Sveučilište u Zagrebu
Fakultet elektrotehnike i računarstva
Zavod za elektroniku, mikroelektroniku, računalne i inteligentne sustave

Prevođenje programskih jezika

Ak. God. 2014/2016

2. Laboratorijska vježba

Tema druge laboratorijske vježbe je sintaksna analiza. Vaš zadatak je programski ostvariti sintaksni analizator za programski jezik *PJ* koji je opisan u uputi za prvu laboratorijsku vježbu. Preciznije, trebate implementirati parser za LL(1)-gramatiku jezika *PJ* zadanu kasnije u ovoj uputi.

Ulaz i izlaz programa sintaksnog analizatora

Vaš program će na ulaz dobiti niz uniformnih znakova dobiven leksičkom analizom nekog programa iz jezika PJ u istom formatu kao izlaz iz leksičkog analizatora u prvoj laboratorijskoj vježbi. Tijekom ove upute će se ilustracije radi pojavljivati odsječci programskog koda u jeziku *PJ*, ali sintaksni analizator neće dobivati na ulaz takav programski kod, već rezultat leksičke analize nad njim.

Program na standardni izlaz (stdin) treba ispisati generativno stablo zadanog niza uniformnih znakova po gramatici koja je dana u nastavku upute. Podsjetnik iz UTR-a: generativno stablo prikazuje na koji način je neki niz nastao iz početnog nezavršnog znaka gramatike. Preciznija definicija ispisa dana je uz primjer kasnije u uputi.

Gramatika jezika *PJ*

Sintaksna pravila jezika *PJ* zadana su sljedećom LL(1)-gramatikom:

```
<program> ::= <lista_naredbi>
<lista_naredbi> ::= <naredba> <lista_naredbi>
<lista_naredbi> ::= $
<naredba> ::= <naredba_pridruzivanja>
<naredba> ::= <za_petlja>
<naredba_pridruzivanja> ::= IDN OP_PRIDRUZI <E>
<za_petlja> ::= KR_ZA IDN KR_OD <E> KR_DO <E> <lista_naredbi> KR_AZ
<E> ::= <T> <E_lista>
<E_lista> ::= OP_PLUS <E>
<E_lista> ::= OP_MINUS <E>
<E_lista> ::= $
<T> ::= <P> <T_lista>
<T_lista> ::= OP_PUTA <T>
<T_lista> ::= OP_DIJELI <T>
<T_lista> ::= $
<P> ::= OP_PLUS <P>
<P> ::= OP_MINUS <P>
<P> ::= L_ZAGRADA <E> D_ZAGRADA
<P> ::= IDN
<P> ::= BROJ
```

Početni nezavršni znak je `<program>`. Umjesto znaka ε, u produkcijama se koristi znak `$`.

Ostatak ovog poglavlja ukratko objašnjava gramatiku i nije nužan da biste riješili laboratorijsku vježbu. Prirodnim jezikom, dane produkcije definiraju da je program u jeziku *PJ* niz nijedne ili više naredbi (i prazan program je sintaksno ispravan). Nadalje, postoje dvije vrste naredbi: naredba pridruživanja i `za`-petlja. U naredbi pridruživanja, vrijednost nekog izraza pridružuje se identifikatoru (varijabli). S druge strane, `za`-petlja ima zaglavlje koje definira identifikator koji se koristi kao brojač i granične vrijednosti te listu naredbi koje se u petlji ponavljaju.

Preostali dio gramatike odnosi se na izraze (kako produkcije ne bi bile preduge, koriste se uobičajene skraćenice za sintaksne cjeline izraza: E - expression, T - term, P - primary). Produkcije osiguravaju da zagrade i unarni `+` i `-` imaju najviši prioritet. Sljedeći po prioritetu su operatori `*` i `/`, a binarni operatori `+` i `-` imaju najniži prioritet.

Produkcije za izraze imaju tipični oblik za LL(1)-gramatiku izraza u infiksnoj notaciji koji možete vidjeti i u udžbeniku. Nezavršni znak koji definira pravila za operatore nižeg prioriteta (npr. `<E>`) generira nezavršni znak za operatore višeg prioriteta (npr. `<T>`) i proizvoljan nastavak koji se sastoji od operatora i novih podizraza iste ili niže razine prioriteta (npr. `<E_lista>`).

Primjer

U ovom poglavlju prikazana je sintaksna analiza sljedećeg jednostavnog *PJ* programa:

```
n = 5
rez = 0
za i od n do n+5
  rez = rez - i*i + i/3
az
```

Leksičkom analizom dobili bismo

```
IDN 1 n
OP_PRIDRUZI 1 =
BROJ 1 5
IDN 2 rez
OP_PRIDRUZI 2 =
BROJ 2 0
KR_ZA 3 za
IDN 3 i
KR_OD 3 od
IDN 3 n
KR_DO 3 do
IDN 3 n
OP_PLUS 3 +
BROJ 3 5
IDN 4 rez
OP_PRIDRUZI 4 =
IDN 4 rez
OP_MINUS 4 -
IDN 4 i
```

```
OP_PUTA 4 *
IDN 4 i
OP_PLUS 4 +
IDN 4 i
OP_DIJELI 4 /
BROJ 4 3
KR_AZ 5 az
```

i to bi bio ulaz u rješenje ove vježbe. U obliku datoteke, ovaj primjer ulaza nalazi se ovdje.

