



Typy

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

Dnes bude:

- jednoduchý typovaný λ -calcul
 - type-checking a type-inference
 - unifikácia ako nástroj riešenia typových rovníc
- Curry-Howardov izomorfizmus
- zložitejšie typovacie systémy



Jednoduchý typovaný λ -calcul

Základná neformálna predstava o typoch:

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

Pravidlá:

- 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
 - $t_1 \rightarrow t_2 \rightarrow t_3 = t_1 \rightarrow (t_2 \rightarrow t_3)$



Jednoduchý λ -calcul, F_1

Termy

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

Typy

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

Definujeme reláciu $\Gamma \vdash t:a$, znamená, že term t je v kontexte Γ typu a
Kontext Γ obsahuje informáciu o typoch premenných $x:a$

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

[VAR]

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

[ABS]

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

[APPL]

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

[INT]

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

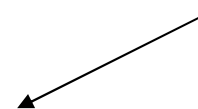
[PLUS]

S intuitívnou predstavou o
typoch sa (bezhlavo) vrhnime
do typovania výrazov

Tipujeme typy

- $\lambda x.x$
 - $(\lambda x^a.x^a)^\beta$, potom $\beta = a$
- $\lambda xy.x$
 - $(\lambda x^a y^\beta.x^a)^\delta$, potom $\delta = a$
- $(\lambda x.x)y$
 - $(\lambda x^a.x^a)^\beta y^\delta$, potom $\beta = a$, $\delta = a$
 - $[(\lambda x^a.x^a)^a y^a]^a$
- $\lambda xyz.(xz)(yz)$
 - $\lambda x^a y^\beta z^\delta.[(x^a z^\delta)^\eta (y^\beta z^\delta)^\varepsilon]^\theta$, $a = \delta$, $\eta = \beta = \delta$, $\varepsilon = \delta$, $\theta = \delta$
 - $x:a = \delta$, $y:\beta = \delta$, $z:\delta$
 - $(\delta) (\varepsilon) (\delta) (\delta) \delta$, výsledok je typu θ
- $\lambda x.(x x)$
 - $\lambda x^a.(x^a x^a)^\beta$, potom $a = \beta$, nemá riešenie...

Sústava rovníc





Vlastnosti F_1

- niektoré λ -termy nie sú otypovateľné, $Y = \lambda f.(\lambda x. f(x x)) (\lambda x. f(x x)) :-)$ ☹
 - dôvod: podobne ako $\lambda x.(x x)$,
 - neformálne: sústava zodpovedajúcich typových rovníc nemá riešenie
- Church-Rosserova vlastnosť platí aj pre typovaný λ -kalkul ☺
 - typované λ -termy sú pomnožinou lambda termov
- typovaný λ -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
- žiaden operátor pevného bodu nie je otypovateľný, lebo $X = F X = F (F X) = \dots$



Normálne formy

(stupeň termu)

Cieľ: typovaný λ -term má normálnu formu = neexistuje nekonečné odvodenie

- **stupeň typu :**
 - pre základné: A, B, \dots je 1,
 - pre funkcionálne: $\alpha \quad \beta$ je $1 + \max(\text{stupeň } \alpha, \text{stupeň } \beta)$
- **stupeň redexu :**
 - $[(\lambda x^\alpha. M^\beta)^\alpha \quad \beta \ N^\alpha]^\beta$ je stupeň typu $(\alpha \quad \beta)$
- **stupeň termu M :**
 - 0 ak neobsahuje redex,
 - max.stupeň redexu v M

Lemma (stupeň po substitúcii/redukcii):

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



Normálne formy

(konečné odvodenie)

Veta: typovaný λ -term má normálnu formu

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

- je taký,
 - stupeň $M = \text{stupeň } R$,
 - R už neobsahuje redexy stupňa ($\text{stupeň } R$), ... ale len menšie !
-
- Pri β -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: β -redukciou nevzrastie stupeň termu a klesne počet redexov max.stupňa

Takže dvojica (stupeň M , počet redexov stupňa M) klesá v lexikografickom usporiadaní...

Keďže je to dobre usporiadanie, nemôže existovať nekonečná postupnosť.



Type checking vs. inference

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

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

`checkType :: LExp → Type → Bool`

Problém (rozhodnuteľný):

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

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

Problém (rozhodnuteľný) :

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

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

Problém (rozhodnuteľný) :

- ako zistiť term M predpísaného typu ?

