FUNKCIONALNO PROGRAMIRNI SISTEMI

Borut Robič, Jurij Šilc in Branko Mihovilovič
Institut 'Jožef Stefan', Ljubljana

UDK: 681.3.06

Redukcijske računalniške arhitekture zahtevajo funkcionalno programirne jezike. Funkcionalno programirni sistemi predstavljajo enega izmed možnih načinov funkcionalnega programiranja. Čeprav so šibkejši od lambda računa, pa so predvsem zaradi preglednosti in hierarhične zgradbe primernejši za praktično uporabo.

FUNCTIONAL PROGRAMMING SYSTEMS - Some new kinds of computer architectures require functional programming languages. Besides lambda style there is another functional style of programming, namely functional programming systems. These systems have properties which make them more useful than lambda style although they are not as powerful.

Uvod

Sodobne računalniške arhitekture, ki temeljijo na paralelnem procesiranju, zahtevajo programske jezike brez stranskih učinkov. Primer takega jezika je LISP, ki temelji na lambda računu. V lambda računu so izvor novih funkcij lambda izrazi skupaj s pripadajočimi pravili zamene. Z lambda izrazi lahko definiramo vse izračunljive funkcije s poljubno mnogo argumenti. To moč lambda računa pa je zaradi nepreglednosti njegovih izrazov praktično zelo težko izkoristiti.
Lambda račun je le en način ti. funkcionalnega programiranja. Drug pristop k funkcionalnemu programiranju pa predstavljajo funkcionalno programirni sistemi (v nadaljnem FP sistemi). Ti sistemi so šibkejši od lambda računa, toda mnogo preglednejši. Tudi FP sistemi ne poznajo stranskih učinkov.
Program v FP sistemu (v nadaljnem FP program) je funkcija, ki preslika en objekt v drugega. Objekt je bodisi a) nedefiniran , b) atom ali pa c) zaporedje, sestavljeno iz atomov ali zaporedij. Zaporedje je lahko tudi prazno. FP program je lahko a) osnoven, b) sestavljen na podlagi funkcijske oblike, t.j. pravila, s katerim lahko obstoječe FP programe in objekte sestavimo v nov FP program ali pa c) je definiran s pomočjo definicije.
FP program f deluje nad objektom x in vrne rezultat f:x. Pravimo, da f:x opisuje uporabo (aplikacijo) programa f nad objektom x.
Podobno kot lamda račun ima tudi FP sistem zelo enostavno semantiko.

Primerjava proceduralnega in FP programa

Če zapišemo zaporedje stavkov za izračun skalarnega produkta dveh vektorjev v enem od proceduralnih (von-Neumannovih) jezikov

```
c:=0;
for i:=1 to n do
  c := c + a[i]*b[i] ;
```

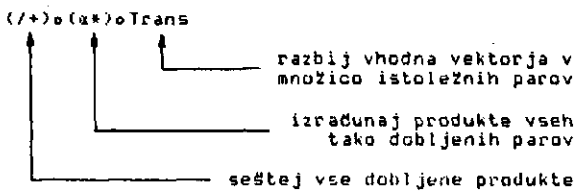
opazimo sledeče lastnosti :

a) računanje temelji na množici različnih pravil, kot so prireditev, indeksiranje , inkrementiranje, testiranje, skoki ;
b) nehierarhično zgradbo, kar pomeni, da kompleksnejši objekti niso sestavljeni iz enostavnejših. Izjema je le desni del prireditvenega stavka c+a[i]*b[i], ki je sestavljen iz dveh enostavnejših c in a[i]*b[i], pri čemer je a[i]*b[i] zopet sestavljen iz dveh enostavnejših a[i] in b[i] ;
c) čeprav problem vsebuje visoko stopnjo potencialne paralelnosti, ta ni izkoriščena. Program deluje nad posameznimi komponentami vektorjev a,b,c, ne pa nad celimi vektorji hkrati. Vzrok je v von-Neumannovi arhitekturi, ki ne omogoča paralelnega množenja istoležnih komponent vektorjev in seštevanja tako dobljenih delnih rezultatov ;
d) konstanta n zmanjšuje uporabnost programa, saj ga lahko uporabimo le nad vektorji dolžine n ;
f) program navaja imena svojih argumentov, s čimer je zmanjšana njegova uporabnost samo na vektorje z imeni a,b,c. Če pa so a,b,c imena formalnih parametrov, nujno sledi uporaba relativno zapletenih mehanizmov prenašanja parametrov ;
g) nekatere operacije so opisane s simboli, ki so porazdeljeni po programu, namesto da bi bile opisane s simbolom na enem mestu.

