Paradigme de Programare

Conf. dr. ing. Andrei Olaru

andrei.olaru@cs.pub.ro | cs@andreiolaru.ro
Departamentul de Calculatoare

2020

Cursul 3

Calcul Lambda

Introducere

- ne punem problema dacă putem realiza un calcul sau nu → pentru a demonstra trebuie să avem un model simplu al calculului (cum realizăm calculul, în mod formal).
- un model de calculabilitate trebuie să fie cât mai simplu, atât ca număr de operații disponibile cât și ca mod de construcție a valorilor.
- corectitudinea unui program se demonstrează mai ușor dacă limbajul de programare este mai apropiat de mașina teoretică (modelul abstract de calculabilitate).

Calculul Lambda

λ

- Model de calculabilitate (Alonzo Church, 1932) – introdus în cadrul cercetărilor asupra fundamentelor matematicii. [http://en.wikipedia.org/wiki/Lambda_calculus]
  - sistem formal pentru exprimarea calculului.
- Echivalent cu Mașina Turing (v. Teza Church-Turing)
- Axat pe conceptul matematic de funcție – totul este o funcție

Aplicații
ale calculului λ

- Aplicații importante în
  - programare
  - demonstrarea formală a corectitudinii programelor, datorită modelului simplu de execuție

- Baza teoretică a numeroase limbaje:
  LISP, Scheme, Haskell, ML, F#, Clean, Clojure, Scala, Erlang etc.

Lambda-expresii

λ-expresii
Exemple

Ex Exemplu

1. $x \rightarrow$ variabila (numele) x

λ-expresii
Exemple

Ex Exemplu

1. $x \rightarrow$ variabila (numele) x
2. $\lambda x.x \rightarrow$ funcția identitate

λ-expresii
Exemple

Exemplu

1. $x \rightarrow$ variabila (numele) x
2. $\lambda x.x \rightarrow$ funcția identitate
3. $\lambda x.\lambda y.x \rightarrow$ funcție selector

λ-expresii
Exemple

Ex Exemplu

1. $x \rightarrow$ variabila (numele) x
2. $\lambda x.x \rightarrow$ funcția identitate
3. $\lambda x.\lambda y.x \rightarrow$ funcție selector
4. $(\lambda x.x\ \ y) \rightarrow$ aplicația funcției identitate asupra parametrului actual y

λ-expresii
Exemple

Exemplu

1. x → variabila (numele) x
2. $\lambda x.x$ → funcția identitate
3. $\lambda x.\lambda y.x$ → funcție selector
4. $(\lambda x.x\ \ y)$ → aplicația funcției identitate asupra parametrului actual y
5. $(\lambda x.(x\ \ x)\ \ \lambda x.x)$ → ?

λ-expresii
Exemple

Exemplu

1. $x \rightarrow$ variabila (numele) x
2. $\lambda x.x \rightarrow$ funcția identitate
3. $\lambda x.\lambda y.x \rightarrow$ funcție selector
4. $(\lambda x.x\ \ y) \rightarrow$ aplicația funcției identitate asupra parametrului actual y
5. $(\lambda x.(x\ \ x)\ \ \lambda x.x) \rightarrow$?

Intuitiv, evaluarea aplicației $(\lambda x.x\ \ y)$ presupune substituția textuală a lui x, în corp, prin $y \rightarrow$ rezultat y.

+ λ-expresie

- Variabilă: o variabilă x este o λ-expresie;
- Funcție: dacă x este o variabilă și E este o λ-expresie, atunci $\lambda x.E$ este o λ-expresie, reprezentând funcția anonimă, unară, cu parametrul formal x și corpul E;
- Aplicație: dacă F și A sunt λ-expresii, atunci $(F\ A)$ este o λ-expresie, reprezentând aplicația expresiei F asupra parametrului actual A.

Evaluare

Intuitiv

$$((\lambda x.\lambda y.x\ \ z)\ \ t)$$

Evaluare

Intuitiv

$$(\boxed{(\lambda x.\lambda y.x\ \ z)}\ \ t) \leftarrow$$

parametru formal
parametru actual

Evaluare

Intuitiv

$$(\boxed{(\lambda x.\lambda y.x\ \ z)}\ \ t) \leftarrow$$

parametru formal
parametru actual

$$\Vert$$

substituție

$$\Downarrow$$

$$(\lambda y.z\ \ t)$$

Evaluare

Intuitiv

$$(\boxed{(\lambda x.\lambda y.x\ \ z)}\ \ t) \quad \leftarrow$$

parametru formal
parametru actual

$$\Downarrow$$

substituție

$$\Downarrow$$

$$\boxed{(\lambda y.z\ \ t)} \quad \leftarrow$$

parametru formal
parametru actual

$$(\boxed{(\lambda x.\lambda y.x \; z)} \; t) \quad \leftarrow \begin{array}{l}\text{parametru formal}\\ \text{parametru actual}\end{array}$$

$$\Downarrow \text{ substituție}$$

$$\boxed{(\lambda y.z \; t)} \quad \leftarrow \begin{array}{l}\text{parametru formal}\\ \text{parametru actual}\end{array}$$

$$\Downarrow \text{ substituție}$$

$$z$$

Evaluare

Intuitiv

$$(\boxed{(\lambda x.\lambda y.x\ \ z)}\ \ t) \quad \leftarrow \begin{array}{l}\text{parametru formal}\\ \text{parametru actual}\end{array}$$

$$\|$$

substituție

$$\Downarrow$$

$$\boxed{(\lambda y.z\ \ t)} \quad \leftarrow \begin{array}{l}\text{parametru formal}\\ \text{parametru actual}\end{array}$$

$$\|$$

substituție

$$\Downarrow$$

$$z$$

nu mai este nicio funcție de aplicat

$$\left(\boxed{\left(\lambda x.\lambda y.x \;\; z\right)} \;\; t\right) \quad \leftarrow$$

parametru formal
parametru actual

$$\Downarrow$$

substituție

$$\boxed{\left(\lambda y.z \;\; t\right)} \quad \leftarrow$$

parametru formal
parametru actual

$$\Downarrow$$

substituție

$$z$$

nu mai este nicio funcție de aplicat

- cum știm ce reducem, cum reducem, în ce ordine, și ce apariții ale variabilelor înlocuim?

Reducere

β-redex

Cum arată (Formal, vedem mai târziu)

- β-redex: o λ-expresie de forma: $(\lambda x.E\ \ A)$
  - E – λ-expresie – este corpul funcției
  - A – λ-expresie – este parametrul actual
- β-redexul se reduce la $E_{[A/x]}$ – E cu toate aparițiile libere ale lui x din E înlocuite cu A prin substituție textuală.

Apariții ale variabilelor
Legate vs libere

+ Apariție legată O apariție x_n a unei variabile x este legată într-o expresie E dacă:

- $E = \lambda x.F$ sau
- $E = \ldots\ \lambda x_n.F\ \ldots$ sau
- $E = \ldots\ \lambda x.F\ \ldots$ și x_n apare în F.

+ Apariție liberă O apariție a unei variabile este liberă într-o expresie dacă nu este legată în acea expresie.

- Atenție! În raport cu o expresie dată!

Apariții ale variabilelor

Mod de gândire

· O apariție legată în expresie este o apariție a parametrului formal al unei funcții definite în expresie, în corpul funcției; o apariție liberă este o apariție a parametrului formal al unei funcții definite în exteriorul expresiei, sau nu este parametru formal al niciunei funcții.

- $\underset{<1>}{x}$ ← apariție liberă

Mod de gândire

· O apariție legată în expresie este o apariție a parametrului formal al unei funcții definite în expresie, în corpul funcției; o apariție liberă este o apariție a parametrului formal al unei funcții definite în exteriorul expresiei, sau nu este parametru formal al niciunei funcții.

