje

- pomijanie $\cdot$ przy aplikacji $(T \cdot S) \equiv (TS)$
- pomijanie nawiasów – domyślne nawiasowanie do lewej
  $RST \equiv ((RS)T)$
- grupowanie abstrahowanych zmiennych $\lambda xy.T \equiv \lambda x.(\lambda y.T)$

Redukcja

β redukcja

$$(\lambda x.T)S \ \rightarrow_\beta \ T[x \leftarrow S]$$

(pod warunkiem że żadne wolne wystąpienie zmiennej w S nie zostaje związane w $T[x \leftarrow S]$)

α równoważność

Termy, które różnią się tylko nazwami zmiennych związanych, są równoważne i można je sobą zastępować.

$$(\lambda s\ z.s(s(z)))(\lambda s\ z.s(s(z)))$$

Obliczenia w lambda rachunku

Postać normalna

Term jest w *postaci (beta) normalnej* jeśli nie zawiera β-redex'u.

program	$\leftrightarrow$	term
ewaluacja programu	$\leftrightarrow$	wykonywanie β redukcji
wynik obliczenia	$\leftrightarrow$	term w postaci normalnej

Theorem (Church-Rosser)

Jeśli $P\ {}_{\beta}\!\twoheadleftarrow M \twoheadrightarrow_{\beta} Q$ *to istnieje* M' *taki że* $P \twoheadrightarrow_{\beta} M'\ {}_{\beta}\!\twoheadleftarrow Q$.

Wniosek

Każdy term ma co najwyżej jedną postać normalną.

W stronę języków funkcyjnych

Wzbogacamy λ rachunek o:

1. wyrażenia `let` i `letrec`
2. pattern-matching lambda abstractions
3. operator []
4. wyrażenia `case`
5. stałe: małe liczby, znaki, funkcje na małych liczbach itp.

let

Składnia (bez pattern-matchingu)

let v = B **in** E

let
    w = A
    v = B
in E

=

let w = A **in**
    (**let** v = B **in** E)

Tłumaczenie

(**let** v = B **in** E) $\equiv$ $((\lambda v.E)B)$

letrec

Składnia (bez pattern-matchingu)

letrec

a = A

b = B

...

n = N

in E

Tłumaczenie (dla jednej zmiennej)

(**letrec** v = B **in** E) $\equiv$ (**let** v = Y (λ v.B) **in** E)

(Y jest kombinatorem punktu stałego)

letrec

```
letrec f= \n. IF (n=1) THEN 1 ELSE (n * (f (n-1))) in f 4
```

$$F = \lambda f\ n.\mathbf{if}(n = 1)\mathbf{then}\ 1\ \mathbf{else}(n * f(n-1))$$

$$\begin{array}{rc}
 & (Y\ F)\ 4 \twoheadrightarrow F\ (Y\ F)\ 4 \\
= & (\lambda f\ n.\mathbf{if}(n = 1)\mathbf{then}\ 1\ \mathbf{else}(n * f(n-1)))\ (Y\ F)\ 4 \\
\twoheadrightarrow & \mathbf{if}(4 = 1)\mathbf{then}\ 1\ \mathbf{else}(4 * ((Y\ F)(4-1))) \\
\twoheadrightarrow & 4 * ((Y\ F)(4-1)) \twoheadrightarrow 4 * (F(Y\ F)(4-1)) \\
\twoheadrightarrow & 4 * (\mathbf{if}(4 - 1 = 1)\mathbf{then}\ 1\ \mathbf{else}((4-1) * ((Y\ F)(4-1-1)))) \\
\twoheadrightarrow & 4 * (\mathbf{if}(3 = 1)\mathbf{then}\ 1\ \mathbf{else}(3 * ((Y\ F)(3-1)))) \\
\twoheadrightarrow & 4 * (3 * ((Y\ F)\ (3-1))) \twoheadrightarrow 4 * (3 * (F\ (Y\ F)\ (3-1))) \\
\twoheadrightarrow & 4 * (3 * (2 * (F\ (Y\ F)\ (2-1)))) \twoheadrightarrow 4 * (3 * (2 * 1)) \\
\twoheadrightarrow & 4 * (3 * 2) \twoheadrightarrow 4 * 6 \twoheadrightarrow 24
\end{array}$$

let ?

Tłumaczenie

$$(\textbf{let } \mathrm{v} = \mathrm{B} \textbf{ in } \mathrm{E}) \quad \equiv \quad ((\lambda v.E)B)$$

1. polimorfizm - system typowania inaczej traktuje konstrukcje `let` i λ abstrakcje
2. wydajność - specyficzna aplikacja $(\lambda v.E)$ jest aplikowana tylko do konkretnego argumentu B
3. `letrec` generuje Y
   1. można wydajniej obliczać bezpośrednio na termach z `letrec`
   2. termy z Y nie mogą być ewaluowane gorliwie

Wzbogacony Rachunek Lambda

Wzbogacamy λ rachunek o:

1. wyrażenia `let` i `letrec`
2. pattern-matching lambda abstractions
3. operator []
4. wyrażenia `case`

Structured Types (algebraic types)

Typy wyliczeniowe

```
datatype color = RED | GREEN | BLUE
datatype bool = TRUE | FALSE
```

Tuples/Typy produktowe

```
datatype pair 'a 'b = PAIR 'a * 'b
datatype triple 'a 'b 'c = TRIPLE 'a * 'b * 'c
```

Union types

```
datatype shape = CIRCLE color * int
        | RECTANGLE color * int *int
datatype intList = INIL | ICONS int * intList
datatype list 'a = NIL | CONS 'a * ('a list)
```

Pattern matching - przykłady

Overlapping patterns & Constant patterns

```
factorial 0 = 1
factorial n = n * factorial (n-1)
```

Nested patterns

```
getEven [] = []
getEven [x] = []
getEven (x:(y:ys)) = y: (getEven ys)
```

Multiple arguments

```
xor FALSE y = y
xor TRUE FALSE = TRUE
xor TRUE TRUE = FALSE
```

Pattern matching - przykłady

Non-exhaustive sets of equations

```
head (x:xs) = x
```

Conditional equations

```
fibb n = 1,  n<2
fibb n = fibb (n-1) + (fibb (n-2))
```

Repeated variables (?)

```
noDups [] = []
noDups [x] = [x]
noDups (x:x:xs) = noDups (x:xs)
noDups (x:y:ys) = x: (noDups (y:ys))
```

Pattern matching w rozszerzonym lambda rachunku

Definicja

Wzorzec (Pattern)

1. *zmienna jest wzorcem*
2. *stała jest wzorcem (`int`, `char`, `bool` itp.)*
3. *jeśli c jest konstruktorem arności r, oraz $p_1, \ldots, p_r$ są wzorcami, to wzorcem jest*

$$(c \;\; p_1 \; \ldots \; p_r)$$

Dodatkowo wszystkie nazwy zmiennych we wzorcu muszą być różne.