FP program za računanje skalarnega produkta lahko definiramo kot:

```
DEF SP = (/+)∘(α*)∘Trans
```

Sestavljajo ga tri osnovne funkcije: seštevanje + , množenje * in transpozicija Trans ter tri funkcijske oblike: vstavljanje / , uporaba nad vsemi elementi α in kompozitum ∘ .
Program ima sledeč pomen :



Uporabo (aplikacijo) funkcije f nad argumentom x zapišemo z f:x . Tedaj lahko izvajanje zgornjega programa nad argumentom, ki je par vektorjev <<1,2,3>,<6,5,4>>, opišemo tako:

```
SP:< <1,2,3>,<5,5,4> > =
= (/+) (α*) Trans:< <1,2,3>,<6,5,4> > =
= (/+) (α*):( Trans:< <1,2,3>,<6,5,4> > ) =
= (/+) (α*):< <1,6>,<2,5>,<3,4> > =
= (/+):( (α*):< <1,6>,<2,5>,<3,4> > ) =
= (/+):< *:<1,6>, *:<2,5>, *:<3,4> > =
= (/+):< 6,10,12 > =
= +:< 6, +:< 10,12 > > =
= +:< 6, 22 > )=
= 28
```

Opazimo sledeče lastnosti programa SP :

a) računanje temelji le na dveh pravilih: na pravilu uporabe funkcije nad objektom in pravilu razvoja funkcijske oblike ;
b) program je hierarhičen, kar pomeni, da je sestavljen na podlagi pravil, ki enostavnejše dele programa sestavljajo v kompleksnejše. Tudi ti enostavnejši deli so sestavljeni na podlagi istih pravil, ali pa so elementarni (osnovne funkcije ali objekti);
c) je statičen, kar pomeni, da lahko njegov pomen razumemo že, če ga beremo z desne v levo , seveda ob predpostavki, da poznamo pomen osnovnih funkcij in funkcijskih oblik
d) deluje nad celimi vektorji hkrati
e) uporabimo ga lahko nad poljubno dolgimi vektorji, saj ne vsebuje nobene konstante ;
f) ne poimenuje svojih argumentov , ker deluje le nad njihovimi vrednostmi .

Funkcionalno programirni sistem

FP sistem sestavljajo:

- množica objektov O
- množica F funkcij, ki preslikavajo objekte v objekte
- operator : aplikacije funkcije nad objektom
- množica E funkcijskih oblik, s katerimi sestavljamo obstoječe funkcije in objekte v nove funkcije
- množica D definicij, ki nekaterim funkcijam iz F prirejajo nova imena

Objekt

Objekt je lahko:

- nedefiniran, in ga označimo z ⊥ ;
- atom ;
- zaporedje $\langle x_1, x_2, \ldots, x_n \rangle$, kjer je x_i objekt.

Zaporedje, ki ne vsebuje nobenega elementa označimo z ∅. Zaporedje je nedefinirano, torej enako ⊥ , če vsebuje vsaj en nedefiniran element.
Torej izbrana množica atomov A in pravilo sestavljanja v zaporedje določata množico objektov O. Za množico atomov navadno izberemo množico končnih nizov črk, cifer ali posebnih znakov. Atoma T in F uporabljamo za opis logičnih vrednosti True in False.

Primeri objektov : ⊥ 1.5 ∅ AB3 <AB,1,2.3>
<A,<<1>,C>,D> <A,⊥>

Prvi in zadnja objekta so nedefinirani.

