This page is intentionally left blank.

Kazimir Majorinc

Programski jezik LISP
Operacije i aplikacije (II)

Povijest Lispa 25.

Razmjena vještina
Hacklab u mami
16. ožujka 2013.

Tehnike korištenja Lispa za automatsko otkrivanje zanimljivih odnosa među podacima.

Autor, Edward Fredkin, pokušava razviti program za otkrivanje pravilnosti u nizu podataka, u najjednostavnijem slučaju nizu prirodnih brojeva.

Jedna od metoda kojom se želi služiti je otkrivanje podnizova. Primjerice, niz brojeva S se može „razbiti“ u podnizove

S1,S2,S3,...

pri čemu je svaki Si niz brojeva. U ovom članku se, ipak, ne bavi tom mogućnošću.

Uočava mogućnost da se nizovi brojeva opišu sa dvije operacije: (1) lokalne operacije - koje uključuju ograničeni niz brojeva, i važniju (2) operaciju sljedbenika.

Original: 0,1,8,27,64,125,216

Prva razlika: 1,7,19,37,61,91

Druga razlika: 6,12,18,24,30

Treća razlika: 6,6,6,6,6

Funkcija P1 koja provjerava jesu li svi elementi u nizu isti

$P1[x]=null[x] \vee P2[car[x]; cdr[x]]$

$P2[a;y]=null[y] \vee [equal[a;car[y]] \wedge P2[a; cdr[y]]]$

Funkcija koja originalnom nizu x pridružuje niz „generaliziranih razlika“

s[d;x] = [null [cdr[x]] → NIL, T → cons[fn[cadr[x];car[x]]; s[d; cdr[x]]]

Ako je d = minus onda je to niz prvih razlika.

Funkcija *encode* svodi niz na bitne elemente, u našem primjeru 0,1,6,6 na temelju kojih se može rekonstruirati cijeli niz.

encode[x]

=[null[cdr[x]] ∨ P1[x] → list[car[x]];T→ cons[car[x];encode[d[x]]]

Treba eksperimentirati sa različitim funkcijama d i vidjeti koja najbolje enkodira niz.

Automatizacija, korištenje LISP-a i induktivno izvođenje na nizovima

Autori, **Pivar** i **Filkenstein** uvode male promjene u odnosu na prethodni članak.

Rezultat nije tek „encode“ nego funkcija koja generira niz.

Podržavaju se i složeniji nizovi, tako da se traže razlike dok god većina razlika ne postane konstantna.

Onda od iznimaka konstruira dva niza - mjesta na kojima se iznimke pojavljuju i vrijednosti iznimaka. Na njih primjeni isti postupak.

Primjena LISPa za provjeru matematičkih dokaza.

Abrahams tvrdi da programi trivijalno mogu provjeriti dokaze iz knjiga, ako su ti dokazi dovoljno rigorozni. No, dokazi to nisu. Zato program prvo od dokaza u formi kakvi su u knjizi konstruira rigorozne dokaze.

Primjenjuje program na Principia Mathematica.

Konstrukcija rigoroznih dokaza je gotovo trivijalna. Prevodi se matematička, infix forma sa implicitnim prioritetom u S-izraze. Primjerice,

$$p \vee q \rightarrow q \vee p$$

se prevodi u

```
(IMPLIES (OR P Q) (OR Q P))
```

Pri tom prevođenju otkrivene su brojne greške u Principia Mathematica.

„Polurigorozni dokazi“ se tako prevode u nizove S-izraza, primjerice

```
(18 (IMPLIES (AND (EQUAL (GMULT A B) (GMULT A C)) (EQUAL
(GMULT A C) (GMULT D C))) (EQUAL (GMULT A B) (GMULT D C)))
NIL)
```

Abrahams uočava da se neki od operatora mogu definirati kao makroi, na primjer

```
(IMPLIES P Q)
```

se transformira u

```
(OR (NOT P) Q)
```

i da se neka pojednostavljenja i uspoređivanja izraza mogu postići evaluiranjem podizraza. Dakle, podaci se „spajaju“ sa Lispom.

Za evaluiranje izraza mu treba mogućnost „hvatanja grešaka“ koja postoji u tadašnjem Lispu, ali Abrahams predlaže daljnja unapređenja.

Program je vrlo spor - i provjera jednog jedinog koraka u dokazu zahtjevala je do trideset sekundi. (Današnji PC-i su nekoliko milijuna puta brži.)

Članak je na mjestima vrlo tehnički, ali stvarni program nije opisan sa svim potrebnim podacima za rekonstrukciju i nije objavljen.

Abrahams bi se želio riješiti S-izraza, prevođenje matematičkog izraza u S-izraze je problem.

Misli da jezik može biti efikasniji. (Jedino mjesto koga ga, možda koči je to što Lisp mora pretraživati cijelu listu da bi došao do mjesta na kojem treba izvršiti promjenu - KM).

Kompajliranje?