FUNDAMENTELE LIMBAJELOR DE PROGRAMARE

Cursul 1

Functiile prin grafic
= cele clasice de la matematica (cu domeniu si codomeniu fixat + functie)
= definite extensional (singurul lucru pe care il observam e cum functia duce intrarile in iesiri)
Functiile ca reguli sau formule
= nu e mereu necesar sa stim domeniu si codomeniu
= definite astfel incat sa putem observa si alte detalii

Functii extensional egale	Functii intensional egale
pentru aceeasi intrare obtin aceeasi iesire	au aceeasi formula
$F(x) = G(x)$ pentru orice x din domeniu	$F(x) = x^2 + 1$ $G(x) = (x+1)(x-1)$

Lambda calculul = teoria functiilor ca formule
= sistem care permite manipularea functiilor ca expresii

Fie f functia $x \rightarrow x^2$ si A = f (5). In lambda calcul avem A = $(\lambda x.x^2)$ (5).
functia care duce x in x^2
x este locala/legata in termenul $\lambda x.x^2$

Functiile de nivel inalt = intrarile/iesirile lor sunt tot functii
f o f = λx.f (f (x))
f -> f o f = λf.λx.f (f (x))

Exemplu de evaluare a unei functii:
((λf .λx.f (f (x)))(λy.y2))(5) = 625

Lambda Calcul		
Fara tipuri	Cu tipuri simple	Cu tipuri polimorfice
Nu specificam tipul expresiei, domeniul sau codomeniul functiilor	Specificam mereu tipul oricarei expresii, iar expresiile de forma f(f) dispar	Specificam ca o expresie are tipul X -> X, dar nu specificam cine este X
Avem flexibilitate, dar putem avea erori cand aplicam o functie unui argument pe care nu il poate procesa	Nu avem flexibilitate, nu putem aplica functii unui argument care are alt tip fata de domeniul functiei	

Functie calculabila = exista o metoda pe foaie pentru a calcula f(n) pentru orice n

1. Turing = ddaca poate fi calculata de o masina Turing
2. Godel = ddaca este recursiva
3. Church = ddaca poate fi scrisa ca un lambda termen

Toate aceste trei modele de calculabilitate sunt echivalente.

Lambda calcul – Logica constructiva (o demonstratie trebuie sa fie o constructie, un program, iar lambda calculul este o notatie pentru astfel de programe)

- Logica clasica = plec de la presupuneri si ajung la contradictie
- Logica constructiva = ca sa arat ca un obiect exista, il construiesc explicit

Cursul 2

V = multime infinita de variabile (x, u, z ...)
Lambda termeni: lambda termen = variabila | aplicare | abstractizare

M, N :: = x | (MN) | (λx.M)

În Haskell, \ e folosit în locul simbolului si λ în locul punctului:

λx.x*x ----------> \x -> x * x
λx.x > 0 ----------> \x -> x > 0

Conventii:

1. Se elimina parantezele exterioare
2. Aplicarea este asociativa la stanga: MNP = (MN)P
3. Corpul abstractizarii se extinde la dreapta cat mai mult: λx.MN = λx.(MN)
4. Mai multi λ pot fi comprimati: λxyz.M = λx.λy.λz.M

Variabiile libere si variabile legate:

- λ_._ = operator de legare
- x din λx._ = variabila de legare
- N din λx.N = domeniu de legare a lui x

Toate aparitiile lui x in N sunt legate.
O aparitie cacre nu e legata = libera
Un termen fara variabile libere = inchis sau combinator

Cum gasim variabilele libere?
Multimea lor e notata FV(M) si FV(x) = {x}

FV(M N) = FV(M) reunit cu FV(N)
FV(λx.M) = FV(M) \ {x}

Ce inseamna sa redenumim o variabila intr-un termen?

Dacă x, y sunt variabile și M este un termen, $M\langle y/x\rangle$ este rezultatul obținut după redenumirea lui x cu y în M.

$$
\begin{aligned}
x\langle y/x\rangle &\equiv y, \\
z\langle y/x\rangle &\equiv z, && \text{dacă } x \neq z \\
(M\,N)\langle y/x\rangle &\equiv (M\langle y/x\rangle)\,(N\langle y/x\rangle) \\
(\lambda x.M)\langle y/x\rangle &\equiv \lambda y.(M\langle y/x\rangle) \\
(\lambda z.M)\langle y/x\rangle &\equiv \lambda z.(M\langle y/x\rangle), && \text{dacă } x \neq z
\end{aligned}
$$

Obs: inlocuim toate aparitiile lui x cu y, indiferent daca e libera, legata sau de legare si se foloseste doar daca y nu apare deja in M

α-echivalenta = cea mai mica relatie de congruenta pe multimea lambda termenilor

λx.M =$_\alpha$ λ.(M<y/x>)