- $\underset{<1>}{x}$ ← apariție liberă
- $(\lambda y.\ \underset{<1>}{x}\ \ z)$ ← apariție încă liberă, nu o leagă nimeni

Apariții ale variabilelor

Mod de gândire

· O apariție legată în expresie este o apariție a parametrului formal al unei funcții definite în expresie, în corpul funcției; o apariție liberă este o apariție a parametrului formal al unei funcții definite în exteriorul expresiei, sau nu este parametru formal al niciunei funcții.

- $\underset{<1>}{x}$ ← apariție liberă
- $(\lambda y.\underset{<1>}{x}\ z)$ ← apariție încă liberă, nu o leagă nimeni
- $\lambda\underset{<2>}{x}.(\lambda y.\underset{<1>}{x}\ z)$ ← $\lambda\underset{<2>}{x}$ *leagă* apariția $\underset{<1>}{x}$

Apariții ale variabilelor

λ

Mod de gândire

· O apariție legată în expresie este o apariție a parametrului formal al unei funcții definite în expresie, în corpul funcției; o apariție liberă este o apariție a parametrului formal al unei funcții definite în exteriorul expresiei, sau nu este parametru formal al niciunei funcții.

- $\underset{<1>}{x}$ $\leftarrow$ apariție liberă
- $(\lambda y.\ \underset{<1>}{x}\ \ z)$ $\leftarrow$ apariție încă liberă, nu o leagă nimeni
- $\lambda \underset{<2>}{x}.(\lambda y.\ \underset{<1>}{x}\ \ z)$ $\leftarrow$ $\lambda \underset{<2>}{x}$ *leagă* apariția $\underset{<1>}{x}$
- $(\lambda \underset{<2>}{x}.\underbrace{(\lambda y.\ \underset{<1>}{x}\ \ z)}_{\text{corp } \lambda x_2}\ \underset{<3>}{x})$ $\leftarrow$ apariția x_3 este liberă – este în exteriorul corpului funcției cu parametrul formal x (λx_2)

Apariții ale variabilelor

Mod de gândire

· O apariție legată în expresie este o apariție a parametrului formal al unei funcții definite în expresie, în corpul funcției; o apariție liberă este o apariție a parametrului formal al unei funcții definite în exteriorul expresiei, sau nu este parametru formal al niciunei funcții.

- $\underset{<1>}{x}$ ← apariție liberă
- $(\lambda y. \underset{<1>}{x}\ z)$ ← apariție încă liberă, nu o leagă nimeni
- $\lambda \underset{<2>}{x}.(\lambda y. \underset{<1>}{x}\ z)$ ← $\lambda \underset{<2>}{x}$ *leagă* apariția $\underset{<1>}{x}$
- $(\lambda \underset{<2>}{x}.\underbrace{(\lambda y. \underset{<1>}{x}\ z)}_{\text{corp } \lambda x_2}\ \underset{<3>}{x})$ ← apariția x_3 este liberă – este în exteriorul corpului funcției cu parametrul formal x (λx_2)
- $\lambda \underset{<4>}{x}.(\lambda \underset{<2>}{x}.(\lambda y. \underset{<1>}{x}\ z)\ \underset{<3>}{x})$ ← $\lambda \underset{<4>}{x}$ *leagă* apariția $\underset{<3>}{x}$

Variabile

Legate vs libere

+ O variabilă este legată într-o expresie dacă toate aparițiile sale sunt legate în acea expresie.

+ O variabilă este liberă într-o expresie dacă nu este legată în acea expresie i.e. dacă cel puțin o apariție a sa este liberă în acea expresie.

- Atenție! În raport cu o expresie dată!

Variabile și Apariții ale lor

Exemplu 1

λ

Exemplu

În expresia $E = (\lambda x.x\ \ x)$, evidențiem aparițiile lui x:

$(\lambda \underset{<1>}{x} . \underbrace{\underset{<2>}{x}}_{F} \ \underset{<3>}{x})$.

- $\underset{<1>}{x}, \underset{<2>}{x}$ în E
- $\underset{<3>}{x}$ în E
- $\underset{<2>}{x}$ în F!
- x în E și F

Variabile și Apariții ale lor

Exemplu 1

Exemplu

În expresia $E = (\lambda x.x\ x)$, evidențiem aparițiile lui x:

$$(\lambda \underset{<1>}{x} . \underbrace{\underset{<2>}{x}}_{F} \ \underset{<3>}{x}).$$

- $\underset{<1>}{x}$, $\underset{<2>}{x}$ legate în E
- $\underset{<3>}{x}$ în E
- $\underset{<2>}{x}$ în F!
- x în E și F

Exemplu

Ex

În expresia $E = (\lambda x.x\ x)$, evidențiem aparițiile lui x:
$(\lambda \underset{<1>}{x} . \underbrace{\underset{<2>}{x}}_{F} \underset{<3>}{x})$.

- $\underset{<1>}{x}$, $\underset{<2>}{x}$ legate în E
- $\underset{<3>}{x}$ liberă în E
- $\underset{<2>}{x}$ în F!
- x în E și F

Ex
Exemplu

În expresia $E = (\lambda x.x\ \ x)$, evidențiem aparițiile lui x:
$(\lambda \underset{<1>}{x} . \underbrace{\underset{<2>}{x}}_{F}\ \ \underset{<3>}{x})$.

- $\underset{<1>}{x}$, $\underset{<2>}{x}$ legate în E
- $\underset{<3>}{x}$ liberă în E
- $\underset{<2>}{x}$ liberă în F!
- x în E și F

Variabile și Apariții ale lor
Exemplu 1

Exemplu

În expresia $E = (\lambda x.x\ x)$, evidențiem aparițiile lui x:

$(\lambda \underset{<1>}{x} . \underbrace{\underset{<2>}{x}\ \ \underset{<3>}{x}}_{F})$.

- $\underset{<1>}{x}$, $\underset{<2>}{x}$ legate în E
- $\underset{<3>}{x}$ liberă în E
- $\underset{<2>}{x}$ liberă în F!
- x liberă în E și F

Variabile și apariții ale lor

Exemplu 2

λ

Ex Exemplu

În expresia $E = (\lambda x.\lambda z.(z\ x)\ (z\ y))$, evidențiem aparițiile:

$$(\lambda \underset{<1>}{x}.\underbrace{\lambda \underset{<1>}{z}.(\underset{<2>}{z}\ \ \underset{<2>}{x})}_{F}\ (\underset{<3>}{z}\ \ \underset{<1>}{y})).$$

- $\underset{<1>}{x}, \underset{<2>}{x}, \underset{<1>}{z}, \underset{<2>}{z}$ în E
- $\underset{<1>}{y}, \underset{<3>}{z}$ în E
- $\underset{<1>}{z}, \underset{<2>}{z}$ în F
- $\underset{<2>}{x}$ în F
- x în E, în F
- y în E
- z în E, în F

λ

Exemplu

În expresia $E = (\lambda x.\lambda z.(z\ x)\ (z\ y))$, evidențiem aparițiile:

$$(\lambda \underset{<1>}{x}.\underbrace{\lambda \underset{<1>}{z}.(\underset{<2>}{z}\ \underset{<2>}{x})}_{F}\ (\underset{<3>}{z}\ \underset{<1>}{y})).$$

- $\underset{<1>}{x}, \underset{<2>}{x}, \underset{<1>}{z}, \underset{<2>}{z}$ legate în E
- $\underset{<1>}{y}, \underset{<3>}{z}$ în E
- $\underset{<1>}{z}, \underset{<2>}{z}$ în F
- $\underset{<2>}{x}$ în F
- x în E, în F
- y în E
- z în E, în F

Variabile și apariții ale lor

Exemplu 2

Ex Exemplu

În expresia $E = (\lambda x.\lambda z.(z\ x)\ (z\ y))$, evidențiem aparițiile:

$$(\lambda \underset{<1>}{x}.\underbrace{\lambda \underset{<1>}{z}.(\underset{<2>}{z}\ \ \underset{<2>}{x})}_{F}\ (\underset{<3>}{z}\ \ \underset{<1>}{y})).$$

