λ-calcul

Variabile libere si legate

$\lambda x.t$ - toate aparitiile lui x in t sunt aparitii legate

$\lambda x.xy$ - x este legata, y este libera
$\lambda x.zt$ - toate variabilele (z si t) sunt libere
$\lambda xy.xyx$ - toate variabilele sunt legate. Acesta este un λ-termen inchis.

Exercitiu

$FV(\lambda x.xy) = FV(xy) - \{x\} = (FV(x) \cup FV(y)) - \{x\} = (\{x\} \cup \{y\}) - \{x\} = \{y\}$

$FV(x\lambda x.xy) = FV(x) \cup FV(\lambda x.xy) = \{x\} \cup \{y\} = \{x, y\}$

$FV(x(\lambda xy.xyz)(\lambda v.yv)) = FV(x) \cup FV(\lambda xy.xyz) \cup FV(\lambda v.yv) = \{x, y, z\}$

1. $FV(x) = \{x\}$
2. $FV(\lambda xy.xyz) = FV(\lambda x.\lambda y.xyz) = FV(\lambda y.xyz) - \{x\} = (FV(xyz) - \{y\}) - \{x\} = (FV(x) \cup FV(y) \cup FV(z) - \{y\}) - \{x\} = \{x, y, z\} - \{y\} - \{x\} = \{z\}$
3. $FV(\lambda v.yv) = \{y\}$

$FV(\lambda t.((\lambda xyz.yzx)t)) = \emptyset$, deci avem un λ-termen inchis.

Substitutii

1. $[y/x]\lambda z.x = \lambda z.[y/x]x = \lambda z.y$
2. $[y/x]\lambda y.x$ - nu pot efectua substitutia

- am $x \neq y$;
- dar nu satisfac conditia $y \notin FV(y) = \{y\}$

3. $[\lambda z.z/x](\lambda x.yx)$ - nu pot efectua substitutia

- nu satisfac $x \neq x$

4. $[\lambda z.z/x](\lambda y.yx) = \lambda y.[\lambda z.z/x]yx = \lambda y.([\lambda z.z/x]y)([\lambda z.z/x]x) = \lambda y.y(\lambda z.z)$

α-conversie

$\lambda x.xyz =_\alpha \lambda r.ryz$

$\lambda x.x =_\alpha \lambda z.z$

β-reductia

$(\lambda x.t)u \to_\beta [u/x]t$

```
(\x -> x + 1) 2 -beta-> 2 + 1 = 3
```

$t \to_\beta^* t_1$
$t \to_\beta^* t_2$
$t \to_\beta^* t_3$

$t_1 =_\alpha t_2 =_\alpha t_3$

Daca rescriu t intr-un t_1, si din t_1 nu mai pot aplica nicio β-reductie, atunci t_1 se numeste β-forma normala.

Pentru un λ-termen, β-forma normala este unica modulo α-conversie.

Sistemul de rescriere prin β-reductii este confluent.

$t \to_\beta^* \lambda z.z$
$t \to_\beta^* \lambda x.x$

Exercitiu

$(\lambda x.(\lambda y.yx)z)v \rightarrow_\beta [v/x]((\lambda y.yx)z) =_\alpha (\lambda y.yv)z =_\alpha [z/y](yv) =_\alpha zv$ - β forma normala

$(\lambda x.(\lambda y.yx)z)v \rightarrow_\beta (\lambda x.[z/y](yx))v =_\alpha (\lambda x.zx)v \rightarrow_\beta [v/x](zx) =_\alpha zv$ - β forma normala

Cerinte de laborator

Implementarea λ-calculului in Haskell

Fie urmatoarele tipuri de date:

Tipul pentru variabile:

```
type Variable = Char
```

Tipul de date algebric pentru expresiile din λ-calcul:

```
data Term = V Variable
    | App Term Term
    | Lam Variable Term
    deriving (Eq, Show)
```

$l_1 = z\lambda x.xy$

```
lambda1 = App (V 'z') (Lam 'x' (App (V 'x') (V 'y')))
```

Cerinta 1 Definiti urmatorii λ-termeni in Haskell:

1. $\lambda x.xyz$
2. $(\lambda x.xy)(y\lambda s.sz)$
3. $\lambda sz.ssz$

Cerinta 2 Definiti o functie care sa calculeze multimea variabilelor libere dintr-un λ-termen.

```
freeVars :: Term -> [Variable]
freeVars (V x) = [x]
```

```
freeVars (App t1 t2) = nub $ (freeVars t1) ++ (freeVars t2)
freeVars (Lam x t) = undefined
```

Cerinta 3 Definiti o functie care sa calculeze substitutia unei variabile cu un termen intr-un alt termen.

```
--         u        x          t      result
-- result = [u / x] t
subst :: Term -> Variable -> Term -> Term
subst u x (V y)
    | x == y = u
    | otherwise = V y
subst u x (App t1 t2) = App (subst u x t1) (subst u x t2)
subst u x (Lam y t) = undefined
```

Solutiile se trimit pe adresa *bogdan.macovei.fmi@gmail.com* cu subiectul **Grupa, Nume Prenume, Lambda calcul** pana pe 19 mai 2022, seara (23:59 sau putin dupa). Si acest laborator este notat pentru punctajul bonus.

Laboratorul/Seminarul 7 FLP

Puncte fixe. Teorema Knaster-Tarski.

Teorie pentru Exercitiul 1

O multime partial ordonata (poset) **mpo** este o pereche $(M, \leq)$, unde $\leq \subseteq M^2$ care respecta urmatoarele proprietati:

- reflexiva;
- antisimetrica;
- tranzitiva.
  O multime partial ordonata este completa **cpo** daca exista un prim element $\perp \in M$ astfel incat oricum alegem $x \in M$, $\perp \leq x$ si, in plus, pentru orice lant $x_1 \leq x_2 \leq \ldots \leq x_n$ exista un supremum, notat $\vee_n x_n$, care exista in aceasta multime.