Wzorce postaci $(c \;\; p_1 \; \ldots \; p_r)$ nazywamy *sum construction patterns* jeśli c jest konstruktorem dla typu będącego sumą (*union types*).

Wzorce postaci $(c \;\; p_1 \; \ldots \; p_r)$ nazywamy *product construction patterns* jeśli c jest konstruktorem dla typu produktowego (*product types*).

Pattern matching w rozszerzonym lambda rachunku

Rozszerzenie składni - term:

1. wzorzec (!)
2. aplikacja $(T_1 \cdot T_2)$
3. abstrakcja $(\lambda p. T)$ (gdzie p jest wzorcem a T jest termem)
4. ...
5. `let, letrec` ...

Przykład

`fst (x,y) = x` $\quad \longrightarrow \quad$ $fst = \lambda(PAIR\ x\ y).x$

Pattern matching w rozszerzonym lambda rachunku

Problem

```
null NIL = true
null (CONS x xs) = false
```

Rozszerzenie składni - term:

1. wzorzec (!) (w tym stała `FAIL`)
2. aplikacja $(T_1 \cdot T_2)$
3. abstrakcja $(\lambda p.T)$ (gdzie p jest wzorcem a T jest termem)
4. operator binarny [] (będziemy zapisywać infiksowo)
5. `let`, `letrec` ...

operator []

$$
\begin{aligned}
a[]b &= a && \text{jeśli } a \neq FAIL \text{ oraz } a \neq \bot \\
FAIL[]b &= b \\
\bot[]b &= \bot
\end{aligned}
$$

Pattern matching w rozszerzonym lambda rachunku

Przykład

```
null NIL = true
null (CONS x xs) = false
```

$$\lambda v.(\ ((\lambda NIL.true)v)\ []\ ((\lambda(CONS\ x\ xs).false)v)\ )$$

Przykład

```
reflect (LEAF n) = LEAF n
reflect (BRANCH t1 t2) = BRANCH (reflect t2) (reflect t1)
```

$$
\begin{array}{ll}
 & letrec \\
reflect = \lambda t.(& (\lambda(LEAF\ n).LEAF\ n)t \\
\quad [] & ((\lambda(BRANCH\ t1\ t2).BRANCH\ (reflect\ t2)(reflect\ t1))t) \\
\quad [] & ERROR) \\
 & in\ \ E
\end{array}
$$

Pattern matching - wiele argumentów

```
f p1 ... p_m = E
```

$$\lambda v_1 \dots \lambda v_n.((\lambda p_1 \dots \lambda p_m.E) v_1 \dots v_m \; [] \; ERROR)$$

Dodatkowa reguła dla FAIL

$$(FAIL\ A) \rightarrow FAIL$$

Przykład

```
xor False y = y
xor True False = True
xor True True = False
```

$$\begin{aligned} xor = \quad & \lambda x.\lambda y.(& (\lambda FALSE.\lambda y.y)\ x\ y \\ & [] & ((\lambda TRUE.\lambda FALSE.TRUE)\ x\ y) \\ & [] & ((\lambda TRUE.\lambda TRUE.FALSE)\ x\ y) \\ & [] & ERROR) \end{aligned}$$

Pattern matching - guards

Przykład

```
foo (x:xs) = x, x<0
foo (x:[]) = x
foo (x:xs) = foo xs
```

$$\begin{aligned} foo = \quad & \lambda v.(((\lambda(CONS\ x\ xs).IF\ (x < 0)\ THEN\ x\ ELSE\ FAIL)\ v) \\ & \;[]((\lambda(CONS\ x\ NIL).x)\ v) \\ & \;[]((\lambda(CONS\ x\ xs).foo\ xs)\ v) \\ & \;[]ERROR) \end{aligned}$$

Przykład - guard TRUE

```
fac n = 1,  n=0
fac n = n * (factorial (n-1))
```

$$\begin{aligned} fac = \quad & \lambda v.((\lambda n.(IF\ (n = 0)\ THEN\ 1\ ELSE\ (n * (fac(n-1))))\ v \\ & \;\;[]ERROR) \end{aligned}$$

Powtórzone zmienne

```
nasty x x True = 1
nasty x y z    = 2

nasty' x y True = 1, x=y
nasty' x y z    = 2
```

$$(nasty\ \bot\ 3\ False) \neq (nasty'\ \bot\ 3\ False)$$

```
multi p q q p = 1
multi p q r s = 2
```

$$multi\ \bot\ 1\ 2\ 3\ = ?$$

Dopasowania typów algebraicznych

$$
\begin{aligned}
(\lambda(s\ p_1 \ldots p_r).E)\ (s\ a_1 \ldots a_r) &\to && (\lambda p_1 \ldots p_r.E)\ a_1 \ldots a_r \\
(\lambda(s\ p_1 \ldots p_r).E)\ (s'\ a_1 \ldots a_r) &\to && \mathit{FAIL} \\
\mathit{Eval}[[(\lambda(s\ p_1 \ldots p_r).E)]]\ \bot &= && \bot
\end{aligned}
$$

(Leniwe) Dopasowywanie typów produktowych

Przykład

```
zeroAny     x     = 0
zeroList    []    = 0
zeroPair    (x,y)= 0
```

$$\mathit{zeroAny}\ \bot = 0$$
$$\mathit{zeroList}\ \bot = \bot$$
$$\mathit{zeroPair}\ \bot = ?$$

Destruktory par

```
addPair     (x,y) = x+y
```

$$\mathit{addPair} = \lambda p.((\texttt{SEL-PAIR-1}\ p) + (\texttt{SEL-PAIR-2}\ p))$$

