# 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.
- $\lambda$-calcul cu implementare in Haskell.

## Implementarea $\beta$-reductiei in $\lambda$-calcul

Vom implementa $\beta$-reductia in Haskell cu scopul de a obtine o $\beta$-forma normala.

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

- $tt_1 \rightarrow_\beta tt_2$
- $t_1 t \rightarrow_\beta t_2 t$
- $\lambda x.t_1 \rightarrow_\beta \lambda x.t_2$

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, $\beta$-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 $\beta$-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

- $\tau$ este un tip

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

- $e$ este un termen
- $\Gamma$ 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 $\texttt{exp}/1$ 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 $\texttt{type}/3$, cu urmatoarele argumente:

- $\Gamma$ - mediul de tipuri;
- $e$ - expresia LAMBDA;
- $\tau$ - 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).
```