Lambda calculus 4

Bolo:

- viac o rekurzii – trochu technickejšie rozprávanie bez kódenia ☹
- de Bruijnove indexy (odstránenie mien premenných) – slajd 28..
- logika kombinátorov SKi (odstránenie mien premenných inak)

Dnes:

- jednoducho typovaný λ-calcul, F_1
- type-checking&inference, typové rovnice
- algoritmus riešenia: unifikačný algoritmus
- výhliadky do okolia

Cvičenie:

- unifikácia v prázdnej teórii
- Curry-Howardov izomorfizmus

Typy

- motto: *každý slušný programovací jazyk je typovaný*

- napriek tomu, mnoho beztypových jazykov sa teší veľkej obľube (Basic, PHP, Prolog, Smalltalk, Scheme, Python, Ruby, ...)

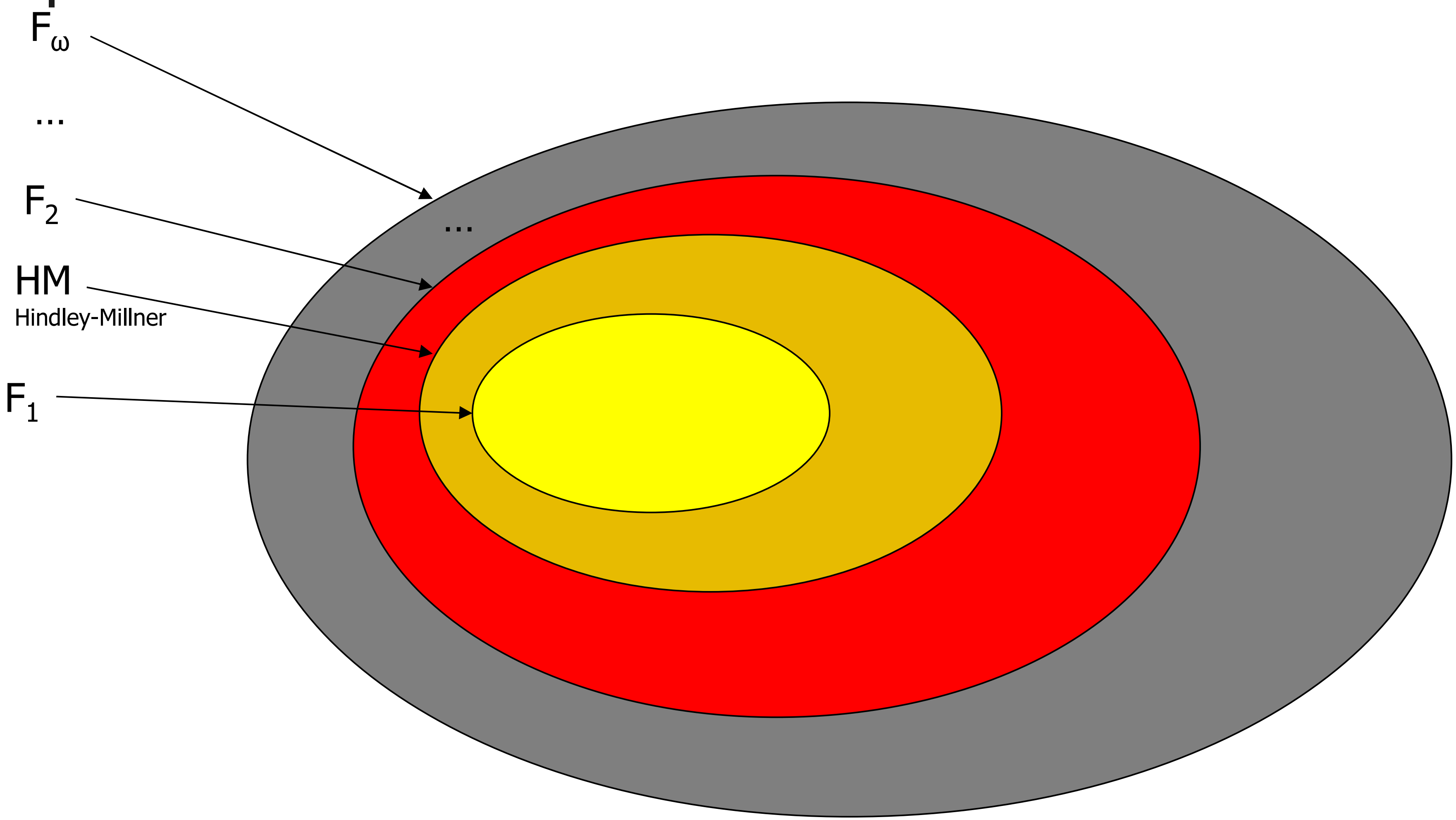
- typy však obmedzujú programátora v písaní nezmyselných konštrukcií,
- ich úlohou je odhaliť chyby už v čase kompilácie, najmä tie, ktoré by sa objavili v run-time (možno..., ak by tou vetvou program išiel)
- typy obmedzujú vyjadrovaciu (event. výpočtovú ?) silu jazyka (True or False)?