(Leniwe) Dopasowywanie typów produktowych

Destruktory par

```
addPair    (x,y) = x+y
```

$$addPair = \lambda p.((\texttt{SEL-PAIR-1}\ p) + (\texttt{SEL-PAIR-2}\ p))$$

Ogólnie

$$(\lambda(t\ p_1 \ldots p_r).E)\ a \quad \rightarrow \quad (\lambda p_1 \ldots p_r.E)\ (\texttt{SEL-t-1}\ a) \ldots (\texttt{SEL-t-r}\ a)$$

Case expressions

Wzbogacamy λ rachunek o:

1. wyrażenia `let` i `letrec`
2. pattern-matching lambda abstractions
3. operator []
4. wyrażenia `case`

Case expressions

$$
\begin{array}{ll}
\texttt{case}\ v\ \texttt{of} & \\
\quad c_1\ \ v_{1,1}\ldots v_{1,r_1} & \Rightarrow E_1 \\
\quad \ldots & \\
\quad c_n\ \ v_{n,1}\ldots v_{n,r_n} & \Rightarrow E_n
\end{array}
$$

$$
\begin{array}{lll}
\mathit{reflect} = & \lambda t.\texttt{case}\ t\ \texttt{of} & \\
& \quad \mathit{LEAF}\ n & \Rightarrow \mathit{LEAF}\ n \\
& \quad \mathit{BRANCH}\ t1\ t2 & \Rightarrow \mathit{BRANCH}\ (\mathit{reflect}\ t1)\ (\mathit{reflect}\ t2)
\end{array}
$$

$$
\begin{array}{l}
\ (\lambda(c_1\ \ v_{1,1}\ldots v_{1,r_1}).E_1)v \\
[]\ldots \\
[](\lambda(c_n\ \ v_{1,1}\ldots v_{1,r_n}).E_n)v
\end{array}
$$

Powrót do zwykłego λ rachunku

…

Twierdzenie Churcha-Rossera

Twierdzenie (Church-Rosser)

Jeśli $P\,{}_\beta\!\twoheadleftarrow M \twoheadrightarrow_\beta Q$ *to istnieje* M' *taki że* $P \twoheadrightarrow_\beta M'\,{}_\beta\!\twoheadleftarrow Q$.

Wniosek

Każdy λ-term ma co najwyżej jedną postać normalną.

Twierdzenie Churcha-Rossera

Twierdzenie (Church-Rosser)

Jeśli $P\ {}_{\beta}\!\twoheadleftarrow M \twoheadrightarrow_{\beta} Q$ *to istnieje* M' *taki że* $P \twoheadrightarrow_{\beta} M'\ {}_{\beta}\!\twoheadleftarrow Q$.

Wniosek

Każdy λ-term ma co najwyżej jedną postać normalną.

C-R rozgrzewka

Słaba własność Churcha-Rossera (WCR)

Relacja $\rightarrow$ ma *słabą własność Churcha-Rossera* jeśli $b \leftarrow a \rightarrow c$ implikuje istnienie d takiego że $b \twoheadrightarrow d \twoheadleftarrow c$.

$$\mathrm{WCR} \not\Rightarrow \mathrm{CR}$$

Własność silnej normalizacji (SN)

Relacja $\rightarrow$ ma własność *silnej normalizacji* jeśli nie istnieją nieskończone ciągi $(a_n)_{\in\mathbb{N}}$ takie że $a_n \rightarrow a_{n+1}$, dla każdego $n \in \mathbb{N}$.

Lemat Newmanna

$$\mathrm{WCR} + \mathrm{SN} \Rightarrow \mathrm{CR}$$

C-R rozgrzewka

Słaba własność Churcha-Rossera (WCR)

Relacja $\rightarrow$ ma *słabą własność Churcha-Rossera* jeśli $b \leftarrow a \rightarrow c$ implikuje istnienie d takiego że $b \twoheadrightarrow d \twoheadleftarrow c$.

$$\text{WCR} \not\Rightarrow \text{CR}$$

Własność silnej normalizacji (SN)

Relacja $\rightarrow$ ma własność *silnej normalizacji* jeśli nie istnieją nieskończone ciągi $(a_n)_{\in\mathbb{N}}$ takie że $a_n \rightarrow a_{n+1}$, dla każdego $n \in \mathbb{N}$.

Lemat Newmanna

$$\text{WCR} + \text{SN} \Rightarrow \text{CR}$$

C-R rozgrzewka

Słaba własność Churcha-Rossera (WCR)

Relacja $\to$ ma *słabą własność Churcha-Rossera* jeśli $b \leftarrow a \to c$ implikuje istnienie d takiego że $b \twoheadrightarrow d \twoheadleftarrow c$.

$$\mathrm{WCR} \not\Rightarrow \mathrm{CR}$$

Własność silnej normalizacji (SN)

Relacja $\to$ ma własność *silnej normalizacji* jeśli nie istnieją nieskończone ciągi $(a_n)_{\in \mathbb{N}}$ takie że $a_n \to a_{n+1}$, dla każdego $n \in \mathbb{N}$.

Lemat Newmanna

$$\mathrm{WCR} + \mathrm{SN} \Rightarrow \mathrm{CR}$$

C-R rozgrzewka

Słaba własność Churcha-Rossera (WCR)

Relacja $\rightarrow$ ma *słabą własność Churcha-Rossera* jeśli $b \leftarrow a \rightarrow c$ implikuje istnienie d takiego że $b \twoheadrightarrow d \twoheadleftarrow c$.

$$\text{WCR} \not\Rightarrow \text{CR}$$

Własność silnej normalizacji (SN)

Relacja $\rightarrow$ ma własność *silnej normalizacji* jeśli nie istnieją nieskończone ciągi $(a_n)_{\in \mathbb{N}}$ takie że $a_n \rightarrow a_{n+1}$, dla każdego $n \in \mathbb{N}$.

Lemat Newmanna

$$\text{WCR} + \text{SN} \Rightarrow \text{CR}$$

C-R dla $\longrightarrow_\beta$

Definicja (1)

1. $x \xrightarrow{1} x$ *gdy* x *jest zmienną,*
2. *jeśli* $M \xrightarrow{1} M'$, *to* $\lambda x.M \xrightarrow{1} \lambda x.M'$,
3. *jeśli* $M \xrightarrow{1} M'$ *oraz* $N \xrightarrow{1} N'$ *to:*
    1. $MN \xrightarrow{1} M'N'$,
    2. *oraz* $(\lambda x.M)N \xrightarrow{1} M'[x := N']$.

