

Sveučilište u Zagrebu
Fakultet elektrotehnike i računarstva

Samostalni studentski projekt iz predmeta
„Prevođenje programskih jezika“

Zadatak broj 2008

Zagreb, prosinac 2010.

Samostalni studentski projekt iz predmeta „Prevođenje programskih jezika“

Student : Deu5

JMBAG : 0036xxxxxx

Zadatak 2008 : Programski ostvariti parser koji **parsira tehnikom prednosti operatora**.

Sadržaj

Zadatak 2008	2
1. Uvod	4
1.1. Parsiranje	4
1.2. Parsiranje od dna prema vrhu	4
1.3. Ostvarenje parsiranja od dna prema vrhu	5
1.4. Parsiranje tehnikom prednosti operatora	5
1.5. Određivanje <i>relacija prednosti</i> na temelju zadane gramatike	7
1.6. Određivanje <i>relacija prednosti</i> na temelju prednosti operatora i asocijativnosti	8
1.7. Algoritam parsiranja tehnikom prednosti operatora	9
2. Ostvarenje	10
2.1. Tablica prednosti operatora	10
2.2. Gramatika	12
2.3. Parser	13
2.4. Parser s gradnjom generativnog stabla	15
2.5. Izvođenje programa	19
3. Zaključak	24
4. Literatura	25

1. Uvod

1.1. Parsiranje

Parsiranje je analiza niza, tj. postupak prepoznavanja ulaznog niza i gradnja pripadajućeg generativnog stabla na temelju produkcija zadane kontekstno neovisne gramatike. Postoji parsiranje od vrha prema dnu i parsiranje od dna prema vrhu.

1.2. Parsiranje od dna prema vrhu

Parsiranje od dna prema vrhu gradnju pripadajućeg generativnog stabla započinje gradnjom listova generativnog stabla. Listovi generativnog stabla označavaju završne znakove zadane gramatike. Gradnja generativnog stabla nastavlja se primjenom desnih strana produkcija gramatika na već izgrađene čvorove sve do korijena stabla. Kod parsiranja od dna prema vrhu znakovi niza w se čitaju također slijedno slijeva nadesno. Pošto u svakom koraku smanjuje se (reducira) ili ostaje isti broj znakova u međunizu, produkcije se u ovom postupku parsiranja nazivaju *redukcije*. Ako je korijen stvorenog generativnog stabla označen sa početnim nezavršnim znakom zadane gramatike, niz w je u zadanom jeziku. Redukcija se bira na temelju pročitanih znakova niza. Učinkovitost parsiranja od dna prema vrhu ovisi o preciznosti kojom je moguće odrediti redukciju koju je potrebno primijeniti u određenom koraku samog parsiranja.

Uobičajeno je da se kod gradnje generativnog stabla koristi obrnuti postupak generiranja niza w zamjenom krajnje desnog nezavršnog znaka. Parsiranje od dna prema vrhu započinje ulaznim nizom završnih znakova w koji se slijedno čitaju slijeva nadesno. Podnizovi završnih i nezavršnih znakova međuniza koji su jednaki desnoj strani određene produkcije zamjenjuju se nezavršnim znakovima lijevih strana odgovarajućih produkcija zadane gramatike. Ako se cijeli niz završnih znakova ulaznog niza w uspješno zamijeni sa početnim nezavršnim znakom gramatike, niz w je u zadanom jeziku te parsiranje je uspješno završilo. Tijek parsiranja se može opisati na slijedeći način:

$$w \stackrel{*}{\leftarrow} \alpha x \stackrel{*}{\leftarrow} \langle S \rangle$$

gdje je w niz završnih znakova, αx je međuniz koji se dobije postupkom generiranja zamjenom krajnje desnog nezavršnog znaka, a $\langle S \rangle$ je početni nezavršni znak zadane gramatike. Međuniz αx sastavljen je od niza završnih znakova x i niza završnih i nezavršnih znakova α . Neka je $\langle B \rangle \rightarrow \beta$ produkcija zadane gramatike, a $\delta\beta y$ je međuniz dobiven redukcijama niza w . Onda se na podniz znakova β dobivenog međuniza $\delta\beta y$ primjenjuje se redukcija $\langle B \rangle \rightarrow \beta$ na slijedeći način:

$$\delta\beta y \leftarrow \delta \langle B \rangle y$$

Pri čemu se podniz β koji je jednak desnoj strani produkcije $\langle B \rangle \rightarrow \beta$ naziva *uzorak za zamjenu*.