Se satisfac urmatoarele reguli:

$$(refl)\ \frac{}{M = M} \qquad (cong)\ \frac{M = M' \quad N = N'}{MN = M'N'}$$

$$(symm)\ \frac{M = N}{N = M} \qquad (\xi)\ \frac{M = M'}{\lambda x.M = \lambda x.M'}$$

$$(trans)\ \frac{M = N \quad N = P}{M = P} \qquad (\alpha)\ \frac{y \notin M}{\lambda x.M = \lambda y.(M\{y/x\})}$$

Substitutii = vrem sa substituim variabile cu lambda termeni

1. Vrem sa inlocuim doar variabilele libere
2. Nu vrem sa legam variabile libere neintentionat

Definim substitutia aparitiilor libere ale lui x cu N in M (M[N/x]):

$$\begin{array}{lcll} x[N/x] & \equiv & N & \\ y[N/x] & \equiv & y & \text{dacă } x \neq y \\ (M\,P)[N/x] & \equiv & (M[N/x])\,(P[N/x]) & \\ (\lambda x.M)[N/x] & \equiv & \lambda x.M & \\ (\lambda y.M)[N/x] & \equiv & \lambda y.(M[N/x]) & \text{dacă } x \neq y \text{ și } y \notin FV(N) \\ (\lambda y.M)[N/x] & \equiv & \lambda y'.(M\langle y'/y\rangle[N/x]) & \text{dacă } x \neq y,\ y \in FV(N) \\ & & & \text{si } y' \text{ variabilă nouă} \end{array}$$

Cursul 3

- **β-reductie** = procesul de a evalua lambda termeni prin pasarea de argumente functiilor
- **β-redex** = un termen de forma (λx.M)N
- redusul unui redex (λx.M)N = M[N/x]
- forma normala = un lambda termen fara redex-uri

Conventie: spunem ca doi termeni sunt egali daca sunt alfa-echivalenti.

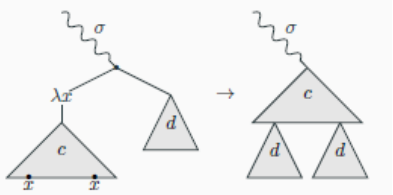
Reducem lambda termeni prin gasirea unui subtermen care este redex si apoi inlocuirea acelui redex cu redusul sau. Repetam acest proces de cate ori putem, pana nu mai sunt redex-uri.

$$(\beta)\ \frac{}{(\lambda x.M)N \to_\beta M[N/x]}$$

$$(cong_1)\ \frac{M \to_\beta M'}{MN \to_\beta M'N}$$

$$(cong_2)\ \frac{N \to_\beta N'}{MN \to_\beta MN'}$$

$$(\xi)\ \frac{M \to_\beta M'}{\lambda x.M \to_\beta \lambda x.M'}$$



Exemple + Observatii:

$$
\begin{array}{lll}
(\lambda x.y)\,(\underline{(\lambda z.zz)\,(\lambda w.w)}) & \to_\beta & (\lambda x.y)\,((z\,z)[\lambda w.w/z]) \\
& \equiv & (\lambda x.y)\,((z[\lambda w.w/z])\,(z[\lambda w.w/z]) \\
& \equiv & (\lambda x.y)\,(\underline{(\lambda w.w)\,(\lambda w.w)}) \\
& \to_\beta & \underline{(\lambda x.y)\,(\lambda w.w)} \\
& \to_\beta & y
\end{array}
$$

$$
\begin{array}{lll}
(\lambda x.y)\,(\underline{(\lambda z.zz)\,(\lambda w.w)}) & \to_\beta & (\lambda x.y)\,(\underline{(\lambda w.w)\,(\lambda w.w)}) \\
& \to_\beta & \underline{(\lambda x.y)\,(\lambda w.w)} \\
& \to_\beta & y
\end{array}
$$

$$
\begin{array}{lll}
\underline{(\lambda x.y)\,((\lambda z.zz)\,(\lambda w.w))} & \to_\beta & y[(\lambda z.zz)\,(\lambda w.w)/x] \\
& \equiv & y
\end{array}
$$

- reducerea unui redex poate crea noi redex-uri sau sterge alte redex-uri
- numarul de pasi necesari poate varia
- rezultatul final nu depinde de alegerea redex-urilor

$$
\begin{array}{lll}
\omega \equiv \underline{(\lambda x.x\,x)\,(\lambda y.y\,y)} & \to_\beta & (\lambda y.y\,y)\,(\lambda y.y\,y) \equiv \omega \\
& \to_\beta & \ldots
\end{array}
$$