Fie $(C, \leq)$ **cpo**. Un element al lui C - $c \in C$ se numeste punct fix pentru o functie f, daca $f(c) = c$. Numim cel mai mic punct fix al lui f - lpf acel element care este mai mic decat toate celelalte puncte fixe. Pentru orice $p \in C$ punct fix, $lpf \leq p$.

Exercitiul 1

Care sunt punctele fixe pentru urmatoarele functii? Indicati si care este cel mai mic punct fix pentru fiecare situatie data.

1. $f_1 : \mathcal{P}(\{1,2,3\}) \to \mathcal{P}(\{1,2,3\})$, $f_1(Y) = Y \cup \{1\}$.
   Observatie: toate submultimile care il contin pe $\{1\}$ sunt puncte fixe.
   Aceste elemente sunt: $\{1\}, \{1,2\}, \{1,3\}, \{1,2,3\}$.
   Care este cel mai mic punct fix? $\{1\}$

2. $f_2 : \mathcal{P}(\{1,2,3\}) \to \mathcal{P}(\{1,2,3\})$, $f_2(Y) = \begin{cases} \{1\} & 1 \in Y \\ \emptyset & \text{altfel} \end{cases}$
   $f_2(\emptyset) = \emptyset$ punct fix
   $f_2(\{1\}) = \{1\}$ punct fix

$f_2(\{2\}) = \emptyset$
$f_2(\{1,2\}) = \{1\}$
...
Singurele doua puncte fixe sunt $\emptyset$ si $\{1\}$, iar cel mai mic punct fix este $\emptyset$.

3. $f_3 : \mathcal{P}(\{1,2,3\}) \to \mathcal{P}(\{1,2,3\})$, $f_3(Y) = \begin{cases} \emptyset & 1 \in Y \\ \{1\} & \text{altfel} \end{cases}$
   Observam ca nu avem puncte fixe.

Teorie pentru Exercitiul 2

Fie $(A, \leq_A)$, $(B, \leq_B)$ doua multimi partial ordonate **mpo**. O functie $f : A \to B$ este monotona (crescatoare) daca $a_1 \leq_A a_2$ implica $f(a_1) \leq_B f(a_2)$, pentru orice $a_1, a_2 \in A$.

O clauza definita propozitionala este o formula care poate avea una dintre urmatoarele doua forme:

- q (clauza unitate).
- $p_1 \wedge p_2 \wedge ... \wedge p_n \to q$
  unde $p_1, p_2, ...p_n, q$ sunt variabile propozitionale.

Fie S o multime de clauze definite propozitionale. Fie $\mathcal{A}$ multimea variabilelor propozitionale $p_1, p_2, ..., p_n$ care apar in S. Fie $Baza = \{p_i | p_i \in S\}$ multimea clauzelor unitate. Definim functia $f_S : \mathcal{P}(\mathcal{A}) \to \mathcal{P}(\mathcal{A})$ prin

$$f_S(Y) = Y \cup Baza \cup \{a \in \mathcal{A} | (s_1 \wedge ... \wedge s_n \to a) \in S, s_1 \in Y, ..., s_n \in Y\}$$

Exercitiul 2

Sa se demonstreze ca functia f_S este monotona.

Solutie.

Fie Y_1 si Y_2 astfel incat $Y_1 \subseteq Y_2$. Trebuie sa demonstrez ca $f_S(Y_1) \subseteq f_S(Y_2)$.

$f_S(Y_1) = Y_1 \cup Baza \cup Z_1$
$f_S(Y_2) = Y_2 \cup Baza \cup Z_2$

Trebuie sa demonstrez ca $Z_1 \subseteq Z_2$.

$Z_1 = \{a \in \mathcal{A} | (s_1 \wedge ... \wedge s_n \rightarrow a), s_1, ..., s_n \in Y_1\}$
$Z_2 = \{a \in \mathcal{A} | (s_1 \wedge ... \wedge s_n \rightarrow a), s_1, ..., s_n \in Y_2\}$

Fie $a \in Z_1$. Inseamna ca exista $s_1, ...s_n \in Y_1$ astfel incat $s_1 \wedge ... \wedge s_n \rightarrow a$. Dar $Y_1 \subseteq Y_2$ (din ipoteza). Inseamna ca $s_1, ..., s_n \in Y_2$. Rezulta ca $a \in Z_2$. Am demonstrat $Z_1 \subseteq Z_2$.
Deci $f_S(Y_1) \subseteq f_S(Y_2)$, deci f_S este monotona.

Teorie Exercitiul 3

Fie $(A, \leq_A), (B, \leq_B)$ doua **cpo**. O functie $f : A \rightarrow B$ este continua daca $f(\vee_n a_n) = \vee_n f(a_n)$ pentru orice lant $\{a_n\}_n$ din A.
Observam ca orice functie continua este si crescatoare.
Pentru orice multime de clauze definite propozitionale S, functia f_S este continua.

Teorema Knaster-Tarski. Fie $(C, \leq)$ o multime partial ordonata completa si $F : C \rightarrow C$ o functie continua. Atunci, elementul

$$a = \vee_n F^n(\perp)$$

este cel mai mic punct fix al lui F.

Exercitiul 3

Calculati cel mai mic punct fix pentru functia f_S pentru urmatoarele multimi de clauze definite propozitionale.

