

*[A type system is a] tractable syntactic method for proving the absence of certain program behaviors by classifying phrases according to the kinds of values they compute.*  
— Benjamin Pierce



# Lambda calculus 4

---

Bolo:

- viac o rekurzii – trochu technickejšie rozprávanie bez kódovania ☹
- de Bruijnové indexy (odstránenie mien premenných) – slajd 28..
- logika kombinátorov SKI (odstránenie mien premenných inak)

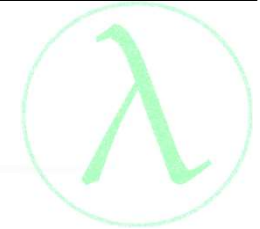
Dnes:

- jednoducho typovaný  $\lambda$ -calcul,  $F_1$
- type-checking & inference, typové rovnice (constraints)
- algoritmus ich riešenia: unifikačný algoritmus
- pohľad do okolia  $F_1$

Cvičenie:

- unifikácia v prázdnej teórii
- Curry-Howardov izomorfizmus

Well-typed programs don't go wrong, but not every program that never goes wrong is well-typed. It's easy to exhibit programs that don't go wrong but are ill-typed in ... any ... decidable type system. Many such programs are useful, which is why dynamically-typed languages like Erlang and Lisp are justly popular.  
— Simon Peyton Jones



# Typy

- motto: *každý slušný programovací jazyk je typovaný*
- napriek tomu, mnoho beztypových jazykov sa teší veľkej obľube (Basic, PHP, Prolog, Smalltalk, Scheme, Python, Ruby, ...)
- typy však obmedzujú programátora v písaní nezmyselných konštrukcií,
- ich úlohou je odhaliť chyby už v čase kompilácie, najmä tie, ktoré by sa objavili v run-time (možno..., ak by tou vetvou program išiel)
- typy obmedzujú vyjadrovaciu (event. výpočtovú ?) silu jazyka (True or False)?

Dnes bude:

- jednoduchý typovaný  $\lambda$ -calcul ([Simply typed lambda calculus](https://en.wikipedia.org/wiki/Simply_typed_lambda_calculus) - F1)  
[https://en.wikipedia.org/wiki/Simply\\_typed\\_lambda\\_calculus](https://en.wikipedia.org/wiki/Simply_typed_lambda_calculus)
  - rozdiel medzi type-checking a type-inference
  - unifikácia ako nástroj riešenia typových rovníc
- Curry-Howardov izomorfizmus
- zložitejšie typovacie systémy (Hindley-Millner a F2)

*Well-typed programs cannot “go wrong”.*  
— Robin Milner

# Hierarchia

$F_\omega$

...

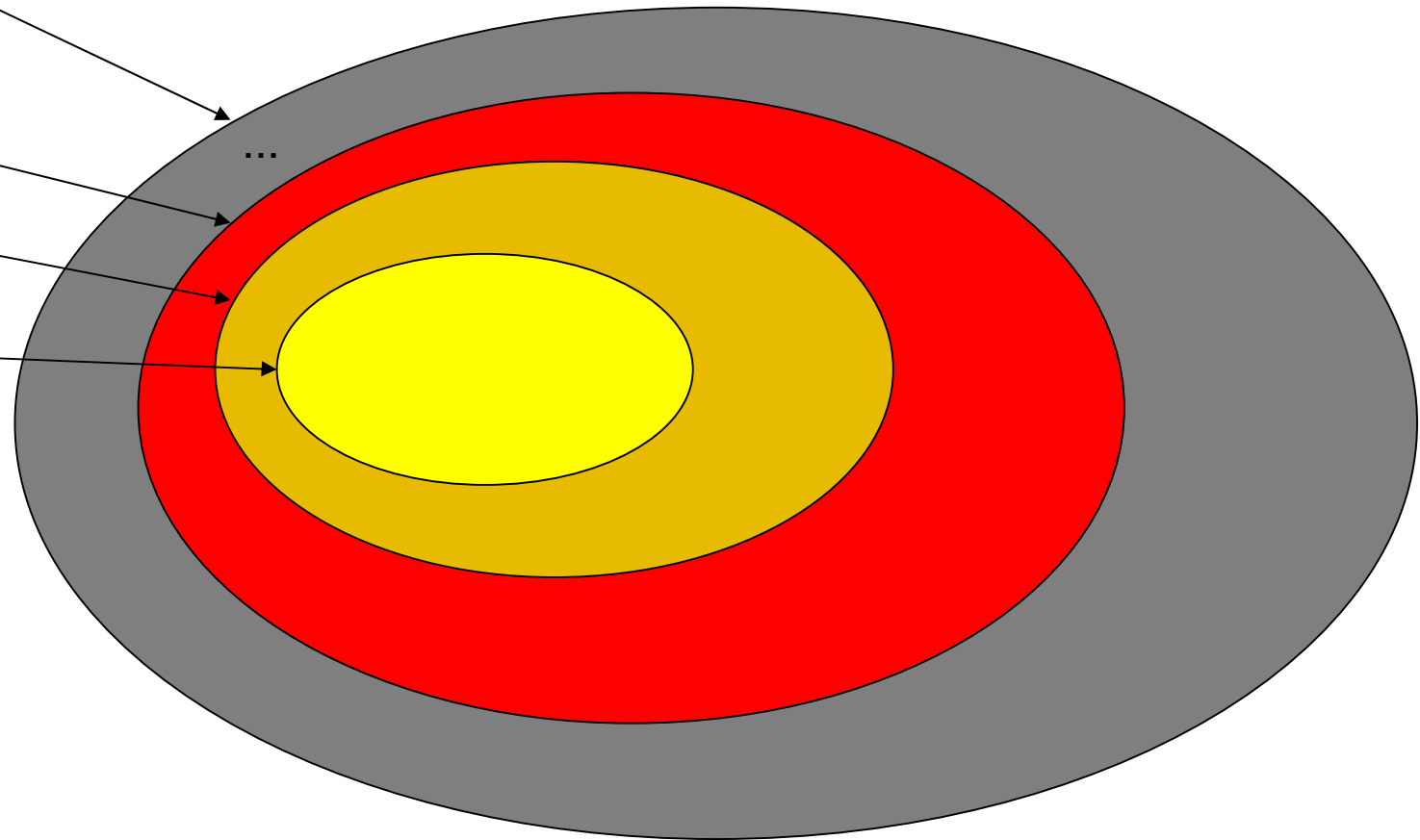
$F_2$

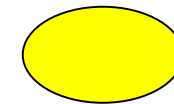
...

