nedeljainformatike v2.0

Funkcionalni pristup konstrukciji kompajlera

Nikola Jovanović

Matematička gimnazija
Nedelja informatike v2.0

17. decembar 2015.

Šta je FP?

- Programska paradigma koja tretira izračunavanje kao evaluaciju matematičkih funkcija
- Deklarativni jezici definišu izraz koji opisuje rešenje, nema naredbi niti redosleda izvršavanja
- Teorijska osnova: λ račun
- Haskell*, ML, Common Lisp, Erlang (funkcionalni jezici)
- F#, OCaml, Scala (hibridi)

Čiste funkcije

- Funkcija u FP $\neq$ funkcija u IP
- Nema sporednih efekata
- Referencijalna transparencija (rezultat funkcije zavisi samo od ulaza)
- Sve je izraz (if then else)
- Lakoća verifikacije, modularnost, konciznost, paralelizacija …

Stroga tipiziranost

- Statička provera tipova (tipovi svih promenljivih poznati pre pokretanja)
- Inferencija tipova (Hindli-Milnerov algoritam)
- Veća pouzdanost

Evaluacione strategije

- call-by-value (ML)
  - striktna strategija
  - evaluira nepotrebne argumente
- call-by-name
  - nestriktna strategija
  - evaluira potrebne argumente više puta
- call-by-need (Haskell)
  - call-by-name + memoizacija
  - lenja evaluacija
  - rad sa beskonačnim strukturama

```
f 2 (1/0)
take 6 [1..]
```

Rekurzija i pattern matching

- Rekurzija kao glavni mehanizam izračunavanja u nedostatku iteracije
- Traženje uzoraka (pattern matching)

```
fib 1 = 1
fib 2 = 1
fib n = fib (n-1) + fib (n-2)
```

- Izuzetno korisno pri radu sa listama

```
[], [x], (x:xs)
```

List comprehension

- Sažet zapis liste nalik na skupovnu notaciju
  `[(x, y) | x<-[1, 2] , y<-[1..10] , x+y == 10]`

Lambde

- In-place anonimne funkcije
  `(\x -> x + 1)`

Funkcije višeg reda i currying

- Funkcije su u FP vrednosti prve klase (mogu da se dodele promenljivoj ili proslede kao argument drugoj funkciji)
- Funkcije višeg reda su funkcije koje kao argumente uzimaju druge funckije: izvod, integral
- Currying (ne postoje funkcije sa više argumenata)
  `saberi x y = x + y` -> funkcija sa jednim argumentom (x) koja vraća funkciju sa jednim argumentom (y) koja vraća vrednost
- Parcijalna aplikacija
  `saberi_sa_5 = saberi 5`

map i filter

- Transformacije podataka
- `map f l`: funkcija višeg reda koja transformiše listu `l` primenom funkcije `f` na pojedinačne elemente
  `map imena [3, 4, 5] = [''tri'', ''cetiri'', ''pet'']`
- `filter p l`: funkcija višeg reda koja transformiše listu `l` primenom predikata `p` na pojedinačne elemente
  `filter neparni [3, 4, 5] = [3, 5]`

Par primera

- ```
  sum . takeWhile(<1000) . filter odd . map (^2) $ [1..]
  ```
- ```
  quick [] = []
  quick (p:xs) = (quick manji) ++ [p] ++ (quick veci)
  where
       manji = filter (< p) xs
       veci = filter (>= p) xs
  ```

Gde je FP?

- U industriji (Facebook, Google, Intel, Microsoft, Jane Street)
- U drugim programskim jezicima (hibridi, koncepti FP u imperativnim jezicima)
- U knjigama o FP (`http://learnyouahaskell.com/`)

xkcd: Haskell

Šta je kompajler?

- Programski prevodilac
- Izvorni i ciljni jezik
- Kreiranje izvršnog fajla

Komponente

- Leksička analiza
- Sintaksna analiza
- Optimizacija*
- Generisanje koda

Ideja

- Kompajler u Haskell-u
- Koncepti FP su primenljivi na problem konstrukcije kompajlera
- Transformacije podataka, rekurzija, pattern matching …
- Ulazni jezik: prost imperativni jezik PV
- Ciljna mašina: virtuelna stek mašina (stek + memorija, implementirana u C++)

Ulazni jezik PV

- `x, y` ∈ `Var` (promenljive, alfanumerički stringovi koji počinju slovom)
- `n` ∈ `Num` (brojevi, celobrojne vrednosti)
- `opa` ∈ `OPa` (aritmetičke operacije)
  `opa ::= + | *`
- `opb` ∈ `OPb` (logičke operacije)
  `opb ::= and | or`
- `opr` ∈ `OPr` (relacije)
  `opr ::= > | < | =`

Ulazni jezik PV (2)

- a ∈ Aexp (aritmetički izrazi)
  `a ::= n | x | a opa a | (a)`
- b ∈ Bexp (logički izrazi)
  `b ::= true | false | !b | a opr a | b opb b | (b)`
- S ∈ Stmt (naredbe)
  `S ::= skip | x := a | S;S | if b then S else S | while b do S`
- Standardan prioritet operacija
- Komentari