Aplikacija

FP sistem ima samo eno operacijo, ki jo imenujemo aplikacija. Če je f funkcija in x objekt, potem aplikacijo f nad x zapišemo z f:x . Tu je f operator, x pa operand aplikacije f:x . f:x je objekt, ki je rezultat izvršitve funkcije f nad objektom x.

Primeri aplikacije : +:<1,2>=3 2:<A,B,C>=B
tl:<A,B,C>=<B,C>

Funkcije

Vse funkcije preslikavajo objekte v objekte in tudi ohranjajo nedefiniranost, torej velja f:⊥ = ⊥ za vsako funkcijo f .

Funkcija je lahko :

- osnovna (primitivna) ;
- sestavljena ;
- definirana .

Osnovne funkcije podaja že sam FP sistem. Sestavljeno funkcijo dobimo na podlagi obstoječih osnovnih, sestavljenih ali definiranih funkcij, če nad njimi uporabimo eno ali več pravil sestavljanja tj. funkcijskih oblik. Če je funkcija osnovna, je tudi sestavljena. Definirana funkcija je funkcija f, za katero v množici definicij obstaja definicija oblike DEF f ≡ E(f,g,...,h), pri čemer je desni del definicije sestavljena funkcija.
Razlikujemo dva primera, ko je f:x = ⊥ . Če se računanje f:x konča v končno mnogo korakih in vrne vrednost ⊥ , je f nedefinirana pri x. Torej se izračun f nad x konča, vendar ne vrne smiselnega rezultata. Če pa se izračun f nad x ne konča v končno mnogo korakih, zopet velja f:x = ⊥ , vendar tokrat pravimo, da je f pri x neustavljiva .
Žaljeno je, da FP sistem vsebuje široko uporaben nabor osnovnih funkcij. V naslednjih primerih je opisanih nekaj osnovnih funkcij, stroge definicije pa so podane v dodatku A.

'Selekcija' je funkcija i, ki vrne i-ti element danega zaporedja. Če takega elementa ni, vrne vrednost ⊥ . Primeri selekcije:

3:<A,B,C>=C 3:<A,B>=⊥ 1:<<1,2>,3>=<1,2>

'Rep' je funkcija tl, ki izloči prvi element zaporedja in vrne njegov preostanek . Primeri:

tl:<A,B,C>=<B,C> tl:<A>=∅ tl:∅ = ⊥

'Atom' ugotovi, če je dani objekt atom. Primeri atoma: Atom:B=T Atom:<A,B>=F

'Enakost' je funkcija eq, ki ugotavlja enakost dveh elementov zaporedja. Primeri enakosti:

eq:<A,B>=F eq:<<C,D>,<C,D>>=T

'Praznost' je funkcija null, ki ugotovi, če je zaporedje prazno. Primer:

null:∅ =T null: ⊥=⊥ null:<A,B>=F

'Obrat' je funkcija reverse, ki obrne vrstni red elementov zaporedja. Primer obrata:

reverse:<A,B,<C,D>> = <<C,D>,B,A>

'Transpozicija' je funkcija trans, ki istoležne elemente zaporedij združi v nova zaporedja. Primer transpozicije:

```
trans:<<A,B,C>,<D,E,F>,<G,H,J>,<K,L,M>> =
      = <<A,D,G,K>,<B,E,H,L>,<C,F,J,M>>
```

Funkcijske oblike

Funkcijska oblika je izraz, ki označuje funkcijo. Le-ta je odvisna od drugih funkcij ali objektov - parametrov funkcijske oblike. Npr., če sta f in g funkciji, je fog funkcijska oblika, ki ji pravimo kompozitum funkcij f in g. f in g sta parametra te funkcijske oblike. fog je funkcija, katere pomen opišemo takole: (fog):x = f:(g:x). Funkcijske oblike imajo lahko za parametre tudi objekte. če je x objekt, je $\overline{x}$ funkcija, ki nad vsakim definiranim objektom zavzame vrednost x.
V naslednjih primerih je opisanih nekaj funkcijskih oblik, njihove stroge definicije pa so podane v dodatku B.