HM

Hindley-Millner

$F_1$





# Jednoduchý typovaný $\lambda$ -calcul

Základná neformálna predstava o typoch:

- Typy sú základné a funkčné-funkcionálne:
  - základné:  $A, B, \dots$
  - funkcionálne:  $t_1 \rightarrow t_2$

Pravidlá (2 intuitívne):

- Aplikácia: ak  $M:t_1 \rightarrow t_2$ ,  $N:t_1$ , potom  $(M\ N):t_2$
- Abstrakcia: ak  $x:t_1$ ,  $M:t_2$ , potom  $(\lambda x.M):t_1 \rightarrow t_2$
- Konvencia:  
operátor funkčného typu  $\rightarrow$  je asociatívny doprava (ako v Haskell):
  - $t_1 \rightarrow t_2 \rightarrow t_3 = t_1 \rightarrow (t_2 \rightarrow t_3)$



# Jednoduchý $\lambda$ -calculus, $F_1$

$\lambda$ -Termy (LExp)

Typy (typové termy)

$t ::= x \mid \lambda x.t \mid (t \ t) \mid n \mid +$

$a, \beta ::= \text{Int} \mid a \rightarrow \beta$

Definujeme reláciu  $\Gamma \vdash t:a$ , znamená, že  $\lambda$ -term  $t$  je v kontexte  $\Gamma$  typu  $a$   
Kontext  $\Gamma$  obsahuje informáciu o typoch premenných  $x:a$ ,  
Kontext  $\Gamma$  je zoznam/množina tvaru  $[(\text{String}, \text{Typ})]$ , resp.  $\text{Map String Typ}$

$\{x:a\}:\Gamma \vdash x:a$

[VAR]

$$\frac{\{x:a\}:\Gamma \vdash N:\beta}{\Gamma \vdash (\lambda x.N):a \rightarrow \beta}$$

[ABS]

$$\frac{\Gamma \vdash M:a \rightarrow \beta, \Gamma \vdash N:a}{\Gamma \vdash (M \ N):\beta}$$

[APPL]

$\Gamma \vdash n:\text{Int}$

[INT]

$\Gamma \vdash +:\text{Int} \rightarrow \text{Int} \rightarrow \text{Int}$

[PLUS]



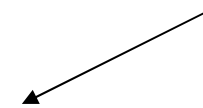
# Tipujeme typy

S intuitívnou predstavou o typoch sa (bezhľavo) vrhnime do typovania:

- k rovnakým premenným napíšme rovnaké typové dekorácie (grécke písmená pri hornom indexe)
- typovacími pravidlami ich propagujeme cez abstrakcie a aplikácie

- $I = \lambda x.x$ 
  - $(\lambda x^a.x^a)^\beta$ , potom  $\beta = a \rightarrow a$
- $K = \lambda x.\lambda y.x$ 
  - $(\lambda x^a.\lambda y^\beta.x^a)^\delta$ , potom  $\delta = a \rightarrow (\beta \rightarrow a)$
- $((\lambda x.x) y)$ 
  - $((\lambda x^a.x^a)^\beta y^\delta)$ , potom  $\beta = a \rightarrow a$ ,  $\delta = a$
  - $((\lambda x^a.x^a)^{a \rightarrow a} y^a)^a$
- $S = \lambda x.\lambda y.\lambda z.((x z) (y z))$ 
  - $\lambda x^a y^\beta z^\delta.[(x^a z^\delta)^\eta (y^\beta z^\delta)^\epsilon]^\theta$ ,  $a = \delta \rightarrow \eta$ ,  $\beta = \delta \rightarrow \epsilon$ ,  $\eta = \epsilon \rightarrow \theta$
  - $x:a=\delta \rightarrow (\epsilon \rightarrow \theta)$ ,  $y:\beta = \delta \rightarrow \epsilon$ ,  $z:\delta$
  - $(\delta \rightarrow (\epsilon \rightarrow \theta)) \rightarrow (\delta \rightarrow \epsilon) \rightarrow \delta \rightarrow \theta$ , výsledok je typu  $\theta$
- $\omega = \lambda x.(x x)$ 
  - $\lambda x^a.(x^a x^a)^\beta$ , potom  $a = ? a \rightarrow \beta$ , ... asi nemá riešenie...

Sústava rovníc





- [illegible]



# Vlastnosti $F_1$

- niektoré  $\lambda$ -termy nie sú otypovateľné,  $Y = \lambda f.(\lambda x. f(x x)) (\lambda x. f(x x)) :-)$  ☹
  - dôvod: podobne ako  $\lambda x.(x x)$  – tzv. *self-application*,
  - neformálne: sústava zodpovedajúcich typových rovníc nemá riešenie
- Church-Rosserova vlastnosť platí aj pre typovaný  $\lambda$ -kalkul ☺
  - typované  $\lambda$ -termy sú podmnožinou  $\lambda$ -termov, a  $\beta$ -redukcia je rovnaká
- typovaný  $\lambda$ -kalkul je silne normalizovateľný = noetherovský = neexistuje nekonečné odvodenie ☺
  - dokázané až 1966-1968, a to  $\geq 6x$ .
  - Dôkaz: napr. v H. Barendregt – Lambda Calculus, Its Syntax and Semantics
- **Dôsledok:**
  - žaden operátor pevného bodu nie je otypovateľný,  $X = F X = F (F X) = F (F (F X)) \dots$
- $F_1$  nemá nekonečný výpočet, cyklus/rekurziu, pevný bod, nie je Turing complete
- **Dôsledok:**
  - jednoducho-otypovateľný  $\lambda$ -kalkul nemá žaden fix-point operátor.





# Normálne formy

(stupeň termu)

Cieľ: typovaný  $\lambda$ -term má normálnu formu = neexistuje nekonečné odvodenie

- **stupeň typu :**

- pre základné:  $A, B, \dots$  je 1,
- pre funkcionálne:  $\alpha \rightarrow \beta$  je  $1 + \max(\text{stupeň } \alpha, \text{stupeň } \beta)$

- **stupeň redexu :**

- $[(\lambda x^\alpha. M^\beta)^\alpha \rightarrow^\beta N^\alpha]^\beta$  je stupeň typu  $(\alpha \rightarrow \beta)$

- **stupeň termu  $M$  :**

- 0 ak neobsahuje redex,
- max.stupeň redexu v  $M$

**Lemma (stupeň termu po substitúcii/  $\beta$ -redukcii):**  $(\lambda x^\alpha. M) N$

- **stupeň termu  $M[x^\alpha:N] \leq \max(\text{stupeň } M, \text{stupeň } N, \text{stupeň } \alpha)$**

- redexy v  $M[x^\alpha:N]$  buďto boli už v  $M$ , resp.  $N$ , alebo vznikli substitúciou
- ak v  $M$  je podterm niekde tvaru  $(x Q)$  a za  $N$  dosadíme  $N = \lambda y. P$ , tak vznikne nový redex  $((\lambda y. P) Q)$
- $\alpha = \beta \rightarrow \eta, Q :: \beta, (x Q) :: \eta, \lambda y. P :: \alpha = \beta \rightarrow \eta, y :: \beta, P :: \eta$
- $\text{stupeň } (x Q) = \text{stupeň } (\beta \rightarrow \eta) = \text{stupeň } ((\lambda y. P) Q)$



# Normálne formy

(stupeň termu)

**Lemma (stupeň termu po substitúcii/  $\beta$ -redukcii):**

- **stupeň termu  $M[x^a:N] \leq \max(\text{stupeň } M, \text{stupeň } N, \text{stupeň } a)$** 
  - redexy v  $M[x^a:N]$  sú v
    - z  $M$ ,
    - z  $N$
  - ak v  $M$  je podterm tvaru  $(x Q)$  a za  $N$  dosadíme  $N = \lambda y.P$ , tak vznikne nový redex  $((\lambda y.P) Q)$

Pod'me merať stupeň:

- $x^a \cdots a = \beta \rightarrow \eta, Q::\beta, (x Q)::\eta$ ,
- $\lambda y.P::a = \beta \rightarrow \eta, y::\beta, P::\eta$
- $\text{stupeň } (x Q) = \text{stupeň } a = \text{stupeň } (\beta \rightarrow \eta)$
- $\text{stupeň } ((\lambda y.P) Q) = \text{stupeň } (\lambda y.P) = \text{stupeň } a = \text{stupeň } (\beta \rightarrow \eta)$



# Normálne formy

(konečné odvođenje)

**Veta:** (každý) typovaný  $\lambda$ -term má normálnu formu

Nájdeme  $R$  je redex v  $M$  maximálneho stupňa, ktorý obsahuje len redexy ostro menších stupňov, t.j.