- $\underset{<1>}{x}, \underset{<2>}{x}, \underset{<1>}{z}, \underset{<2>}{z}$ legate în E
- $\underset{<1>}{y}, \underset{<3>}{z}$ libere în E
- $\underset{<1>}{z}, \underset{<2>}{z}$ în F
- $\underset{<2>}{x}$ în F
- x în E, în F
- y în E
- z în E, în F

Variabile și apariții ale lor
Exemplu 2

λ

Ex

Exemplu

În expresia $E = (\lambda x.\lambda z.(z\ x)\ (z\ y))$, evidențiem aparițiile:

$$(\lambda \underset{<1>}{x}.\underbrace{\lambda \underset{<1>}{z}.(\underset{<2>}{z}\ \underset{<2>}{x})}_{F}\ (\underset{<3>}{z}\ \underset{<1>}{y})).$$

- $\underset{<1>}{x}, \underset{<2>}{x}, \underset{<1>}{z}, \underset{<2>}{z}$ legate în E
- $\underset{<1>}{y}, \underset{<3>}{z}$ libere în E
- $\underset{<1>}{z}, \underset{<2>}{z}$ legate în F
- $\underset{<2>}{x}$ în F
- x în E, în F
- y în E
- z în E, în F

Variabile și apariții ale lor

Exemplu 2

λ

Ex Exemplu

În expresia $E = (\lambda x.\lambda z.(z\ x)\ (z\ y))$, evidențiem aparițiile:

$$(\lambda \underset{<1>}{x}.\underbrace{\lambda \underset{<1>}{z}.(\underset{<2>}{z}\ \ \underset{<2>}{x})}_{F}\ (\underset{<3>}{z}\ \ \underset{<1>}{y})).$$

- $\underset{<1>}{x}, \underset{<2>}{x}, \underset{<1>}{z}, \underset{<2>}{z}$ legate în E
- $\underset{<1>}{y}, \underset{<3>}{z}$ libere în E
- $\underset{<1>}{z}, \underset{<2>}{z}$ legate în F
- $\underset{<2>}{x}$ liberă în F
- x în E, în F
- y în E
- z în E, în F

Variabile și apariții ale lor

Exemplu 2

λ

Exemplu

În expresia $E = (\lambda x.\lambda z.(z\ x)\ (z\ y))$, evidențiem aparițiile:

$$(\lambda \underset{<1>}{x}.\underbrace{\lambda \underset{<1>}{z}.(\underset{<2>}{z}\ \ \underset{<2>}{x})}_{F}\ (\underset{<3>}{z}\ \ \underset{<1>}{y})).$$

- $\underset{<1>}{x}, \underset{<2>}{x}, \underset{<1>}{z}, \underset{<2>}{z}$ legate în E
- $\underset{<1>}{y}, \underset{<3>}{z}$ libere în E
- $\underset{<1>}{z}, \underset{<2>}{z}$ legate în F
- $\underset{<2>}{x}$ liberă în F
- x legată în E, dar liberă în F
- y în E
- z în E, în F

Variabile și apariții ale lor

Exemplu 2

λ

Exemplu

În expresia $E = (\lambda x.\lambda z.(z\ x)\ (z\ y))$, evidențiem aparițiile:

$$(\lambda \underset{<1>}{x} .\underbrace{\lambda \underset{<1>}{z} .(\underset{<2>}{z}\ \underset{<2>}{x})}_{F}\ (\underset{<3>}{z}\ \underset{<1>}{y})).$$

- $\underset{<1>}{x}, \underset{<2>}{x}, \underset{<1>}{z}, \underset{<2>}{z}$ legate în E
- $\underset{<1>}{y}, \underset{<3>}{z}$ libere în E
- $\underset{<1>}{z}, \underset{<2>}{z}$ legate în F
- $\underset{<2>}{x}$ liberă în F
- x legată în E, dar liberă în F
- y liberă în E
- z în E, în F

Variabile și apariții ale lor

Exemplu 2

Ex Exemplu

În expresia $E = (\lambda x.\lambda z.(z\ \ x)\ \ (z\ \ y))$, evidențiem aparițiile:

$$(\lambda \underset{<1>}{x}.\underbrace{\lambda \underset{<1>}{z}.(\underset{<2>}{z}\ \ \underset{<2>}{x})}_{F}\ (\underset{<3>}{z}\ \ \underset{<1>}{y})).$$

- $\underset{<1>}{x}$, $\underset{<2>}{x}$, $\underset{<1>}{z}$, $\underset{<2>}{z}$ legate în E
- $\underset{<1>}{y}$, $\underset{<3>}{z}$ libere în E
- $\underset{<1>}{z}$, $\underset{<2>}{z}$ legate în F
- $\underset{<2>}{x}$ liberă în F
- x legată în E, dar liberă în F
- y liberă în E
- z liberă în E, dar legată în F

Determinarea variabilelor libere și legate

O abordare formală

Variabile libere (*free variables*)

- $FV(x) = \{x\}$
- $FV(\lambda x.E) = FV(E) \setminus \{x\}$
- $FV((E_1\ E_2)) = FV(E_1) \cup FV(E_2)$

Variabile legate (*bound variables*)

- $BV(x) = \emptyset$
- $BV(\lambda x.E) = BV(E) \cup \{x\}$
- $BV((E_1\ E_2)) = BV(E_1) \setminus FV(E_2) \cup BV(E_2) \setminus FV(E_1)$

Expresii închise

λ

+ | O expresie închisă este o expresie care nu conține variabile libere.

Ex | Exemplu

- $(\lambda x.x\ \ \lambda x.\lambda y.x)\ \cdots$
- $(\lambda x.x\ \ a)\ \cdots$

- Variabilele libere dintr-o λ-expresie pot sta pentru alte λ-expresii
- Înaintea evaluării, o expresie trebuie adusă la forma închisă.
- Procesul de înlocuire trebuie să se termine.

Expresii închise

λ

+ | O expresie închisă este o expresie care nu conține variabile libere.

Ex | Exemplu

- $(\lambda x.x\ \lambda x.\lambda y.x)$ $\rightarrow$ închisă
- $(\lambda x.x\ a)$ $\cdots$

- Variabilele libere dintr-o λ-expresie pot sta pentru alte λ-expresii
- Înaintea evaluării, o expresie trebuie adusă la forma închisă.
- Procesul de înlocuire trebuie să se termine.

Expresii închise

+ O expresie închisă este o expresie care nu conține variabile libere.

Ex Exemplu

- $(\lambda x.x \ \lambda x.\lambda y.x)$ $\rightarrow$ închisă
- $(\lambda x.x \ a)$ $\rightarrow$ deschisă, deoarece a este liberă

- Variabilele libere dintr-o λ-expresie pot sta pentru alte λ-expresii
- Înaintea evaluării, o expresie trebuie adusă la forma închisă.
- Procesul de înlocuire trebuie să se termine.

β-reducere
Definiție

+ | β-reducere: Evaluarea expresiei $(\lambda x.E\ A)$, cu E și A λ-expresii, prin substituirea textuală a tuturor aparițiilor libere ale parametrului **formal** al funcției, x, din corpul acesteia, E, cu parametrul **actual**, A:

$$(\lambda x.E\ A) \rightarrow_\beta E_{[A/x]}$$

+ | β-redex Expresia $(\lambda x.E\ A)$, cu E și A λ-expresii – o expresie pe care se poate aplica β-reducerea.