1. $S_1 = \{x_1 \wedge x_2 \rightarrow x_3, x_4 \wedge x_2 \rightarrow x_5, x_2, x_6, x_6 \rightarrow x_1\}$

$\mathcal{A} = \{x_1, x_2, x_3, x_4, x_5, x_6\}$
$Baza = \{x_2, x_6\}$

$f_S(\emptyset) = Baza = \{x_2, x_6\}$
$f_S(\{x_2, x_6\}) = \{x_2, x_6, x_1\}$

$f_S(\{x_2, x_6, x_1\}) = \{x_2, x_6, x_1, x_3\}$
$f_S(\{x_2, x_6, x_1, x_3\}) = \{x_2, x_6, x_1, x_3\}$

am obtinut, conform th. Knaster-Tarski, ca $\{x_1, x_2, x_3, x_6\}$ este cel mai mic punct fix al lui f_S.

```
x2.
x6.
x3 :- x1, x2.
x5 :- x4, x2.
x1 :- x6.
```

2. $S_2 = \{x_1 \wedge x_2 \to x_3, x_4 \to x_1, x_5 \to x_2, x_2 \to x_5, x_4\}$

$\mathcal{A} = \{x_1, x_2, x_3, x_4, x_5\}$
$Baza = \{x_4\}$

$f_S(\emptyset) = Baza = \{x_4\}$
$f_S(\{x_4\}) = \{x_4, x_1\}$
$f_S(\{x_4, x_1\}) = \{x_4, x_1\}$

am obtinut, conform th. Knaster-Tarski, ca $\{x_1, x_4\}$ este cel mai mic punct fix al lui f_S.

```
x4.
x3 :- x1, x2.
x1 :- x4.
x2 :- x5.
x5 :- x2.
```

3. $S_3 = \{x_1 \to x_2, x_1 \wedge x_3 \to x_1, x_3\}$.

$\mathcal{A} = \{x_1, x_2, x_3\}$
$Baza = \{x_3\}$

$f_S(\emptyset) = Baza = \{x_3\}$
$f_S(\{x_3\}) = \{x_3\}$

Am obtinut ca $\{x_3\}$ este cel mai mic punct fix, conform th. Knaster-Tarski.

```
x3.
x2 :- x1.
x1 :- x1, x3.
```

Laboratorul 9 Prolog

Implementarea IMP - semantica operationala

In acest laborator vom implementa un limbaj care contine:

- expresii *aritmetice* si *booleene*;
  - `x + 3`
  - `x >= 7`
- instructiuni *de atribuire*, *conditionale* si *de ciclare*;
  - `x = 5`
  - `if(x >= 7, x = 5, x = 0)`
  - `while (x >= 7, x = x - 1)`
- compunerea instructiunilor;
  - `x = 7; while(x >= 0; x = x - 1)`
- blocuri de instructiuni.
  - `{x = 7; while(x >= 0, x = x - 1)}`

Un exemplu de program in IMP este

```
{
    x = 10;
    sum = 0;
    while (0 =< x,
    {
        sum = sum + x;
        x = x - 1
    }),
    sum
}
```

9.1. Definitia limbajului (BNF)

```
    E ::= n | x
        | E + E | E - E | E * E
```

```
B ::= true | false
    | E =< E | E >= E | E == E
    | not(B) | and(B, B) | or(B, B)

C ::= skip
    | x = E
    | if(B, C, C)
    | while(B, C)
    | { C }
    | C ; C

P ::= { C }, E
```

9.2. Implementarea limbajului in Prolog

Avem urmatorii doi operatori:

```
:- op(100, xf, {}).
:- op(1100, yf, ;).
```

Definim un predicat pentru fiecare neterminal din descrierea BNF de mai sus. Vom avea:

- `aexp/1` pentru expresii aritmetice;
- `bexp/1` pentru expresii booleene;
- `stmt/1` pentru instructiunile propriu-zise;
- `program/1` pentru structura programului.

9.2.1. Implementarea `aexp/1`

```
E ::= n | x | E + E | E - E | E * E
```

```
aexp(I) :- integer(I).      % intregii sunt cei din Prolog
aexp(X) :- atom(X).         % identificatorii din IMP sunt atomii
aexp(A1 * A2) :- aexp(A1), aexp(A2).
aexp(A1 + A2) :- aexp(A1), aexp(A2).
aexp(A1 - A2) :- aexp(A1), aexp(A2).
```

9.2.2. Implementarea `bexp/1`

```
    B ::= true | false
        | E =< E | E >= E | E == E
        | not(B) | and(B, B) | or(B, B)
```

```
bexp(true).
bexp(false).
bexp(A1 =< A2) :- aexp(A1), aexp(A2).
bexp(A1 >= A2) :- aexp(A1), aexp(A2).
bexp(A1 == A2) :- aexp(A1), aexp(A2).
bexp(and(BE1,BE2)) :- bexp(BE1), bexp(BE2).
bexp(or(BE1,BE2)) :- bexp(BE1), bexp(BE2).
bexp(not(BE)) :- bexp(BE).
```

9.2.3. Implementarea `stmt/1`

```
    C ::= skip
        | x = E
        | if(B, C, C)
        | while(B, C)
        | { C }
        | C ; C
```

```
stmt(skip).
stmt(X = AE) :- atom(X), aexp(AE).
stmt(if(BE,St1,St2)) :- bexp(BE), stmt(St1), stmt(St2).
stmt(while(BE,St)) :- bexp(BE), stmt(St).
stmt(St1;St2) :- stmt(St1), stmt(St2).
stmt({St}) :- stmt(St).
```

9.2.4. Implementarea `program/1`

```
    P ::= { C }, E
```

```
program(St,AE) :-
  stmt(St),
  aexp(AE).
```

Urmatorul predicat trebuie sa raspunda `true` :

```
test0 :- program({x = 10; sum = 0; while(0 =< x, {sum = sum + x; x = x-1)}}, s

?- test0. => true
```

9.3. Semantica operationala

- descrie cum se executa un program pe o masina abstracta;
- semantica **small-step** descrie cum avanseaza o executie in functie de reduceri succesive. Avem mereu un cuplu (cod, starea memoriei), notat in general $< cod, \sigma >$, iar o tranzitie este $< cod, \sigma > \rightarrow < cod', \sigma' >$

Vom defini care sunt regulile structurale pentru toate neterminalele din programul nostru, dand astfel o semantica a programului.