Dnes bude:

- jednoduchý typovaný λ-calcul (Simply typed lambda calculus - F1) 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)

Hierarchia



Jednoduchý typovaný λ-calcul

Základná neformálna predstava o typoch:

- Typy sú základné a funkčné-funkcionálne:
  - základné: A, B, …
  - funkcionálne: $t_1 \to t_2$

Pravidlá (2 intuitívne):

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

Jednoduchý λ-calcul, F_1

λ-Termy (LExp) $t ::= x \mid \lambda x.t \mid (t\ t) \mid n \mid +$

Typy (typové termy) $\alpha, \beta ::= Int \mid \alpha \to \beta$

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

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

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

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

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

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

Tipujme typy

S intuitívnou predstavou o typoch sa (bezhlavo) vrhnime do typovania:
- k rovnakým premenným napíšme rovnaké typové dekorácie (grécke písmená pri hornom indexe)
- typovacími pravidlami ich propagujeme cez abstrakcie a aplikácie

- $I = \lambda x.x$
  - $(\lambda x^{\alpha}.x^{\alpha})^{\beta}$, potom $\beta = \alpha \to \alpha$
- $K = \lambda x.\lambda y.x$
  - $(\lambda x^{\alpha}.\lambda y^{\beta}.x^{\alpha})^{\delta}$, potom $\delta = \alpha \to (\beta \to \alpha)$
- $((\lambda x.x)\ y)$
  - $((\lambda x^{\alpha}.x^{\alpha})^{\beta}\ y^{\delta})$, potom $\beta = \alpha \to \alpha$, $\delta = \alpha$
  - $((\lambda x^{\alpha}.x^{\alpha})^{\alpha \to \alpha}\ y^{\alpha})^{\alpha}$

Sústava rovníc

- $S = \lambda x.\lambda y.\lambda z.((x\ z)\ (y\ z))$
  - $\lambda x^{\alpha} y^{\beta} z^{\delta}.[(x^{\alpha} z^{\delta})^{\eta} (y^{\beta} z^{\delta})^{\varepsilon}]^{\theta}$, $\alpha = \delta \to \eta$, $\beta = \delta \to \varepsilon$, $\eta = \varepsilon \to \theta$
  - $x{:}\alpha = \delta \to (\varepsilon \to \theta)$, $y{:}\beta = \delta \to \varepsilon$, $z{:}\delta$
  - $(\delta \to (\varepsilon \to \theta)) \to (\delta \to \varepsilon) \to \delta \to \theta$, výsledok je typu θ
- $\omega = \lambda x.(x\ x)$
  - $\lambda x^{\alpha}.(x^{\alpha}\ x^{\alpha})^{\beta}$, potom $\alpha =^{?} \alpha \to \beta$, … asi nemá riešenie...