β-reducere
Exemple

λ

Ex Exemplu

- $(\lambda x.x \;\; y) \rightarrow_\beta x_{[y/x]} \rightarrow y$
- $(\lambda x.\lambda x.x \;\; y)$
- $(\lambda x.\lambda y.x \;\; y)$

β-reducere
Exemple

Ex Exemplu

- $(\lambda x.x \ \ y) \rightarrow_\beta x_{[y/x]} \rightarrow y$
- $(\lambda x.\lambda x.x \ \ y) \rightarrow_\beta \lambda x.x_{[y/x]} \rightarrow \lambda x.x$
- $(\lambda x.\lambda y.x \ \ y)$

β-reducere
Exemple

λ

Ex Exemplu

- $(\lambda x.x \;\; y) \rightarrow_\beta x_{[y/x]} \rightarrow y$
- $(\lambda x.\lambda x.x \;\; y) \rightarrow_\beta \lambda x.x_{[y/x]} \rightarrow \lambda x.x$
- $(\lambda x.\lambda y.x \;\; y) \rightarrow_\beta \lambda y.x_{[y/x]} \rightarrow \lambda y.y$

β-reducere
Exemple

λ

Exemplu

- $(\lambda x.x \;\; y) \rightarrow_\beta x_{[y/x]} \rightarrow y$
- $(\lambda x.\lambda x.x \;\; y) \rightarrow_\beta \lambda x.x_{[y/x]} \rightarrow \lambda x.x$
- $(\lambda x.\lambda y.x \;\; y) \rightarrow_\beta \lambda y.x_{[y/x]} \rightarrow \lambda y.y$ Greșit! Variabila liberă y devine legată, schimbându-și semnificația. $\rightarrow \lambda y^{(a)}.y^{(b)}$

β-reducere

Exemple

Ex Exemplu

- $(\lambda x.x \;\; y) \rightarrow_\beta x_{[y/x]} \rightarrow y$
- $(\lambda x.\lambda x.x \;\; y) \rightarrow_\beta \lambda x.x_{[y/x]} \rightarrow \lambda x.x$
- $(\lambda x.\lambda y.x \;\; y) \rightarrow_\beta \lambda y.x_{[y/x]} \rightarrow \lambda y.y$ Greșit! Variabila liberă y devine legată, schimbându-și semnificația.
  $\rightarrow \lambda y^{(a)}.y^{(b)}$

  Care este problema?

β-reducere
Coliziuni

- **Problemă**: în expresia $(\lambda x.E\ A)$:
  - dacă variabilele libere din A nu au nume comune cu variabilele legate din E: $\mathrm{FV}(A) \cap \mathrm{BV}(E) = \emptyset$
    $\rightarrow$ reducere întotdeauna corectă
  - dacă există variabilele libere din A care au nume comune cu variabilele legate din E: $\mathrm{FV}(A) \cap \mathrm{BV}(E) \neq \emptyset$
    $\rightarrow$ reducere potențial greșită
- Soluție: redenumirea variabilelor legate din E, ce coincid cu cele libere din A $\rightarrow$ α-conversie.

- **Problemă**: în expresia $(\lambda x.E \;\; A)$:
  - dacă variabilele libere din A nu au nume comune cu variabilele legate din E: $\mathrm{FV}(A) \cap \mathrm{BV}(E) = \emptyset$
    $\rightarrow$ reducere întotdeauna corectă
  - dacă există variabilele libere din A care au nume comune cu variabilele legate din E: $\mathrm{FV}(A) \cap \mathrm{BV}(E) \neq \emptyset$
    $\rightarrow$ reducere potențial greșită
- Soluție: redenumirea variabilelor legate din E, ce coincid cu cele libere din A $\rightarrow$ α-conversie.

Ex Exemplu

$$(\lambda x.\lambda y.x \;\; y) \rightarrow_\alpha (\lambda x.\lambda z.x \;\; y) \rightarrow_\beta \lambda z.x_{[y/x]} \rightarrow \lambda z.y$$

α-conversie
Definiție

+ | α-conversie: Redenumirea sistematică a variabilelor legate dintr-o funcție: $\lambda x.E \rightarrow_\alpha \lambda y.E_{[y/x]}$. Se impun două condiții.

Ex Exemplu

- $\lambda x.y \rightarrow_\alpha \lambda y.y_{[y/x]} \rightarrow \lambda y.y$
- $\lambda x.\lambda y.x \rightarrow_\alpha \lambda y.\lambda y.x_{[y/x]} \rightarrow \lambda y.\lambda y.y$

+ | α-conversie: Redenumirea sistematică a variabilelor legate dintr-o funcție: $\lambda x.E \rightarrow_\alpha \lambda y.E_{[y/x]}$. Se impun două condiții.

Ex

Exemplu

- $\lambda x.y \rightarrow_\alpha \lambda y.y_{[y/x]} \rightarrow \lambda y.y \rightarrow$ Greșit!
- $\lambda x.\lambda y.x \rightarrow_\alpha \lambda y.\lambda y.x_{[y/x]} \rightarrow \lambda y.\lambda y.y$

α-conversie
Definiție

+ | α-conversie: Redenumirea sistematică a variabilelor legate dintr-o funcție: $\lambda x.E \rightarrow_\alpha \lambda y.E_{[y/x]}$. Se impun două condiții.

Ex Exemplu

- $\lambda x.y \rightarrow_\alpha \lambda y.y_{[y/x]} \rightarrow \lambda y.y \rightarrow$ Greșit!
- $\lambda x.\lambda y.x \rightarrow_\alpha \lambda y.\lambda y.x_{[y/x]} \rightarrow \lambda y.\lambda y.y \rightarrow$ Greșit!

α-conversie
Definiție

+ | α-conversie: Redenumirea sistematică a variabilelor legate dintr-o funcție: $\lambda x.E \rightarrow_\alpha \lambda y.E_{[y/x]}$. Se impun două condiții.

Ex — Exemplu

- $\lambda x.y \rightarrow_\alpha \lambda y.y_{[y/x]} \rightarrow \lambda y.y \rightarrow$ Greșit!
- $\lambda x.\lambda y.x \rightarrow_\alpha \lambda y.\lambda y.x_{[y/x]} \rightarrow \lambda y.\lambda y.y \rightarrow$ Greșit!

⋮ Condiții

- y nu este o variabilă liberă, existentă deja în E
- orice apariție liberă în E rămâne liberă în $E_{[y/x]}$

α-conversie
Exemple

Exemplu

- $\lambda x.(x\ y) \rightarrow_\alpha \lambda z.(z\ y) \rightarrow$ Corect!
- $\lambda x.\lambda x.(x\ y) \rightarrow_\alpha \lambda y.\lambda x.(x\ y)$
- $\lambda x.\lambda y.(y\ x) \rightarrow_\alpha \lambda y.\lambda y.(y\ y)$
- $\lambda x.\lambda y.(y\ y) \rightarrow_\alpha \lambda y.\lambda y.(y\ y)$

α-conversie

Exemple

Ex Exemplu

- $\lambda x.(x\ y) \rightarrow_\alpha \lambda z.(z\ y)$ → Corect!
- $\lambda x.\lambda x.(x\ y) \rightarrow_\alpha \lambda y.\lambda x.(x\ y)$ → Greșit! y este liberă în $\lambda x.(x\ y)$
- $\lambda x.\lambda y.(y\ x) \rightarrow_\alpha \lambda y.\lambda y.(y\ y)$
- $\lambda x.\lambda y.(y\ y) \rightarrow_\alpha \lambda y.\lambda y.(y\ y)$

α-conversie

Exemple

Ex Exemplu

- $\lambda x.(x\ y) \rightarrow_\alpha \lambda z.(z\ y) \rightarrow$ Corect!
- $\lambda x.\lambda x.(x\ y) \rightarrow_\alpha \lambda y.\lambda x.(x\ y) \rightarrow$ Greșit! y este liberă în $\lambda x.(x\ y)$
- $\lambda x.\lambda y.(y\ x) \rightarrow_\alpha \lambda y.\lambda y.(y\ y) \rightarrow$ Greșit! Apariția liberă a lui x din $\lambda y.(y\ x)$ devine legată, după substituire, în $\lambda y.(y\ y)$
- $\lambda x.\lambda y.(y\ y) \rightarrow_\alpha \lambda y.\lambda y.(y\ y)$

α-conversie
Exemple

λ

Exemplu

- $\lambda x.(x\ y) \rightarrow_\alpha \lambda z.(z\ y)$ → Corect!
- $\lambda x.\lambda x.(x\ y) \rightarrow_\alpha \lambda y.\lambda x.(x\ y)$ → Greșit! y este liberă în $\lambda x.(x\ y)$
- $\lambda x.\lambda y.(y\ x) \rightarrow_\alpha \lambda y.\lambda y.(y\ y)$ → Greșit! Apariția liberă a lui x din $\lambda y.(y\ x)$ devine legată, după substituire, în $\lambda y.(y\ y)$
- $\lambda x.\lambda y.(y\ y) \rightarrow_\alpha \lambda y.\lambda y.(y\ y)$ → Corect!