U skladu s navedenim sintaksnim pravilima, program je sintaksno ispravan i grafički prikaz njegovog generativnog stabla možete vidjeti ovdje. Kako bi se izlaz vaših rješenja mogao usporediti s očekivanim izlazom, sintaksni analizator treba ispisati generativno stablo na standardni izlaz (stdout) *preorder* obilaskom. Za zadani primjer, očekivani izlaz može se vidjeti ovdje. Format ispisa možete intuitivno shvatiti iz primjera, a u nastavku je dana preciznija definicija. Stablo se obilazi od korijena prema listovima, pri čemu se prvo ispiše oznaka korijena stabla i nakon toga se ispisuju podstabla s lijeva na desno. Oznaka svakog čvora stabla zapisuje se u vlastiti redak prefiksirana nizom razmaka čiji broj odgovara dubini čvora u stablu (dubina korijena je 0, dubina djece korijena je 1, itd.). Za listove stabla koji predstavljaju neki uniformni znak iz ulaza, ispisuje se uniformni znak, redak u kojem se uniformni znak nalazio u izvornom programu i pripadna leksička jedinka odvojeni po jednim razmakom (dakle ispisuje se redak iz ulaza prefiksiran odgovarajućim brojem razmaka ovisno o dubini u stablu - pogledajte primjer). Za listove stabla označene znakom $, ispisuje se samo znak $ prefiksiran odgovarajućim brojem razmaka.

Napomena za bolje razumijevanje zadatka i gradiva predmeta (nije nužna za rješavanje laboratorijske vježbe): U generativnom stablu izraza s desne strane naredbe pridruživanja u `za`-petlji vide se dvije značajne činjenice (koje vrijede općenito za danu gramatiku). Prvo, stablo **ispravno** prikazuje prioritet operatora (množenje i dijeljenje imaju viši prioritet od zbrajanja i oduzimanja). Drugo, stablo **pogrešno** prikazuje asocijativnost operatora; aritmetički operatori su lijevo asocijativni, dok dana gramatika prikazuje desno asocijativne operatore. Ovo je tipična situacija za LL(1)-gramatike izraza jer se lijeva asocijativnost u gramatici postiže lijevom rekurzijom što nije moguće u LL(1)-gramatici (postupkom izbacivanja lijeve rekurzije mijenja se asocijativnost operatora). Ipak, ovaj problem može se lako riješiti transformacijom stabla čime se nećemo baviti u ovoj laboratorijskoj vježbi.

Sintaksne greške

Za razliku od prve laboratorijske vježbe, u drugoj laboratorijskoj vježbi moguće su (sintaksne) greške. Sintaksni analizator ne smije raditi nikakav oporavak od pogreške već jednostavno treba ispisati niz
`err <uniformni_znak> <br_retka> <leksicka_jedinka>`
i završiti izvođenje. Uniformni znak koji treba ispisati je onaj na kojem je greška detektirana, a ostale informacije (broj retka i leksička jedinka) su oni koji su zadani u ulazu uz taj uniformni znak. Nadalje, ako se sintaksna greška detektira kada je ulaz prazan (tj. svi uniformni znakovi iz ulaza su već obrađeni), onda treba ispisati
`err kraj`

Na primjer, ako se u gornjem primjeru izbaci operator pridruživanja u drugom retku (dakle drugi redak bi bio "`rez 0`"), očekivani ispis sintaksnog analizatora je:
`err BROJ 2 0`

Izlaz je takav jer analizator nakon identifikatora u naredbi pridruživanja očekuje operator pridruživanja, a na ulazu se nalazi uniformni znak `BROJ` koji predstavlja konstantu 0 iz izvornog programa.

Ako se pak umjesto operatora pridruživanja izbaci ključna riječ `az` koja završava `za`-petlju, očekivani ispis je:

```
err kraj
```

Izlaz je takav jer parser očekuje ključnu riječ `az` koja će završiti petlju ili novu naredbu (ova greška je ekvivalentna greški “missing } at end of file” koju dobijete od kompilatora kada negdje zaboravite zatvoriti blok u jezicima iz C sintaksne obitelji). Ova greška nije posebno vezana uz petlju; npr. ako program završava operatorom pridruživanja, opet bi očekivani ispis bio `err` kraj (naravno, ako prije toga nema drugih grešaka).

Važna napomena: Ovaj način signalizacije sintaksne pogreške onemogućuje ispis generativnog stabla tijekom rada parsera jer je nemoguće unaprijed znati hoće li doći do pogreške. Zato generativno stablo treba izgraditi u memoriji i onda ga ispisati na kraju rada programa ako nije bilo sintaksne pogreške.

