This page is intentionally left blank.

Kazimir Majorinc

MATCHLESS i Aktori

Povijest Lispa 44.

Razmjena vještina
Hacklab u mami
4. siječnja 2014.

Četvrto poglavlje Hewittove dizertacije; nešto razrađeniji od ranijih opisa o kojima je bilo riječi.

Osnovne ideje MATCHLESSA.

MATCHLESS je pattern-directed jezik korišten za implementaciju PLANNER-a, sličan jezicima kao što su CONVERT i SNOBOL.

0. Jezik poštuje „Fundamental Principle of Pattern Recognition:“ tijelo procedure je uzorak koji opisuje svrhu procedure. Taj je princip razvijen dublje u PLANNERU- gdje se procedure pozivaju „on the basis of their intent.“

1. Jezik mora biti moćan ali jednostane konstrukcije moraju biti efikasno kompajlirane. Ugradnjom više znanja u program, mora biti moguće povećati efikasnost programa do granica određenih strojem.

2. Funkcije se moraju nezavisno kompajlirati

3. Funkcije ne moraju biti parsirane da bi se efikasno interpretirale.

4. Jezik je dio deduktivnog sustava.

5. Jezik mora tretirati stringove, liste, vektore, n-torke i čvorove simetrično, tako da isti program može biti primijenjen na različite podatke.

6. Jezik ne pretvara (engl. coercion) podatke automatski. To rade procedure.

7. Jezik ima samo jedan mod evaluacije. „Locatives should be generated explicitly in the same way.“

8. Jezik garantira da su sve petlje u jeziku trebaju biti pravilno ugnježđene.

Prefix operatori za identifikatore.

Kao što je uobičajeno u pattern matching jezicima, konstante kao 3, a, (a b) i (e (f g)) se uklapaju samo sa njima samima.

.x se uklapa samo samo vrijednosti identifikatora x. Ako x ima vrijednost (a 3) onda se .x uklapa samo sa (a 3).

_x se uklapa sa vrijednošću identifikatora x, ali zadržava novu vrijednost samo ako ostala uklapanja uspiju.

Jako puno varijacija na tu temu.

Ako x ima vrijednost (a 1), onda (b .x 4) → (b (a 1) 4)

! je „escape character“. (b !.x 4) → (b a 1 4).

Nakon uklapanja (c !:x d) i (c 3 a d) x ima vrijednost (3 a).

Sintaksa izraza u MATCHLESSU.

MATCHLESS koristi poljsku notaciju za pozive funkcija, ograničenu sa < i >. Znakovi (i) se koriste za označavanje liste. Znakovi [i] za vektore.

<+ 2 3> → 5

ako y ima vrijednost 4, onda < + .y 1> → 5

(.y) → (4)

(<+ .y 1> (4 a) .y) → (5 (4 a) 4).

<rest (a b c)> → (b c).

(1 {rest (a b c)} e f) → (1 b c e f)

(a b {rest (1 (e f) g)} k) → (a b (e f) g k)

<2 (a b c)> → b

<3 [(a) e 5]> → 5

<get |i| |x|> vraća lokaciju i-te komponente strukture |x|.

[.y (a b) .y] → [4 (a b) 4].

Argumenti koji se mogu izračunavati paralelno su ograničeni između |< i >.

Primjerice, <• |<+ 7 3> <+ 2 4>>, 7+3 se izračunava paralelno sa 2+4.

Hijerarhija tipova:

<?>univerzalni tip

<word> primitivni tipovi koji nisu pointeri

FALSE; sve druge vrijednosti su true u uvjetnim izrazima.