Reducere
Definiții

λ

+ | Pas de reducere: O secvență formată dintr-o α-conversie și o β-reducere, astfel încât a doua se produce fără coliziuni:
$E_1 \rightarrow E_2 \equiv E_1 \rightarrow_\alpha E_3 \rightarrow_\beta E_2$.

+ | Secvență de reducere: Succesiune de zero sau mai mulți pași de reducere:
$E_1 \rightarrow^* E_2$.
Reprezintă un element din închiderea reflexiv-tranzitivă a relației $\rightarrow$.

λ

: Reducere

- $E_1 \rightarrow E_2 \implies E_1 \rightarrow^* E_2$ – un pas este o secvență
- $E \rightarrow^* E$ – zero pași formează o secvență
- $E_1 \rightarrow^* E_2 \wedge E_2 \rightarrow^* E_3 \Rightarrow E_1 \rightarrow^* E_3$ – tranzitivitate

Ex Exemplu

$$((\lambda x.\lambda y.((y\ x)\ y)\ \lambda x.x) \rightarrow (\lambda z.(z\ y)\ \lambda x.x) \rightarrow (\lambda x.x\ y) \rightarrow y$$

$$\Rightarrow$$

$$((\lambda x.\lambda y.((y\ x)\ y)\ \lambda x.x) \rightarrow^* y$$

Evaluare

Întrebări λ

Pentru construcția unei mașini de calcul

· Dacă am vrea să construim o mașină de calcul care să aibă ca program o λ-expresie și să aibă ca operație de bază pasul de reducere, ne punem câteva întrebări:

1. Când se termină calculul? Se termină întotdeauna?
2. Dacă mai multe secvențe de reducere se termină, obținem întotdeauna același rezultat?
3. Comportamentul depinde de secvența de reducere?
4. Dacă rezultatul este unic, cum îl obținem?

λ

Ex

Exemplu

$$\Omega = (\lambda x.(x\ x)\ \lambda x.(x\ x)) \rightarrow (\lambda x.(x\ x)\ \lambda x.(x\ x)) \rightarrow^* \ldots$$

Terminarea reducerii (reductibilitate)

Exemplu și definiție

Ex Exemplu

$\Omega = (\lambda x.(x\ x)\ \lambda x.(x\ x)) \rightarrow (\lambda x.(x\ x)\ \lambda x.(x\ x)) \rightarrow^* \ldots$

Ω nu admite nicio secvență de reducere care se termină.

+ **Expresie reductibilă** este o expresie care admite (cel puțin o) secvență de reducere care se termină.

· expresia Ω nu este reductibilă.

λ

Dar!

Exemplu

$E = (\lambda x.y\ \ \Omega)$

$\to y$ *sau*

$\to E \to y$ *sau*

$\to E \to E \to y$ *sau...*

$\ldots$

$\xrightarrow{n}^{*} y,\ n \geq 0$

$\xrightarrow{\infty}^{*} \ldots$

Secvențe de reducere

și terminare

λ

Dar!

Exemplu

$E = (\lambda x.y\ \Omega)$
$\rightarrow y$ *sau*
$\rightarrow E \rightarrow y$ *sau*
$\rightarrow E \rightarrow E \rightarrow y$ *sau...*
$\ldots$
$\xrightarrow{n}^{*} y,\ n \geq 0$
$\xrightarrow{\infty}^{*} \ldots$

- E are o secvență de reducere care nu se termină;
- dar E are forma normală $y \Rightarrow E$ este reductibilă;
- lungimea secvențelor de reducere ale E este nemărginită.

Forme normale

Cum știm că s-a terminat calculul?

· Calculul se termină atunci când expresia nu mai poate fi redusă $\rightarrow$ expresia nu mai conține β-redecși.

+ | Forma normală a unei expresii este o formă (la care se ajunge prin reducere, care nu mai conține β-redecși i.e. care nu mai poate fi redusă.

Forme normale

Este necesar să mergem până la Forma Normală?

+ | Forma normală funcțională – FNF este o formă $\lambda x.F$, în care F *poate* conține β-redecși.

Ex | Exemplu

$(\lambda x.\lambda y.(x\ \ y)\ \lambda x.x) \rightarrow_{FNF} \lambda y.(\lambda x.x\ \ y) \rightarrow_{FN} \lambda y.y$

- FN a unei expresii închise este în mod necesar FNF.
- într-o FNF nu există o necesitate imediată de a evalua eventualii β-redecși interiori (funcția nu a fost încă aplicată).

Unicitatea formei normale

Rezultate

λ

T | Teorema Church-Rosser / diamantului Dacă $E \rightarrow^* E_1$ și $E \rightarrow^* E_2$, atunci există E_3 astfel încât $E_1 \rightarrow^* E_3$ și $E_2 \rightarrow^* E_3$.

$$
\begin{tikzcd}
 & E_1 \arrow[dr, "*"] & \\
E \arrow[ur, "*"] \arrow[dr, "*"'] & & E_3 \\
 & E_2 \arrow[ur, "*"'] &
\end{tikzcd}
$$

C | Corolar Dacă o expresie este reductibilă, forma ei normală este unică. Ea corespunde valorii expresiei.

Unicitatea formei normale

Exemplu

Ex Exemplu

$$(\lambda x.\lambda y.(x\ \ y)\ \ (\lambda x.x\ \ y))$$

Unicitatea formei normale
Exemplu

Ex Exemplu

$(\lambda x.\lambda y.(x\ y)\ (\lambda x.x\ y))$

- $\rightarrow \lambda z.((\lambda x.x\ y)\ z) \rightarrow \lambda z.(y\ z) \rightarrow_\alpha \lambda a.(y\ a)$
- $\rightarrow (\lambda x.\lambda y.(x\ y)\ y) \rightarrow \lambda w.(y\ w) \rightarrow_\alpha \lambda a.(y\ a)$

Exemplu

Ex

$(\lambda x.\lambda y.(x\ y)\ (\lambda x.x\ y))$

- $\rightarrow \lambda z.((\lambda x.x\ y)\ z) \rightarrow \lambda z.(y\ z) \rightarrow_{\alpha} \lambda a.(y\ a)$
- $\rightarrow (\lambda x.\lambda y.(x\ y)\ y) \rightarrow \lambda w.(y\ w) \rightarrow_{\alpha} \lambda a.(y\ a)$

- Forma normală corespunde unei clase de expresii, echivalente sub redenumiri sistematice.
- Valoarea este un anumit membru al acestei clase de echivalență.

$\Rightarrow$ Valorile sunt echivalente în raport cu redenumirea.

Modalități de reducere

Cum putem *organiza* reducerea?

λ

+ Reducere stânga-dreapta: Reducerea celui mai superficial și mai din stânga β-redex.

Ex Exemplu

$((\underline{\lambda x.x\ \lambda x.y)}\ (\lambda x.(x\ x)\ \lambda x.(x\ x))) \rightarrow \underline{(\lambda x.y\ \Omega)} \rightarrow y$

+ Reducere dreapta-stânga: Reducerea celui mai adânc și mai din dreapta β-redex.

Ex Exemplu

$(\lambda x.(\lambda x.x\ \lambda x.y)\ \underline{(\lambda x.(x\ x)\ \lambda x.(x\ x))}) \rightarrow$
$(\lambda x.(\lambda x.x\ \lambda x.y)\ \underline{\Omega}) \rightarrow \ldots$

Ce modalitate alegem?

λ

T | Teorema normalizării Dacă o expresie este reductibilă, evaluarea stânga-dreapta a acesteia se termină.