Prečo $\alpha =^? \alpha \to \beta$ nemá

- $x = x + 5$
- $x = x + 0$
- $x = x * 1$
- $x = x * 2$
- $x = f(x)$
- je toto riešenie $x = f(f(f(f(f(\ldots)))))$
- $\beta \quad =^? \alpha \to \alpha \to \alpha \to \alpha \to \alpha \to \alpha \to \alpha \to \alpha \to$
- $\alpha \to \beta =^? \alpha \to \alpha \to \alpha \to \alpha \to \alpha \to \alpha \to \alpha \to \alpha \to \alpha \to$

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, a β-redukcia je rovnaká

- 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, X=F X=F (F X)= F (F (F X))…
- F_1 nemá nekonečný výpočet, cyklus/rekurziu, pevný bod, nie je Turing complete

- **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 termu po substitúcii/ β-redukcii): $(\lambda x^{\alpha}.M)\ N$

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

Normálne formy

(stupeň termu)

Lemma (stupeň termu po substitúcii/ β-redukcii):

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

  - $\mathbf{x^{\alpha}}$ ··· $\alpha = \beta \to \eta$, $Q::\beta$, $(x\ Q)::\eta$,
  - $\lambda y.P::\alpha = \beta \to \eta$, $y::\beta$, $P::\eta$
  - stupeň (x Q) = stupeň α = stupeň (β →η)
  - stupeň ((λy.P) Q) = stupeň (λy.P) = stupeň α = stupeň (β →η)

Normálne formy
(konečné odvodenie)

Veta: (každý) 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

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 |
            APP LExp LExp |
            CON String | CN Integer
            deriving(Show, Read, Eq)

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

checkType :: LExp → Type → Bool

Problém (rozhodnuteľný):

- checkType (A B) T_2 = checkType A ($T_1 \rightarrow T_2$) && 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 |
            APP LExp LExp |
            CON String | CN Integer
            deriving(Show, Read, Eq)

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

Termy

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

- ak $x{:}T_1$, $M{:}T_2$, potom $(\lambda x{:}T_1.M){:}\ T_1 \to T_2$

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

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

inferType ctx (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]
```

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)	$\dfrac{\Gamma \vdash M : (\sigma \to \tau) \quad \Gamma \vdash N : \sigma}{\Gamma \vdash (M\,N) : \tau};$	
(→-introduction)	$\dfrac{\Gamma, x{:}\sigma \vdash M : \tau}{\Gamma \vdash (\lambda x{:}\sigma.M) : (\sigma \to \tau)}.$	

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

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

$\dfrac{\Gamma \vdash M{:}\alpha \to \beta,\ \Gamma \vdash N{:}\alpha}{\Gamma \vdash (M\ N){:}\beta}$ [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 $\text{Church}_\lambda \rightarrow \text{Curry}_\lambda$

- 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 |
            APP 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 (APP m n) eqs = (t2, {t1 = t3}:eqs”) where          -- [APP]
                              (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 = λf.(λx. f(x x)) (λx. f(x x)):T, Ø, Ø

(λx. f(x x)) (λx. f(x x)):T2, f:T1, {T=T1→T2}

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

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

(x x):T7, (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 →

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 Θ, že $t_1 \Theta = t_2 \Theta$

Θ sa potom nazýva **unifikátorom.**

Príklad:

X = f(a, Y) sú unifikovateľné, lebo napr. Θ = {X/f(a,b), Y/b}

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

Pre X = f(a, Y) je **najvšeobecnejší** Θ = {X/f(a,Y)}

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

Dnes

- 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

	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

$f(t_1, \ldots, t_n) = f(s_1, \ldots, s_n), C$ =» $t_1=s_1, \ldots, t_n=s_n, C$ (DECOMPOSE)

$f(t_1, \ldots, t_n) = g(s_1, \ldots, s_m), C$ =» FAIL, C (FAILURE)

$x=t, C$ =» $x=t, C[x:t]$ if $x \notin t$ (SUBSTITUTE)

$x=t, C$ =» FAIL else, if $x \in t$ (OCCUR CHECK)

$x=x, C$ =» C

f(a, g(Y)) = f(Y,Z) =» a = Y, g(Y) = Z =» Y = a, g(a) = Z

f(X,X) = f(Y,g(Y)) =» X=Y, X = g(Y) =» X=Y, Y = g(Y) =» FAIL

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 Θ = {X/f(X)} nie je unifikátor, lebo

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

a nie, ako by sa zdalo, že

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

A v algoritme to zachráni occur check

x=t, C =» FAIL if $x \in t$

ale v naivnej implementácii occur check robí z lineárneho algoritmu kvadratický.

Preto mnoho nástrojov (Prolog) neimplementuje korektne unifikáciu, len aby „algoritmus"ostal lineárny, a „bug" prezentujú ako feature (nekonečné termy)

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 □ 3 = 5 nemá riešenie, lebo nikto netuší (interpretáciu □)

a □ X = Y □ b má riešenie aj v prázdnej teórii, Y = a, X = b

Typové rovnice sú teória s jediným funkčným symbolom →

T8 = T3 → T7 =» SUBSTITUTE

T8 = **T8** → T7, **T8** = T2 → **T8**, =» FAIL, occur check

T5 → T2 = T3 → T7, =» DECOMPOSE, T5=T3, T2=T7

Komutatívna teória – nevieme, čo □ robí, ale vieme, že je komutatívne

a □ X = Y □ Z má 2 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 == 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])
```