Avem deja implementate predicatele auxiliare de mai jos:

```
% get ne da informatii despre starea memoriei
% vi/2 este o pereche variabila - valoare
% de exemplu, putem avea S = [vi(X, 0), vi(Y, 1)] care spune
% ca X |-> 0 si Y |-> 1 in stadiul curent
% apelam get(S, X, -I) - dam starea memoriei, o variabila, si in I gasim valoa
get(S,X,I) :- member(vi(X,I),S).
get(_,_,0).

% elimina vi(X, I) din starea curenta a memoriei
set(S,X,I,[vi(X,I)|S1]) :- del(S,X,S1).
del([vi(X,_)|S],X,S).
del([H|S],X,[H|S1]) :- del(S,X,S1) .
del([],_,[]).
```

9.3.1. Regula ID

$$(\text{ID}) < x, \sigma > \rightarrow < i, \sigma >$$

daca $i = \sigma(x)$

```
smallstepA(X, S, I, S) :-
  atom(X),
  get(S, X, I).
```

9.3.2. Regula ADD

$$(\text{ADD1}) < i_1 + i_2, \sigma > \rightarrow < i, \sigma >$$

daca $i = i_1 + i_2$

```
smallstepA(I1 + I2, S, I, S) :-
  integer(I1), integer(I2),
  I is I1 + I2.
```

$$(\text{ADD2}) \frac{< a_1, \sigma > \rightarrow < a_1', \sigma >}{< a_1 + a_2, \sigma > \rightarrow < a_1' + a_2, \sigma >}$$

```
smallstepA(AE1 + I, S, AE2 + I, S) :-
  integer(I),
  smallstepA(AE1, S, AE2, S).
```

$$(\text{ADD3}) \frac{< a_2, \sigma > \rightarrow < a_2', \sigma >}{< a_1 + a_2, \sigma > \rightarrow < a_1 + a_2', \sigma >}$$

```
smallstepA(I + AE1,S,I + AE2,S) :-
  integer(I),
  smallstepA(AE1, S, AE2, S).
```

9.3.3. Regula DIFF [de lucrat]

$$(\text{DIFF1}) < i_1 - i_2, \sigma > \rightarrow < i, \sigma >$$

daca $i = i_1 - i_2$

```
smallstepA(I1-I2, S, I, S) :-
    integer(I1), integer(I2),
    I is I1-I2.
```

$$(\text{DIFF2}) \frac{< a_1, \sigma > \rightarrow < a_2, \sigma >}{< i - a_1, \sigma > \rightarrow < i - a_2, \sigma >}$$

```
smallstepA(I-AE1, S, I-AE2,S) :-
    integer(I),
    smallstepA(AE1, S, AE2, S)
```

$$(\text{DIFF3})\frac{< a_1, \sigma >\rightarrow< a_2, \sigma >}{< a_1 - i, \sigma >\rightarrow< a_2 - i, \sigma >}$$

```
smallstepA(AE1-I, S, AE2-I, S) :-
    integer(I),
    smallstepA(AE1, S, AE2, S).
```

9.3.4. Regula MUL [de lucrat]

$$(\text{MUL1}) < i_1 * i_2, \sigma >\rightarrow< i, \sigma >$$

daca $i = i_1 * i_2$

```
smallstepA(I1*I2, S, I, S) :-
    integer(I1), integer(I2),
    I is I1*I2.
```

$$(\text{MUL2})\frac{< a_1, \sigma >\rightarrow< a_1', \sigma >}{< a_1 * a_2, \sigma >\rightarrow< a_1' * a_2, \sigma >}$$

```
smallstepA(AE1*I, S, AE2*I,S) :-
    integer(I),
    smallstepA(AE1, S, AE2, S)
```

$$(\text{MUL3})\frac{< a_2, \sigma >\rightarrow< a_2', \sigma >}{< a_1 * a_2, \sigma >\rightarrow< a_1 * a_2', \sigma >}$$

```
smallstepA(I*AE1, S, I*AE2,S) :-
    integer(I),
    smallstepA(AE1, S, AE2, S)
```

9.3.5. Reguli de comparatie LEQ

$$(\text{LEQ-FALSE}) < i_1 =< i_2, \sigma >\rightarrow< \text{false}, \sigma >$$

daca $i_1 > i_2$

```
smallstepB(I1 =< I2, S, false, S) :-
    integer(I1), integer(I2),
    I1 > I2.
```

$$(\text{LEQ-TRUE}) < i_1 =< i_2, \sigma > \rightarrow < \text{true}, \sigma >$$

daca $i_1 \leq i_2$

```
smallstepB(I1 =< I2, S, true, S):-
    integer(I1), integer(I2),
    I1 =< I2.
```

$$(\text{LEQ1}) \frac{< a_1, \sigma > \rightarrow < a_1', \sigma >}{< a_1 =< a_2, \sigma > \rightarrow < a_1' =< a_2, \sigma >}$$

```
smallstepB(AE1 =< I, S, AE2 =< I, S) :-
    integer(I),
    smallstepA(AE1, S, AE2, S).
```

$$(\text{LEQ2}) \frac{< a_2, \sigma > \rightarrow < a_2', \sigma >}{< a_1 =< a_2, \sigma > \rightarrow < a_1 =< a_2', \sigma >}$$

```
smallstepB(I =< AE1, S, I =< AE2, S) :-
     integer(I),
     smallstepA(AE1, S, AE2, S).
```

9.3.6. Reguli de comparatie GEQ

$$(\text{GEQ-FALSE}) < i_1 >= i_2, \sigma > \rightarrow < \text{false}, \sigma >$$

daca $i_1 > i_2$

```
smallstepB(I1 >= I2, S, false, S) :-
    integer(I1), integer(I2),
    I1 < I2.
```

$$(\text{GEQ-TRUE}) < i_1 >= i_2, \sigma > \rightarrow < \text{true}, \sigma >$$

daca $i_1 \leq i_2$

```
smallstepB(I1 >= I2, S, true, S):-
    integer(I1), integer(I2),
    I1 >= I2.
```

$$(\text{GEQ1})\frac{< a_1, \sigma > \rightarrow < a_1', \sigma >}{< a_1 >= a_2, \sigma > \rightarrow < a_1' >= a_2, \sigma >}$$

```
smallstepB(AE1 >= I, S, AE2 >= I, S) :-
    integer(I),
    smallstepA(AE1, S, AE2, S).
```

$$(\text{GEQ2})\frac{< a_2, \sigma > \rightarrow < a_2', \sigma >}{< a_1 >= a_2, \sigma > \rightarrow < a_1 >= a_2', \sigma >}$$

```
smallstepB(I >= AE1, S, I >= AE2, S) :-
     integer(I),
     smallstepA(AE1, S, AE2, S).
```