- Teorema normalizării (normalizare = aducere la forma normală) nu garantează terminarea evaluării oricărei expresii, ci doar a celor reductibile!
- Dacă expresia este ireductibilă, nicio reducere nu se va termina.

Răspunsuri la întrebări

λ

1. Când se termină calculul? Se termină întotdeauna?
$\rightarrow$ se termină cu forma normală [funcțională]. NU se termină decât dacă expresia este reductibilă.
2. Comportamentul depinde de secvența de reducere?
$\rightarrow$ DA.
3. Dacă mai multe secvențe de reducere se termină, obținem întotdeauna același rezultat?
$\rightarrow$ DA.
4. Dacă rezultatul este unic, cum îl obținem?
$\rightarrow$ Reducere stânga-dreapta.
5. Care este valoarea expresiei?
$\rightarrow$ Forma normală [funcțională] (FN[F]).

λ

- $\boxed{+}$ **Evaluare aplicativă** (*eager*) – corespunde unei reduceri *mai degrabă* dreapta-stânga. Parametrii funcțiilor sunt evaluați înaintea aplicării funcției.
- $\boxed{+}$ **Evaluare normală** (*lazy*) – corespunde reducerii stânga-dreapta. Parametrii funcțiilor sunt evaluați la cerere.
- $\boxed{+}$ **Funcție strictă** – funcție cu evaluare aplicativă.
- $\boxed{+}$ **Funcție nestrictă** – funcție cu evaluare normală.

Ordine de evaluare

În practică

λ

- Evaluarea aplicativă prezentă în majoritatea limbajelor: C, Java, Scheme, PHP etc.

Exemplu

`(+ (+ 2 3) (* 2 3))` $\rightarrow$ `(+ 5 6)` $\rightarrow$ `11`

- Nevoie de funcții nestricte, chiar în limbajele aplicative: `if`, `and`, `or` etc.

Exemplu

`(if (< 2 3) (+ 2 3) (* 2 3))` $\rightarrow$ `(< 2 3)` $\rightarrow$ `#t` $\rightarrow$ `(+ 2 3)` $\rightarrow$ `5`

Limbajul lambda-0 și incursiune în TDA

- Am putea crea o mașină de calcul folosind calculul λ – mașină de calcul ipotetică;
- Mașina folosește limbajul $\lambda_0 \equiv$ calcul lambda;
- Programul $\rightarrow$ λ-expresie;
  - + Legări top-level de expresii la nume.
- Datele $\rightarrow$ λ-expresii;
- Funcționarea mașinii $\rightarrow$ reducere – substituție textuală
  - evaluare normală;
  - terminarea evaluării cu forma normală funcțională;
  - se folosesc numai expresii închise.

Tipuri de date

Cum reprezentăm datele? Cum interpretăm valorile?

- Putem reprezenta toate datele prin funcții cărora, convențional, le dăm o semnificație abstractă.

  Ex | Exemplu

  $T \equiv_{\text{def}} \lambda x.\lambda y.x \qquad F \equiv_{\text{def}} \lambda x.\lambda y.y$

- Pentru aceste tipuri de date abstracte (TDA) creăm operatori care transformă datele în mod coerent cu interpretarea pe care o dăm valorilor.

  Ex | Exemplu

  $not \equiv_{\text{def}} \lambda x.((x\ F)\ T)$

  $(not\ T) \rightarrow (\lambda x.((x\ F)\ T)\ T) \rightarrow ((T\ F)\ T) \rightarrow F$

+ Tip de date abstract – TDA – Model matematic al unei mulțimi de valori și al operațiilor valide pe acestea.

: Componente

- constructori de bază: cum se generează valorile;
- operatori: ce se poate face cu acestea;
- axiome: cum lucrează operatorii / ce restricții există.

TDA *Bool*
Specificare

- Constructori: $\left| \begin{array}{l} T: \quad \rightarrow \mathit{Bool} \\ F: \quad \rightarrow \mathit{Bool} \end{array} \right.$

TDA *Bool*
Specificare

λ

· Constructori: $\left|\begin{array}{ll} T: & \to Bool \\ F: & \to Bool \end{array}\right.$

· Operatori: $\left|\begin{array}{ll} not: & Bool \to Bool \\ and: & Bool^2 \to Bool \\ or: & Bool^2 \to Bool \\ if: & Bool \times A \times A \to A \end{array}\right.$

TDA *Bool*
Specificare

· Constructori:

$$\begin{array}{ll} T: & \rightarrow Bool \\ F: & \rightarrow Bool \end{array}$$

· Operatori:

$$\begin{array}{ll} not: & Bool \rightarrow Bool \\ and: & Bool^2 \rightarrow Bool \\ or: & Bool^2 \rightarrow Bool \\ if: & Bool \times A \times A \rightarrow A \end{array}$$

· Axiome:

$$\begin{array}{lll} not: & not(T) = F & not(F) = T \\ and: & and(T, a) = a & and(F, a) = F \\ or: & or(T, a) = T & or(F, a) = a \\ if: & if(T, a, b) = a & if(F, a, b) = b \end{array}$$

Intuiție bazat pe comportamentul necesar pentru if: selecția între cele două valori

- $T \equiv_{\text{def}} \lambda x.\lambda y.x$
- $F \equiv_{\text{def}} \lambda x.\lambda y.y$

- $if \equiv_{\text{def}} \lambda c.\lambda x.\lambda y.((c\ \ x)\ \ y)$

TDA *Bool*

Implementarea operatorilor

- $if \equiv_{\text{def}} \lambda c.\lambda x.\lambda y.((c\ x)\ y)$
- $and \equiv_{\text{def}} \lambda x.\lambda y.((x\ y)\ F)$

TDA *Bool*

Implementarea operatorilor

λ

- $if \equiv_{\text{def}} \lambda c.\lambda x.\lambda y.((c\ x)\ y)$
- $and \equiv_{\text{def}} \lambda x.\lambda y.((x\ y)\ F)$
  - $((and\ T)\ a) \rightarrow ((\lambda x.\lambda y.((x\ y)\ F)\ T)\ a) \rightarrow ((T\ a)\ F) \rightarrow a$

- $if \equiv_{\text{def}} \lambda c.\lambda x.\lambda y.((c\ \ x)\ \ y)$
- $and \equiv_{\text{def}} \lambda x.\lambda y.((x\ \ y)\ \ F)$
  - $((and\ \ T)\ \ a) \rightarrow ((\lambda x.\lambda y.((x\ \ y)\ \ F)\ \ T)\ \ a) \rightarrow ((T\ \ a)\ \ F) \rightarrow a$
  - $((and\ \ F)\ \ a) \rightarrow ((\lambda x.\lambda y.((x\ \ y)\ \ F)\ \ F)\ \ a) \rightarrow ((F\ \ a)\ \ F) \rightarrow F$

- $if \equiv_{\text{def}} \lambda c.\lambda x.\lambda y.((c\ x)\ y)$

- $and \equiv_{\text{def}} \lambda x.\lambda y.((x\ y)\ F)$
  - $((and\ T)\ a) \rightarrow ((\lambda x.\lambda y.((x\ y)\ F)\ T)\ a) \rightarrow ((T\ a)\ F) \rightarrow a$
  - $((and\ F)\ a) \rightarrow ((\lambda x.\lambda y.((x\ y)\ F)\ F)\ a) \rightarrow ((F\ a)\ F) \rightarrow F$

- $or \equiv_{\text{def}} \lambda x.\lambda y.((x\ T)\ y)$

λ

- $if \equiv_{\text{def}} \lambda c.\lambda x.\lambda y.((c\ x)\ y)$

- $and \equiv_{\text{def}} \lambda x.\lambda y.((x\ y)\ F)$
  - $((and\ T)\ a) \rightarrow ((\lambda x.\lambda y.((x\ y)\ F)\ T)\ a) \rightarrow ((T\ a)\ F) \rightarrow a$
  - $((and\ F)\ a) \rightarrow ((\lambda x.\lambda y.((x\ y)\ F)\ F)\ a) \rightarrow ((F\ a)\ F) \rightarrow F$