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 | TApl 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:         inferType 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:

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

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

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 $A\to A$?

Typy a formule

Hilbertov axiomatický systém:

Ax1) $A \to (B \to A)$

Ax2) $((A \to (B \to C)) \to ((A \to B) \to (A \to C))$

Platí tu, že $A \to A$?

1) $A \to ((B \to A) \to A)$ je šikovne zvolená inštancia axiomy Ax1, teda platí...
2) $((A \to ((B \to A) \to A)) \to ((A \to (B \to A)) \to (A \to A))$ je inštancia axiomy Ax2, teda platí...

MP 1) a 2)

3) $((A \to (B \to A)) \to (A \to A))$

MP 3) a Ax1)

4) A -> A q.e.d.

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

Curry-Howardov izomorfizmus

- ((S K) K) = I

$S{:}(\delta \to \varepsilon \to \theta) \to (\delta \to \varepsilon) \to (\delta \to \theta)$
$K{:}\alpha \to \beta \to \alpha$
$K{:}\varphi \to \psi \to \varphi$

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

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

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

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

$\delta = \theta$

$S{:}(\delta \to (\psi \to \delta) \to \delta) \to (\delta \to (\psi \to \delta)) \to (\delta \to \delta)$
$K{:}(\delta \to (\psi \to \delta) \to \delta)$
$K{:}(\delta \to (\psi \to \delta))$