1.3. Ostvarenje parsiranja od dna prema vrhu

Pomoću *potisnog stoga* i *ulaznog spremnika* je moguće ostvariti parsiranje od dna prema vrhu. Na stog se spremaju i završni i nezavršni znakovi gramatike koje sve parser i čita. U ulaznom nizu nalazi se niz w koji se parsira, a čita se slijedno slijeva nadesno. Na početku rada na stogu se nalazi oznaka kraja stoga ∇ , a u ulaznom spremniku se nalazi ulazni niz w . Parser prvo čita krajnje lijevi znak a niza $w = w'a$. Na temelju pročitane ulaznog znaka a i znaka na vrhu stoga, parser odlučuje o primjeni jedne od četiri akcija: *Pomakni*, *Reduciraj*, *Prihvati* i *Odbaci*.

- Akcija *Pomakni* stavlja pročitani znak a na vrh stoga i pomiče kazaljku za čitanje ulaznog niza znakova w za jedan znak udesno.
- Akcija *Reduciraj* se primjenjuje ako na vrhu stoga postoji uzorak za zamjenu. Uzimaju se znakovi desne strane produkcije s vrha stoga, a umjesto njih se na vrh stoga stavlja nezavršni znak lijeve strane produkcije. Ova akcija zahtjeva uspoređivanje znakova stoga i znakova desnih strana produkcija.
- Akcija *Prihvati* se primjenjuje ako se pročita oznaka kraja niza iz ulaznog spremnika \perp , a na stogu se nalazi početni nezavršni znak gramatike $\langle S \rangle$ i oznaka dna stoga ∇ , i tada parser prihvća niz w .
- Akcija *Odbaci* se primjenjuje ako se nije moguća daljnja gradnja generativnog stabla. Odbacuje se niz w jer nije u jeziku. Akcija *Odbaci* primjenjuje se ako nema definirane akcije za ulazni znak i znak vrha stoga i tijekom akcije *Reduciraj* ako znakovi sa stoga nisu jednaki znakovima nijedne desne strane produkcija.

1.4. Parsiranje tehnikom prednosti operatora

„Kontekstno neovisna gramatika je *operatorska* ako i samo ako produkcije nisu ε -produkcije i ni na jednom mjestu desne strane produkcije dva susjedna znaka nisu nezavršna.“¹

Kada imamo operatorsku gramatiku, onda generiranjem bilo kojeg međuniza postupkom generiranja niza zamjenom krajnje desnog nezavršnog znaka ima slijedeći oblik :

$$\beta_0 \alpha_1 \beta_1 \cdots \alpha_n \beta_n$$

pri tome vrijedi da su α_i završni znakovi gramatike, a niz β_i je prazni niz ε ili niz kojeg čini isključivo jedan nezavršni znak.

¹ Preuzeto iz Srbljić, Siniša : Prevođenje programskih jezika, 1. izdanje, Element, Zagreb, 2007. ; 130. str.

Pri postupku parsiranja tehnikom prednosti operatora koristi se *relacija prednosti* koja se definira za sve završne znakove zadane gramatike, uključujući oznaku dna stoga ∇ i oznaku kraja niza \perp . Relacije prednosti se definiraju na slijedeći način:

Relacija	Značenje
$a \leftarrow b$	Prednost a je manja od prednosti b
$a \leftrightarrow b$	a i b su jednake prednosti
$a \rightarrow b$	Prednost a je veća od prednosti b

Pri tome je moguće da u istom jeziku za dva završna znaka zadane gramatike a i b vrijedi $a \rightarrow b$ i $b \rightarrow a$, a isto tako je moguće da za dva završna znaka gramatike ne vrijedi nijedna relacija.