- je taký,
  - stupeň  $M$  = stupeň  $R$ ,
  - $R$  už neobsahuje redexy stupňa (stupeň  $R$ ), ... ale len menšie !
- 
- Pri  $\beta$ -redukcii redexu  $R$  v  $M$  sa redexy
    - mimo  $R$  nezmenia,
    - vnútri  $R$  sa nahradia sa inými, ale stupeň termu  $M$  nevzrastie (vid' lemma),
    - $R$  zmizne a je nahradený redexami s menším stupňom.

**Dôsledok:**  $\beta$ -redukciou nevzrastie stupeň termu a klesne počet redexov max.stupňa

Dvojica(max.stupeň, počet redexov max.stupňa) klesá v lexikografickom usporiadaní

Keďže je to dobre usporiadanie, **nemôže existovať nekonečná postupnosť**.



# Type checking vs. inference

```
data LExp = LAMBDA String LExp |
           ID String |
           APP LExp LExp |
           CON String | CN Integer
           deriving(Show, Read, Eq)
data Type = Tvar Integer |
           Type -> Type
           deriving(Show, Eq)
```

**checkType :: LExp → Type → Bool**

Problém (rozhodnuteľný):

- $\text{checkType } (A \ B) \ T_2 = \text{checkType } A \ (T_1 \rightarrow T_2) \ \&\& \ \text{checkType } B \ T_1$
- nevieme, ako uhádnuť typ  $T_1$  ???

**typeInference :: LExp → Maybe Type = (Just Type | Nothing)**

Problém (rozhodnuteľný) :

- $\text{inferType } (\lambda x.M) = T_1 \rightarrow (\text{inferType } M)$
- ako zistiť typ premennej  $x$  viazanej v  $\lambda$ -abstrakcii, teda  $T_1$  ???

**inhabitation :: Type → Maybe LExp = (Just LExp | Nothing)**

Problém (rozhodnuteľný) :

- ako zistiť term  $M$  predpísaného typu ?

# Anotácie premenných

(typovaný  $\lambda$ -kalkul podľa Churcha)

Termy

$t ::= x \mid \lambda x:a.t \mid (t \ t) \mid n \mid +$

- ak  $x:T_1$ ,  $M:T_2$ , potom  $(\lambda x:T_1.M): T_1 \rightarrow T_2$

`type Context = [(String, Type)]`

`inferType :: Context → LExp → Maybe Type`

-- premenná a jej typ

`inferType ctx (ID var) = lookup var ctx`

-- [VAR]

`inferType ctx (APP m n) = t2 where`

-- [APPL]

`t1 :→ t2 = inferType ctx m`

-- môže výjsť Nothing, potom propaguj

`t3 = inferType ctx n`

-- Nothing aj do výsledku

`{t1 == t3}`

-- tieto dva typy musia byť rovnaké

`inferType ctx (LAMBDA x t1 m) = t1 :→ inferType ((x,t1):ctx) m`

-- [ABS]

```
data LExp = LAMBDA String Type LExp |
  ID String |
  APP LExp LExp |
  CON String | CN Integer
  deriving(Show, Read, Eq)
```

```
data Type = Tvar Integer |
  Type :→ Type
  deriving(Show, Read, Eq)
```

# Typovaný $\lambda$ -kalkul podľa Churcha

(axiom)  $\Gamma \vdash x : \sigma,$  if  $(x:\sigma) \in \Gamma;$

( $\rightarrow$ -elimination) 
$$\frac{\Gamma \vdash M : (\sigma \rightarrow \tau) \quad \Gamma \vdash N : \sigma}{\Gamma \vdash (MN) : \tau};$$

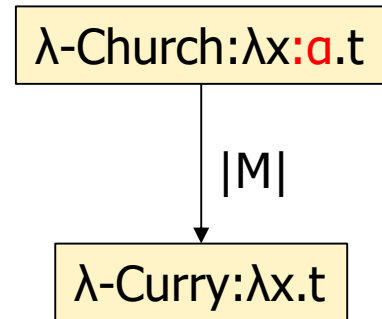
( $\rightarrow$ -introduction) 
$$\frac{\Gamma, x:\sigma \vdash M : \tau}{\Gamma \vdash (\lambda x:\sigma. M) : (\sigma \rightarrow \tau)}.$$

$\{x:a\}:\Gamma \vdash x:a$  [VAR]

$$\frac{\{x:a\}:\Gamma \vdash N:\beta}{\Gamma \vdash (\lambda x:\underline{a}.N):a \rightarrow \beta}$$
 [ABS]

$$\frac{\Gamma \vdash M:a \rightarrow \beta, \Gamma \vdash N:a}{\Gamma \vdash (M N):\beta}$$
 [APPL]

# Typovaný $\lambda$ -kalkul Church vs. Curry



Nech  $|M|$  je term  $M$  bez typových anotácií, t.j. zobrazenie  $\text{Church}_\lambda \rightarrow \text{Curry}_\lambda$

- Ak  $\Gamma \vdash M:\mathbf{a}$  podľa Church, potom  $\Gamma \vdash |M|:\mathbf{a}$  podľa Curry.
- Ak  $\Gamma \vdash M:\mathbf{a}$  podľa Curry, potom existuje anotovaný  $M'$  taký, že
  - $\Gamma \vdash M':\mathbf{a}$  podľa Church,
  - $|M'| = M$ .

# Typové premenné

(typovaný  $\lambda$ -kalkul podľa Curry)

```
data LExp = LAMBDA String LExp |
            ID String |
            APP LExp LExp |
            CON String | CN Integer
            deriving(Show, Read, Eq)
data Type = Tvar Integer |
            Type  $\rightarrow$  Type
            deriving(Show, Read, Eq)
```

- ak nepoznáme konkrétny typ, zavedieme typovú premennú,
- algoritmus zozbiera podmienky (rovnosti) pre typové premenné,
- ak máme šťastie, podarí sa nám ich vyriešiť,
- typové premenné:  $\alpha$ ,  $\beta$ ,  $\delta$ ,  $\eta$ ,  $\delta$ ,  $\varepsilon$ ,  $\theta$  budeme radšej indexovať

```
type Constraints = [(Type,Type)] -- zoznam rovností, eqs
```

```
inferType :: Context -> LExp -> Constraints -> (Type, Constraints)
```

```
inferType ctx (APP m n) eqs = (t2, {t1 = t3}:eqs'') where -- [APP]
```

```
    (t3  $\rightarrow$  t2, eqs') = inferType ctx m eqs
```

```
    (t1, eqs'') = inferType ctx n eqs'
```

```
inferType ctx (LAMBDA x m) eqs = (t1  $\rightarrow$  t2, eqs') -- [ABS]
```

```
    -- t1 je nová typová premenná
```

```
    where (t2,eqs') = inferType ((x,t1):ctx) m eqs
```

```
inferType ctx (ID var) eqs = lookup var ctx, eqs -- [VAR]
```

Dostaneme sústavu rovníc (constraints), ktorú riešime ...