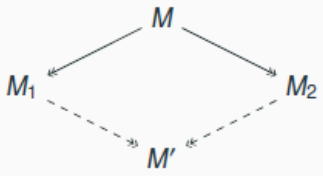
- exista lambda termeni care nu pot fi redusi
- lungimea unui termen nu trebuie sa scada in acest proces (creste sau ramane neschimbata)

$$
\begin{array}{lll}
\underline{(\lambda xy.y)\,((\lambda o.o\,o)\,(\lambda p.p\,p))}\,(\lambda z.z) & \to_\beta & \underline{(\lambda y.y)\,(\lambda z.z)} \\
& \to_\beta & \lambda z.z
\end{array}
$$

$$
\begin{array}{lll}
(\lambda xy.y)\,(\underline{(\lambda o.o\,o)\,(\lambda p.p\,p)})\,(\lambda z.z) & \to_\beta & (\lambda xy.y)\,(\underline{(\lambda p.p\,p)\,(\lambda p.p\,p)})\,(\lambda z.z) \\
& \to_\beta & \ldots
\end{array}
$$

- exista lambda termeni care desi pot fi redusi la o forma normala, pot sa nu o atinga niciodata

Teorema Church-Rosser. Dacă $M \twoheadrightarrow_\beta M_1$ și $M \twoheadrightarrow_\beta M_2$ atunci există M' astfel încât $M_1 \twoheadrightarrow_\beta M'$ și $M_2 \twoheadrightarrow_\beta M'$.



Consecință. Un lambda termen are cel mult o β-formă normală (modulo α-echivalență).

$M \twoheadrightarrow_\beta M'$ = M poate fi β-redus pana la M' in 0 sau mai multi pasi

- M slab normalizabil = exista N in forma normala astfel incat $M \twoheadrightarrow_\beta N$
- M puternic normalizabil = nu exista reduceri infinite care incep din M

Orice termen puternic normalizabil este si slab normalizabil

STRATEGII DE EVALUARE:

1. Strategia normala = alegem redex-ul cel mai din stanga care nu e continut de alt redex

$$\begin{aligned} \underline{((\lambda a.a)\,(\lambda xy.y))}\,((\lambda o.o\,o)\,(\lambda p.p\,p))\,(\lambda z.z) &\to_\beta \\ \underline{(\lambda xy.y)\,((\lambda o.o\,o)\,(\lambda p.p\,p))}\,(\lambda z.z) &\to_\beta \underline{(\lambda y.y)\,(\lambda x.x)} \\ &\to_\beta \lambda x.x \end{aligned}$$

2. Strategia aplicativa = alegem redex-ul cel mai din stanga care nu contine alte redex-uri

$$(\lambda xy.y)\,(\underline{(\lambda x.x\,x)\,(\lambda x.x\,x)})\,(\lambda z.z) \quad \to_\beta \quad (\lambda xy.y)\,((\lambda x.x\,x)\,(\lambda x.x\,x))\,(\lambda z.z)$$

3. Strategia call-by-name (CBN) = strategia normala fara a face reduceri in corpul unei λ-abstractizari

4. Strategia call-by-value (CBV) = strategia aplicativa fara a face reduceri in corpul unei λ-abstractizari

CBV	CBN
Majoritatea limbajelor de programare functionala	Haskell
Functiile pot fi evaluate doar prin valori	Amanam evaluarea argumentelor cat mai mult (lazy evaluation)

Strategia CBV:

$$\begin{aligned} (\lambda x.succ\,x)\,((\lambda y.succ\,y)\,3) &\twoheadrightarrow_\beta (\lambda x.succ\,x)\,(succ\,3) \\ &\to (\lambda x.succ\,x)\,4 \\ &\twoheadrightarrow_\beta succ\,4 \\ &\to 5 \end{aligned}$$

Strategia CBN:

$$\begin{aligned} (\lambda x.succ\,x)\,((\lambda y.succ\,y)\,3) &\twoheadrightarrow_\beta succ\,((\lambda y.succ\,y)\,3) \\ &\twoheadrightarrow_\beta succ\,(succ\,3) \\ &\to succ\,4 \\ &\to 5 \end{aligned}$$

Expresivitatea λ-calculului

- Valori booleene (Bool)
- Valori optiune (Maybe a)
- Perechi (Pair a b)
- Liste (List l)
- Numere naturale

Pentru Bool, definim T si F:

Bool ifTrue ifFalse b = b ifTre ifFalse

$$\mathbf{T} \triangleq \lambda xy.x \qquad \mathbf{F} \triangleq \lambda xy.y \qquad \mathbf{bool} \triangleq \lambda tfb.btf$$

$$\begin{aligned} \mathbf{if} &\triangleq \lambda btf.\mathbf{bool}\, t\, f\, b \\ \mathbf{and} &\triangleq \lambda b_1 b_2.\mathbf{if}\, b_1\, b_2\, \mathbf{F} \\ \mathbf{or} &\triangleq \lambda b_1 b_2.\mathbf{if}\, b_1\, \mathbf{T}\, b_2 \\ \mathbf{not} &\triangleq \lambda b_1.\mathbf{if}\, b_1\, \mathbf{F}\, \mathbf{T} \end{aligned}$$

Pentru Maybe, definim Nothing si Just:

$\mathbf{Nothing} \triangleq \lambda nj.n$ (dintre cele două alternative o alege pe prima)

$\mathbf{Just} \triangleq \lambda anj.ja$ (**Just** *a* aplică al doilea argument valorii *a*)

Pentru Pair, definim:

$\mathbf{Pair} \triangleq \lambda abf.fab$ (**Pair** *a b* aplică funcția valorilor încapsulate)

Pentru List, definim:

$\mathbf{Nil} \triangleq \lambda fi.i$ (alege valoarea inițială)

$\mathbf{Cons} \triangleq \lambda alfi.fa(lfi)$ (**Cons** *a l* agregează lista, apoi agregheză valorea *a* în rezultat)

Pentru numere naturale:

$\mathbf{Zero} \triangleq \lambda fi.i \qquad \mathbf{Succ} \triangleq \lambda nfi.f(nfi) \qquad \mathbf{iterate} \triangleq \lambda fin.nfi$

Numeralul Church pentru numărul $n \in \mathbb{N}$ este notat $\overline{n}$.

Numeralul Church $\overline{n}$ este forma normală a λ-termenului $\mathbf{Succ}^n\, \mathbf{Zero}$, adică $\lambda fi.f^n\, i$, unde f^n reprezintă compunerea lui f cu ea însăși de n ori:

$$\begin{aligned} \overline{0} &\triangleq \lambda fi.f^0\, i = \lambda fi.i \\ \overline{1} &\triangleq \lambda fi.f^1\, i = \lambda fi.f\, i \\ \overline{2} &\triangleq \lambda fi.f^2\, i = \lambda fi.f\,(f\, i) \\ \overline{3} &\triangleq \lambda fx.f^3\, i = \lambda fi.f\,(f\,(f\, i)) \\ &\vdots \\ \overline{n} &\triangleq \lambda fi.f^n\, i = \lambda fi.\underbrace{f(f(\ldots(f}_{n}\, i)\ldots)) \end{aligned}$$

Obs: Succ pe argumentul n returneaza o functie care primeste ca argument o functie f, îi aplica n pentru a obtine compunerea de n ori a lui f cu ea însasi, apoi aplica iar f pentru a obtine compunerea de n + 1 ori a lui f cu ea insasi.

$$
\begin{aligned}
\mathbf{Succ}\,\overline{n} &= (\lambda nfx.f\,(n\,f\,x))\,\overline{n} \\
&\twoheadrightarrow_\beta \lambda fx.f\,(\overline{n}\,f\,x) \\
&\twoheadrightarrow_\beta \lambda fx.f\,(f^n\,x) \\
&= \lambda fx.f^{n+1}\,x \\
&= \overline{n+1}
\end{aligned}
$$

Acum, putem defini adunarea:

$$\mathbf{add} \triangleq \lambda mnfx.m\,f\,(n\,f\,x)$$

Pentru argumentele $\overline{m}$ și $\overline{n}$, obținem:

$$
\begin{aligned}
\mathbf{add}\,\overline{m}\,\overline{n} &= (\lambda mnfx.m\,f\,(n\,f\,x))\,\overline{m}\,\overline{n} \\
&\twoheadrightarrow_\beta \lambda fx.\overline{m}\,f\,(\overline{n}\,f\,x) \\
&\twoheadrightarrow_\beta \lambda fx.f^m\,(f^n\,x) \\
&= \lambda fx.f^{m+n}\,x \\
&= \overline{m+n}
\end{aligned}
$$

Am folosit compunerea lui f^m cu f^n pentru a obține f^{m+n}.

Putem defini adunarea și ca aplicarea repetată a funcției succesor:

$$\mathbf{add'} \triangleq \lambda mn.m\,\mathbf{Succ}\,n$$

$$
\begin{aligned}
\mathbf{add'}\,\overline{m}\,\overline{n} &= (\lambda mn.m\,\mathbf{Succ}\,n)\,\overline{m}\,\overline{n} \\
&\twoheadrightarrow_\beta \overline{m}\,\mathbf{Succ}\,\overline{n} \\
&= (\lambda fx.f^m\,x)\,\mathbf{Succ}\,\overline{n} \\
&\twoheadrightarrow_\beta \mathbf{Succ}^m\,\overline{n} \\
&= \underbrace{\mathbf{Succ}(\mathbf{Succ}(\ldots(\mathbf{Succ}}_{m}\,\overline{n})\ldots)) \\
&\twoheadrightarrow_\beta \underbrace{\mathbf{Succ}(\mathbf{Succ}(\ldots(\mathbf{Succ}}_{m-1}\,\overline{n+1})\ldots)) \\
&\twoheadrightarrow_\beta \overline{m+n}
\end{aligned}
$$

Pentru a defini inmultirea:

Similar înmulțirea este adunare repetată, iar ridicarea la putere este înmulțire repetată:

$$
\begin{aligned}
\mathbf{mul} &\triangleq \lambda mn.m\,(\mathbf{add}\,n)\,\overline{0} \\
\mathbf{exp} &\triangleq \lambda mn.m\,(\mathbf{mul}\,n)\,\overline{1}
\end{aligned}
$$

In plus, putem avea o functie de la numere naturale la booleeni care verifica daca un numar natural este 0 sau nu:

$$\mathbf{isZero} \triangleq \lambda nxy.n\,(\lambda z.y)\,x$$

Cursul 4

Puncte fixe = x punct fix al lui f daca f(x) = x
- o functie poate avea mai multe puncte fixe sau chiar o infinitate (f(x)=x)

Am notat cu M $\twoheadrightarrow_\beta$ M' faptul ca M poate fi β-redus pâna la M' în 0 sau mai multi pasi de β-reductie.
$\twoheadrightarrow_\beta$ este închiderea reflexiva si tranzitiva a relatiei $\rightarrow_\beta$

Notam cu M $=_\beta$ M' faptul ca M poate fi transformat în M' în 0 sau mai multi pasi de β-reductie, transformare în care pasii de reductie pot fi si întorsi.
$=_\beta$ este închiderea reflexiva, simetrica si tranzitiva a relatiei $\rightarrow_\beta$.

De exemplu, avem $(\lambda y.y\,v)\,z =_\beta (\lambda x.z\,x)\,v$ deoarece avem

$$(\lambda y.y\,v)\,z \rightarrow_\beta z\,v \leftarrow_\beta (\lambda x.z\,x)\,v$$

Daca F si M sunt λ-termeni, spunem ca M este un punct fix al lui F daca F M $=_\beta$ M.
Teorema: În lambda calculul fara tipuri, orice termen are un punct fix.

Combinatorii de puncte fixe = termeni inchisi care construiesc un punct fix pentru un termen arbitrar

Combinatorul de punct fix al lui Curry
$\mathbf{Y} \triangleq \lambda y.(\lambda x.y\,(x\,x))\,(\lambda x.y\,(x\,x))$
Pentru orice termen F, $\mathbf{Y}F$ este un punct fix al lui F deoarece
$\mathbf{Y}F \twoheadrightarrow_\beta F\,(\mathbf{Y}\,F)$.

Combinatorul de punct fix al lui Turing
$\Theta \triangleq (\lambda xy.y\,(x\,x\,y))\,(\lambda xy.y\,(x\,x\,y))$
Pentru orice termen F, ΘF este un punct fix al lui F deoarece
$\Theta F \twoheadrightarrow_\beta F\,(\Theta\,F)$.

Punctele fixe ne permit sa rezolvam ecuatii.
Un model de ecuatie:
fact n = if (isZero n) (1) (mul n (fact(pred n)))

Să rezolvăm ecuația de mai sus. Rescriem problema puțin

$$\begin{aligned} \mathbf{fact} &= \lambda n.\mathbf{if}\,(\mathbf{isZero}\,n)\,(\overline{1})\,(\mathbf{mul}\,n\,(\mathbf{fact}(\mathbf{pred}\,n))) \\ \mathbf{fact} &= (\lambda fn.\mathbf{if}\,(\mathbf{isZero}\,n)\,(\overline{1})\,(\mathbf{mul}\,n\,(f(\mathbf{pred}\,n))))\,\mathbf{fact} \end{aligned}$$

Notăm termenul $\lambda fn.\mathbf{if}\,(\mathbf{isZero}\,n)\,(\overline{1})\,(\mathbf{mul}\,n\,(f(\mathbf{pred}\,n)))$ cu F.

Ultima ecuație devine $\mathbf{fact} = F\,\mathbf{fact}$, o ecuație de punct fix.

Am văzut că $\mathbf{Y}\,F$ este un punct fix pentru F (adică $\mathbf{Y}\,F \twoheadrightarrow_\beta F\,(\mathbf{Y}\,F)$), de aceea putem rezolva ecuația de mai sus luând

$$\begin{aligned} \mathbf{fact} &\triangleq \mathbf{Y}\,F \\ \mathbf{fact} &\triangleq \mathbf{Y}\,(\lambda fn.\mathbf{if}\,(\mathbf{isZero}\,n)\,(\overline{1})\,(\mathbf{mul}\,n\,(f(\mathbf{pred}\,n)))) \end{aligned}$$

Observați că **fact** a dispărut din partea dreaptă.

Cursul 5

De ce nu e prea ok lambda calcul fara tipuri?