9.3.7. Reguli de egalitate EQ

$$(\text{EQ-FALSE}) < i_1 == i_2, \sigma > \rightarrow < \text{false}, \sigma >$$

daca $i_1 \neq i_2$

```
smallstepB(I1 == I2, S, false, S) :-

integer(I1), integer(I2),

I1 \= I2.
```

$$(\text{EQ-TRUE}) < i_1 == i_2, \sigma > \rightarrow < \text{true}, \sigma >$$

daca $i_1 = i_2$

```
smallstepB(I1 == I2, S, true, S):-

integer(I1), integer(I2),
```

```
I1 == I2.
```

$$(\text{EQ1})\frac{< a_1, \sigma > \rightarrow < a_1', \sigma >}{< a_1 == a_2, \sigma > \rightarrow < a_1' == a_2, \sigma >}$$

```
smallstepB(AE1 == I, S, AE2 == I, S) :-
integer(I),
smallstepA(AE1, S, AE2, S).
```

$$(\text{EQ2})\frac{< a_2, \sigma > \rightarrow < a_2', \sigma >}{< a_1 == a_2, \sigma > \rightarrow < a_1 == a_2', \sigma >}$$

```
smallstepB(I == AE1, S, I == AE2, S) :-
integer(I),
smallstepA(AE1, S, AE2, S).
```

9.3.8. Reguli pentru conjunctie

$$(\text{AND1}) < and(\text{true}, BE_2), \sigma > \rightarrow < BE_2, \sigma >$$

```
smallstepB(and(true, BE2), S, BE2, S).
```

$$(\text{AND2}) < and(\text{false}, \_), \sigma > \rightarrow < \text{false}, \sigma >$$

```
smallstepB(and(false,_), S, false, S).
```

$$(\text{AND3})\frac{< BE_1, \sigma > \rightarrow < BE_2, \sigma >}{< and(BE_1, BE), \sigma > \rightarrow < and(BE_2, BE), \sigma >}$$

```
smallstepB(and(BE1, BE),S,and(BE2, BE),S) :-
    smallstepB(BE1, S, BE2, S).
```

9.3.9. Reguli pentru disjunctie

$$(\text{OR1}) < or(\text{true}, \_), \sigma > \rightarrow < \text{true}, \sigma >$$

```
smallstepB(or(true, _), S, true, S).
```

$$(\text{OR2}) < and(\text{false}, BE2), \sigma > \rightarrow < BE2, \sigma >$$

```
smallstepB(or(false,BE2), S, BE2 , S).
```

$$(\text{OR3}) \frac{< BE_1, \sigma > \rightarrow < BE_2, \sigma >}{< or(BE_1, BE), \sigma > \rightarrow < or(BE_2, BE), \sigma >}$$

```
smallstepB(or(BE1, BE),S,or(BE2, BE),S) :-
    smallstepB(BE1, S, BE2, S).
```

9.3.10. Regulile pentru negatie

$$\text{!-TRUE} < not(\text{true}), \sigma > \rightarrow < \text{false}, \sigma >$$

```
smallstepB(not(true),S,false,S) .
```

$$\text{!-FALSE} < not(\text{false}), \sigma > \rightarrow < \text{true}, \sigma >$$

```
smallstepB(not(false),S,true,S) .
```

$$(\text{NEG}) \frac{< a, \sigma > \rightarrow < a', \sigma >}{< not(a), \sigma > \rightarrow < not(a'), \sigma >}$$

```
smallstepB(not(BE1),S,not(BE2),S) :-
    smallstepB(BE1,S,BE2,S).
```

9.3.11. Regula ASGN

$$(\text{ASGN1}) < x = i, \sigma > \rightarrow < \text{skip}, \sigma' >$$

daca $\sigma' = \sigma[i/x]$

```
smallstepS(X = AE, S, skip, S1) :-
  integer(AE), set(S, X, AE, S1) .
```

$$(\text{ASGN2}) \frac{< e_1, \sigma > \rightarrow < e_2, \sigma >}{< x = e_1, \sigma > \rightarrow < x = e_2, \sigma >}$$

```
smallstepS(X = AE1, S, X = AE2, S) :-
  smallstepA(AE1, S, AE2, S).
```

9.3.12. Regula NEXT-STMT

$$(\text{NEXT-STMT1}) < \text{skip}; s_2, \sigma > \rightarrow < s_2, \sigma >$$

```
smallstepS((skip;St2), S, St2, S).
```

$$(\text{NEXT-STMT2}) \frac{< s_1, \sigma > \rightarrow < s_1', \sigma' >}{< s_1; s_2, \sigma > \rightarrow < s_1'; s_2, \sigma >}$$

```
smallstepS((St1;St),S1,(St2;St),S2) :-
  smallstepS(St1,S1,St2,S2) .
```

9.3.13. Regula pentru blocuri de cod

$$(\text{BLOCK}) < \{E\}, \sigma > \rightarrow < E, \sigma >$$

```
smallstepS({E}, S, E, S).
```

9.3.14. Reguli pentru IF

$$(\text{IF-TRUE}) < if(\text{true}, St_1, \_), \sigma > \rightarrow < St_1, \sigma >$$

```
smallstepS(if(true, St1, _), S, St1, S).
```

$$(\text{IF-FALSE}) < if(\text{false}, \_, St_2), \sigma > \rightarrow < St_2, \sigma >$$

```
smallstepS(if(false, _, St2), S, St2, S).
```

$$(\text{IF})\frac{< BE_1, \sigma > \rightarrow < BE_2, \sigma >}{< if(BE_1, St_1, St_2), \sigma > \rightarrow < if(BE_2, St_1, St_2), \sigma >}$$

```
smallstepS(if(BE1,St1,St2),S,if(BE2,St1,St2),S) :-
    smallstepB(BE1, S, BE2, S).
```