PRIMIJENI skupovi za produkcije gramatike

Kako ne biste morali programski ili ručno određivati PRIMIJENI skupove produkcija gramatike, oni su dani u nastavku. PRIMIJENI skup je zadan nakon produkcije u vitičastim zagradama i od produkcije je odvojen znakom “=”. U skupu su navedeni uniformni znakovi i znak ⊥ (znak za kraj ulaza), odvojeni razmacima.

```
<program> ::= <lista_naredbi> = {IDN KR_ZA ⊥}
<lista_naredbi> ::= <naredba> <lista_naredbi> = {IDN KR_ZA}
<lista_naredbi> ::= $ = {KR_AZ ⊥}
<naredba> ::= <naredba_pridruzivanja> = {IDN}
<naredba> ::= <za_petlja> = {KR_ZA}
<naredba_pridruzivanja> ::= IDN OP_PRIDRUZI <E> = {IDN}
<za_petlja> ::= KR_ZA IDN KR_OD <E> KR_DO <E> <lista_naredbi> KR_AZ = {KR_ZA}
<E> ::= <T> <E_lista> = {IDN BROJ OP_PLUS OP_MINUS L_ZAGRADA}
<E_lista> ::= OP_PLUS <E> = {OP_PLUS}
<E_lista> ::= OP_MINUS <E> = {OP_MINUS}
<E_lista> ::= $ = {IDN KR_ZA KR_DO KR_AZ D_ZAGRADA ⊥}
<T> ::= <P> <T_lista> = {IDN BROJ OP_PLUS OP_MINUS L_ZAGRADA}
<T_lista> ::= OP_PUTA <T> = {OP_PUTA}
<T_lista> ::= OP_DIJELI <T> = {OP_DIJELI}
<T_lista> ::= $ = {IDN KR_ZA KR_DO KR_AZ OP_PLUS OP_MINUS D_ZAGRADA ⊥}
<P> ::= OP_PLUS <P> = {OP_PLUS}
<P> ::= OP_MINUS <P> = {OP_MINUS}
<P> ::= L_ZAGRADA <E> D_ZAGRADA = {L_ZAGRADA}
<P> ::= IDN = {IDN}
<P> ::= BROJ = {BROJ}
```

Kako početi

Kako je zadana LL(1)-gramatika, preporuča se da parser ostvarite ili tabličnim LL(1)-parserom (potisni automat) koji ste učili na predavanjima i opisan je u udžbeniku u poglavlju 3.5 ili rekurzivnim spustom koji je načelno opisan u udžbeniku za UTR u poglavlju 3.1.3. U udžbeniku za UTR nisu eksplicitno definirani PRIMIJENI skupovi koji igraju ključnu ulogu u prediktivnom parsiranju od vrha prema dnu. Prema tome, pseudokod koji je dan u udžbeniku (ako se odlučite na rekurzivni spust) trebate prilagoditi novim znanjima stečenim na PPJ-u.

Za svaki nezavršni znak gramatike definirali biste jednu funkciju koja obrađuje taj znak. Ovisno o

uniformnom znaku na ulazu, jednoznačno je moguće odrediti koju produkciju treba primijeniti ili je li došlo do sintaksne pogreške. Primjena produkcije znači usporedba uniformnih znakova s ulaza s uniformnim znakovima na desnoj strani produkcije i pozivanje odgovarajućih potprograma nezavršnih znakova s desne strane produkcije.

Izgradnja generativnog stabla

Bez obzira koji pristup odaberete, ključan zadatak parsera je izgraditi generativno stablo niza uniformnih znakova sa ulaza. Ako odaberete pristup zasnovan na potisnom automatu, na stog uz znakove treba stavljati i čvorove generativnog stabla. Pri zamjeni znaka s vrha stoga sa znakovima s desne strane odgovarajuće produkcije, za svaki znak s desne strane treba napraviti novi čvor stabla i povezati te znakove s čvorom stabla znaka na vrhu stoga (tj. znaka s lijeve strane te produkcije koja se primjenjuje).

Slično vrijedi i za rekurzivni spust, samo se tamo čvorovi stabla prenose kao argumenti funkcija koje predstavljaju pojedine nezavršne znakove gramatike.

Predaja rješenja na SPRUT

Rješenje se predaje kao zip arhiva u kojoj se nalazi sav potreban kod za prevođenje/izvođenje. Vrijede ista pravila kao na UTR-u (specifično, u **Javi nemojte koristiti pakete**). Ulazna točka za rješenja u Javi je razred SintaksniAnalizator, a za Python datoteka SintaksniAnalizator.py. Za ostale jezike je organizacija koda proizvoljna (naravno, mora postojati točno jedna funkcija/metoda main itd.). Vremensko ograničenje za izvođenje sintaksnog analizatora je dvije minute (što je daleko više od očekivanog vremena izvođenja).