Definicja (•)

1. $x^\bullet = x$ *gdy* x *jest zmienną,*
2. $(\lambda x.M)^\bullet = \lambda x.M^\bullet$,
3. $(MN)^\bullet = M^\bullet N^\bullet$, *gdy* MN *nie jest redeksem,*
4. $((\lambda x.M)N)^\bullet = M^\bullet[x := N^\bullet]$.

C-R dla $\longrightarrow_\beta$

Definicja (1)

1. $x \xrightarrow{1} x$ gdy x jest zmienną,
2. jeśli $M \xrightarrow{1} M'$, to $\lambda x.M \xrightarrow{1} \lambda x.M'$,
3. jeśli $M \xrightarrow{1} M'$ oraz $N \xrightarrow{1} N'$ to:
   1. $MN \xrightarrow{1} M'N'$,
   2. oraz $(\lambda x.M)N \xrightarrow{1} M'[x := N']$.

Definicja (•)

1. $x^\bullet = x$ gdy x jest zmienną,
2. $(\lambda x.M)^\bullet = \lambda x.M^\bullet$,
3. $(MN)^\bullet = M^\bullet N^\bullet$, gdy MN nie jest redeksem,
4. $((\lambda x.M)N)^\bullet = M^\bullet[x := N^\bullet]$.

C-R dla $\longrightarrow_\beta$

Definicja (1)

1. $x \xrightarrow{1} x$ *gdy* x *jest zmienną,*
2. *jeśli* $M \xrightarrow{1} M'$, *to* $\lambda x.M \xrightarrow{1} \lambda x.M'$,
3. *jeśli* $M \xrightarrow{1} M'$ *oraz* $N \xrightarrow{1} N'$ *to:*
   1. $MN \xrightarrow{1} M'N'$,
   2. *oraz* $(\lambda x.M)N \xrightarrow{1} M'[x := N']$.

Definicja (•)

1. $x^\bullet = x$ *gdy* x *jest zmienną,*
2. $(\lambda x.M)^\bullet = \lambda x.M^\bullet$,
3. $(MN)^\bullet = M^\bullet N^\bullet$, *gdy* MN *nie jest redeksem,*
4. $((\lambda x.M)N)^\bullet = M^\bullet[x := N^\bullet]$.

C-R dla $\rightarrow_\beta$

Definicja (1)

1. $x \xrightarrow{1} x$ *gdy* x *jest zmienną,*
2. *jeśli* $M \xrightarrow{1} M'$, *to* $\lambda x.M \xrightarrow{1} \lambda x.M'$,
3. *jeśli* $M \xrightarrow{1} M'$ *oraz* $N \xrightarrow{1} N'$ *to:*
   1. $MN \xrightarrow{1} M'N'$,
   2. *oraz* $(\lambda x.M)N \xrightarrow{1} M'[x := N']$.

Definicja ($\bullet$)

1. $x^\bullet = x$ *gdy* x *jest zmienną,*
2. $(\lambda x.M)^\bullet = \lambda x.M^\bullet$,
3. $(MN)^\bullet = M^\bullet N^\bullet$, *gdy* MN *nie jest redeksem,*
4. $((\lambda x.M)N)^\bullet = M^\bullet[x := N^\bullet]$.

C-R dla $\longrightarrow_\beta$

Definicja (1)

1. $x \xrightarrow{1} x$ *gdy* x *jest zmienną,*
2. *jeśli* $M \xrightarrow{1} M'$, *to* $\lambda x.M \xrightarrow{1} \lambda x.M'$,
3. *jeśli* $M \xrightarrow{1} M'$ *oraz* $N \xrightarrow{1} N'$ *to:*
   1. $MN \xrightarrow{1} M'N'$,
   2. *oraz* $(\lambda x.M)N \xrightarrow{1} M'[x := N']$.

Definicja (•)

1. $x^\bullet = x$ *gdy* x *jest zmienną,*
2. $(\lambda x.M)^\bullet = \lambda x.M^\bullet$,
3. $(MN)^\bullet = M^\bullet N^\bullet$, *gdy* MN *nie jest redeksem,*
4. $((\lambda x.M)N)^\bullet = M^\bullet[x := N^\bullet]$.

C-R dla $\longrightarrow_\beta$

Definicja (1)

1. $x \xrightarrow{1} x$ *gdy* x *jest zmienną,*
2. *jeśli* $M \xrightarrow{1} M'$, *to* $\lambda x.M \xrightarrow{1} \lambda x.M'$,
3. *jeśli* $M \xrightarrow{1} M'$ *oraz* $N \xrightarrow{1} N'$ *to:*
   1. $MN \xrightarrow{1} M'N'$,
   2. *oraz* $(\lambda x.M)N \xrightarrow{1} M'[x := N']$.

Definicja ($\bullet$)

1. $x^\bullet = x$ *gdy* x *jest zmienną,*
2. $(\lambda x.M)^\bullet = \lambda x.M^\bullet$,
3. $(MN)^\bullet = M^\bullet N^\bullet$, *gdy* MN *nie jest redeksem,*
4. $((\lambda x.M)N)^\bullet = M^\bullet[x := N^\bullet]$.

C-R dla $\to_\beta$

Definicja (1)

1. $x \xrightarrow{1} x$ *gdy* x *jest zmienną,*
2. *jeśli* $M \xrightarrow{1} M'$, *to* $\lambda x.M \xrightarrow{1} \lambda x.M'$,
3. *jeśli* $M \xrightarrow{1} M'$ *oraz* $N \xrightarrow{1} N'$ *to:*
    1. $MN \xrightarrow{1} M'N'$,
    2. *oraz* $(\lambda x.M)N \xrightarrow{1} M'[x := N']$.

Definicja (•)

1. $x^\bullet = x$ *gdy* x *jest zmienną,*
2. $(\lambda x.M)^\bullet = \lambda x.M^\bullet$,
3. $(MN)^\bullet = M^\bullet N^\bullet$, *gdy* MN *nie jest redeksem,*
4. $((\lambda x.M)N)^\bullet = M^\bullet[x := N^\bullet]$.