9.3.15. Regula pentru WHILE

$$(\text{WHILE}) < while(BE, St), \sigma > \\ \rightarrow < if(BE, (St; while(BE, St)), \text{skip}), \sigma >$$

```
smallstepS(while(BE, St), S, if(BE, (St; while(BE, St)), skip), S).
```

9.3.16. Reguli pentru programe

```
smallstepP(skip, AE1, S1, skip, AE2, S2) :-
    smallstepA(AE1, S1, AE2, S2).

smallstepP(St1, AE, S1, ST2, AE, S2) :-
    smallstepS(St1, S1, St2, S2).


run(skip, I, _, I) :- integer(I).
run(St1, AE1, S1, I) :-
    smallstepP(St1, AE1, S1, ST2, AE2, S2),
        run(St2, AE2, S2, I).


run_program(Name) :- defpg(Name, {P}, E),
    run(P, E, [], I),
    write(I).


defpg(pg1, {nr = 0; while(nr =< 10, nr=nr+1)}, nr).
```

Saptamana 13 FLP

Reactualizare materie FLP

- Sintaxa limbajului Prolog. Recursivitate, liste, **cum raspunde Prolog intrebarilor?** - algoritmul de unificare (in prima faza).
- Elemente de logica propozitionala, sistemul deductiei naturale, puncte fixe si teorema Knaster-Tarski, rezolutia propozitionala si rezolutia SLD.
- Semantica operationala - implementarea IMP in Prolog.
- Semantica denotationala - implementarea Hask in Haskell.
- λ-calcul cu implementare in Haskell.

Implementarea β-reductiei in λ-calcul

Vom implementa β-reductia in Haskell cu scopul de a obtine o β-forma normala.

[Aplicare] $(\lambda x.t)u \to_\beta [u/x]t$
[Compatibilitate] $t_1 \to_\beta t_2$ implica

- $tt_1 \to_\beta tt_2$
- $t_1t \to_\beta t_2t$
- $\lambda x.t_1 \to_\beta \lambda x.t_2$

Avem deja implementata substitutia $[u/x]t$:

```
subst :: Term -> Variable -> Term -> Term
subst u x (V y)
    | x == y = u
    | otherwise = V y
subst u x (App t1 t2) = App (subst u x t1) (subst u x t2)
subst u x (Lam y t)
    | x == y = Lam y t
    | notElem y (freeVars u) = Lam y (subst u x t)
    | otherwise = error "Nu putem efectua substitutia"
```

In acest caz, β-reductia este urmatoarea:

```
betaReduction :: Term -> Maybe Term
betaReduction (App (Lam x t) u) = Just $ subst u x t -- vezi [Aplicare]
betaReduction (App t1 t2)
    | isJust t1' = Just $ App (fromJust t1') t2
    | isJust t2' = Just $ App t1 (fromJust t2')
    | otherwise = Nothing
    where
        t1' = betaReduction t1
        t2' = betaReduction t2
betaReduction (Lam x t)
    | isJust t' = Just $ Lam x (fromJust t')
    | otherwise = Nothing
    where
        t' = betaReduction t
betaReduction _ = Nothing
```

Putem defini o functie care sa ne calculeze β-forma normala:

```
betaNormalForm :: Term -> Term
betaNormalForm t
    | isJust t' = betaNormalForm $ fromJust t'
    | otherwise = t
    where
        t' = betaReduction t
```

Sistemul de inferenta pe tipuri

Fie relatia $\Gamma \vdash e : \tau$ unde

- τ este un tip

$$\tau ::= \mathbf{int} \mid \mathbf{bool} \mid \tau \to \tau \mid a$$

- e este un termen
- Γ este mediul de tipuri, o functie partiala finita care asociaza tipuri variabilelor

Axiome

$\texttt{(:VAR)}\ \Gamma \vdash x : \tau$ daca $\Gamma(x) = \tau$

$\texttt{(:INT)}\ \Gamma \vdash n : \texttt{int}$ daca n intreg

$\texttt{(:BOOL)}\ \Gamma \vdash b : \texttt{bool}$ daca b este true sau false

Expresii

$$\texttt{(:IOP)}\ \frac{\Gamma \vdash e_1 : \texttt{int}\ \ \Gamma \vdash e_2 : \texttt{int}}{\Gamma \vdash e_1 \circ e_2 : \texttt{int}} \text{ daca } \circ \in \{+, -, *, /\}$$

$$\texttt{(:COP)}\ \frac{\Gamma \vdash e_1 : \texttt{int}\ \ \Gamma \vdash e_2 : \texttt{int}}{\Gamma \vdash e_1 \circ e_2 : \texttt{bool}} \text{ daca } \circ \in \{\leq, \geq, <, >, =\}$$

$$\texttt{(:BOP)}\ \frac{\Gamma \vdash e_1 : \texttt{bool}\ \ \Gamma \vdash e_2 : \texttt{bool}}{\Gamma \vdash e_1 \circ e_2 : \texttt{bool}} \text{ daca } \circ \in \{\texttt{and}, \texttt{or}\}$$

$$\texttt{(:IF)}\ \frac{\Gamma \vdash e_b : \texttt{bool}\ \ \Gamma \vdash e_1 : \tau\ \ \Gamma \vdash e_2 : \tau}{\Gamma \vdash \texttt{if}\ \ e_b\ \ \texttt{then}\ \ e_1\ \ \texttt{else}\ \ e_2 : \tau}$$

Fragmentul functional

$$\texttt{(:FN)}\ \frac{\Gamma' \vdash e : \tau'}{\Gamma \vdash \lambda x.e : \tau \to \tau'} \text{ daca } \Gamma' = \Gamma[x \to \tau]$$

$$\texttt{(:APP)}\ \frac{\Gamma \vdash e_1 : \tau' \to \tau\ \ \Gamma \vdash e_2 : \tau'}{\Gamma \vdash e_1\ e_2 : \tau}$$

Limbajul LAMBDA

Vom exemplifica sistemul de inferenta pe tipuri in Prolog, implementand urmatorul limbaj LAMBDA.

