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, …
  - 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 $\alpha, \beta ::= Int \mid \alpha \to \beta$

Definujeme reláciu $\Gamma \vdash t:\alpha$, znamená, že term t je v kontexte Γ typu α
Kontext Γ obsahuje informáciu o typoch premenných x:α, [(String,Typ)]

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

$\dfrac{\{x:\alpha\}:\Gamma \vdash N:\beta}{\Gamma \vdash (\lambda x.N):\alpha\to\beta}$ [ABS]

$\dfrac{\Gamma \vdash M:\alpha\to\beta,\ \Gamma \vdash N:\alpha}{\Gamma \vdash (M\ N):\beta}$ [APPL]

$\Gamma \vdash n:Int$ [INT]

$\Gamma \vdash +:Int\to Int\to Int$ [PLUS]

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

Tipujme typy

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

Sústava rovníc

Vlastnosti F_1

- niektoré λ-termy nie sú otypovateľné, Y= λf.(λx. f(x x)) (λx. f(x x)) :-) ☹
  - dôvod: podobne ako λx.(x x) – tzv. *self-application*,
  - 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 >= 6x.

  Dôkaz: napr. v H. Barendregt – Lambda Calculus, Its Syntax and Semantics
- Dôsledok:

  žiaden operátor pevného bodu nie je otypovateľný, lebo X = F X = F (F X) = …
- Dôsledok:

  jednoducho-typovateľný λ-kalkul nemá žiaden fix-point operátor.

Normálne formy
(stupeň termu)

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

- **stupeň typu :**
  - pre základné: A, B, … je 1,
  - pre funkcionálne: $\alpha \rightarrow \beta$ je 1+max(stupeň α, stupeň β)

- **stupeň redexu :**
  - $[(\boldsymbol{\lambda x^{\alpha}.M^{\beta}})^{\alpha\rightarrow\beta}\ \mathbf{N}^{\alpha}]^{\beta}$ je stupeň typu $(\alpha \rightarrow \beta)$

- **stupeň termu M :**
  - 0 ak neobsahuje redex,
  - max.stupeň redexu v **M**

Lemma (stupeň po substitúcii/redukcii):

- **stupeň termu $M[x^{\alpha}:N] <= \max$(stupeň M, stupeň N, stupeň α)**
  - 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 =λy.P, tak vznikne nový redex ((λy.P) Q)
  - $\alpha = \beta \rightarrow \eta$, Q::$\beta$, (x Q):: η, λy.P::$\alpha = \beta \rightarrow \eta$, y::$\beta$, P::$\eta$
  - stupeň (x Q) = stupeň($\beta \rightarrow \eta$) = stupeň ((λy.P) Q)

Normálne formy
(konečné odvodenie)

Veta: typovaný λ-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 β-redukcii redexu R v M sa redexy
  - mimo R nezmenia,
  - vnútri R sa nahradia sa inými, ale stupeň termu M nevzrastie (viď 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):

- checkType (A B) T_2 = checkType A (T_1→T_2) && checkType B T_1
- nevieme, ako uhádnuť typ T_1 ???

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

Problém (rozhodnuteľný) :

- inferType (λx.M) = T_1 → (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)

```
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)
```

Termy

t ::= x | λx:α.t | (t t) | n | +

- ak x:T_1, M:T_2, potom (λx:T_1.M): $T_1 \to T_2$

```
type Context = [(String, Type)]                  -- premenná a jej typ
inferType :: Context → LExp → Maybe Type

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]
```

Tento kód nie je v Haskelli, len ilustruje ideu

Typovaný λ-kalkul podľa Churcha

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

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

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

{x:α}:ΓƊ x:α [VAR]

$\frac{\text{\{x:α\}:ΓƊ N:β}}{\text{ΓƊ (λx:α.N):α→β}}$ [ABS]

$\frac{\text{ΓƊ M:α→β, ΓƊ N:α}}{\text{ΓƊ (M N):β}}$ [APPL]

http://plato.stanford.edu/entries/type-theory-church/

Typovaný λ-kalkul
Church vs. Curry

Nech |M| je term M bez typových anotácií, t.j. zobrazenie $Church_{\lambda} \rightarrow Curry_{\lambda}$

- Ak ΓƊ M:α podľa Church, potom ΓƊ |M|:α podľa Curry.