Virtuelna stek mašina

- Instrukcije za rad sa stekom i memorijom
  - **PUSH n** – dodaje celobrojnu vrednost n na vrh steka
  - **POP** – izbacuje element sa vrha steka (**S1**)
  - **LOAD s** – dodaje vrednost promenljive s na vrh steka
  - **STORE s** – postavlja vrednost promenljive s na **S1**, i izbacuje **S1** sa steka

Virtuelna stek mašina (2)

- Aritmetičke i logičke instrukcije
  - **ADD** – uvećava drugi element na steku (**S2**) za **S1** i izbacuje **S1**
  - **SUB** – umanjuje **S2** za **S1** i izbacuje **S1**
  - **MUL** – množi **S2** sa **S1** i izbacuje **S1**
  - **OR** – postavlja **S2** na logičku disjunkciju **S2** i **S1** i izbacuje **S1**
  - **AND** – postavlja **S2** na logičku konjukciju **S2** i **S1** i izbacuje **S1**
  - **NOT** – vrši logičku negaciju nad **S1**

Virtuelna stek mašina (3)

- Skokovi
  - **JMP n** – bezuslovni skok na instrukciju n
  - **JZ n** – skok na instrukciju n ako je **S1** jednak nuli
  - **JP n** – skok na instrukciju n ako je **S1** veći od nule
  - **JM n** – skok na instrukciju n ako je **S1** manji od nule
- **NOP** – prazna instrukcija
- **HALT** – zaustavlja izvršenje programa

Korak 0: izvorni kod

```
b := 1;
if (b > 3)
then { a := 0 }
else { a := b + 1 }
```

Regularni izrazi

- **Alfabet** je konačan skup Σ čije elemente zovemo **simbolima**. Na primer, skup malih slova engleske abecede $\Sigma = \{a, b, c, \dots, z\}$ je validan primer alfabeta dok skup prirodnih brojeva $\mathbb{N} = \{1, 2, 3, \dots\}$ ne može biti alfabet, jer nije konačan.
- String (nenulte) dužine n nad alfabetom Σ je uredjena n-torka elemenata iz Σ. Za $n = 0$ postoji jedinstveni string, tzv **prazan string** koji se označava sa ε. Sa Σ^* označavamo skup svih stringova nad skupom Σ.

Regularni izrazi (2)

- $\emptyset$ je regularan izraz
- ε je regularan izraz
- Svaki simbol a iz skupa Σ je regularan izraz
- Ukoliko su R i S regularni izraz tada je i RS regularni izraz
- Ukoliko su R i S regularni izraz tada je i $(R|S)$ regularni izraz
- Ukoliko je R regularni izraz tada je i R^* regularni izraz

Regularni izrazi (3)

- Regularni izrazi su jezik za predstavljanje šablona stringova. Pogledajmo pravila uparivanja stringova iz Σ^* sa regularnim izrazima:
- nijedan string ne odgovara $\emptyset$
- u odgovara ε ukoliko $u = \varepsilon$
- u odgovara $a \in \Sigma$ ukoliko $u = a$
- u odgovara $R|S$ ukoliko u odgovara R ili u odgovara S
- u odgovara RS ukoliko se može izraziti kao konkatenacija dva stringa vw gde v odgovara R i w odgovara S
- u odgovara R^* ukoliko je $u = \varepsilon$ ili se u može izraziti kao konkatenacija dva ili više stringova od kojih svaki odgovara R

Regularni izrazi (4)

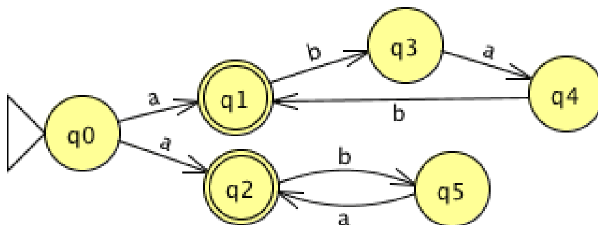
- Regularni jezik odredjen regularnim izrazom R nad Σ definišemo kao:

$$L(r) \stackrel{\text{def}}{=} \{u \in \Sigma^* \mid u \text{ odgovara } R\}$$

- Uz **determinističke konačne automate** možemo efikasno da odredimo da li string odgovara nekom regularnom izrazu tj. da li pripada jeziku

Konačni automati

- Matematički model izračunavanja
- Stanja: $q_0, q_1, q_2, q_3, q_4, q_5$
- Ulazni simboli: a, b
- Prelazi: $q_0 \xrightarrow{a} q_1$, $q_0 \xrightarrow{a} q_2$, $q_1 \xrightarrow{b} q_3 \ldots$
- Početno stanje: q_0
- Prihvatajuća stanja: q_1, q_2



Konačni automati (2)