Sintaxa BNF este urmatoarea:

```
e ::= x | n | true | false
    | e + e | e < e | not(e)
    | if e then e else e
    | x -> e | e $ e
```

Fie definiti urmatorii operatori in Prolog:

```
:- op(100,xfy,or).
:- op(100,xfy,and).
:- op(200,yfx,$).
```

Definiti un predicat `exp/1` care sa verifice sintaxa limbajului LAMBDA. Amintiti-va de implementarea pentru IMP.

```
% exp / 1
exp(true).
exp(false).
exp(Id) :- atom(Id).
exp(Lit) :- integer(Lit).
exp(E1 + E2) :- exp(E1), exp(E2).
exp(if(E1, E2, E3)) :- exp(E1), exp(E2), exp(E3).
exp(Id -> Exp) :- atom(Id), exp(Exp).
exp(Exp1 $ Exp2) :- exp(Exp1), exp(Exp2).
```

Implementarea sistemului de inferenta pe tipuri pentru LAMBDA

Avem urmatorii operatori si urmatoarele predicate deja date:

```
:- op(100,xfy,or).
:- op(100,xfy,and).
:- op(200,yfx,$).

remove(Gamma, X, Gamma1) :- select((X,_), Gamma, Gamma1),!.
remove(Gamma, _, Gamma).

set(Gamma, X, T, [(X,T) | Gamma1]) :- remove(Gamma, X, Gamma1).
get(Gamma, X, T) :- member((X, T), Gamma).
```

Vom utiliza predicatul `type/3`, cu urmatoarele argumente:

- Γ - mediul de tipuri;
- e - expresia LAMBDA;
- τ - tipul expresiei.

```
% Axiomele
% (:VAR)
```

```
type(Gamma, X, T) :- atom(X), get(Gamma, X, T).

% (:INT)
type(_, I, int) :- integer(I).

% (:BOOL)
type(_, true, bool).
type(_, false, bool).
```

Pentru expresii, implementam similar cu modul de implementare din semantica operationala.

Operatii intregi

```
type(Gamma, E1 + E2, int) :-
    type(Gamma, E1, int),
    type(Gamma, E2, int).

type(Gamma, E1 - E2, int) :-
    type(Gamma, E1, int),
    type(Gamma, E2, int).

type(Gamma, E1 * E2, int) :-
    type(Gamma, E1, int),
    type(Gamma, E2, int).

type(Gamma, E1 / E2, int) :-
    type(Gamma, E1, int),
    type(Gamma, E2, int).
% completati pentru -, *, /
```

Operatii de comparatie

```
type(Gamma, E1 =< E2, bool) :-
    type(Gamma, E1, int),
    type(Gamma, E2, int).

type(Gamma, E1 >= E2, bool) :-
    type(Gamma, E1, int),
    type(Gamma, E2, int).

type(Gamma, E1 < E2, bool) :-
```

```
    type(Gamma, E1, int),
    type(Gamma, E2, int).

type(Gamma, E1 > E2, bool) :-
    type(Gamma, E1, int),
    type(Gamma, E2, int).

% completati pentru >=, <, >, =
```

Operatii booleene

```
type(Gamma, E1 and E2, bool) :-
    type(Gamma, E1, bool),
    type(Gamma, E2, bool).

type(Gamma, E1 or E2, bool) :-
    type(Gamma, E1, bool),
    type(Gamma, E2, bool).

    type(Gamma, not E1, bool) :-
    type(Gamma, E1, bool).
% completati si pentru or si not
```

IF

```
type(Gamma, if(E, E1, E2), T) :-
    type(Gamma, E, bool),
    type(Gamma, E1, T),
    type(Gamma, E2, T).
```

Fragmentul functional - :FN

```
type(Gamma, X -> E, TX -> TE) :-
    atom(X),
    set(Gamma, X, TX, GammaX),
    type(GammaX, E, TE).
```

Aplicarea functiilor

```
type(Gamma, E1 $ E2, T) :-
    type(Gamma, E1, T2 -> T),
```

```
            type(Gamma, E2, T2).
```

Saptamana 14 FLP

Exercitii recapitulative - Deductie naturala, Algoritm de unificare, Rezolutie SLD

Structura examenului

- 2h, fizic, in laboratoare / amfiteatre;
- aveti voie cu materiale ajutatoare de la curs / seminar / laborator, dar fara internet;
- 1p din oficiu, si se trece de la o nota > 4.99
- este format din doua parti:
  - parte teoretica - 4p unde vor fi 3 probleme din lista urmatoare:
    - unificare
    - deductie naturala
    - rezolutie SLD + arbori de rezolutie si de executie
    - puncte fixe
    - pasi in semantica operationala
    - substitutii si β-reductii in λ-calcul
  - parte practica - 5p
    - o problema de Prolog - 2p
    - o problema de limbaj de programare - 3p (se da sintaxa unui limbaj de programare, se va verifice daca un sir de caractere primit respecta sintaxa *verificare sintactica*; si apoi se va interpreta respectivul sir de caractere - *semantica*)

Exercitii - Deductia Naturala

Exercitiul 1 Sa se demonstreze ca urmatorul secvent este valid

$$p \wedge q \to \neg u, p \to u, p, q \vdash \neg r$$

Demonstrez ca formula logica din dreapta relatiei $\vdash$ se poate obtine din

- ipotezele de deductie din stanga relatiei $\vdash$;

- regulile de inferenta din sistemul deductiei naturale pentru logica propozitionala.