„null function call“ <> se izračunava u FALSE

CHARACTER za znakove

<NUMBER> za brojeve

<FIXED>

FIX - small fixed point number

BIG - big fixed point number

FLCAT - floating point number

<POINTER> za pointere

ATOM za atome

<STRUCTURE> za strukturirane podatke; operacija REST i selektiranje n-tog elementa su definirani za sve strukture

TUPLE alocirani na steku, brišu se izlaskom iz procedure; duljina neprojmjenjiva

VECTOR vektor, duljina promjenjiva

STRING

LIST

NODE (najopćenitija struktura, komponente se dohvaćaju uz pomoć hash-codinga, ostale strukture su potrebne samo radi efikasnosti.)

još mnogi drugi koje Hewitt nije objasnio: JUNCTION, ACTIVATION, STATE, ARC, BIND, <LOCATIVE>, LABEL, PROCESS, STACK, RING, ELEMENT-CALL, SEGMENT-CALL, SEGMENT-VALUE-CALL

Jednostavni primjeri uklapanja

Primitivnom funkcijom

<IS? |pattern| |expression|>

odlučujemo da li se |pattern| uklapa sa |expression|. Funkcija IS ima vrijednost true ako uklapanje uspije, a <>, što je isto što i FALSE ako ne uspije.

Izraz

<_ |pattern| |expression|>

je dobro definiran samo ako se |pattern| i |expression| uklapaju. Rezultat izračunavanja je tada vrijednost |expression|.

```
<prog [ a [!= atom h ] c]
        ; program u kojem su deklarirani a, h (tipa atom) i c
  <is? (_a k _h !_c) ( (l) k b o a)>>
        ; a dobija vrijednost (l)
        ; h dobija vrijednost b
        ; c dobija vrijednost (o a)
        ; funkcija IS? vraća true, jer uklapanje uspjeva

<prog [ c [!=atom h] a]
  <is? (!_c _h k _a) (a j b k q)>>
         ; c dobija vrijednost (a j)
         ;  h dobija vrijednost b
         ; a dobija vrijednost q
         ; funkcija IS? vraća true, jer uklapanje uspjeva
```

Funkcija MATCH? je nešto moćnija od funkcije IS? jer može uklapati uzorke sa uzorcima.

```
<prog [x y]
   <match ?x ?y>
  ; povezuje x i y
  <match ?x 3> >
  ; Nakon ovoga i x i y imaju vrijednost 3

<prog [x]
  <match ?x <less 5≫
  <match 6 ?x≫
; ne uspijeva jer 6 nije manje od 5.
```

```
<prog [ x y z]
    <match ?x [?y !?z]>
    <match (a b c) ?x> >
  ; y dobija vrijednost a
  ; z dobija vrijednost (b c).
```

Funkcionalne procedure:

```
<<function ['x] .x> <+ 2 2>> → <+ 2 2>
<<function [¬ "rest" x] <2 .x>> 11 21 33> →  21
jer  <2 [11 21 33]> je 21
<<function [¬"rest" 'x] .x> a <+ 3 4> c> → [a <+ 3 4> c]
<<function [x] .x> 3> →  3
<<function [x] .x> a> → a
<define f <function … ≫
```

Checker

<<function odd [x] .x> 4> ima vrijednost FALSE, jer rezultat funkcije ne zadovoljava uvjet odd (pretpostavljamo, otprije poznata funkcija).

Makroi ili makro procedure

Makroi se ekspandiraju od strane interpretera ili kompajlera. Rezultat se interpretira i kompajlira.

Aktori ili aktor procedure

Primarna razlika između funkcija i aktora je da funkcije „proizvode“ vrijednosti dok ih aktori uklapaju.

<<actor ['x] .x> <+ 2 2>> se uklapa sa <+ 2 2>

<<actor [¬ "rest" x] <2 .x>> 11 21 33> se uklapa sa 21

jer <2 [11 21 33]> se uklapa sa 21

<<actor [¬"rest" 'x] .x> a <+ 3 4> c> se uklapa sa [a <+ 3 4> c]

<<actor [x] .x> 3> se uklapa sa 3

<<actor [x] .x> a> se uklapa sa a

<define a <actor … ≫

<prog [a b c]

 <is? (k {all _a _b} _c) (k x y z) >>

  Uzorak {all _a _b} se uklapa u izraz samo ako se _a i _b uklapaju

  a dobija vrijednost (x y)

  b dobija vrijednost (x y)

  c dobija vrijednost z

```
<define palindrome
  <actor [ ]
      <either <empty>
              <declaration [x] <list _x {palindrome} .x> > > >
```

<is? <palindrome> (a 1 1 a)> → true

kraj