Tento kód nie je v Haskellí, len ilustruje ideu



# Inferencia typu – príklad

$\lambda f. \lambda a. \lambda b. \lambda c. ((c (f a)) (f b)):T, \emptyset, \emptyset$

4steps-in-1

$((c (f a)) (f b)):T4, f:T0, a:T1, b:T2, c:T3, \{T=T0 \rightarrow T1 \rightarrow T2 \rightarrow T3 \rightarrow T4\}$

$(c (f a)):T5, (f b):T6, f:T0, a:T1, b:T2, c:T3, \{T=T0 \rightarrow T1 \rightarrow T2 \rightarrow T3 \rightarrow T4, T5=T6 \rightarrow T4\}$

$c:T3, (f a):T8, (f b):T6, f:T0, a:T1, b:T2, c:T3, \{T=T0 \rightarrow T1 \rightarrow T2 \rightarrow T3 \rightarrow T4, T5=T6 \rightarrow T4, T3=T8 \rightarrow T5\}$

$c:T3, f:T0, a:T1, f:T0, b:T2, f:T0, a:T1, b:T2, c:T3, \{T=T0 \rightarrow T1 \rightarrow T2 \rightarrow T3 \rightarrow T4, T5=T6 \rightarrow T4, T3=T8 \rightarrow T5, T0=T1 \rightarrow T8, T0=T2 \rightarrow T6\}$

Sústava rovníc:

$\{T=T0 \rightarrow T1 \rightarrow T2 \rightarrow T3 \rightarrow T4, T5=T6 \rightarrow T4, T3=T8 \rightarrow T5, T0=T1 \rightarrow T8, T0=T2 \rightarrow T6\}$

$\{T=T0 \rightarrow T1 \rightarrow T2 \rightarrow T3 \rightarrow T4, T5=T6 \rightarrow T4, T3=T8 \rightarrow T5, T0=T1 \rightarrow T8, T2=T1, T8=T6\}$

Riešenie:

$T=T0 \rightarrow T1 \rightarrow T2 \rightarrow T3 \rightarrow T4 = (T1 \rightarrow T8) \rightarrow T1 \rightarrow T1 \rightarrow (T8 \rightarrow T5) \rightarrow T4 =$

$(T1 \rightarrow T8) \rightarrow T1 \rightarrow T1 \rightarrow (T8 \rightarrow T8 \rightarrow T4) \rightarrow T4$

Domáca úloha 1:

$\text{inferType} :: [(\text{LExp}, \text{Type})] \rightarrow \text{Context} \rightarrow \text{Constraint} \rightarrow \text{Int} \rightarrow \text{Constraint}$

# Inferencia typu – príklad 2

$\lambda xyz.(xz)(yz):T, \emptyset, \emptyset$

$\lambda yz.(xz)(yz):T2, x:T1, \{T=T1 \rightarrow T2\}$

$\lambda z.(xz)(yz):T4, y:T3, x:T1, \{T=T1 \rightarrow T2, T2=T3 \rightarrow T4\}$

$(xz)(yz):T6, z:T5, y:T3, x:T1, \{T=T1 \rightarrow T2, T2=T3 \rightarrow T4, T4=T5 \rightarrow T6\}$

$(xz):T7, (yz):T8, z:T5, y:T3, x:T1, \{T=T1 \rightarrow T2, T2=T3 \rightarrow T4, T4=T5 \rightarrow T6, T7=T8 \rightarrow T6\}$

$x:T9, z:T10, (yz):T8, z:T5, y:T3, x:T1, \{T=T1 \rightarrow T2, T2=T3 \rightarrow T4, T4=T5 \rightarrow T6, T7=T8 \rightarrow T6, T9=T10 \rightarrow T7, T9=T1, T10=T5\}$

$y:T11, z:T12, z:T5, y:T3, x:T1, \{T=T1 \rightarrow T2, T2=T3 \rightarrow T4, T4=T5 \rightarrow T6, T7=T8 \rightarrow T6, T9=T10 \rightarrow T7, T9=T1, T10=T5, T11=T12 \rightarrow T8, T11=T3, T12=T5\}$

Sústava rovníc:

$\{T=T1 \rightarrow T2, T2=T3 \rightarrow T4, T4=T5 \rightarrow T6, T7=T8 \rightarrow T6, T9=T10 \rightarrow T7, T9=T1, T10=T5, T11=T12 \rightarrow T8, T11=T3, T12=T5\}$

$\{T=T1 \rightarrow T2, T2=T3 \rightarrow T4, T4=T5 \rightarrow T6, T7=T8 \rightarrow T6, T1=T5 \rightarrow T7, T3=T5 \rightarrow T8\}$

Riešenie:

$T=T1 \rightarrow T2=T1 \rightarrow (T3 \rightarrow T4)=T1 \rightarrow (T3 \rightarrow (T5 \rightarrow T6))=$   
 $(T5 \rightarrow T8 \rightarrow T6) \rightarrow (T5 \rightarrow T8) \rightarrow T5 \rightarrow T6$

# Inferencia typu – príklad 3

$Y = \lambda f.(\lambda x. f(x x)) (\lambda x. f(x x)):T, \emptyset, \emptyset$   
 $(\lambda x. f(x x)) (\lambda x. f(x x)):T2, f:T1, \{T=T1 \rightarrow T2\}$   
 $(\lambda x. f(x x)):T3, (\lambda x. f(x x)):T4, f:T1, \{T=T1 \rightarrow T2, T3=T4 \rightarrow T2\}$   
 $(f(x x)):T6, (\lambda x. f(x x)):T4, f:T1, x:T5, \{T=T1 \rightarrow T2, T3=T4 \rightarrow T2, T3=T5 \rightarrow T6\}$   
 $(x x):T7, (\lambda x. f(x x)):T4, f:T1, x:T5, \{T=T1 \rightarrow T2, T3=T4 \rightarrow T2, T3=T5 \rightarrow T6, T1=T7 \rightarrow T6\}$   
 $(\lambda x. f(x x)):T4, f:T1, x:T5, \{T=T1 \rightarrow T2, T3=T4 \rightarrow T2, T3=T5 \rightarrow T6, T1=T7 \rightarrow T6, T5=T5 \rightarrow T7\}$

Riešenie:

nemá  $\emptyset$



# Unifikácia

- unifikácia je spôsob riešenia rovníc v rôznych teóriach
- najjednoduchším prípadom je syntaktická (prázdna, či Herbrandova teória)
- v našom prípade je problém zredukovaný na jediný funkčný symbol →  
[https://en.wikipedia.org/wiki/Unification\\_\(computer\\_science\)#Syntactic\\_unification\\_of\\_first-order\\_terms](https://en.wikipedia.org/wiki/Unification_(computer_science)#Syntactic_unification_of_first-order_terms)

Unifikácia v syntaktickej teórii:

Termy  $t_1$  a  $t_2$  sú **unifikovateľné**, ak existuje substitúcia  $\Theta$ , že  $t_1 \Theta = t_2 \Theta$   
 $\Theta$  sa potom nazýva **unifikátorom**.