Primer kompozituma funkcij 'rep' in 'obrat' :

```
  tl∘reverse:<A,B,C> = tl:<C,B,A> = <B,A> .
```

Konstrukcija funkcij f in g je funkcija [f,g]. Funkcijo [f,g] uporabimo nad objektom tako, da nad njim istočasno uporabimo funkciji f in g ter dobljena rezultata združimo v zaporedje. Primer konstrukcije :

```
        [1,2]:<A,B,C> = <<B,C>,B>
```

Levo vstavljanje funkcije f opišemo z /f. Primer levega vstavljanja funkcije + :

```
    /+:<1,2,3>  =  +:<1,/+:<2,3>> =
=   +:<1,+:<2,/+:<3>>>  =  +:<1,+:<2,3>> =
=   +:<1,5>  =  6
```

Istočasno uporabo funkcije f nad vsemi elementi zaporedja opišemo z αf . Primer uporabe funkcije * nad vsemi:

```
α*:<<1,2>,<3,4>> = <*:<1,2>,*:<3,4>> = <2,12>
```

Pogojno izvršitev funkcij f in g glede na funkcijo p opišemo s (p -->f ; g). Rezultat izvršitve funkcije (p -->f ; g) nad objektom x je odvisen od vrednosti p:x. če je le-ta enaka T (True), je rezultat f:x, sicer pa g:x. Primer:

```
  (Atom -->id; reverse) : <A,B,C> = <C,B,A> .
```

Definicije

Definicija v FP sistemu je izraz oblike

```
DEF l <==> r
```

kjer je na levi strani nek še neuporabljen simbol, na desni pa funkcija. Leva stran definicije lahko vstopa kot parameter v desni del, s čimer omogočimo tudi rekurzivne definicije.

Primeri definicij :

```
DEF u <==> [1,tl,2]

Npr. u:<A,B,C> = [1,tl,2]:<A,B,C>= <A,<B,C>,B>

DEF last1 <==> 1∘reverse
```

Funkcija last1 izbere zadnji element zaporedja.
Npr. last1:<1,2,3> = 1∘rev:<1,2,3>=1:<3,2,1>=3

```
DEF last <==> nullotl --> 1 ; lastotl
```

definira isto funkcijo kot last1, tokrat na rekurziven način.

```
last:<A,B> = (nullotl --> 1; lastotl):<A,B>
           = lastotl:<A,B>
           = last:(tl:<A,B>)
           = last:<B>
           = (nullotl --> 1; lastotl):<B>
           = 1:<B>
           = B
```

Lahko se zgodi, da kaka definicija definira funkcijo, ki ni ustavljiva. Množica definicij je dobro oblikovana , če ne vsebuje nobene definicije z enako levo in desno stranjo.

Semantika FP sistema

FP sistem je določen, če poznamo:

- množico atomov A ;
- množico osnovnih funkcij P ;
- množico funkcijskih oblik F ;
- dobro oblikovano množico definicij D .

Za to, da bi poznali semantiko danega FP sistema, moramo za vsako funkcijo f in objekt x vedeti, kako uporabimo f nad x. Obstajajo štiri možnosti :

- f je osnovna funkcija ;
- f je sestavljena ;
- f je definirana ;
- za f ne velja nobena od zgornjih točk .

Uporabo osnovne funkcije pojasnjuje že sam FP sistem. Uporaba sestavljene funkcije temelji na razvoju funkcijske oblike, na podlagi katere je nastala, ter na uporabi parametrov te funkcijske oblike. Definirano funkcijo uporabimo tako, da uporabimo desni del njene definicije. če za f ne velja nobena od zgornjih točk, velja f:x = ⊥ za vsak objekt x.