- Ak ΓƊ M:α podľa Curry, potom existuje anotovaný M' taký, že
  - ΓƊ M':α 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

λf.λa.λb.λc. c (f a) (f b):T, Ø, Ø

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

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

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

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

Sústava rovníc:

{T=T0→T1→T2→T3→T4, T5=T6→T4, T3=T8→T5, T0=T1→T8, T0=T2→T6}

{T=T0→T1→T2→T3→T4, T5=T6→T4, T3=T8→T5, T0=T1→T8, T2=T1, T8=T6}

Riešenie:

T=T0→T1→T2→T3→T4 =(T1→T8)→T1→T1→(T8→T5)→T4=

(T1→T8)→T1→T1→(T8→T8→T4) →T4

Domáca úloha 1:

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

Inferencia typu – príklad 2

$\underline{\lambda xyz.(xz)(yz)}$:T, Ø, Ø

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

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

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

$\underline{(xz)}$: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, $\underline{(yz)}$: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 = <u>λf.(λx. f(x x)) (λx. f(x x))</u>:T, Ø, Ø

<u>(λx. f(x x)) (λx. f(x x))</u>:T2, f:T1, {T=T1→T2}

<u>(λx. f(x x)):T3</u>, (λx. f(x x)):T4, f:T1, {T=T1→T2 , T3=T4→T2}

<u>(f(x x)):T6</u>, (λx. f(x x):T4, f:T1,x:T5, {T=T1→T2 , T3=T4→T2,T3=T5→T6}

<u>(x x):T7</u>, (λx. f(x x):T4, f:T1,x:T5, {T=T1→T2 , T3=T4→T2,T3=T5→T6, T1=T7→T6}

(λx. 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 →

$f(t_1, \ldots, t_n) = f(s_1, \ldots, s_n), C$	=» $t_1=s_1, \ldots, t_n=s_n, C$
$f(t_1, \ldots, t_n) = g(s_1, \ldots, s_m)$, C	=» FAIL, C
$x=t, C$	=» $x=t, C[x:t]$ if $x \notin t$ [OCCUR CHECK]
$x=t, C$	=» FAIL if $x \in t$
$x=x, C$	=» C

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

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 == t_i && not(t_i ∈ T)          = unify(C′[t_i:T]) ++ [t_i=T]
    | t_i == T && not(t_i ∈ S)          = unify(C′[t_i:S]) ++ [t_i=S]
    | S == S_1→S_2 and T == T_1→T_2     = unify(S_1=T_1:S_2=T_2:C′)
    | otherwise                         = fail              -- Nothing
```

- unify :: Constraints→Maybe Constraints

```
    | S == t_i && not(t_i ∈ Free(T))    = Just ([t_i=T] : subst) where
                                          Just subst = unify(C′[t_i:T])
```

Hlavné funkcie

(návrh)

```
data Type          = TVar Int | TApl Type Type deriving (Eq)
type Constraint    = (Type,Type)
type Constraints   = [Constraint]

contains           :: Type -> Int -> Bool
substitute         :: Int -> Type -> Type -> Type
substitute'        :: Int -> Type -> Constraints -> Constraints
unify              :: Constraints -> Maybe Constraints
add2Maybe          :: Constraint -> Maybe Constraints -> Maybe Constraints
add2Maybe          :: a -> Maybe [a] -> Maybe [a]
```

Unifikácia

(odstránenie substitúcie)

```
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 (t_i,b) s@(Just subst)         = unify (a,b) s where a = deref t_i subst
unify (a,t_i) s@(Just subst)         = unify (a,b) s where b = deref t_i subst
--predpokladáme, že t_i je dereferencovaná a že platí occur check
unify (t_i,b) (Just subst)           = Just ((t_i,b):subst) if t_i not in b
unify (a,t_i) (Just subst)           = Just ((t_i,a):subst) if t_i not in b
– zabránenie vzniku cyklu medzi premennými
unify (t_i,t_j) s@(Just subst)       | i < j = Just ((t_i , t_j):subst)
                                     | j < i = Just ((t_j , t_i):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

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

Hilbertov axiomatický systém:

$A \to (B \to A)$

$((A \to (B\to C)) \to ((A\to B) \to(A\to C))$

Modus ponens:

$$\frac{A\to B \qquad A}{B}$$

Platí tu, že $A\to A$?

K = (λxy.x)$\alpha \to \beta \to \alpha$
S = λxyz.xz(yz)$(\delta \to \varepsilon \to \theta) \to (\delta \to \varepsilon) \to \delta \to \theta$

Curry-Howardov izomorfizmus

- S K K = I

S:$(\delta \to \varepsilon \to \theta) \to (\delta \to \varepsilon) \to (\delta \to \theta)$
K:$\alpha \to \beta \to \alpha$
K:ű$\to$ɣ$\to$ű

$(\delta \to \varepsilon \to \theta) = (\alpha \to \beta \to \alpha)$

  - $\delta = \alpha$, $\varepsilon = \beta$

$(\delta \to \varepsilon)$=(ű$\to$ɣ$\to$ű)

  - δ=ű, ε=ɣ$\to$ű

$\delta = \theta$

S:$(\delta \to (\gamma \to \delta) \to \delta) \to (\delta \to (\gamma \to \delta)) \to (\delta \to \delta)$
K:$(\delta \to (\gamma \to \delta) \to \delta)$
K:$(\delta \to (\gamma \to \delta))$

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

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

Curry-Howardov izomorfizmus

$S:(\delta\to(\gamma\to\delta)\to\delta) \to (\delta\to(\gamma\to\delta)) \to (\delta\to\delta)$