C-R dla $\to_\beta$

Definicja (1)

1. $x \xrightarrow{1} x$ *gdy* x *jest zmienną,*
2. *jeśli* $M \xrightarrow{1} M'$, *to* $\lambda x.M \xrightarrow{1} \lambda x.M'$,
3. *jeśli* $M \xrightarrow{1} M'$ *oraz* $N \xrightarrow{1} N'$ *to:*
    1. $MN \xrightarrow{1} M'N'$,
    2. *oraz* $(\lambda x.M)N \xrightarrow{1} M'[x := N']$.

Definicja (•)

1. $x^\bullet = x$ *gdy* x *jest zmienną,*
2. $(\lambda x.M)^\bullet = \lambda x.M^\bullet$,
3. $(MN)^\bullet = M^\bullet N^\bullet$, *gdy* MN *nie jest redeksem,*
4. $((\lambda x.M)N)^\bullet = M^\bullet[x := N^\bullet]$.

C-R dla $\to_\beta$

Definicja (1)

1. $x \xrightarrow{1} x$ *gdy* x *jest zmienną,*
2. *jeśli* $M \xrightarrow{1} M'$, *to* $\lambda x.M \xrightarrow{1} \lambda x.M'$,
3. *jeśli* $M \xrightarrow{1} M'$ *oraz* $N \xrightarrow{1} N'$ *to:*
   1. $MN \xrightarrow{1} M'N'$,
   2. *oraz* $(\lambda x.M)N \xrightarrow{1} M'[x := N']$.

Definicja ($\bullet$)

1. $x^\bullet = x$ *gdy* x *jest zmienną,*
2. $(\lambda x.M)^\bullet = \lambda x.M^\bullet$,
3. $(MN)^\bullet = M^\bullet N^\bullet$, *gdy* MN *nie jest redeksem,*
4. $((\lambda x.M)N)^\bullet = M^\bullet[x := N^\bullet]$.

C-R dla $\to_\beta$

Definicja (1)

1. $x \xrightarrow{1} x$ *gdy* x *jest zmienną,*
2. *jeśli* $M \xrightarrow{1} M'$, *to* $\lambda x.M \xrightarrow{1} \lambda x.M'$,
3. *jeśli* $M \xrightarrow{1} M'$ *oraz* $N \xrightarrow{1} N'$ *to:*
   1. $MN \xrightarrow{1} M'N'$,
   2. *oraz* $(\lambda x.M)N \xrightarrow{1} M'[x := N']$.

Definicja (•)

1. $x^\bullet = x$ *gdy* x *jest zmienną,*
2. $(\lambda x.M)^\bullet = \lambda x.M^\bullet$,
3. $(MN)^\bullet = M^\bullet N^\bullet$, *gdy* MN *nie jest redeksem,*
4. $((\lambda x.M)N)^\bullet = M^\bullet[x := N^\bullet]$.

C-R dla $\to_\beta$

Definicja (1)

1. $x \xrightarrow{1} x$ *gdy* x *jest zmienną,*
2. *jeśli* $M \xrightarrow{1} M'$, *to* $\lambda x.M \xrightarrow{1} \lambda x.M'$,
3. *jeśli* $M \xrightarrow{1} M'$ *oraz* $N \xrightarrow{1} N'$ *to:*
   1. $MN \xrightarrow{1} M'N'$,
   2. *oraz* $(\lambda x.M)N \xrightarrow{1} M'[x := N']$.

Definicja (•)

1. $x^\bullet = x$ *gdy* x *jest zmienną,*
2. $(\lambda x.M)^\bullet = \lambda x.M^\bullet$,
3. $(MN)^\bullet = M^\bullet N^\bullet$, *gdy* MN *nie jest redeksem,*
4. $((\lambda x.M)N)^\bullet = M^\bullet[x := N^\bullet]$.

C-R dla $\rightarrow_\beta$

Lemat

1. Dla dowolnego M mamy $M \xrightarrow{1} M$ oraz $M \xrightarrow{1} M^\bullet$.
2. Jeśli $M \xrightarrow{1} M'$ oraz $N \xrightarrow{1} N'$ to $M[x := N] \xrightarrow{1} M'[x := N']$.
3. Jeśli $M \xrightarrow{1} M'$ to $M' \xrightarrow{1} M^\bullet$.

Wnioski

1. Relacja $\xrightarrow{1}$ ma własność rombu.
2. Relacja $\rightarrow_\beta$ ma własność C-R.

C-R dla $\longrightarrow_\beta$

Lemat

1. Dla dowolnego M mamy $M \xrightarrow{1} M$ oraz $M \xrightarrow{1} M^\bullet$.
2. Jeśli $M \xrightarrow{1} M'$ oraz $N \xrightarrow{1} N'$ to $M[x := N] \xrightarrow{1} M'[x := N']$.
3. Jeśli $M \xrightarrow{1} M'$ to $M' \xrightarrow{1} M^\bullet$.

Wnioski

1. Relacja $\xrightarrow{1}$ ma własność rombu.
2. Relacja $\longrightarrow_\beta$ ma własność C-R.

C-R dla $\longrightarrow_\beta$

Lemat

1. Dla dowolnego M mamy $M \xrightarrow{1} M$ oraz $M \xrightarrow{1} M^\bullet$.
2. Jeśli $M \xrightarrow{1} M'$ oraz $N \xrightarrow{1} N'$ to $M[x := N] \xrightarrow{1} M'[x := N']$.
3. Jeśli $M \xrightarrow{1} M'$ to $M' \xrightarrow{1} M^\bullet$.

Wnioski

1. Relacja $\xrightarrow{1}$ ma własność rombu.
2. Relacja $\longrightarrow_\beta$ ma własność C-R.

C-R dla $\longrightarrow_\beta$

Lemat

1. Dla dowolnego M mamy $M \xrightarrow{1} M$ oraz $M \xrightarrow{1} M^\bullet$.
2. Jeśli $M \xrightarrow{1} M'$ oraz $N \xrightarrow{1} N'$ to $M[x := N] \xrightarrow{1} M'[x := N']$.
3. Jeśli $M \xrightarrow{1} M'$ to $M' \xrightarrow{1} M^\bullet$.

Wnioski

1. Relacja $\xrightarrow{1}$ ma własność rombu.
2. Relacja $\longrightarrow_\beta$ ma własność C-R.

C-R dla $\longrightarrow_\beta$

Lemat

1. Dla dowolnego M mamy $M \xrightarrow{1} M$ oraz $M \xrightarrow{1} M^\bullet$.
2. Jeśli $M \xrightarrow{1} M'$ oraz $N \xrightarrow{1} N'$ to $M[x := N] \xrightarrow{1} M'[x := N']$.
3. Jeśli $M \xrightarrow{1} M'$ to $M' \xrightarrow{1} M^\bullet$.