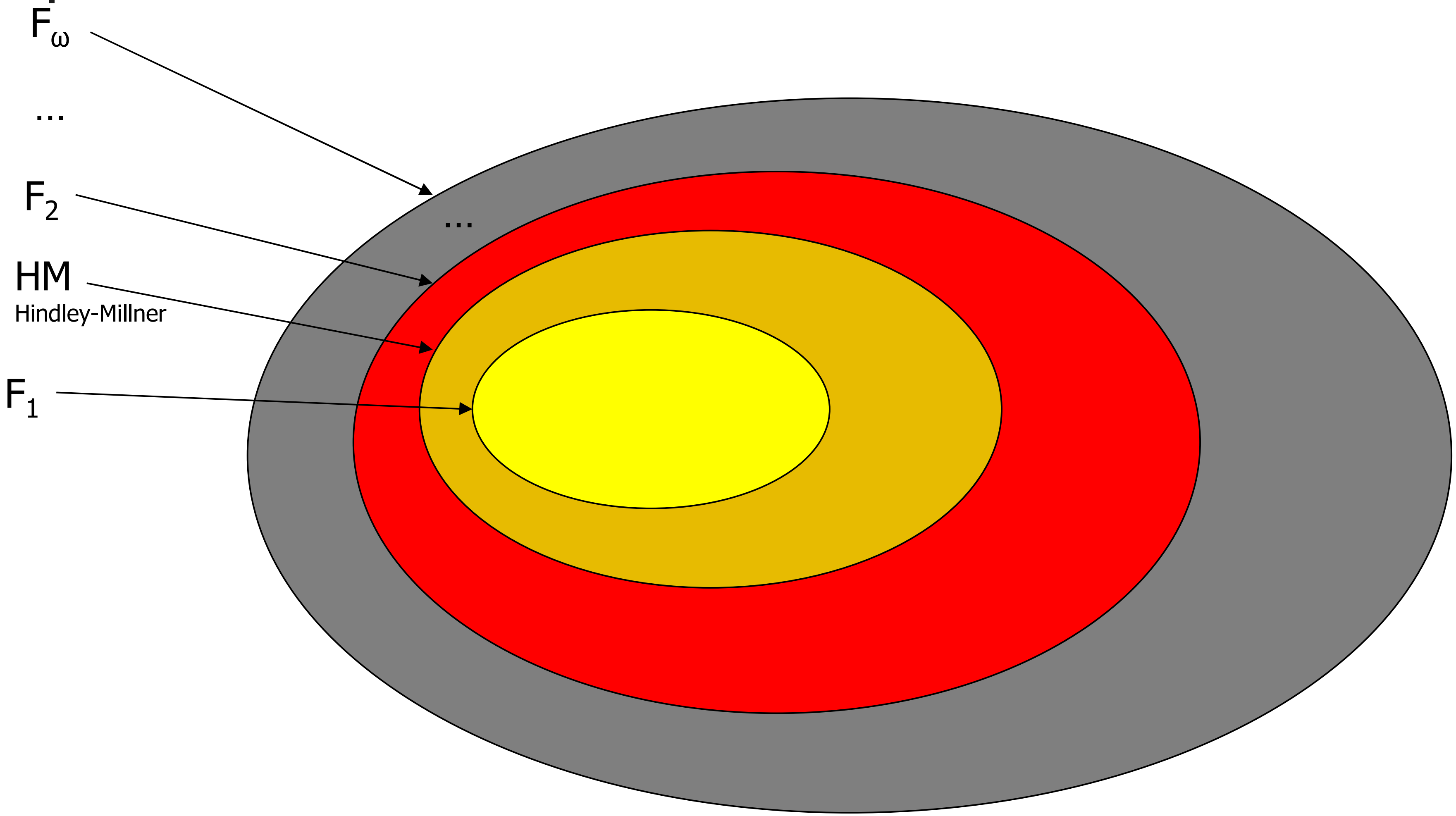
$K = (\lambda xy.x)\alpha\to\beta\to\alpha$
$S = \lambda xyz.xz(yz)(\delta\to\varepsilon\to\theta)\to(\delta\to\varepsilon)\to\delta\to\theta$

Curry-Howardov izomorfizmus

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

$$\dfrac{\dfrac{S:(A\to(B\to A)\to A) \to (A\to(B\to A)) \to (A\to A) \qquad K:(A\to(B\to A)\to A)}{(S\ K):(A\to(B\to A)) \to (A\to A) \qquad K:(A\to(B\to A))}}{((S\ K)\ K):(A\to A)}$$

Hierarchia



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

$\sigma ::= Int \mid \sigma \to \sigma \mid \alpha \mid \forall\alpha.\sigma$

- všeobecne kvantifikovaný typ
- α je typová premenná
- v Haskelli, id :: $\alpha \to \alpha$

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

Dôsledky:

- v takomto λ-kalkule vieme otypovať aj to, čo v F_1 nevieme (ω) ... 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, … $\to F_\omega$
- dependent type (t:Type, t)