FP sistemi kot programirni jeziki

FP sisteme lahko smatramo za programirne jezike. Osnovne funkcije in funkcijske oblike so osnovni stavki danega programirnega jezika. Izbira različnih osnovnih funkcij in funkcijskih oblik omogoča izgradnjo programirnih jezikov različnih zmogljivosti. Množica definicij je programska knjižnica.
Funkcija f je program, objekt x je vsebina pomnilnika (vhodni podatek), f:x pa vsebina pomnilnika po izvršitvi programa f nad x.
Dokazovanje pravilnosti programov v konvencionalnih (von Neumannovih) jezikih je zapleteno. Z definiranjem algebre programov, prirejene FP sistemom, je podana osnova za mehanično ugotavljanje in dokazovanje lastnosti programov FP sistema. Algebra programov omogoča tudi uporabniškemu programerju, da ugotavlja in dokazuje lastnosti svojih programov, ne da bi zahtevala od njega pretirano teoretično znanje. Prednost takega dokazovalnega sistema je tudi ta, da programer uporablja pri dokazovanju isti jezik, v katerem so napisani programi. Torej pri dokazovanju lastnosti programov ni potrebno preiti na nek višji, zunanji logični nivo.
Jedro algebre programov sestavlja množica zakonov in izrekov, ki povedo, kdaj je neka funkcija ekvivalentna drugi funkciji.

Spremenljivke algebre programov so funkcije ustreznega FP sistema, operacije algebre pa so funkcijske oblike tega sistema. Tako je npr. [f,g]oh izraz v algebri programov, v katerem so spremenljivke f,g,h poljubne funkcije FP sistema. Zakon v algebri programov ima npr. obliko [f,g]oh <==> [foh,goh]. Ta zakon pravi, da za poljubne funkcije f,g,h sistema velja, da je funkcija na levi strani ekvivalentna funkciji na desni.
Nekaj zakonov algebre programov:

[f,g]oh <==> [foh,goh]

(p-->f;g)oh <==> (poh-->foh;goh)

ho(p-->f;g) <==> (p-->hof;hog)

[f,(p-->g;h)] <==> (p --> [f,g];[f,h])

Dodatek A

Pri naslednjih definicijah uporabljamo izraze

$$p_1 \text{-->} e_1 ; p_2 \text{-->} e_2 ; \ldots p_{k-1} \text{-->} e_{k-1} ; e_k$$

katerih pomen je sledeč:

če p_1 tedaj e_1 sicer pa
če p_2 tedaj e_2 sicer pa
.....
če p_{k-1} tedaj e_{k-1} sicer pa e_k .

V definicijah so x, x_i, y, y_i, z, z_i objekti.

Selekcija

1:x <==> x=<x1,...,xn> --> x1 ; ⊥
in za vsako pozitivno število s:
s:x <==> x=<x1,...,xn> in n≥s --> xs ; ⊥

Rep

tl:x <==> x=<x1> --> ∅ ;
x=<x1,...,xn> in n≥2 --> <x2,...,xn> ; ⊥

Identiteta id:x <==> x

Atom

atom:x <==> x je atom --> T ; x ni ⊥ --> F ; ⊥

Enakost

eq:x <==> x=<y,z> in y=z --> T ;
x=<y,z> in y≠z --> F ; ⊥

Praznost

null:x <==> x=∅ --> T ; x ni ⊥ --> F ; ⊥

Obrat

rev:x <==> x=∅ --> ∅ ;
x=<x1,...,xn> --> <xn,...,x1> ; ⊥

Distribucija z leve

disl:x <==> x=<y,∅> --> ∅ ;
x=<y,<z1,...,zn>> -->
<<y,z1>,...,<y,zn>> ; ⊥

Distribucija z desne

disr:x <==> x=<∅,y> --> ∅ ;
x=<<z1,...,zn>,y> -->
<<z1,y>,...,<zn,y>> ; ⊥

Dolžina

len:x <==> x=<x1,..,xn> --> n ; x=∅ --> 0 ; ⊥

Vsota,Razlika,Produkt,Kvocient

+:x <==> x=<y,z> in y,z števili --> y+z ; ⊥
-:x <==> x=<y,z> in y,z števili --> y-z ; ⊥
*:x <==> x=<y,z> in y,z števili --> y*z ; ⊥
div:x <==> x=<y,z> in y,z števili --> y/z ; ⊥