(1) $p \wedge q \to \neg u$ [ipoteza]
(2) $p \to u$ [ipoteza]
(3) p [ipoteza]
(4) q [ipoteza]
(5) u [$\to e(3, 2)$]
(6) $p \wedge q$ [$\wedge i(3, 4)$]
(7) $\neg u$ [$\to e(6, 1)$]
(8) $\bot$ [$\neg e$(5, 7)]
(9) $\neg r$ [$\bot e(8)$]

Exercitiul 2 Sa se demonstreze ca urmatorul secvent este valid:

$$\neg q, p \vee q, s \to \neg p, s \vdash \neg r$$

(1) $\neg q$ [ipoteza]
(2) $p \vee q$ [ipoteza]
(3) $s \to \neg p$ [ipoteza]
(4) s [ipoteza]
(5) $\neg p$ [$\to e(4, 3)$]
    ---
(6) | p [asumptie]
(7) | $\bot$ [$\neg e(6, 5)$]
(8) | $\neg r$ [$\bot e(7)$]

    ---
(9) | q [asumptie]
(10) | $\bot$ [$\neg e(9, 1)$]
(11) | $\neg r$ [$\bot e(10)$]
    ---
(12) $\neg r$ [$\vee e(2, 6 - 8, 9 - 11)$]

Exercitiul 3 Sa se demonstreze ca urmatorul secvent este valid:

$$p, s \to \neg q, \neg p \vee q, s \vdash r$$

(1) p [ipoteza]
(2) $s \to \neg q$ [ipoteza]

(3) $\neg p \vee q$ [ipoteza]
(4) s [ipoteza]
(5) $\neg q$ [$\rightarrow e(4,2)$]
    ---
(6) | $\neg p$ [asumptie]
(7) | $\perp$ [$\neg e(1,6)$]
(8) | r [$\perp e(7)$]

    ---
(9) | q [asumptie]
(10) | $\perp$ [$\neg e(9,5)$]
(11) | r [$\perp e(10)$]
    ---
(12) r [$\vee e(3, 6-8, 9-11)$]

Exercitiu - Rezolutia SLD

Gasiti o SLD-respingere pentru programul de mai jos, cu tinta
`?- p(X), m(Y, X), p(Y)`. Indicati, la fiecare pas, regula si substitutiile folosite.

```
1. m(e, c).
2. m(d, b).
3. f(a, b).
4. f(a, c).
5. p(a).
6. p(d).
7. p(X) :- f(Y, X), p(Y).
```

Mereu, cand lucram un exercitiu bazat pe rezolutia SLD, primul pas este sa transpunem cerinta in formule logice.

Facts se transpun in formule atomice (termeni din FOL)
Regulile de inferenta din Prolog se transpun astfel:

```
P :- Q1, Q2, ..., Qn.
```

$$Q_1 \wedge Q_2 \wedge ... \wedge Q_n \rightarrow P$$
$$\equiv \neg(Q_1 \wedge Q_2 \wedge ... \wedge Q_n) \vee P$$

$$\equiv \neg Q_1 \vee \neg Q_2 \vee ... \vee \neg Q_n \vee P$$

(1) $m(e, c)$
(2) $m(d, b)$
(3) $f(a, b)$
(4) $f(a, c)$
(5) $p(a)$
(6) $p(d)$
(7) $\neg f(Y, X) \vee \neg p(Y) \vee p(X)$

`?- p(X), m(Y, X), p(Y)`
Cand fac rezolutia, tinta devine $\neg p(X) \vee \neg m(Y, X) \vee \neg p(Y)$

$G_0 = \neg p(X) \vee \neg m(Y, X) \vee \neg p(Y)$

$G_1' = \neg m(Y, a) \vee \neg p(Y)$
aplicand $Rez(G_0, 5)$ cu $\theta(X) = a$ - path blocant

$G_1'' = \neg m(Y, d) \vee \neg p(Y)$
aplicand $Rez(G_0, 6)$ cu $\theta(X) = d$ - path blocant
----------\

$G_0 = \neg p(X) \vee \neg m(Y, X) \vee \neg p(Y)$
$G_1 = \neg f(Z, X) \vee \neg p(Z) \vee \neg m(Y, X) \vee \neg p(Y)$
aplicand $Rez(G_0, 7)$ cu $\theta(X) = X$
$G_2 = \neg f(a, X) \vee \neg m(Y, X) \vee \neg p(Y)$
aplicand $Rez(G_1, 5)$ cu $\theta(Z) = a$
$G_3 = \neg m(Y, b) \vee \neg p(Y)$
aplicand $Rez(G_2, 3)$ cu $\theta(X) = b$
$G_4 = \neg m(d, b)$
aplicand $Rez(G_3, 6)$ cu $\theta(Y) = d$
$G_5 = \square$
aplicand $Rez(G_4, 2)$

Am gasit o SLD-respingere pentru tinta data, deci programul Prolog raspunde cu **`true`**.

Exercitiu - Algoritmul de unificare

Aplicati algoritmul de unificare pentru urmatorii doi termeni.

$g(y, f(x), b) = g(x, y, b)$

Avem x, y variabile, b constanta ($b/0$), $f/1$, $g/3$

REZOLVA: $x = t$ sau $t = x$
se muta in multimea solutiei informatia $x = t$
iar in restul multimii de rezolvat, toate aparitiile lui x sunt substituite cu t
Este necesar sa verificam ca, in egalitatea $x = t$ sau $t = x$, x nu apare in t. Daca x apare in t, atunci suntem pe un caz de $ESEC$.

Un alt caz de esec este la aplicarea descompunerii, cand incercam sa unificam simboluri de functii diferite. $f(x) = g(y). b = f(x)$ - sunt cazuri de ESEC

Multimea solutiei	**Multimea de rezolvat**	**Operatie aplicata**
$\emptyset$	$g(y, f(x), b) = g(x, y, b)$	DESCOMPUNE
$\emptyset$	$y = x, f(x) = y, b = b$	SCOATE
$\emptyset$	$y = x, f(x) = y$	REZOLVA
$y = f(x)$	$f(x) = x$	ESEC

Pentru exemplul de mai sus, nu exista un unificator pentru termenii $g(y, f(x), b)$ si $g(x, y, b)$.

Incercam pentru $f(a, x, g(x)) = f(a, y, y)$

Multimea solutiei	**Multimea de rezolvat**	**Operatie aplicata**
$\emptyset$	$f(a, x, g(x)) = f(a, y, y)$	DESCOMPUNE
$\emptyset$	$a = a, x = y, g(x) = y$	SCOATE
$\emptyset$	$x = y, g(x) = y$	REZOLVA
$x = y$	$g(y) = y$	ESEC