



THIS PAGE IS  
INTENTIONALLY  
LEFT BLANK.

Kazimir Majorinc

# REKURZIVNE FUNKCIJE NA SIMBOLIČKIM IZRAZIMA (II)

Povijest Lispa 8.

Razmjena vještina  
Hacklab u mami  
13. listopada 2012.

1. Popis uobičajenih metoda za izgradnju matematičkih izraza i posebno, definicija funkcija. Npr.  $\lambda((x,y),y\vee x)$  umjesto  $(x,y)\mapsto y\vee x$ .

2. Definiraju se simbolički izrazi (S-izrazi), **A**, **B** ... **Z**, **AA**, **AB**, ..., **(A,A)**, **(A,B)**, ..., **(A,(A,A))** ... **(A,B,C)**. Logičke formule se mogu zapisati u tom obliku.

3. Koristeći metode iz 1, definiraju se funkcije na S-izrazima. No, svaka definicija funkcije je i sama izraz; meta-izraz, M-izraz. Mala slova, uglate zagrade i točka-zarezi.

4. Pokazuje se kako prevesti meta-izraze u simboličke-izraze. Npr.  $\lambda[[x;y];y\vee x]$  se prevodi u **(LAMBDA, (X, Y), (OR, Y, X))**

5. Definira se univerzalna S-funkcija *apply*. Npr.

$$f[\mathbf{X};\mathbf{Y}] = \mathit{apply}[f^*; (\mathbf{X}, \mathbf{Y})]$$

# 4. Prevođenje meta-izraza u simboličke izraze

## 4. Prevođenje meta-izraza u simboličke izraze.

Oznaka:  $e^*$  je prijevod meta-izraza u  $e$  S-izraz.

(1) Posebit slučaj meta-izraza su S-izrazi. (kao što su konstante 1, 2, ... posebit slučaj aritmetičkih izraza.) Prevode se konkateniranjem između **(QUOTE i )**. Primjerice,

prijevod **x** je **(QUOTE, x)**,

prijevod **(x, (y, z))** je **(QUOTE, (x, (y, z)))**,

prijevod **(QUOTE, ABA)** je **(QUOTE, (QUOTE, ABA))**.

(2) Varijable i funkcije se u meta-izrazima označavaju kao nizovi malih slova, npr.  $x$ ,  $y$ ,  $append$  itd. Prijevodi takvih nizova su iste nizovi, ali velikih slova, primjerice  $X$ ,  $Y$ ,  $APPEND$ .

**QUOTE** nam treba da ne bi preveli  $append[x,x]$  i  $append[x,X]$  u isti izraz.

(3) Primjene funkcija se označavaju meta-izrazima

$$f [e_1; \dots; e_n].$$

Prevode se u

$$(f^*, e_1^*, \dots, e_n^*).$$

Npr.  $append[x; (X, X)]$  u  $(APPEND, X, (QUOTE, (X, X)))$

(4) Uvjetni meta-izrazi imaju oblik

$$[p_1 \rightarrow e_1; \dots; p_n \rightarrow e_n]$$

Prevode se u

$$(\text{COND}, ((p_1^*, e_1^*), \dots, (p_n^*, e_n^*)))$$

(5) Meta-izrazi koji definiraju “anonimne” funkcije imaju oblik:

$$\lambda[[v_1; \dots; v_n]; E].$$

Prevode se u

$$(\text{LAMBDA}, (v_1^*, \dots, v_n^*), E^*).$$

(6) Analogno i *label* meta-izrazi kojima se definiraju funkcije sa imenom vidljivim “iznutra”.

# McCarthyjev primjer.

label [subst;  $\lambda[[x; y; z]; [\text{atom } [z] \rightarrow [\text{eq } [y; z] \rightarrow x; T \rightarrow z]; T \rightarrow$   
cons [subst[x; y; car [z]]; subst [x; y; cdr [z]]]]]]

se prevodi u

```
( LABEL , SUBST , ( LAMBDA , ( X , Y , Z ) , ( COND ( ( ATOM , Z ) , ( COND , ( EQ , Y , Z ) , X ) , ( ( QUOTE , T ) , Z ) ) ) , ( ( QUOTE , T ) , ( CONS , ( SUBST , X , Y , ( CAR Z ) ) , ( SUBST , X , Y , ( CDR , Z ) ) ) ) ) ) )
```

*“This notation is writable and somewhat readable.”*

## 5. Univerzalna S-funkcija *apply*.

Definira se univerzalna S-funkcija *apply* takva da za svaku primjenu “obične” rekurzivne S-funkcije *f* vrijedi

$$f[e_1; \dots; e_n] = \text{apply}[f^*; (e_1^*, \dots, e_n^*)]$$

Što dobijamo tom funkcijom?

**John McCarthy** je funkciju razvio iz teorijskog interesa.

**Steve Russell** je uočio da implementiranjem takve funkcije dobija interpreter.



# Eval.

$apply[f^*; (e_1^*, \dots, e_n^*)]$  se svodi na  $eval[(f^*, e_1^*, \dots, e_n^*); NIL]$

gdje je *eval* još zanimljivija funkcija od *apply*. Eval izračunava vrijednost izraza koji sadrži slobodne varijable, pri čemu su vrijednosti slobodnih varijabli zadane listom.

Primjerice,

```
(EVAL (QUOTE (CONS S1 S2)) (QUOTE ((S1 A) (S2 (B C)))))  
(A . (B . (C . NIL)))
```

```

eval [e; a] = [
atom [e] → assoc [e; a];
atom [car [e]] → [
eq [car [e]; QUOTE] → cadr [e];
eq [car [e]; ATOM] → atom [eval [cadr [e]; a]];
eq [car [e]; EQ] → [eval [cadr [e]; a] = eval [caddr [e]; a]];
eq [car [e]; COND] → cvcon [cdr [e]; a];
eq [car [e]; CAR] → car [eval [cadr [e]; a]];
eq [car [e]; CDR] → cdr [eval [cadr [e]; a]];
eq [car [e]; CONS] → cons [eval [cadr [e]; a]; eval [caddr [e];
a]]; T → eval [cons [assoc [car [e]; a];
evlis [cdr [e]; a]; a]];
eq [caar [e]; LABEL] → eval [cons [caddr [e]; cdr [e]];
cons [list [cadar [e]; car [e]; a]];
eq [caar [e]; LAMBDA] → eval [caddr [e];
append [pair [cadar [e]; evlis [cdr [e]; a]; a]]]

```

```

(define (EVAL e a)
  (COND ((ATOM e) (ASSOC e a))
        ((ATOM (CAR e))
         (COND ((EQ (CAR e) (QUOTE QUOTE)) (EVAL-QUOTE e a))
               ((EQ (CAR e) (QUOTE ATOM)) (EVAL-ATOM e a))
               ((EQ (CAR e) (QUOTE EQ)) (EVAL-EQ e a))
               ((EQ (CAR e) (QUOTE COND)) (EVAL-COND e a))
               ((EQ (CAR e) (QUOTE CAR)) (EVAL-CAR e a))
               ((EQ (CAR e) (QUOTE CDR)) (EVAL-CDR e a))
               ((EQ (CAR e) (QUOTE CONS)) (EVAL-CONS e a))
               ((QUOTE T) (EVAL-REST e a))))
        ((EQ (CAAR e) (QUOTE LABEL)) (EVAL-LABEL e a))
        ((EQ (CAAR e) (QUOTE LAMBDA)) (EVAL-LAMBDA e a))
        ((QUOTE T)
         (throw-error
          (string "EVAL unexpected case, e=" e ", a=" a ".")))))

```