Wnioski

1. Relacja $\xrightarrow{1}$ ma własność rombu.
2. Relacja $\longrightarrow_\beta$ ma własność C-R.

C-R dla $\longrightarrow_\beta$

Lemat

1. Dla dowolnego M mamy $M \xrightarrow{1} M$ oraz $M \xrightarrow{1} M^\bullet$.
2. Jeśli $M \xrightarrow{1} M'$ oraz $N \xrightarrow{1} N'$ to $M[x := N] \xrightarrow{1} M'[x := N']$.
3. Jeśli $M \xrightarrow{1} M'$ to $M' \xrightarrow{1} M^\bullet$.

Wnioski

1. Relacja $\xrightarrow{1}$ ma własność rombu.
2. Relacja $\longrightarrow_\beta$ ma własność C-R.

C-R dla $\to_\beta$

Lemat

1. Dla dowolnego M mamy $M \xrightarrow{1} M$ oraz $M \xrightarrow{1} M^\bullet$.
2. Jeśli $M \xrightarrow{1} M'$ oraz $N \xrightarrow{1} N'$ to $M[x := N] \xrightarrow{1} M'[x := N']$.
3. Jeśli $M \xrightarrow{1} M'$ to $M' \xrightarrow{1} M^\bullet$.

Wnioski

1. Relacja $\xrightarrow{1}$ ma własność rombu.
2. Relacja $\to_\beta$ ma własność C-R.

```
red((\x.A) B) = let
          B'= red B
     in red( A [x:=B'] )
```

Eager

$$
\begin{array}{rcl}
 & (\lambda xy.x)(\lambda v.v)(\underline{(\lambda z.f(zz))(\lambda z.f(zz))}) & \equiv KI(Yf) \\
\to_\beta & (\lambda xy.x)(\lambda v.v)(f(\underline{(\lambda z.f(zz))(\lambda z.f(zz))})) & \equiv KI(f(Yf)) \\
\to_\beta & (\lambda xy.x)(\lambda v.v)(f(f(\underline{(\lambda z.f(zz))(\lambda z.f(zz))}))) & \equiv KI(f(f(Yf))) \\
\to_\beta & \ldots &
\end{array}
$$

$$
\begin{array}{rcl}
 & \underline{(\lambda xy.x)(\lambda v.v)}((\lambda z.f(zz))(\lambda z.f(zz))) & \equiv KI(Yf) \\
\to_\beta & \underline{(\lambda y.\lambda v.v)((\lambda z.f(zz))(\lambda z.f(zz)))} & \equiv .. \\
\to_\beta & \lambda v.v & \equiv I
\end{array}
$$

Definicja

Ciąg redukcji:

$$M_0 \to M_1 \to \cdots \to M_n$$

nazywamy standardowym jeśli w kroku $M_i \to M_{i+1}$ *redukujemy redeks zaczynający się nie dalej od początku termu niż redeks redukowany w kroku* $M_{i+1} \to M_{i+2}$.

Twierdzenie

Jeśli $M \twoheadrightarrow_\beta N$ *to istnieje standardowa redukcja z* M *do* N.

normal order

```
red((\x.A)B)=red(A[x:=B])
```

eager

```
red((\x.A) B) = let
          B'= red B
     in red( A [x:=B'] )
```

$$
\begin{array}{lcl}
 & \underline{(\lambda xy.x)(\lambda v.v)}((\lambda z.f(zz))(\lambda z.f(zz))) & \equiv KI(Yf) \\
\to_\beta & \underline{(\lambda y.\lambda v.v)((\lambda z.f(zz))(\lambda z.f(zz)))} & \equiv .. \\
\to_\beta & \lambda v.v & \equiv I
\end{array}
$$

Definicja (Czołowa postać normalna)

Każdy term ma jedną z poniższych postaci:

1. $\lambda\vec{x}.x\vec{R}$
2. $\lambda\vec{x}.\underline{(\lambda y.P)Q}\vec{R}$.

Termy postaci (1) są w czołowej postaci normalnej (head normal form). Dla termów postaci (2), podkreślony redeks nazywamy redeksem czołowym.

Redukcję redeksu czołowego nazywamy *redukcją czołową* i oznaczamy

$$M \xrightarrow{h} N.$$

Pozostałe redukcje nazywamy wewnętrznymi i oznaczamy

$$M \xrightarrow{i} N.$$

Leftmost outermost

$$
\begin{array}{rc}
\underline{4}(\lambda x.SKKx) \equiv & (\lambda sz.s(s(s(s(z)))))(\lambda x.SKKx) \\
\to_\beta & \lambda z.(\lambda x.SKKx)((\lambda x.SKKx)((\lambda x.SKKx)((\lambda x.SKKx)(z)))) \\
\twoheadrightarrow_\beta & \lambda z.(\lambda x.x)((\lambda x.SKKx)((\lambda x.SKKx)((\lambda x.SKKx)(z)))) \\
\to_\beta & \lambda z.(\lambda x.SKKx)((\lambda x.SKKx)((\lambda x.SKKx)(z))) \\
\twoheadrightarrow_\beta & \lambda z.(\lambda x.x)((\lambda x.SKKx)((\lambda x.SKKx)(z))) \\
\to_\beta & \lambda z.(\lambda x.SKKx)((\lambda x.SKKx)(z)) \\
\twoheadrightarrow_\beta & \lambda z.(\lambda x.x)((\lambda x.SKKx)(z)) \\
\to_\beta & \lambda z.(\lambda x.SKKx)(z) \\
\twoheadrightarrow_\beta & \lambda z.(\lambda x.x)(z) \\
\to_\beta & \lambda z.z
\end{array}
$$

Lazy

$$
\begin{aligned}
\underline{4}(\lambda x.SKKx) \equiv\;& (\lambda sz.s(s(s(s(z)))))(\lambda x.SKKx) && \\
\to_\beta\;& (\lambda z.r1(r1(r1(r1(z))))) && \text{where } r1 = (\lambda x.SKKx) \\
\twoheadrightarrow_\beta\;& (\lambda z.r1(r1(r1(r1(z))))) && \text{where } r1 = \lambda x.x \\
\to_\beta\;& (\lambda z.r1(r1(r1(z)))) && \text{where } r1 = \lambda x.x \\
\to_\beta\;& (\lambda z.r1(r1(z))) && \text{where } r1 = \lambda x.x \\
\to_\beta\;& (\lambda z.r1(z)) && \text{where } r1 = \lambda x.x \\
\to_\beta\;& (\lambda z.z) &&
\end{aligned}
$$

Typy proste

Składnia

- typy atomowe $p, q, r, \ldots$ są typami,
- jeśli σ i τ są typami to $\sigma \to \tau$ jest typem.