- $or \equiv_{\text{def}} \lambda x.\lambda y.((x\ T)\ y)$
  - $((or\ T)\ a) \rightarrow ((\lambda x.\lambda y.((x\ T)\ y)\ T)\ a) \rightarrow ((T\ T)\ a) \rightarrow T$

TDA *Bool*

Implementarea operatorilor

λ

- $if \equiv_{\text{def}} \lambda c.\lambda x.\lambda y.((c\ x)\ y)$
- $and \equiv_{\text{def}} \lambda x.\lambda y.((x\ y)\ F)$
  - $((and\ T)\ a) \rightarrow ((\lambda x.\lambda y.((x\ y)\ F)\ T)\ a) \rightarrow ((T\ a)\ F) \rightarrow a$
  - $((and\ F)\ a) \rightarrow ((\lambda x.\lambda y.((x\ y)\ F)\ F)\ a) \rightarrow ((F\ a)\ F) \rightarrow F$
- $or \equiv_{\text{def}} \lambda x.\lambda y.((x\ T)\ y)$
  - $((or\ T)\ a) \rightarrow ((\lambda x.\lambda y.((x\ T)\ y)\ T)\ a) \rightarrow ((T\ T)\ a) \rightarrow T$
  - $((or\ F)\ a) \rightarrow ((\lambda x.\lambda y.((x\ T)\ y)\ F)\ a) \rightarrow ((F\ T)\ a) \rightarrow a$

TDA *Bool*
Implementarea operatorilor

λ

- $if \equiv_{\text{def}} \lambda c.\lambda x.\lambda y.((c\ x)\ y)$

- $and \equiv_{\text{def}} \lambda x.\lambda y.((x\ y)\ F)$
  - $((and\ T)\ a) \rightarrow ((\lambda x.\lambda y.((x\ y)\ F)\ T)\ a) \rightarrow ((T\ a)\ F) \rightarrow a$
  - $((and\ F)\ a) \rightarrow ((\lambda x.\lambda y.((x\ y)\ F)\ F)\ a) \rightarrow ((F\ a)\ F) \rightarrow F$

- $or \equiv_{\text{def}} \lambda x.\lambda y.((x\ T)\ y)$
  - $((or\ T)\ a) \rightarrow ((\lambda x.\lambda y.((x\ T)\ y)\ T)\ a) \rightarrow ((T\ T)\ a) \rightarrow T$
  - $((or\ F)\ a) \rightarrow ((\lambda x.\lambda y.((x\ T)\ y)\ F)\ a) \rightarrow ((F\ T)\ a) \rightarrow a$

- $not \equiv_{\text{def}} \lambda x.((x\ F)\ T)$

λ

- $if \equiv_{\text{def}} \lambda c.\lambda x.\lambda y.((c\ x)\ y)$
- $and \equiv_{\text{def}} \lambda x.\lambda y.((x\ y)\ F)$
  - $((and\ T)\ a) \rightarrow ((\lambda x.\lambda y.((x\ y)\ F)\ T)\ a) \rightarrow ((T\ a)\ F) \rightarrow a$
  - $((and\ F)\ a) \rightarrow ((\lambda x.\lambda y.((x\ y)\ F)\ F)\ a) \rightarrow ((F\ a)\ F) \rightarrow F$
- $or \equiv_{\text{def}} \lambda x.\lambda y.((x\ T)\ y)$
  - $((or\ T)\ a) \rightarrow ((\lambda x.\lambda y.((x\ T)\ y)\ T)\ a) \rightarrow ((T\ T)\ a) \rightarrow T$
  - $((or\ F)\ a) \rightarrow ((\lambda x.\lambda y.((x\ T)\ y)\ F)\ a) \rightarrow ((F\ T)\ a) \rightarrow a$
- $not \equiv_{\text{def}} \lambda x.((x\ F)\ T)$
  - $(not\ T) \rightarrow (\lambda x.((x\ F)\ T)\ T) \rightarrow ((T\ F)\ T) \rightarrow F$

TDA *Bool*

Implementarea operatorilor

λ

- $if \equiv_{\text{def}} \lambda c.\lambda x.\lambda y.((c\ x)\ y)$
- $and \equiv_{\text{def}} \lambda x.\lambda y.((x\ y)\ F)$
  - $((and\ T)\ a) \rightarrow ((\lambda x.\lambda y.((x\ y)\ F)\ T)\ a) \rightarrow ((T\ a)\ F) \rightarrow a$
  - $((and\ F)\ a) \rightarrow ((\lambda x.\lambda y.((x\ y)\ F)\ F)\ a) \rightarrow ((F\ a)\ F) \rightarrow F$
- $or \equiv_{\text{def}} \lambda x.\lambda y.((x\ T)\ y)$
  - $((or\ T)\ a) \rightarrow ((\lambda x.\lambda y.((x\ T)\ y)\ T)\ a) \rightarrow ((T\ T)\ a) \rightarrow T$
  - $((or\ F)\ a) \rightarrow ((\lambda x.\lambda y.((x\ T)\ y)\ F)\ a) \rightarrow ((F\ T)\ a) \rightarrow a$
- $not \equiv_{\text{def}} \lambda x.((x\ F)\ T)$
  - $(not\ T) \rightarrow (\lambda x.((x\ F)\ T)\ T) \rightarrow ((T\ F)\ T) \rightarrow F$
  - $(not\ F) \rightarrow (\lambda x.((x\ F)\ T)\ F) \rightarrow ((F\ F)\ T) \rightarrow T$

λ

- Intuiție: pereche → funcție ce așteaptă selectorul, pentru a-l aplica asupra membrilor

TDA *Pair*
Implementare

λ

- Intuiție: pereche $\rightarrow$ funcție ce așteaptă selectorul, pentru a-l aplica asupra membrilor
- $fst \equiv_{\text{def}} \lambda p.(p\ \ T)$

TDA *Pair*
Implementare

- Intuiție: pereche → funcție ce așteaptă selectorul, pentru a-l aplica asupra membrilor

- $fst \equiv_{\text{def}} \lambda p.(p\ T)$
  - $(fst\ ((pair\ a)\ b)) \rightarrow (\lambda p.(p\ T)\ \lambda z.((z\ a)\ b)) \rightarrow (\lambda z.((z\ a)\ b)\ T) \rightarrow ((T\ a)\ b) \rightarrow a$

TDA *Pair*
Implementare

λ

- Intuiție: pereche → funcție ce așteaptă selectorul, pentru a-l aplica asupra membrilor
- $fst \equiv_{\text{def}} \lambda p.(p\ \ T)$
  - $(fst\ \ ((pair\ \ a)\ \ b)) \rightarrow (\lambda p.(p\ \ T)\ \ \lambda z.((z\ \ a)\ \ b)) \rightarrow (\lambda z.((z\ \ a)\ \ b)\ \ T) \rightarrow ((T\ \ a)\ \ b) \rightarrow a$
- $snd \equiv_{\text{def}} \lambda p(F\ \ )$

TDA *Pair*

Implementare

λ

- Intuiție: pereche → funcție ce așteaptă selectorul, pentru a-l aplica asupra membrilor
- $fst \equiv_{\text{def}} \lambda p.(p\ \ T)$
  - $(fst\ \ ((pair\ \ a)\ \ b)) \rightarrow (\lambda p.(p\ \ T)\ \ \lambda z.((z\ \ a)\ \ b)) \rightarrow (\lambda z.((z\ \ a)\ \ b)\ \ T) \rightarrow ((T\ \ a)\ \ b) \rightarrow a$
- $snd \equiv_{\text{def}} \lambda p(F\ \ )$
  - $(snd\ \ ((pair\ \ a)\ \ b)) \rightarrow (\lambda p.(p\ \ F)\ \ \lambda z.((z\ \ a)\ \ b)) \rightarrow (\lambda z.((z\ \ a)\ \ b)\ \ F) \rightarrow ((F\ \ a)\ \ b) \rightarrow b$