kjer y/0 = ⊥

Inkrementiranje,Dekrementiranje

ad:x <==> x število --> x+1 ; ⊥
sb:x <==> x število --> x-1 ; ⊥

Transpozicija

trans:x <==> x=<∅,...,∅> --> ∅ ;
x=<x1,...,xn> --> <y1,...,ym> ; ⊥

kjer $x_i = \langle x_{i1}, x_{i2}, \ldots, x_{im} \rangle$ in
$y_j = \langle x_{1j}, x_{2j}, \ldots, x_{nj} \rangle$, $1 \le i \le n$, $1 \le j \le m$

Konjunkcija,Disjunkcija,Negacija

and:x <==> x=<T,T> -->T ; x=<T,F> ali x=<F,T>
ali x=<F,F> --> F ; ⊥
or:x <==> x=<T,T> ali x=<T,F>
ali x=<F,T> --> T ; x=<F,F> --> F ; ⊥
not:x <==> x=<T> --> F ; x=<F> --> T ; ⊥

Levo in desno pripenjanje

apndl:x <==> x=<y,∅> --> <y> ;
x=<y,<z1,...,zn>> --> <y,z1,...,zn> ; ⊥
apndr:x <==> x=<∅,y> --> <y> ;
x=<<z1,...,zn>,y> --> <z1,...,zn,y> ; ⊥

Desna selekcija, Desni rep

1r:x <==> x=<x1,...,xn> --> xn ; ⊥
2r:x <==> x=<x1,...,xn> in n≥2 --> xn-1 ; ⊥
itd.
tlr:x <==> x=<x1> --> ∅ ; x=<x1,...,xn> in
n≥2 --> <x1,...,xn-1>> ; ⊥

Leva in desna rotacija

rotl:x <==> x=∅ --> ∅ ; x=<x1> --> <x1> ;
x=<x1,...,xn> in n≥2 -->
<x2,...,xn,x1> ; ⊥

rotr:x <==> x=∅ --> ∅ ; x=<x1> --> <x1> ;
x=<x1,...,xn> in n≥2 -->
<xn,x1,x2,...,xn-1 > ; ⊥

Dodatek B

Kompozicija (fog):x <==> f:(g:x)

Konstrukcija [f1,...,fn]:x<==><f1:x,...,fn:x>

Pogoj (p --> f; g):x <==> (p:x)=T --> f:x ;
(p:x)=F --> g:x;⊥

Konstanta $\bar{x}$:y <==> y=⊥ --> ⊥ ; x

Levo in desno vstavljanje

/f:x <==> x=<x1> --> x1 ;
x=<x1,...,xn>
in n≥2 --> f:<x1, /f:<x2,...,xn> > ; ⊥

\f:x <==> x=<x1> --> x1 ;
x=<x1,...,x >
in n≥2 --> f:<\f<x1,...,xn-1 >, xn >; ⊥

Uporaba nad vsemi

af:x <==> x=∅ --> ∅ ;
x=<x1,...,xn> --> <f:x1,...,f:xn> ; ⊥

Dvojiško v eniško

(bu f x):y <==> f:<x,y> kjer je x objekt

Zanka

(while p f):x <==> p:x=T-->(while p f):(f:x) ;
p:x=F-->x ; ⊥

Literatura

[1] W.B.Ackerman: 'Data Flow Languages', Computer, Special Issue on Data Flow Systems, Vol.15, No.2, February 1982

[2] J. Backus: 'Can Programming Be Liberated from von Neumann Style? A Functional Style and Its Algebra of Programs', Comm. of the ACM, Vol 21, No.8, August 1978

[3] J. Backus: 'The Algebra of Functional Programs: Function Level Reasoning, Linear Equations and Extended Definitions', Formalization of Programming Concepts, Lecture Notes in Computer Science, 107

[4] J.Šilc,B.Robič,B.Mihovilović: 'Podatkovno vodene računalniške arhitekture', Posvetovanje in seminarji Informatica '85, Nova Gorica, September 1985