Reguły wnioskowania

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau} \quad [\mathtt{Var}]$$

$$\frac{\Gamma \vdash e_0 : \tau \to \tau' \qquad \Gamma \vdash e_1 : \tau}{\Gamma \vdash e_0\ e_1 : \tau'} \quad [\mathtt{App}]$$

$$\frac{\Gamma,\ x : \tau \vdash e : \tau'}{\Gamma \vdash \lambda\, x\ .\ e : \tau \to \tau'} \quad [\mathtt{Abs}]$$

Przykład

$$SKK : bool \to bool$$

gdzie $(S = \lambda abc.(ac)(bc),\ K = \lambda ab.a))$

Typy proste

Składnia

- typy atomowe $p, q, r, \ldots$ są typami,
- jeśli σ i τ są typami to $\sigma \to \tau$ jest typem.

Reguły wnioskowania

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau} \quad [\mathtt{Var}]$$

$$\frac{\Gamma \vdash e_0 : \tau \to \tau' \qquad \Gamma \vdash e_1 : \tau}{\Gamma \vdash e_0\ e_1 : \tau'} \quad [\mathtt{App}]$$

$$\frac{\Gamma,\ x : \tau \vdash e : \tau'}{\Gamma \vdash \lambda\, x\ .\ e : \tau \to \tau'} \quad [\mathtt{Abs}]$$

Przykład

$$SKK : bool \to bool$$

gdzie $(S = \lambda abc.(ac)(bc),\ K = \lambda ab.a))$

Typy proste

Składnia

- typy atomowe $p, q, r, \ldots$ są typami,
- jeśli σ i τ są typami to $\sigma \to \tau$ jest typem.

Reguły wnioskowania

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau} \quad [\mathtt{Var}]$$

$$\frac{\Gamma \vdash e_0 : \tau \to \tau' \qquad \Gamma \vdash e_1 : \tau}{\Gamma \vdash e_0\ e_1 : \tau'} \quad [\mathtt{App}]$$

$$\frac{\Gamma,\ x : \tau \vdash e : \tau'}{\Gamma \vdash \lambda\, x\ .\ e : \tau \to \tau'} \quad [\mathtt{Abs}]$$

Przykład

$$SKK : bool \to bool$$

gdzie $(S = \lambda abc.(ac)(bc),\ K = \lambda ab.a))$

Typy proste

Składnia

- typy atomowe $p, q, r, \ldots$ są typami,
- jeśli σ i τ są typami to $\sigma \to \tau$ jest typem.

Reguły wnioskowania

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau} \quad [\mathtt{Var}]$$

$$\frac{\Gamma \vdash e_0 : \tau \to \tau' \qquad \Gamma \vdash e_1 : \tau}{\Gamma \vdash e_0\ e_1 : \tau'} \quad [\mathtt{App}]$$

$$\frac{\Gamma,\ x : \tau \vdash e : \tau'}{\Gamma \vdash \lambda\, x\ .\ e : \tau \to \tau'} \quad [\mathtt{Abs}]$$

Przykład

$$SKK : bool \to bool$$

gdzie ($S = \lambda abc.(ac)(bc)$, $K = \lambda ab.a$))

Typy proste

Składnia

- typy atomowe $p, q, r, \ldots$ są typami,
- jeśli σ i τ są typami to $\sigma \to \tau$ jest typem.

Reguły wnioskowania

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau} \qquad [\mathtt{Var}]$$

$$\frac{\Gamma \vdash e_0 : \tau \to \tau' \qquad \Gamma \vdash e_1 : \tau}{\Gamma \vdash e_0\ e_1 : \tau'} \qquad [\mathtt{App}]$$

$$\frac{\Gamma,\ x : \tau \vdash e : \tau'}{\Gamma \vdash \lambda\, x\ .\ e : \tau \to \tau'} \qquad [\mathtt{Abs}]$$

Przykład

$$SKK : bool \to bool$$

gdzie ($S = \lambda abc.(ac)(bc)$, $K = \lambda ab.a$))

Warianty

Wariant Curry'ego

$$\lambda a\, b\, c.(ac)(bc) : (p \to q \to r) \to (p \to q) \to p \to r$$

Wariant Churcha

$$\lambda a^{p \to q \to r} b^{p \to q} c^{p}.(ac)^{q \to r}(bc)^{q} : (p \to q \to r) \to (p \to q) \to p \to r$$

Warianty

Wariant Curry'ego

$$\lambda a\ b\ c.(ac)(bc) : (p \to q \to r) \to (p \to q) \to p \to r$$

Wariant Churcha

$$\lambda a^{p \to q \to r} b^{p \to q} c^{p}.(ac)^{q \to r}(bc)^{q} : (p \to q \to r) \to (p \to q) \to p \to r$$

Własności

Zgodność z redukcją

Jeśli $\Gamma \vdash M : \tau$ oraz $M \to_\beta N$ to $\Gamma \vdash N : \tau$.

$$(\lambda\, id\ a\ b.K(id\ a)(id\ b))(\lambda x.x)$$

Normalizacja

Jeśli $\Gamma \vdash M : \tau$ to M ma postać normalną.

Ranga redeksu $(\lambda x^\sigma.P)Q$ to długość σ. Indukcja ze względu na (n, m) gdzie n jest maksymalną rangą redeksu w termie, a m liczbą takich redeksów.

Normalizacja

Jeśli $\Gamma \vdash M : \tau$ to M jest silnie normalizowalny.

Własności

Zgodność z redukcją

Jeśli $\Gamma \vdash M : \tau$ oraz $M \to_\beta N$ to $\Gamma \vdash N : \tau$.

$$(\lambda\ id\ a\ b.K(id\ a)(id\ b))(\lambda x.x)$$

Normalizacja

Jeśli $\Gamma \vdash M : \tau$ to M ma postać normalną.