**Príklad:**

$X = f(a, Y)$  sú unifikovateľné, lebo napr.  $\Theta = \{X/f(a,b), Y/b\}$       vid'  $X\Theta = f(a,b) = f(a, Y)\Theta$

$\Theta$  sa nazýva **najvšeobecnejším** unifikátorom, ak každý iný  $\eta$  unifikátor je jeho inštanciou, teda  $\eta = \Theta \circ \psi$

Pre  $X = f(a, Y)$  je **najvšeobecnejší**  $\Theta = \{X/f(a,Y)\}$  a  $\{X/f(a,b), Y/b\}$  nie je najvšeobecnejší unif.

V syntaktickej teórii (Herbrandove Univerzum) najvšeobecnejší, ak existuje, je jednoznačne určený až na premenovanie premenných.



- inferType nám vráti sústavu typových rovníc
- prejdeme exponencialny unifikačný algo
- nakódime si ho
- pochopíme aký to ma súvis s typovými rovnicami

ideový vrchol (neprogramujete :)

- Curry-Howardov izomorfizmus
- zložitejšie typovacie systémy (Hindley-Millner a F2)



# Unifikácia algoritmický problém

[https://www.researchgate.net/publication/221562903\\_Comparing\\_Unification\\_Algorithms\\_in\\_First-Order\\_Theorem\\_Proving](https://www.researchgate.net/publication/221562903_Comparing_Unification_Algorithms_in_First-Order_Theorem_Proving)

	čas	pamäť
Robinson, 1965	$exp(n)$	$exp(n)$
Boyer, Moore, 1972	$exp(n)$	$O(n)$
Corbina, Bidoit, 1983	$O(n^2)$	$O(n)$
Ružička, Prívara, 1989	$O(n.\alpha(n))$	$O(n)$
Martelli, Montanari, 1982	$O(n + m.\alpha(m))$	$O(n)$
Huet, 1973	$O(n.\alpha(n))$	$O(n)$
Patterson, Wegman, 1978	$O(n)$	$O(n)$

<https://core.ac.uk/download/pdf/82682021.pdf>



# Unifikácia

$f(t_1, \dots, t_n) = f(s_1, \dots, s_n), C$	$\Rightarrow t_1=s_1, \dots, t_n=s_n, C$	(DECOMPOSE)
$f(t_1, \dots, t_n) = g(s_1, \dots, s_m), C$	$\Rightarrow \text{FAIL}, C$	(FAILURE)
$x=t, C$	$\Rightarrow x=t, C[x:t]$	if $x \notin t$ (SUBSTITUTE)
$x=t, C$	$\Rightarrow \text{FAIL}$	else, if $x \in t$ (OCCUR CHECK)
$x=x, C$	$\Rightarrow C$	

$f(a, g(Y)) = f(Y, Z) \Rightarrow a = Y, g(Y) = Z \Rightarrow Y = a, g(a) = Z$   
 $f(X, X) = f(Y, g(Y)) \Rightarrow X=Y, X = g(Y) \Rightarrow X=Y, Y = g(Y) \Rightarrow \text{FAIL (OCCUR CHECK)}$

tento naivný algoritmus je exponenciálny (generuje exponenciálny výstup...)

Príklad: Vstup  $\{ t_1=g(t_2, t_2), t_2=g(t_3, t_3), t_3=g(t, t) \}$

Riešenie:

SUBSTITÚCIA:  $t_1=g(t_2, t_2), t_2=g(g(t, t), g(t, t)), t_3=g(t, t)$

dtto:  $t_1=g(g(g(t, t), g(t, t)), g(g(t, t), g(t, t))), t_2=g(g(t, t), g(t, t)), t_3=g(t, t)$

Ak zovšeobecníme vstup pre  $t_1..t_n$ , výsledok bude exponenciálny od dĺžky vstupu čo sa dá stihnúť len v exponenciálnom čase...



# Unifikácia2

---

## Príklad:

$X = f(X)$  nie sú unifikovateľné, a  $\Theta = \{X/f(X)\}$  nie je unifikátor, lebo

$$X\Theta = f(X) \quad \neq \quad f(X)\Theta = f(f(X))$$

a nie, ako by sa zdalo, že

$$X\Theta = f(f(f(f(\dots)))) \quad == \quad f(X)\Theta = f(f(f(f(\dots))))$$

A v algoritme to zachráni occur check

$x=t, C \Rightarrow$  **FAIL** if  $x \in t$  **(OCCUR CHECK)**

ale v naivnej implementácii **occur check** robí z lineárneho algoritmu kvadratický. Preto mnoho nástrojov využívajúcich unifikáciu (napr. Prolog) neimplementuje korektne unifikáciu, len aby „algoritmus“ ostal lineárny, a „bug“ prezentujú ako feature (nekonečné termy)

**Matching** ak existuje substitúcia  $\Theta$ , že  $t_1 \Theta = t_2$





# Unifikácia v rôznych teóriach

Riešte rovnice

- $X+3 = 5$  má riešenie, lebo interpretujeme funkčné symboly tušíte, že interpretácia + hovorí, že  $2+3$  je 5

Syntaktická/prázdna teória (Herbrandova) neinterpretuje symboly

$x \square 3 = 5$  nemá riešenie, lebo nikto netuší (interpretáciu  $\square$  )

$a \square X = Y \square b$  má riešenie aj v prázdnej teórii,  $Y = a, X = b$

Typové rovnice sú teória s jediným funkčným symbolom  $\rightarrow$

$T8 = T3 \rightarrow T7$

=» **SUBSTITUTE**

**$T8 = T8 \rightarrow T7, T8 = T2 \rightarrow T8,$**

=» **FAIL, occur check**

$T5 \rightarrow T2 = T3 \rightarrow T7,$

=» **DECOMPOSE  $T5=T3, T2=T7$**

Komutatívna teória – nevieme, čo  $\square$  robí, ale vieme, že je komutatívne

$a \square X = Y \square Z$  má 2 rôzne riešenia  $\{Y = a, X = Z\}$      $\{Z = a, X = Y\}$



# Unifikácia

- `type Constraints = [(Type,Type)]` `-- [(Type=Type)]`
- `unify :: Constraints → Constraints`

```

unify [] = [] -- Just []
unify ( (S=T) : C' )
  | S == T = unify C'
  | S == ti && not(ti ∈ T) = unify(C'[ti:T]) ++ [ti=T]
  | ti == T && not(ti ∈ S) = unify(C'[ti:S]) ++ [ti=S]
  | S == (S1→S2) and T == (T1→T2) = unify( (S1=T1):(S2=T2):C' )
  | otherwise = fail -- Nothing

```

- `unify :: Constraints → Maybe Constraints`

```

| S == ti && not(ti ∈ Free(T)) = Just ([ti=T] : subst) where
                                Just subst = unify(C'[ti:T])

```

`type Maybe t = Just t | Nothing`

`Just [], Just [(T0, T1), (T1,(T2->T3))], Nothing :: Maybe Constraints`



# Hlavné funkcie

(návrh pre domácu úlohu)