$K:(\delta\to(\gamma\to\delta)\to\delta)$

$K:(\delta\to(\gamma\to\delta))$

$$\dfrac{\dfrac{S:(A\to(B\to A)\to A) \to (A\to(B\to A)) \to (A\to A) \quad K:(A\to(B\to A)\to A)}{SK:(A\to(B\to A)) \to (A\to A)} \quad K:(A\to(B\to A))}{SKK:(A\to A)}$$

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

$\sigma ::= \mathrm{Int} \mid \sigma \to \sigma \mid \alpha \mid \forall\alpha.\sigma$ - všeobecne kvantifikovaný typ

- $\lambda x.x : \forall\alpha.\alpha\to\alpha$
- NOT : $\forall\alpha.\alpha\to\alpha$
- K=TRUE $(\lambda x.\lambda y.x) : \forall\alpha.\forall\beta.\alpha\to\beta\to\alpha$, FALSE $(\lambda x.\lambda y.y) : \forall\alpha.\forall\beta.\alpha\to\beta\to\beta$
- $\underline{0}$,$\underline{1}$,$\underline{2}$ $(\lambda f.\lambda x.f(f\ x))$,… : $\forall\alpha.(\alpha\to\alpha)\to\alpha\to\alpha$

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, … $\to F_\omega$
- dependent type (t:Type, t)

System F_2

Typy

$$\sigma ::= \mathrm{Int} \mid \sigma \to \sigma \mid \alpha \mid \forall\alpha.\sigma$$

{x:α}:ΓƊ x:α [VAR]

$$\frac{\{x:\alpha\}:\Gamma Ɗ\ N:\beta}{\Gamma Ɗ\ (\lambda x.N):\alpha\to\beta}$$ [ABS]

$$\frac{\Gamma Ɗ\ M:\alpha\to\beta,\ \Gamma Ɗ\ N:\alpha}{\Gamma Ɗ\ (M\ N):\beta}$$ [APPL]

$$\frac{\Gamma Ɗ\ M:\beta}{\Gamma Ɗ\ M:\forall\alpha.\beta}$$ [GEN] α not free in Γ

$$\frac{\Gamma Ɗ\ M:\forall\alpha.\beta}{\Gamma Ɗ\ M:\beta[\alpha:\theta]}$$ [INST]

λx.x : ∀α.α→α
AND : ∀α.α→α→α
0,1,2,… : ∀t.(t→t)→(t→t)

Príklady v F_2

{x:α}⊢x:α	[VAR]	{x:Int}⊢x:Int	[VAR]
⊢(λx.x):α→α	[ABS]	⊢(λx.x):Int→Int	[ABS]
⊢(λx.x): ∀α.α→α	[GEN]		
⊢(λx.x): Int→Int	[INST]		

{x:∀α.α→α}⊢x:∀α.α→α [VAR]

{x:∀α.α→α}⊢x:(∀β.β→β)→(∀β.β→β) [INST] {x:∀α.α→α}⊢x:∀α.α→α [VAR]

{x:∀α.α→α}⊢(x x):∀α.α→α [APP]

⊢λx.(x x):(∀α.α→α)→(∀α.α→α) [ABS]

- podarilo sa nám otypovať výraz, ktorý v jednoducho-typovanom kalkule sa nedal
- dostali sme však typ (∀α.α→α)→(∀α.α→α), ktorý vnútri obsahuje kvantifikátory (deep type) na rozdieľ od tých, čo ich majú len na najvyššej úrovni, napr. ∀t.(t→t)→(t→t) – shallow type

Let polymorfizmus Hindley-Millner

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

Typy

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

$\psi ::= \mathrm{Int} \mid \psi \rightarrow \psi \mid \alpha$

premenná viazaná λ výrazom nemôže byť polymofrického typu, napr.

f (λx.x) where f = λg.[… (g 1) …. (g True) …], alebo v Haskelli:

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

Nahradíme pravidlo [GEN] pravidlom [LET]

<u>ΓƊ M:β,   Γ, x:∀α.β Ɗ N:δ</u>   [LET] α in β, α not free in Γ

ΓƊ let x=M in N : δ

Rekurzia v systeme Hindley-Millner

- v takomto λ-kalkule vieme otypovať aj to, čo v F_1 nevieme (ω)

Máme Y_α pre každé α:

$Y_\alpha : \forall\alpha.(\alpha \to \alpha) \to \alpha$

a pridáme pravidlo typovanej Y konverzie:

$(Y_\alpha \;\; e^{\alpha \to \alpha})^\alpha \quad \to_Y \quad (e^{\alpha \to \alpha}(Y_\alpha \;\; e^{\alpha \to \alpha})^\alpha)^\alpha$

napr. pre faktorial:

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

pravidlo typovanej Y_{Int} konverzie:

$(Y_{Int} \;\; e^{Int \to Int})^\alpha \quad \to_Y \quad (e^{Int \to Int}(Y_{Int} \;\; e^{Int \to Int})^{Int})^{Int}$