Ranga redeksu $(\lambda x^\sigma.P)Q$ to długość σ. Indukcja ze względu na (n, m) gdzie n jest maksymalną rangą redeksu w termie, a m liczbą takich redeksów.

Normalizacja

Jeśli $\Gamma \vdash M : \tau$ to M jest silnie normalizowalny.

Własności

Zgodność z redukcją

Jeśli $\Gamma \vdash M : \tau$ oraz $M \to_\beta N$ to $\Gamma \vdash N : \tau$.

$$(\lambda\ id\ a\ b.K(id\ a)(id\ b))(\lambda x.x)$$

Normalizacja

Jeśli $\Gamma \vdash M : \tau$ to M ma postać normalną.

Ranga redeksu $(\lambda x^\sigma.P)Q$ to długość σ. Indukcja ze względu na (n, m) gdzie n jest maksymalną rangą redeksu w termie, a m liczbą takich redeksów.

Normalizacja

Jeśli $\Gamma \vdash M : \tau$ to M jest silnie normalizowalny.

Własności

Zgodność z redukcją

Jeśli $\Gamma \vdash M : \tau$ oraz $M \to_\beta N$ to $\Gamma \vdash N : \tau$.

$$(\lambda\ id\ a\ b.K(id\ a)(id\ b))(\lambda x.x)$$

Normalizacja

Jeśli $\Gamma \vdash M : \tau$ to M ma postać normalną.

Ranga redeksu $(\lambda x^\sigma.P)Q$ to długość σ. Indukcja ze względu na (n, m) gdzie n jest maksymalną rangą redeksu w termie, a m liczbą takich redeksów.

Normalizacja

Jeśli $\Gamma \vdash M : \tau$ to M jest silnie normalizowalny.

Własności

Zgodność z redukcją

Jeśli $\Gamma \vdash M : \tau$ oraz $M \to_\beta N$ to $\Gamma \vdash N : \tau$.

$$(\lambda\, id\ a\ b.K(id\ a)(id\ b))(\lambda x.x)$$

Normalizacja

Jeśli $\Gamma \vdash M : \tau$ to M ma postać normalną.

Ranga redeksu $(\lambda x^\sigma.P)Q$ to długość σ. Indukcja ze względu na (n, m) gdzie n jest maksymalną rangą redeksu w termie, a m liczbą takich redeksów.

Normalizacja

Jeśli $\Gamma \vdash M : \tau$ to M jest silnie normalizowalny.

Izomorfizm Curry-Howard(-de Bruijn-Lambek)

Reguły wnioskowania dla logiki minimalnej

$$\frac{\tau \in \Gamma}{\Gamma \vdash \tau} \quad [\mathtt{Ax}]$$

$$\frac{\Gamma \vdash \tau \to \tau' \qquad \Gamma \vdash \tau}{\Gamma \vdash \tau'} \quad [\mathtt{E} \to]$$

$$\frac{\Gamma, \tau \vdash \tau'}{\Gamma \vdash \tau \to \tau'} \quad [\mathtt{I} \to]$$

Reguły wnioskowania dla typów prostych

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau} \quad [\mathtt{Var}]$$

$$\frac{\Gamma \vdash e_0 : \tau \to \tau' \qquad \Gamma \vdash e_1 : \tau}{\Gamma \vdash e_0\ e_1 : \tau'} \quad [\mathtt{App}]$$

$$\frac{\Gamma,\ x : \tau \vdash e : \tau'}{\Gamma \vdash \lambda\, x\ .\ e : \tau \to \tau'} \quad [\mathtt{Abs}]$$

Izomorfizm Curry-Howard(-de Bruijn-Lambek)

Reguły wnioskowania dla logiki minimalnej

$$\frac{\tau \in \Gamma}{\Gamma \vdash \tau} \qquad [\mathtt{Ax}]$$

$$\frac{\Gamma \vdash \tau \to \tau' \qquad \Gamma \vdash \tau}{\Gamma \vdash \tau'} \qquad [\mathtt{E} \to]$$

$$\frac{\Gamma, \tau \vdash \tau'}{\Gamma \vdash \tau \to \tau'} \qquad [\mathtt{I} \to]$$

Reguły wnioskowania dla typów prostych

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau} \qquad [\mathtt{Var}]$$

$$\frac{\Gamma \vdash e_0 : \tau \to \tau' \qquad \Gamma \vdash e_1 : \tau}{\Gamma \vdash e_0 \ e_1 : \tau'} \qquad [\mathtt{App}]$$

$$\frac{\Gamma,\ x : \tau \vdash e : \tau'}{\Gamma \vdash \lambda\, x\ .\ e : \tau \to \tau'} \qquad [\mathtt{Abs}]$$

Type inhabitation

Problem

Dla danych Γ, σ czy istnieje term M taki że $\Gamma \vdash M : \sigma$.

(algorytm Ben-Yellesa)

Twierdzenie (Statman)

'Type inhabitation' jest PSPACE-zupełny.

Type inhabitation

Problem

Dla danych Γ, σ czy istnieje term M taki że $\Gamma \vdash M : \sigma$.

(algorytm Ben-Yellesa)

Twierdzenie (Statman)

'Type inhabitation' jest PSPACE-zupełny.

Type inhabitation

Problem

Dla danych Γ, σ czy istnieje term M taki że $\Gamma \vdash M : \sigma$.

(algorytm Ben-Yellesa)

Twierdzenie (Statman)

'Type inhabitation' jest PSPACE-zupełny.

Type reconstruction/checking

Problem

Dla danych Γ, M czy istnieje typ σ taki że $\Gamma \vdash M : \sigma$.

(unifikacja pierwszego rzędu)

Twierdzenie

'Type reconstruction' jest P-zupełny.

Type reconstruction/checking

Problem

Dla danych Γ, M czy istnieje typ σ taki że $\Gamma \vdash M : \sigma$.

(unifikacja pierwszego rzędu)

Twierdzenie

'Type reconstruction' jest P-zupełny.

Type reconstruction/checking

Problem

Dla danych Γ, M czy istnieje typ σ taki że $\Gamma \vdash M : \sigma$.

(unifikacja pierwszego rzędu)

Twierdzenie

'Type reconstruction' jest P-zupełny.