- Aplicari xx sau MM sunt permise, dar sunt contraintuitive
- Existenta formelor normale pentru λ-termeni nu este garantata si putem avea calcule infinite
- Orice λ-termen are un un punct fix -> nu e in armonie cu ceea ce stim despre functii oarecare

Avem o multime de tipuri variabila V = {α, β, γ, ...}.
Multimea tipurilor simple T e definita prin T = V | T -> T

- Tipul variabila: daca α ∈ V, atunci ∈ T
- Tipul sageata: daca σ, τ ∈ T, atunci (σ -> τ) ∈ T

Tipurile **variabila** = reprezentari abstracte pentru tipuri de baza cum ar fi Nat sau List
Tipurile **sageata** = reprezentari pentru tipuri de functii precum (Nat -> Real)

IN TIPUL SAGEATA, PARANTEZELE SUNT ASOCIATIVE LA DREAPTA.

Ce înseamnă că un termen M are un tip σ?
Vom nota acest lucru cu $M : \sigma$.

Variabilă. Dacă o variabilă x are un tip σ, notăm cu $x : \sigma$.
Convenția Barendregt: variabilele legate sunt distincte.
Presupunem că orice variabilă din M are un unic tip.
Dacă $x : \sigma$ și $x : \tau$, atunci $\sigma \equiv \tau$.

Aplicare. Pentru $M\,N$ este clar că vrem să știm tipurile lui M și N.
Intuitiv, $M\,N$ înseamnă că ("funcția") M este aplicată ("intrării") N.
Atunci M trebuie să aibă un tip funcție, adică $M : \sigma \to \tau$, iar N trebuie să fie "adecvat" pentru această funcție, adică $N : \sigma$.

Dacă $M : \sigma \to \tau$ și $N : \sigma$, atunci $M\,N : \tau$.

Abstractizare. Dacă $M : \tau$, ce tip trebuie sa aibă $\lambda x.\, M$?
Dacă $x : \sigma$ și $M : \tau$, atunci $\lambda x.\, M : \sigma \to \tau$.

Variabilă. $x : \sigma$.
Aplicare. Dacă $M : \sigma \to \tau$ și $N : \sigma$, atunci $M\,N : \tau$.
Abstractizare. Dacă $x : \sigma$ și $M : \tau$, atunci $\lambda x.\, M : \sigma \to \tau$.

M *are tip* (este *typeable*) dacă există un tip σ astfel încât $M : \sigma$.

Exemple.

- Dacă $x : \sigma$, atunci funcția identitate are tipul $\lambda x.\, x : \sigma \to \sigma$.
- Conform convențiilor de la aplicare, $y\,x$ poate avea un tip doar dacă y are un tip săgeată de forma $\sigma \to \tau$ și tipul lui x se potrivește cu tipul domeniu σ. În acest caz, tipul lui $y\,x : \tau$.
- Termenul $x\,x$ nu poate avea nici un tip (nu este typeable). Pe de o parte, x ar trebui să aibă tipul $\sigma \to \tau$ (pentru prima apariție), pe de altă ar trebui să aibă tipul σ (pentru a doua apariție). Cum am stabilit că orice variabilă are un unic tip, obținem $\sigma \to \tau \equiv \sigma$, ceea ce este imposibil.

Asociativitatea la dreapta pentru tipurile sageata VS Asociativitatea la stanga pentru aplicare

Asociere explicita (Church-typing)	Asociere implicita (Curry-typing)
• Consta în prescrierea unui unic tip pentru fiecare variabila, la introducerea acesteia. • Presupune ca tipurile variabilelor sunt explicit stabilite. • Tipurile termenilor mai complecsi se obtin natural, tinând cont de conventiile pentru aplicare si abstractizare.	• Consta în a nu prescriere un tip pentru fiecare variabila, ci în a le lasa "deschise" (implicite). • În acest caz, termenii typeable sunt descoperiti printr-un proces de cautare, care poate presupune "ghicirea" anumitor tipuri.

Multimea λ-termenilor cu pre-tipuri Λ_T este

$\Lambda_T = x \mid \Lambda_T \Lambda_T \mid \lambda x : T. \Lambda_T$

Definitii:

- **Afirmatie** = expresie de forma M:σ, unde M ∈ Λ_T si σ ∈ T (M = subiect si σ = tip)
- **Declaratie** = o afirmatie in care subiectul e o variabila (x :σ)
- **Context** = lista de declaratii cu subiecti diferiti
- **Judecata** = o expresie Γ ⊢ M:σ, unde Γ este context si M:σ este o afirmatie

$$\frac{}{\Gamma \vdash x : \sigma} \text{ dacă } x : \sigma \in \Gamma \ (var)$$

$$\frac{\Gamma \vdash M : \sigma \to \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash M\,N : \tau} \ (app)$$

$$\frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash (\lambda x : \sigma.\, M) : \sigma \to \tau} \ (abs)$$

Un termen M în calculul $\lambda\!\to$ este legal dacă există un context Γ și un tip ρ astfel încât $\Gamma \vdash M : \rho$.

Un exemplu de exercitiu:
Sa aratam ca termenul λy : α → β. λz : α. yz are tipul (α → β) → α → β în contextul vid.