Anotácie premenných

(typovaný λ -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 (APL 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

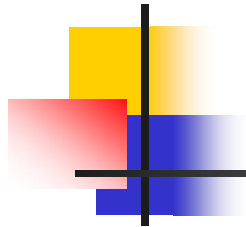
`inferType ctx (LAMBDA x t1 m) = t1 : inferType ((x,t1):ctx) m`

-- [ABS]

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

```
data Type = TInteger |
  Type : Type
  deriving(Show, Read, Eq)
```

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

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

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

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



Typovaný λ -kalkul

Church vs. Curry

Nech $|M|$ je term M bez typových anotácií, t.j. zobrazenie Church_λ Curry_λ

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



Typové premenné

(typovaný λ -kalkul podľa Curry)

```
data LExp = LAMBDA String LExp |
  ID String |
  APL LExp LExp |
  CON String | CN Integer
  deriving(Show, Read, Eq)
data Type = TInteger |
  Tvar Integer |
  Type : 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: α , β , δ , η , δ , ε , θ budeme radšej indexovať

```
type Constraints = [(Type,Type)] -- zoznam rovností, eqs
inferType :: Context->LExp->Constraints-> (Type, Constraints)
```

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

```
  t1, eqs'' = inferType ctx n eqs'
inferType ctx (LAMBDA x m) eqs = t1  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, ktorú riešime

Inferencia typu – príklad

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

$((c (f a)) (f b)):T_4, f:T_0, a:T_1, b:T_2, c:T_3, \{T=T_0 \ T_1 \ T_2 \ T_3 \ T_4\}$

$(c (f a)):T_5, (f b):T_6, f:T_0, a:T_1, b:T_2, c:T_3, \{T=T_0 \ T_1 \ T_2 \ T_3 \ T_4, T_5=T_6 \ T_4\}$

$c:T_3, (f a):T_8, (f b):T_6, f:T_0, a:T_1, b:T_2, c:T_3, \{T=T_0 \ T_1 \ T_2 \ T_3 \ T_4, T_5=T_6 \ T_4, T_3=T_8 \ T_5\}$

$c:T_3, f:T_0, a:T_1, f:T_0, b:T_2, f:T_0, a:T_1, b:T_2, c:T_3, \{T=T_0 \ T_1 \ T_2 \ T_3 \ T_4, T_5=T_6 \ T_4, T_3=T_8 \ T_5, T_0=T_1 \ T_8, T_0=T_2 \ T_6\}$

Sústava rovníc:

$\{T=T_0 \ T_1 \ T_2 \ T_3 \ T_4, T_5=T_6 \ T_4, T_3=T_8 \ T_5, T_0=T_1 \ T_8, T_0=T_2 \ T_6\}$

$\{T=T_0 \ T_1 \ T_2 \ T_3 \ T_4, T_5=T_6 \ T_4, T_3=T_8 \ T_5, T_0=T_1 \ T_8, T_2=T_1, T_8=T_6\}$

Riešenie:

$T=T_0 \ T_1 \ T_2 \ T_3 \ T_4 = (T_1 \ T_8) \ T_1 \ T_1 \ (T_8 \ T_5) \ T_4 =$
 $(T_1 \ T_8) \ T_1 \ T_1 \ (T_8 \ T_8 \ T_4) \ T_4$

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

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

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

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

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

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

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

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

Sústava rovníc:

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

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

Riešenie:

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

Inferencia typu – príklad 3

$Y = \lambda f. (\lambda x. f(x\ x)) (\lambda x. f(x\ x)) : T, \emptyset, \emptyset$

$(\lambda x. f(x\ x)) (\lambda x. f(x\ x)) : T2, f:T1, \{T=T1\ T2\}$

$(\lambda x. f(x\ x)) : T3, (\lambda x. f(x\ x)) : T4, f:T1, \{T=T1\ T2, T3=T4\ T2\}$

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

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

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



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

$$f(t_1, \dots, t_n) = f(s_1, \dots, s_n), C$$

$$\Rightarrow t_1=s_1, \dots, t_n=s_n, C$$

$$f(t_1, \dots, t_n) = g(s_1, \dots, s_m), C$$

$$\Rightarrow \text{FAIL}, C$$

$$x=t, C$$

$$\Rightarrow x=t, C[x:t] \text{ if } x \notin t \text{ [OCCUR CHECK]}$$

$$x=t, C$$

$$\Rightarrow \text{FAIL} \quad \text{if } x \in t$$

$$x=x, C$$

$$\Rightarrow C$$

!!! tento naivný algoritmus je exponenciálny

Príklad:

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

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

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

Ak zovšeobecníme vstup pre $t_1..t_n$, výsledok bude exponenciálny od dĺžky vstupu



Unifikácia

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

`unify [] = []` -- Just []

`unify (S=T:C')`

 | `S == T = unify C'`

 | `S == ti && not(ti ∈ S)` = `unify(C'[ti:S]) ++ [ti=S]`

 | `ti == T && not(ti ∈ T)` = `unify(C'[ti:T]) ++ [ti=T]`

 | `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(S))` = `Just ([ti=S] : subst)` where
Just subst = `unify(C'[ti=S])`



Unifikácia

`unify::(Type,Type)` Maybe Constraints Maybe Constraints

`unify (_,_) Nothing` = Nothing

`unify (a1 b1,a2 b2) subst` = `subst2` where
 `subst1` = `unify (a1,a2) subst`
 `subst2` = `unify (b1,b2) subst1`

--dereferencia premennej miesto aplikacie substitúcie

`unify (ti,b) s@(Just subst)` = `unify (a,b) s` where `a = deref ti subst`

`unify (a,ti) s@(Just subst)` = `unify (a,b) s` where `b = deref ti subst`

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

`unify (ti,b) (Just subst)` = `Just ((ti,b):subst)` if `ti not in b`

`unify (a,ti) (Just subst)` = `Just ((ti,a):subst)` if `ti not in b`

– zabránenie vzniku cyklu medzi premennými

`unify (ti,tj) s@(Just subst)` | `i < j` = `Just ((ti , tj):subst)`
 | `j < i` = `Just ((tj , ti):subst)`
 | otherwise `s`