---

So signatúrou:

```
data Type          = Tvar Int | Type :-> Type deriving (Eq)
type Constraint     = (Type,Type)           typová rovnica
type Constraints    = [Constraint]
```

postupne definujte funkcie:

```
contains          :: Type -> Int -> Bool    -- typ obsahuje typovú premennú
                                           -- potrebujete na occur check

substitute        :: Int -> Type -> Type -> Type
substitute'       :: Int -> Type -> Constraints -> Constraints
unify             :: Constraints -> Maybe Constraints

Pomocná:
add2Maybe        :: Constraint -> Maybe Constraints -> Maybe Constraints
add2Maybe        :: a -> Maybe [a] -> Maybe [a]
```



# Domáca úloha

---

Domáca úloha 1: rozpracujte inferType do podoby

`inferType :: [(LExp, Type)] -> Context -> Constraints -> Int -> Constraints`

použitie: `infType lt = inferType [(lt, Tvar 0)] [] [] 1`

Domáca úloha 2: rozpracujte unify do podoby

`unify :: [(Type, Type)] -> Maybe Constraints -> Maybe Constraints`

použitie: `unify constraint = unify constraint (Just [])`

Demonštrujte spoločne 1 & 2:

```
typeExp lt = case unify (infType lt) of
    Just subst -> lookup (Tvar 0) subst
    Nothing    -> Nothing
```

```

typeExp k = Just (T1->(T3->T1))
typeExp s = Just ((T5->(T7->T6))->((T5->T7)->(T5->T6)))
typeExp izero = Just (T1->(T3->T3))
typeExp ione = Just ((T5->T12)->(T5->T12))

```

# Typy a formule

- $K = (\lambda xy.x)^{\alpha \rightarrow \beta \rightarrow \alpha}$
- $S = \lambda xyz.(x\ z)(y\ z)^{(\delta \rightarrow \varepsilon \rightarrow \theta) \rightarrow (\delta \rightarrow \varepsilon) \rightarrow \delta \rightarrow \theta}$

Hilbertov axiomatický systém:

$$A \rightarrow (B \rightarrow A)$$

$$((A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C)))$$

Modus ponens:

$$\frac{A \rightarrow B \quad A}{B}$$

Platí tu, že  $A \rightarrow A$  ?



# Typy a formule

---

Hilbertov axiomatický systém:

Ax 1)  $A \rightarrow (B \rightarrow A)$

Ax 2)  $((A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C)))$

Platí tu, že  $A \rightarrow A$  ?

1)  $A \rightarrow ((B \rightarrow A) \rightarrow A)$  je šikovne zvolená inštancia axiomy Ax1, teda platí...

2)  $((A \rightarrow ((B \rightarrow A) \rightarrow A)) \rightarrow ((A \rightarrow (B \rightarrow A)) \rightarrow (A \rightarrow A)))$  je inštancia axiomy Ax2, teda platí...

MP 1) a 2)

3)  $((A \rightarrow (B \rightarrow A)) \rightarrow (A \rightarrow A))$

MP 3) a Ax1)

4)  $A \rightarrow A$

q.e.d.

$$K = (\lambda xy.x)a \rightarrow \beta \rightarrow a$$

$$S = \lambda xyz.xz(yz)(\delta \rightarrow \varepsilon \rightarrow \theta) \rightarrow (\delta \rightarrow \varepsilon) \rightarrow \delta \rightarrow \theta$$

# Curry-Howardov izomorfizmus

- $((S K) K) = I$

$$S: (\delta \rightarrow \varepsilon \rightarrow \theta) \rightarrow (\delta \rightarrow \varepsilon) \rightarrow (\delta \rightarrow \theta)$$

$$K: a \rightarrow \beta \rightarrow a$$

$$K: \varphi \rightarrow \psi \rightarrow \varphi$$

$$\begin{array}{c} \text{-----} \\ (\delta \rightarrow \varepsilon \rightarrow \theta) = (a \rightarrow \beta \rightarrow a) \\ \bullet \quad \delta = a, \varepsilon = \beta \end{array}$$

$$(\delta \rightarrow \varepsilon) = (\varphi \rightarrow \psi \rightarrow \varphi)$$

- $\delta = \varphi, \varepsilon = \psi \rightarrow \varphi$

$$\delta = \theta$$

$$S: (\delta \rightarrow (\psi \rightarrow \delta) \rightarrow \delta) \rightarrow (\delta \rightarrow (\psi \rightarrow \delta)) \rightarrow (\delta \rightarrow \delta)$$

$$K: (\delta \rightarrow (\psi \rightarrow \delta) \rightarrow \delta)$$

$$K: (\delta \rightarrow (\psi \rightarrow \delta))$$

$$K = (\lambda xy.x) \alpha \rightarrow \beta \rightarrow \alpha$$

$$S = \lambda xyz.xz(yz)(\delta \rightarrow \varepsilon \rightarrow \theta) \rightarrow (\delta \rightarrow \varepsilon) \rightarrow \delta \rightarrow \theta$$

# Curry-Howardov izomorfizmus

$$S: (\delta \rightarrow (\psi \rightarrow \delta) \rightarrow \delta) \rightarrow (\delta \rightarrow (\psi \rightarrow \delta)) \rightarrow (\delta \rightarrow \delta)$$

$$K: (\delta \rightarrow (\psi \rightarrow \delta) \rightarrow \delta)$$

$$K: (\delta \rightarrow (\psi \rightarrow \delta))$$

$$S: (A \rightarrow (B \rightarrow A) \rightarrow A) \rightarrow (A \rightarrow (B \rightarrow A)) \rightarrow (A \rightarrow A) \quad K: (A \rightarrow (B \rightarrow A) \rightarrow A)$$

---


$$(S K): (A \rightarrow (B \rightarrow A)) \rightarrow (A \rightarrow A) \quad K: (A \rightarrow (B \rightarrow A))$$

---


$$((S K) K): (A \rightarrow A)$$



# Hierarchia



$F_\omega$

...