Putem reprezenta in mai multe moduri rezolvarea:

1. Stilul arbore

$$\emptyset \vdash (\lambda y : \alpha \to \beta.\, \lambda z : \alpha.\, yz) : (\alpha \to \beta) \to \alpha \to \beta$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{y : \alpha \to \beta, z : \alpha \vdash y : \alpha \to \beta}\ (var) \quad \dfrac{}{y : \alpha \to \beta, z : \alpha \vdash z : \alpha}\ (var)}{y : \alpha \to \beta, z : \alpha \vdash (yz) : \beta}\ (app)}{y : \alpha \to \beta \vdash (\lambda z : \alpha.\, yz) : \alpha \to \beta}\ (abs)}{\emptyset \vdash (\lambda y : \alpha \to \beta.\, \lambda z : \alpha.\, yz) : (\alpha \to \beta) \to \alpha \to \beta}\ (abs)$$

2. Stilul liniar

1.	$y:\alpha \to \beta, z:\alpha \vdash y:\alpha \to \beta$	(var)
2.	$y:\alpha \to \beta, z:\alpha \vdash z:\alpha$	(var)
3.	$y:\alpha \to \beta, z:\alpha \vdash (yz):\beta$	(app) cu 1 și 2
4.	$y:\alpha \to \beta \vdash (\lambda z:\alpha.\, yz):\alpha \to \beta$	(abs) cu 3
5.	$\emptyset \vdash (\lambda y:\alpha \to \beta.\, \lambda z:\alpha.\, yz):(\alpha \to \beta) \to \alpha \to \beta$	(abs) cu 4

3. Stilul cutii

1.	$y:\alpha \to \beta$	(*context*)
2.	$z:\alpha$	(*context*)
3.	$(yz):\beta$	(*app*) cu 1 și 2
4.	$(\lambda z:\alpha.\, yz):\alpha \to \beta$	(*abs*) cu 3
5.	$(\lambda y:\alpha \to \beta.\, \lambda z:\alpha.\, yz):(\alpha \to \beta) \to \alpha \to \beta$	(*abs*) cu 4

Cursul 6

Ce **probleme** putem sa rezolvam in teoria tipurilor?

1. **Type Checking** = putem gasi o derivare pentru un context si un tip
   context ⊢ term: type
2. **Well-typedness** (Typability) = verificam daca un termen este legal (cautam un context si un tip)
   ? ⊢ term: ?
3. **Type Assignment** = Typability, dar contextul este dat si trebuie sa gasim tipul
   context ⊢ term: ?
4. **Term Finding** = avem contextul si tipul, trebuie sa stabilim daca exista un termen cu acel tip dat
   context ⊢ ? : type

Toate aceste probleme sunt decidabile pentru calculul Church λ→.

Ce **limitari** avem?

1. Nu mai avem recursie nelimitata (combinatorii de punct fix nu sunt typeable).
   Y ≜ λy. (λx. y (x x)) (λx. y (x x)) nu este typeable.
   Dar avem recursie primitiva (permite doar looping in care nr de iteratii este cunoscut dinainte):
   add ≜ λmnfx.mf (n f x)
2. Tipurile pot fi prea restrictive (ca solutii avem let-polymorphism + cuantificatori de tipuri)
   (λf . if (f T) (f 3) (f 5)) (λx. x) ar trebui sa aiba un tip, dar nu are!

Multimea tipurilor:
T = V | T → T | **Unit**

$$\Lambda_T = x \mid \Lambda_T\,\Lambda_T \mid \lambda x : T.\,\Lambda_T \mid \text{unit}$$

$$\frac{}{\Gamma \vdash \text{unit} : \text{Unit}}\ (unit)$$

Multimea tipurilor:

T = V | T → T | Unit | **Void**

Nu exista regula de tipuri pentru deoarece tipul Void nu are inhabitant (e la fel ca mai sus).

Multimea tipurilor:

T = V | T → T | Unit | Void | **TxT**

$$\Lambda_T = x \mid \Lambda_T\,\Lambda_T \mid \lambda x : T.\,\Lambda_T \mid \text{unit} \mid \langle \Lambda_T, \Lambda_T \rangle \mid fst\ \Lambda_T \mid snd\ \Lambda_T$$

$$\frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash N : \tau}{\Gamma \vdash \langle M, N \rangle : \sigma \times \tau}\ (\times_I)$$

$$\frac{\Gamma \vdash M : \sigma \times \tau}{\Gamma \vdash fst\ M : \sigma}\ (\times_{E_1}) \qquad \frac{\Gamma \vdash M : \sigma \times \tau}{\Gamma \vdash snd\ M : \tau}\ (\times_{E_2})$$

Multimea tipurilor:

T = V | T → T | Unit | Void | TxT | **T + T**

$$\Lambda_T = x \mid \Lambda_T\,\Lambda_T \mid \lambda x : T.\,\Lambda_T \mid \text{unit} \mid \langle \Lambda_T, \Lambda_T \rangle \mid fst\ \Lambda_T \mid snd\ \Lambda_T \mid \text{Left}\ \Lambda_T \mid \text{Right}\ \Lambda_T \mid \text{case}\ \Lambda_T\ \text{of}\ \Lambda_T\ ;\ \Lambda_T$$

$$\frac{\Gamma \vdash M : \sigma}{\Gamma \vdash \text{Left}\ M : \sigma + \tau}\ (+_{I_1}) \qquad \frac{\Gamma \vdash M : \tau}{\Gamma \vdash \text{Right}\ M : \sigma + \tau}\ (+_{I_2})$$

$$\frac{\Gamma \vdash M : \sigma + \tau \quad \Gamma \vdash M_1 : \sigma \to \gamma \quad \Gamma \vdash M_2 : \tau \to \gamma}{\Gamma \vdash \text{case}\ M\ \text{of}\ M_1\ ;\ M_2 : \gamma}\ (+_E)$$

Corespondenta Curry-Howard

Teoria Tipurilor	Logica
Tipuri	Formule
Termeni	Demonstratii
Inhabitation a tipului σ	Demonstratie a lui σ
Tip produs	Conjunctie
Tip functie	Implicatie
Tip suma	Disjunctie
Tipul void	False
Tipul unit	True

Corectitudine = sintaxa implica semantica

Completitudine = sintaxa si semantica coincid

λ-calcul cu tipuri Deductie naturala

$\Gamma \vdash M:\sigma$ $\Gamma \vdash \sigma$

Faptul ca exista un termen de tip σ (inhabitation of type σ) inseamna ca σ este teorema sau ca are o demonstratie in logica.

λ-calcul cu tipuri	Deducție naturală
$\dfrac{\Gamma \vdash M : \sigma \quad \Gamma \vdash N : \tau}{\Gamma \vdash \langle M, N\rangle : \sigma \times \tau}\ (\times_I)$	$\dfrac{\Gamma \vdash \sigma \quad \Gamma \vdash \tau}{\Gamma \vdash \sigma \wedge \tau}\ (\wedge_I)$
$\dfrac{\Gamma \vdash M : \sigma \times \tau}{\Gamma \vdash fst\ M : \sigma}\ (\times_{E_1})$	$\dfrac{\Gamma \vdash \sigma \wedge \tau}{\Gamma \vdash \sigma}\ (\wedge_{E_1})$
$\dfrac{\Gamma \vdash p : \sigma \times \tau}{\Gamma \vdash snd\ p : \tau}\ (\times_{E_2})$	$\dfrac{\Gamma \vdash \sigma \wedge \tau}{\Gamma \vdash \tau}\ (\wedge_{E_2})$
$\dfrac{\Gamma \cup \{x : \sigma\} \vdash M : \tau}{\Gamma \vdash \lambda x.M : \sigma \to \tau}\ (\to_I)$	$\dfrac{\Gamma \cup \{\sigma\} \vdash \tau}{\Gamma \vdash \sigma \supset \tau}\ (\supset_I)$
$\dfrac{\Gamma \vdash M : \sigma \to \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \tau}\ (\to_E)$	$\dfrac{\Gamma \vdash \sigma \supset \tau \quad \Gamma \vdash \sigma}{\Gamma \vdash \tau}\ (\supset_E)$

Propositions are types! ♡

λ-calcul cu tipuri	Deducție naturală
$\dfrac{\{x : \sigma\} \vdash x : \sigma}{\vdash \lambda x.\, x : \sigma \to \sigma}\ (\to_I)$	$\dfrac{\{\sigma\} \vdash \sigma}{\vdash \sigma \supset \sigma}\ (\supset_I)$
$\dfrac{\dfrac{\overline{\{x : \sigma, y : \tau\} \vdash x : \sigma}}{\{x : \sigma\} \vdash \lambda y.\, x : \tau \to \sigma}\ (\to_I)}{\vdash \lambda x.\, (\lambda y.\, x) : \sigma \to (\tau \to \sigma)}\ (\to_I)$	$\dfrac{\dfrac{\overline{\{\sigma, \tau\} \vdash \sigma}}{\{\sigma\} \vdash \tau \to \sigma}\ (\supset_I)}{\vdash \sigma \to (\tau \to \sigma)}\ (\supset_I)$

Proofs are Terms! ♡

Demonstrațiile sunt termeni!

Formulele de mai jos nu sunt demonstrabile in logica intuitionista:

- dubla negatie: ¬¬ϕ ⊃ ϕ
- excluded middle: ϕ ∨ ¬ϕ
- legea lui Pierce: ((ϕ ⊃ τ) ⊃ ϕ) ⊃ ϕ

Nu exista semantica cu tabele de adevar pentru logica intuitionista!

Initial, corespondenta Curry-Howard a fost intre calculul Church λ → si sistemul de deductie naturala al lui Gentzen pentru logica intuitionista.