System F_2

Typy

$$\sigma ::= Int \mid \sigma \rightarrow \sigma \mid \alpha \mid \forall\alpha.\sigma$$

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

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

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

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

$\dfrac{\Gamma \vdash M{:}\forall\alpha.\beta}{\Gamma \vdash M{:}\beta[\alpha{:}\theta]}$ [INST]

$\lambda x.x : \forall\alpha.\alpha\to\alpha$
AND : $\forall\alpha.\alpha\to\alpha\to\alpha$
$\underline{0},\underline{1},\underline{2},\ldots : \forall t.(t\to t)\to(t\to t)$

Príklady v F_2

$$\begin{array}{ll}
\underline{\{x:\alpha\} \vdash x:\alpha} & [VAR] \\
\underline{\vdash(\lambda x.x):\alpha\to\alpha} & [ABS] \\
\underline{\vdash(\lambda x.x): \forall\alpha.\alpha\to\alpha} & [GEN] \\
\vdash(\lambda x.x): Int\to Int & [INST]
\end{array}$$

$$\begin{array}{ll}
\underline{\{x:Int\} \vdash x:Int} & [VAR] \\
\vdash(\lambda x.x):Int\to Int & [ABS]
\end{array}$$

$\underline{\{x:\forall\alpha.\alpha\to\alpha\} \vdash x:\forall\alpha.\alpha\to\alpha}$ [VAR]

$\underline{\boldsymbol{\{x:\forall\alpha.\alpha\to\alpha\} \vdash x:(\forall\beta.\beta\to\beta)\to(\forall\beta.\beta\to\beta)}\ \textbf{[INST]}}$ -- α nahradíme za $(\forall\beta.\beta\to\beta)$

$\underline{\{x:\forall\alpha.\alpha\to\alpha\} \vdash x:\forall\alpha.\alpha\to\alpha\ [VAR]}$

$\underline{\{x:\forall\alpha.\alpha\to\alpha\} \vdash x:\forall\beta.\beta\to\beta}$ -- α nahradíme za β

$\underline{\{x:\forall\alpha.\alpha\to\alpha\} \vdash (x\ x): \forall\beta.\beta\to\beta}$ [APP]

$\underline{\{x:\forall\alpha.\alpha\to\alpha\} \vdash (x\ x): \forall\alpha.\alpha\to\alpha}$ -- β nahradíme za α

$\vdash \lambda x.(x\ x):(\forall\alpha.\alpha\to\alpha)\to(\forall\alpha.\alpha\to\alpha)$ [ABS]

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

Typ v F_2
$\sigma ::= \text{Int} \mid \sigma \to \sigma \mid \alpha \mid \forall\alpha.\sigma$

Let polymorfizmus Hindley-Millner

chceme zakázať kvantifikáciu typov vnútri typového výrazu:

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

Typy (formalizácia):

$\sigma ::= \psi \mid \forall\alpha.\sigma$ -- polymorfné, generické...

$\psi ::= \text{Int} \mid \psi \to \psi \mid \alpha$ -- základné, funkčné a typová premenná

Haskell: 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**))
- foo = let f g = (if (**g True**) then (**g 2**) else (**g 1**)) in f idd

Nahradíme pravidlo [GEN] pravidlom [LET]

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

[LET] α in β, α not free in Γ

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:

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

a pridáme pravidlo typovanej Y konverzie:

$(Y_\alpha \; e^{\alpha \rightarrow \alpha})^\alpha \quad \rightarrow_Y \quad (e^{\alpha \rightarrow \alpha}(Y_\alpha \; e^{\alpha \rightarrow \alpha})^\alpha)^\alpha$

napr. pre faktorial:

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

pravidlo typovanej Y_{Int} konverzie:

$(Y_{Int} \; e^{Int \rightarrow Int})^\alpha \quad \rightarrow_Y \quad (e^{Int \rightarrow Int}(Y_{Int} \; e^{Int \rightarrow Int})^{Int})^{Int}$

- http://dev.stephendiehl.com/fun/006_hindley_milner.html

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]