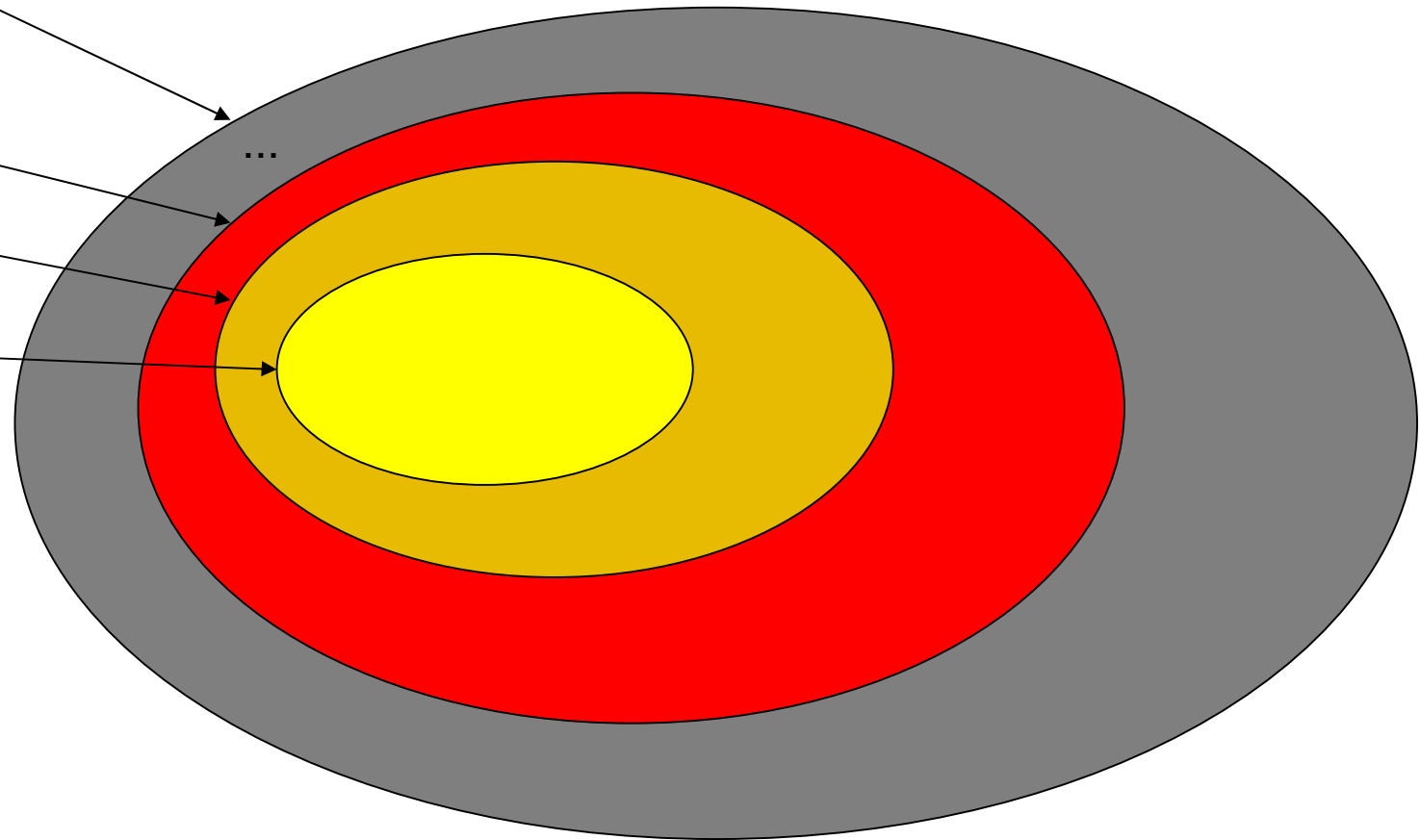
$F_2$

...

HM

Hindley-Millner

$F_1$





# Polymorfický (druhorádový) $\lambda$ -kalkul, System $F_2$ (Girard-Reynold)

V jednoducho-typovanom  $\lambda$ -kalkule doménou premenných sú funkcie,  
v druho-rádovom  $\lambda$ -kalkule doménou premenných sú typy:

Typ

$\sigma ::= \text{Int} \mid \sigma \rightarrow \sigma \mid a \mid \forall a. \sigma$

- všeobecne kvantifikovaný typ
- $a$  je typová premenná
- v Haskell,  $\text{id} :: a \rightarrow a$

- $I = \lambda x. x : \forall a. a \rightarrow a$
- $\text{NOT} : \forall a. a \rightarrow a$
- $\text{K} = \text{TRUE} = (\lambda x. \lambda y. x) : \forall a. \forall \beta. a \rightarrow \beta \rightarrow a,$
- $\text{FALSE} = (\lambda x. \lambda y. y) : \forall a. \forall \beta. a \rightarrow \beta \rightarrow \beta$
- $\underline{0}, \underline{1}, \underline{2} (\lambda f. \lambda x. f(f\ x)), \dots : \forall a. (a \rightarrow a) \rightarrow (a \rightarrow a)$

Dôsledky:

- v takomto  $\lambda$ -kalkule vieme otypovať aj to, čo v  $F_1$  nevieme ( $\omega$ ) ... uvidíme
- Typová inferencia v takomto kalkule (bez typových anotácií, Curry-style) je nerozhodnuteľný problém ☹, otvorený problém v čase 1970-1994
- I napriek tomu, teoretici študujú Martin-Löf hierarchiu:  $F_1, F_2, F_3, \dots \rightarrow F_\omega$
- dependent type ( $t:\text{Type}, t$ )



# System $F_2$

---

Typy

$\sigma ::= \text{Int} \mid \sigma \rightarrow \sigma \mid a \mid \forall a. \sigma$

$\{x:a\}:\Gamma \vdash x:a$  [VAR]

$$\frac{\{x:a\}:\Gamma \vdash N:\beta}{\Gamma \vdash (\lambda x.N):a \rightarrow \beta}$$
 [ABS]

$$\frac{\Gamma \vdash M:a \rightarrow \beta, \Gamma \vdash N:a}{\Gamma \vdash (M N):\beta}$$
 [APPL]

$$\frac{\Gamma \vdash M:\beta}{\Gamma \vdash M:\forall a.\beta}$$
 [GEN]  $a$  not free in  $\Gamma$

$$\frac{\Gamma \vdash M:\forall a.\beta}{\Gamma \vdash M:\beta[a:\theta]}$$
 [INST]

$\lambda x.x : \forall a.a \rightarrow a$   
 $\text{AND} : \forall a.a \rightarrow a \rightarrow a$   
 $\underline{0}, \underline{1}, \underline{2}, \dots : \forall t.(t \rightarrow t) \rightarrow (t \rightarrow t)$

## Príklady v $F_2$

$\{x:a\} \vdash x:a$	[VAR]	$\{x:\text{Int}\} \vdash x:\text{Int}$	[VAR]
$\vdash (\lambda x.x):a \rightarrow a$	[ABS]	$\vdash (\lambda x.x):\text{Int} \rightarrow \text{Int}$	[ABS]
$\vdash (\lambda x.x): \forall a.a \rightarrow a$	[GEN]		
$\vdash (\lambda x.x): \text{Int} \rightarrow \text{Int}$	[INST]		

$\{x:\forall a.a \rightarrow a\} \vdash x:\forall a.a \rightarrow a$  [VAR]  
 $\{x:\forall a.a \rightarrow a\} \vdash x:(\forall \beta.\beta \rightarrow \beta) \rightarrow (\forall \beta.\beta \rightarrow \beta)$  [INST] --  $a$  nahradíme za  $(\forall \beta.\beta \rightarrow \beta)$   
 $\{x:\forall a.a \rightarrow a\} \vdash x:\forall a.a \rightarrow a$  [VAR]  
 $\{x:\forall a.a \rightarrow a\} \vdash x:\forall \beta.\beta \rightarrow \beta$  --  $a$  nahradíme za  $\beta$   
 $\{x:\forall a.a \rightarrow a\} \vdash (x x): \forall \beta.\beta \rightarrow \beta$  [APP]  
 $\{x:\forall a.a \rightarrow a\} \vdash (x x): \forall a.a \rightarrow a$  --  $\beta$  nahradíme za  $a$   
 $\vdash \lambda x.(x x):(\forall a.a \rightarrow a) \rightarrow (\forall a.a \rightarrow a)$  [ABS]

- podarilo sa otypovať výraz  $\omega$ , ktorý v jednoducho-typovanom kalkule nejde
- dostali sme však typ  $(\forall a.a \rightarrow a) \rightarrow (\forall a.a \rightarrow a)$ , ktorý vnútri obsahuje kvantifikátory (deep type) na rozdiel od tých, čo ich majú len na najvyššej úrovni, napr.  $\forall t.(t \rightarrow t) \rightarrow (t \rightarrow t)$  – shallow type

# Let polymorfizmus Hindley-Millner

Typ  $\vee F_2$   
 $\sigma ::= \text{Int} \mid \sigma \rightarrow \sigma \mid a \mid \forall a. \sigma$

chceme zakázať kvantifikáciu typov vnútri typového výrazu:

- zakázať **deep types**  $(\forall a. a \rightarrow a) \rightarrow (\forall a. a \rightarrow a)$ , vnútri obsahuje kvantifikátory
- povoliť len **shallow types**  $\forall a. (a \rightarrow a) \rightarrow (a \rightarrow a)$  na najvyššej úrovni

Typy (formalizácia):

$\sigma ::= \psi \mid \forall a. \sigma$

-- polymorfné, generické...