- Kažemo da konačni automat **prihvata** string S ako se S sastoji od alfabeta ulaznih simbola tog automata i postoje neka (ne obavezno različita) stanja $q_0, q_1, q_2, \ldots, q_n$, takva da je q_0 početno stanje, q_n prihvatajuće stanje, i postoje prelazi oblika: $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \xrightarrow{a_3} \ldots \xrightarrow{a_n} q_n$.
- Nedeterministički automat: jedno stanje, jedan simbol -> više prelaza
- Deterministički automat: jedno stanje, jedan simbol -> jedan prelaz

Leksička analiza

- Prevodi kod izvornog jezika u niz značajnih stringova (tokena)
- Leksički analizator / lekser
- Svaki token ima regularni izraz i prioritet
- Skup DKA prepoznaje tokene

Korak 1: tokeni

```
[ID "b",OP_ASS,NUM 1,SEMICOLON,KWRD_IF,PAREN_LEFT,
ID "b",OP_ORD GT,NUM 3,PAREN_RIGHT,KWRD_THEN,BRKT_LEFT,
ID "a",OP_ASS,NUM 0,BRKT_RIGHT,KWRD_ELSE,BRKT_LEFT,
ID "a",OP_ASS,ID "b",OP_PLUS,NUM 1,BRKT_RIGHT,EOF]
```

Sintaksna analiza

- Proces pretvaranja tokena u **apstraktno sintaksno stablo**
- Sintaksni analizator (parser)
- U osnovi je **nedeterministički pushdown automat** (automat koji koristi stek)

Još malo teorije

- **Formalni jezik** predstavlja skup stringova simbola za koje važe neka pravila (koja zadaje **formalna gramatika** u vidu **produkcionih pravila**, pravila transformacija stringova)
- Definicija jezika PV (konteksno slobodna gramatika)
- Terminali i neterminali
- **Levo rekurzivno pravilo produkcije** je pravilo produkcije oblika **A -> A B**, gde je **A** neterminalni simbol, a **B** sekvenca terminalnih i/ili neterminalnih simbola gramatike

Rekurzivni spust

- Tehnika parsiranja
- Medjusobno rekurzivne procedure, svaka implementira jedno pravilo produkcije
- Moguće parsiranje u jednom prolazu, ako jezik pripada klasi LL(k)
- Potrebno elimisati levu rekurziju

Modifikacije jezika

- Problem:
  `a ::= n | x | a opa a | (a)`
- Hijerarhija neterminala:
  ```
  a3 ::= (a1) | n | x
  a2 ::= a3 * a2 | a3
  a1 ::= a2 + a1 | a2
  ```
- Jezik je sada LL(1) i možemo parsirati rekurzivnim spustom

Korak 2: sintaksno stablo

```
StmtBlock (Ass "b" (Aexp1' (Aexp2' (Number 1))))
(SingleStmt (IfElse (Bexp1'(Bexp2' (ParensB(Bexp1'
(Bexp2' (Relation (Aexp1' (Aexp2' (Id "b"))) GT
(Aexp1' (Aexp2' (Number 3)))))))))(SingleStmt (Ass
"a" (Aexp1' (Aexp2' (Number 0)))))(SingleStmt (Ass
 "a" (Add (Aexp2' (Id "b")) (Aexp1' (Aexp2'(Number 1
 ))))))))
```

Korak 2: sintaksno stablo (graphViz)

StmtBlock
Ass'b'
SingleStmt
Aexp1'
IfElse
Aexp2'
Bexp1'
SingleStmt
SingleStmt
Number1
Bexp2'
Ass'a'
Ass'a'

Generisanje koda

- Generator koda (!)
- Transformacija sintaksnog stabla u kod u ciljnom jeziku

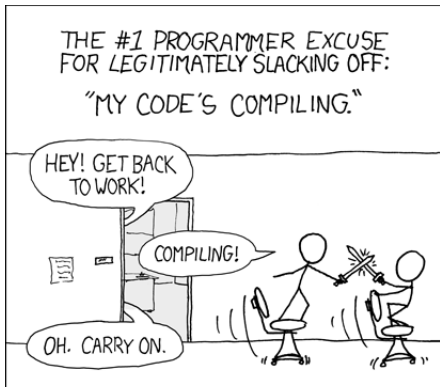
Korak 3: kod za VSM

```
PUSH 1
STORE b
LOAD b
PUSH 3
SUB
JP 9
PUSH 0
JMP 10
PUSH 1
JZ 14
PUSH 0
STORE a
JMP 18
LOAD b
PUSH 1
ADD
STORE a
```

Korak 4: izvršenje na VSM

```
Program se zaustavio na instrukciji br. 18.
-----------------
Stanje memorije:
Velicina programa = 18
Broj promenljivih = 2
Program counter = 18
Pokazivac na vrh steka = 0
b = 1
a = 2
-----------------
```

xkcd: Compiling



https://github.com/Rand0mUsername/pv-vsm

https://goo.gl/QVrT0Q