TDA *Pair*

Implementare

λ

- Intuiție: pereche → funcție ce așteaptă selectorul, pentru a-l aplica asupra membrilor
- $fst \equiv_{\text{def}} \lambda p.(p\ \ T)$
  - $(fst\ \ ((pair\ \ a)\ \ b)) \to (\lambda p.(p\ \ T)\ \ \lambda z.((z\ \ a)\ \ b)) \to (\lambda z.((z\ \ a)\ \ b)\ \ T) \to ((T\ \ a)\ \ b) \to a$
- $snd \equiv_{\text{def}} \lambda p(F\ \ )$
  - $(snd\ \ ((pair\ \ a)\ \ b)) \to (\lambda p.(p\ \ F)\ \ \lambda z.((z\ \ a)\ \ b)) \to (\lambda z.((z\ \ a)\ \ b)\ \ F) \to ((F\ \ a)\ \ b) \to b$
- $pair \equiv_{\text{def}} \lambda x.\lambda y.\lambda z.((z\ \ x)\ \ y)$

λ

- Intuiție: pereche → funcție ce așteaptă selectorul, pentru a-l aplica asupra membrilor
- $fst \equiv_{\text{def}} \lambda p.(p\ \ T)$
  - $(fst\ \ ((pair\ \ a)\ \ b)) \rightarrow (\lambda p.(p\ \ T)\ \ \lambda z.((z\ \ a)\ \ b)) \rightarrow (\lambda z.((z\ \ a)\ \ b)\ \ T) \rightarrow ((T\ \ a)\ \ b) \rightarrow a$
- $snd \equiv_{\text{def}} \lambda p(F\ \ )$
  - $(snd\ \ ((pair\ \ a)\ \ b)) \rightarrow (\lambda p.(p\ \ F)\ \ \lambda z.((z\ \ a)\ \ b)) \rightarrow (\lambda z.((z\ \ a)\ \ b)\ \ F) \rightarrow ((F\ \ a)\ \ b) \rightarrow b$
- $pair \equiv_{\text{def}} \lambda x.\lambda y.\lambda z.((z\ \ x)\ \ y)$
  - $((pair\ \ a)\ \ b) \rightarrow (\lambda x.\lambda y.\lambda z.((z\ \ x)\ \ y)ab\ \ \rightarrow)\lambda z.((z\ \ a)\ \ b)$

TDA *List* și *Natural*

Implementare

Intuiție: listă → pereche (*head*, *tail*)

- $nil \equiv_{\text{def}} \lambda x.T$
- $cons \equiv_{\text{def}} pair$
  - $((cons\ \ e)\ \ L) \rightarrow ((\lambda x.\lambda y.\lambda z.((z\ \ x)\ \ y)\ \ e)\ \ L) \rightarrow \lambda z.((z\ \ e)\ \ L)$
- $car \equiv_{\text{def}} fst \qquad cdr \equiv_{\text{def}} snd$

Intuiție: număr → listă cu lungimea egală cu valoarea numărului

- $zero \equiv_{\text{def}} nil$
- $succ \equiv_{\text{def}} \lambda n.((cons\ \ nil)\ \ n)$
- $pred \equiv_{\text{def}} cdr$

· vezi și [http://en.wikipedia.org/wiki/Lambda_calculus#Encoding_datatypes]

- Modalitate de exprimare a intenției programatorului;
- Documentare: ce operatori acționează asupra căror obiecte;
- Reprezentarea particulară a valorilor de tipuri diferite: `1`, “`Hello`”, `#t` etc.;
- Optimizarea operațiilor specifice;
- Prevenirea erorilor;
- Facilitarea verificării formale;

Absența tipurilor
Consecințe asupra reprezentării obiectelor

- Un număr, o listă sau un arbore, posibil desemnate de aceeași valoare!
- Valori și operatori reprezentați de funcții, semnificația fiind dependentă de context.
- Valoare aplicabilă asupra unei alte valori → operator!

Absența tipurilor

Consecințe asupra corectitudinii calculului

λ

- Incapacitatea Mașinii λde a
  - interpreta semnificația expresiilor;
  - asigura corectitudinea acestora (dpdv al tipurilor).
- Delegarea celor două aspecte programatorului;
- Orice operatori aplicabili asupra oricăror valori;
- Construcții eronate acceptate fără avertisment, dar calcule terminate cu
  - valori fără semnificație *sau*
  - expresii care nu sunt valori (nu au asociată o semnificație), dar sunt ireductibile

  $\rightarrow$ instabilitate.

Absența tipurilor

Consecințe pozitive

- Flexibilitate sporită în reprezentare;
- Potrivită în situațiile în care reprezentarea uniformă obiectelor, ca liste de simboluri, este convenabilă.

…vin cu prețul unei dificultăți sporite în depanare, verificare și mentenanță

λ

· Cum realizăm recursivitatea în λ_0, dacă nu avem nume de funcții?

- Textuală: funcție care se autoapelează, folosindu-și numele;
- Semantică: ce obiect matematic este desemnat de o funcție recursivă, cu posibilitatea construirii de funcții recursive anonime.

Implementare *length*

Problemă

- Lungimea unei liste:
  $length \equiv_{\text{def}} \lambda L.(if\ (null\ L)\ zero\ (succ\ (\underline{length}\ (cdr\ L))))$
- Cu ce înlocuim zona subliniată, pentru a evita recursivitatea textuală? (expresia pentru *length* nu este închisă!)
- Putem primi ca parametru o funcție echivalentă computațional cu *length*?
  $Length \equiv_{\text{def}} \lambda f\ L.(if\ (null\ L)\ zero\ (succ\ (f\ (cdr\ L))))$
- $(Length\ length) = length \rightarrow length$ este un punct fix al lui *Length*!
- Cum obținem punctul fix?

Exemplu

$Fix = \lambda f.(\lambda x.(f\ (x\ x))\ \lambda x.(f\ (x\ x)))$

- $(Fix\ F) \rightarrow (\lambda x.(F\ (x\ x))\ \lambda x.(F\ (x\ x))) \rightarrow$ $(F\ \underline{(\lambda x.(F\ (x\ x))\ \lambda x.(F\ (x\ x)))}) \rightarrow (F\ (Fix\ F))$
- $(Fix\ F)$ este un punct fix al lui F.
- Fix se numește combinator de punct fix.

- $length \equiv_{\text{def}} (Fix\ Length) \sim (Length\ (Fix\ Length)) \sim$ $\lambda L.(if\ (null\ L)\ zero\ (succ\ ((Fix\ Length)\ (cdr\ L))))$

- Funcție recursivă, fără a fi textual recursivă!

Racket vs. lambda-0

Racket vs. λ_0
Construcția expresiilor / sintaxă

	λ	Racket
Variabilă/nume	x	`x`
Funcție	$\lambda x.corp$	`(lambda (x) corp)`
uncurry	$\lambda x\ y.corp$	`(lambda (x y) corp)`
Aplicare	$(F\ \ A)$	`(f a)`
uncurry	$(F\ \ A1\ \ A2)$	`(f a1 a2)`
Legare top-level	-	`(define nume expr)`
Program	λ-expresie închisă	colecție de legări top-level (`define`)
Valori	λ-expresii / TDA	valori de diverse tipuri (numere, liste, etc.)

Racket vs. λ_0
Mai precis

- similar cu λ_0, folosește S-expresii (bază Lisp);
- tipat – dinamic/latent
  - variabilele nu au tip;
  - valorile au tip (`3`, `#f`);
  - verificarea se face la execuție, în momentul aplicării unei funcții;
- evaluare aplicativă;
- permite recursivitate textuală;
- avem legări top-level.

Sfârșitul cursului 3
Elemente esențiale

λ

- Baza formală a calculului λ:
- expresie λ, β-redex, variabile și apariții legate vs. libere, expresie închisă, α-conversie, β-reducere
- FN și FNF, reducere, reductibilitate, evaluare aplicativă și normală
- TDA și recursivitate pentru calcul lambda