$\psi ::= \text{Int} \mid \psi \rightarrow \psi \mid a$

-- základné, funkčné a typová premenná

Haskell: premenná viazaná  $\lambda$  výrazom nemôže byť polymorfného typu, napr.

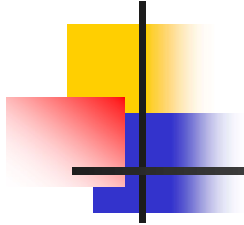
$f (\lambda x. x)$  where  $f = \lambda g. [\dots (g \ 1) \dots (g \ \text{True}) \dots]$ , alebo v Haskell:

- `idd = (\x->x)`
- `foo = f idd` where `f = \g->(if (g True) then (g 2) else (g 1))`
- `foo = let f g = (if (g True) then (g 2) else (g 1)) in f idd`

Nahradíme pravidlo [GEN] pravidlom [LET]

$$\frac{\Gamma \vdash M : \beta, \quad \Gamma, x : \forall a. \beta \vdash N : \delta}{\Gamma \vdash \text{let } x = M \text{ in } N : \delta}$$

[LET]  $a$  in  $\beta$ ,  $a$  not free in  $\Gamma$



- [http://dev.stephendiehl.com/fun/006\\_hindley\\_milner.html](http://dev.stephendiehl.com/fun/006_hindley_milner.html)

$$\frac{x : \sigma \in \Gamma}{\Gamma \vdash x : \sigma} \quad (\text{T-Var})$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1 e_2 : \tau_2} \quad (\text{T-App})$$

$$\frac{\Gamma, x : \tau_1 \vdash e : \tau_2}{\Gamma \vdash \lambda x . e : \tau_1 \rightarrow \tau_2} \quad (\text{T-Lam})$$

$$\frac{\Gamma \vdash e_1 : \sigma \quad \Gamma, x : \sigma \vdash e_2 : \tau}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau} \quad (\text{T-Let})$$

$$\frac{\Gamma \vdash e : \sigma \quad \bar{\alpha} \notin \text{ftv}(\Gamma)}{\Gamma \vdash e : \forall \bar{\alpha} . \sigma} \quad (\text{T-Gen})$$

$$\frac{\Gamma \vdash e : \sigma_1 \quad \sigma_1 \sqsubseteq \sigma_2}{\Gamma \vdash e : \sigma_2} \quad (\text{T-Inst})$$



# Rekurzia v systeme Hindley-Millner

---

- v takomto  $\lambda$ -kalkule vieme otypovať aj to, čo v  $F_1$  nevieme ( $\omega$ )

Máme  $Y_a$  pre každé  $a$ :

$$Y_a : \forall a. (a \rightarrow a) \rightarrow a$$

a pridáme pravidlo typovanej  $Y$  konverzie:

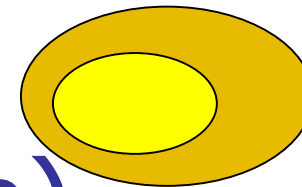
$$(Y_a \ e^{a \rightarrow a})^a \rightarrow_Y (e^{a \rightarrow a} (Y_a \ e^{a \rightarrow a})^a)^a$$

napr. pre faktorial:

$$Y_{Int} : (Int \rightarrow Int) \rightarrow Int$$

pravidlo typovanej  $Y_{Int}$  konverzie:

$$(Y_{Int} \ e^{Int \rightarrow Int})^a \rightarrow_Y (e^{Int \rightarrow Int} (Y_{Int} \ e^{Int \rightarrow Int})^{Int})^{Int}$$



# self-application (MAP map)

(na cvičenie)

vieme otypovať výraz map map ? DU4

- `map :: (a->b)->[a]->[b]` -- polymorický type

`MAP :: (A -> B) -> [A] -> [B]`

`map :: (a -> b) -> ([a] -> [b])`

- preto `(A -> B) = (a -> b) -> ([a] -> [b])`

- ergo `A = (a -> b)`, `B = [a] -> [b]`

Dosadíme:

`MAP :: ((a->b) -> ([a] -> [b])) -> [a->b] -> [[a] -> [b]]`

`(MAP map) :: [a->b] -> [[a] -> [b]]`

ale chcelo by to vidieť a spustiť, tak nech `a = b = Int`

`MAP :: ((Int->Int) -> ([Int] -> [Int])) -> [Int->Int] -> [[Int] -> [Int]]`

`(map map) [(+1),(+2),(*3)]` -- nevypíše nič, lebo je to zoznam funkcií

`length $ (map map) [(+1),(+2),(*3)] == 3`

`((map map) [(+1),(+2),(*3)]!!0) [1..10] == [2,3,4,5,6,7,8,9,10,11]`



# Unifikácia

(odstránenie substitúcie)

$\text{unify} :: (\text{Type}, \text{Type}) \rightarrow \text{Maybe Constraints} \rightarrow \text{Maybe Constraints}$

$\text{unify } (\_, \_) \text{ Nothing}$

$= \text{Nothing}$

$\text{unify } (a1 \rightarrow b1, a2 \rightarrow b2) \text{ subst}$

$= \text{subst2}$  where

$\text{subst1} = \text{unify } (a1, a2) \text{ subst}$

$\text{subst2} = \text{unify } (b1, b2) \text{ subst1}$

--dereferencia premennej miesto aplikacie substitúcie

$\text{unify } (t_i, b) \text{ s@}(\text{Just subst})$

$= \text{unify } (a, b) \text{ s}$  where  $a = \text{deref } t_i \text{ subst}$

$\text{unify } (a, t_i) \text{ s@}(\text{Just subst})$

$= \text{unify } (a, b) \text{ s}$  where  $b = \text{deref } t_i \text{ subst}$

--predpokladáme, že  $t_i$  je dereferencovaná a že platí occur check

$\text{unify } (t_i, b) (\text{Just subst})$

$= \text{Just } ((t_i, b) : \text{subst})$  if  $t_i$  not in  $b$

$\text{unify } (a, t_i) (\text{Just subst})$

$= \text{Just } ((t_i, a) : \text{subst})$  if  $t_i$  not in  $b$

– zabránenie vzniku cyklu medzi premennými

$\text{unify } (t_i, t_j) \text{ s@}(\text{Just subst})$

|  $i < j = \text{Just } ((t_i, t_j) : \text{subst})$

|  $j < i = \text{Just } ((t_j, t_i) : \text{subst})$

| otherwise  $s$

– otherwise

$\text{unify } (\_, \_) \text{ _}$

$= \text{Nothing}$