– otherwise

`unify (_,_) _` = Nothing



Domáca úloha

Domáca úloha 1: rozpracujte inferType do podoby

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

použitie: inferType t = inferType [(t, 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:

typeExp t = case unify (inferType t) of

Just subst -> lookup (T 0) subst

Nothing -> Nothing



Typy a formule

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

Hilbertov axiomatický systém:

$A \quad (B \quad A)$

$((A \quad (B \quad C)) \quad ((A \quad B) \quad (A \quad C)))$

Modus ponens:

$A \quad B \quad A$

B

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

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

Curry-Howardov izomorfizmus

$$\begin{array}{l} \blacksquare S K K = I \\ S: (\delta \quad \varepsilon \quad \theta) \quad (\delta \quad \varepsilon) \quad (\delta \quad \theta) \\ K: a \quad \beta \quad a \\ K: \end{array}$$

$$\begin{array}{l} \hline (\delta \quad \varepsilon \quad \theta) = (a \quad \beta \quad a) \\ \bullet \quad \delta = a, \varepsilon = \beta \end{array}$$

$$\begin{array}{l} (\delta \quad \varepsilon) = (\quad) \\ \blacksquare \quad \delta = \quad, \varepsilon = \end{array}$$

$$\delta = \theta$$

$$\begin{array}{l} S: (\delta \quad (\quad \delta) \quad \delta) \quad (\delta \quad (\quad \delta)) \quad (\delta \quad \delta) \\ K: (\delta \quad \left\{ \begin{array}{l} \delta \\ \delta \end{array} \right\} \quad \delta) \\ K: (\delta \quad (\quad \delta)) \end{array}$$

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

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

Curry-Howardov izomorfizmus

S: $(\delta \quad (\quad \delta) \quad \delta) \quad (\delta \quad (\quad \delta)) \quad (\delta \quad \delta)$

K: $(\delta \quad (\quad \delta) \quad \delta)$

K: $(\delta \quad (\quad \delta))$

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

SK: $(A \quad (B \quad A)) \quad (A \quad A) \quad K: (A \quad (B \quad A))$

SKK: $(A \quad A)$



Polymorfický (druhorádový) λ -kalkul, System F_2 (Girard-Reynold)

V jednoducho-typovanom λ -kalkule doménou premenných sú funkcie,
v druho-rádovom λ -kalkule doménou premenných sú typy:

Typ

$\sigma ::= \text{Int} \mid \sigma \rightarrow \sigma \mid a \mid \forall a. \sigma$ - všeobecne kvantifikovaný typ

- $\lambda x. x : \forall a. a \rightarrow a$
- NOT : $\forall a. a \rightarrow a$
- K=TRUE $(\lambda x. \lambda y. x) : \forall a. \forall \beta. a \rightarrow \beta \rightarrow a$, 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 λ -kalkule vieme otypovať aj to, čo v F_1 nevieme (ω)
- inferencia v takomto kalkule je nerozhodnuteľný problém ☹, 1970-1994
- hierarchia $F_1, F_2, F_3, \dots, F_\omega$
- dependent type (t:Type, t)



System F_2

Typy

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

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

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

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

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

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

$$\frac{\{x:a\} \quad x:a}{(\lambda x.x):a \rightarrow a} \quad \text{[VAR]} \quad \text{[ABS]}$$

$$\frac{(\lambda x.x): \forall a.a \rightarrow a}{(\lambda x.x): \text{Int} \rightarrow \text{Int}} \quad \text{[GEN]} \quad \text{[INST]}$$

$$\frac{\{x:\text{Int}\} \quad x:\text{Int}}{(\lambda x.x):\text{Int} \rightarrow \text{Int}} \quad \text{[VAR]} \quad \text{[ABS]}$$

$$\frac{\{x:\forall a.a \rightarrow a\} \quad x:\forall a.a \rightarrow a}{\{x:\forall a.a \rightarrow a\} \quad x:(\forall \beta.\beta \rightarrow \beta) \rightarrow (\forall \beta.\beta \rightarrow \beta)} \quad \text{[VAR]} \quad \text{[INST]}$$

$$\frac{\{x:\forall a.a \rightarrow a\} \quad (x \ x):\forall a.a \rightarrow a}{\lambda x.(x \ x):(\forall a.a \rightarrow a) \rightarrow (\forall a.a \rightarrow a)} \quad \text{[APP]} \quad \text{[ABS]}$$

- podarilo sa nám otypovať výraz, ktorý v jednoducho-typovanom kalkule sa nedal
- 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

chceme zakázať kvantifikáciu typov vnútri typového výrazu (zakázať deep type)

Typy

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

premenná viazaná λ výrazom nemôže byť polymorfického 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))`

Nahradíme pravidlo [GEN] pravidlom [LET]

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

[LET] a in β , a not free in Γ