Relacije prednosti zapisuju se u obliku tablice (Tablica 1.) koja se koristi pri parsiranju niza.

	a	$+$	$*$	\perp
a		\rightarrow	\rightarrow	\rightarrow
$+$	\leftarrow	\rightarrow	\leftarrow	\rightarrow
$*$	\leftarrow	\rightarrow	\rightarrow	\rightarrow
∇	\leftarrow	\leftarrow	\leftarrow	

Tablica 1. Primjer tablice relacija prednosti

Parser radi parsiranje prednosti operatora tako da uspoređuje završni znak na vrhu stoga i znak u ulaznom nizu i na temelju relacije prednosti za ta dva znaka odlučuje o primjeni akcije *Pomakni* ili *Reduciraj*.

- Akcija *Pomakni* se primjenjuje ako je prednost završnog znaka na vrhu stoga jednaka ili manja od prednosti ulaznog znaka.
- Akcija *Reduciraj* se primjenjuje ako je prednost završnog znaka na vrhu stoga veća od prednosti ulaznog znaka.

Za određivanje *uzorka za zamjenu* parser uzima jedan po jedan znak sa stoga sve do završnog znaka koji ima manju prednost od prednosti prethodno uzetog znaka. U generiranom međunizu koriste se *relacije prednosti* da se odrede granice *uzorka za zamjenu*. *Uzorak za zamjenu* ograđen je *relacijama prednosti* na slijedeći način:

$$a_0 \leftarrow a_1 \leftrightarrow a_2 \leftrightarrow a_3 \leftrightarrow \dots \leftrightarrow a_{n-1} \leftrightarrow a_n \leftrightarrow a_{n+1}$$

gdje je niz završnih znakova $a_1 a_2 a_3 \dots a_n$ *uzorak za zamjenu*.

1.5. Određivanje *relacija prednosti* na temelju zadane gramatike²

Neka su $\langle A \rangle$, $\langle B \rangle$ i $\langle C \rangle$ nezavršni znakovi operatorske gramatike. Za završne znakove l i d određuju se relacije prednosti na slijedeći način:

1) $l \leftarrow d$

Ako je na desnoj strani produkcije završni znak l neposredno ispred nezavršnog znaka $\langle C \rangle$:

$$\langle A \rangle \rightarrow \dots l \langle C \rangle \dots$$

i ako nezavršni znak $\langle C \rangle$ generira međuniz u kojemu je krajnje lijevi *završni* znak znak d , onda se za znakove l i d definira *relacija prednosti* $l \leftarrow d$.

2) $l \leftrightarrow d$

Ako je na desnoj strani produkcije završni znak l neposredno ispred završnog znaka d :

$$\langle A \rangle \rightarrow \dots l d \dots$$

ili je između njih samo jedan nezavršni znak:

$$\langle A \rangle \rightarrow \dots l \langle B \rangle d \dots$$

onda se za znakove l i d definira *relacija prednosti* $l \leftrightarrow d$.

3) $l \rightarrow d$

Ako je na desnoj strani produkcije nezavršni znak $\langle B \rangle$ neposredno ispred završnog znaka d :

$$\langle A \rangle \rightarrow \dots \langle B \rangle d \dots$$

i ako nezavršni znak $\langle B \rangle$ generira međuniz u kojemu je krajnje desni *završni* znak jest znak l , onda se za znakove l i d definira *relacija prednosti* $l \rightarrow d$.

² Algoritam je preuzet iz Sribljčić, Siniša : Prevođenje programskih jezika, 1. izdanje, Element, Zagreb, 2007. ; 133. – 134. str.

1.6. Određivanje *relacija prednosti* na temelju prednosti operatora i asocijativnosti³

Relacije prednosti moguće je odrediti izravno na temelju prednosti binarnih operatora koji se koriste u izrazima:

- 1) Ako binarni operator x ima veću prednost od binarnog operatora y , onda se definiraju relacije prednosti na slijedeći način:
 $x \rightarrow y$ i $y \leftarrow x$.
- 2) Ako su binarni operatori x i y jednake prednosti i ako su operatori lijevo asocijativni, onda se definiraju relacije prednosti na slijedeći način:
 $x \rightarrow y$ i $y \rightarrow x$.
- 3) Ako su binarni operatori x i y jednake prednosti i ako su operatori desno asocijativni, onda se definiraju relacije prednosti na slijedeći način:
 $x \leftarrow y$ i $y \leftarrow x$.
- 4) Za sve operatore x definiraju se slijedeće relacije prednosti:

$$\begin{array}{lll} x \leftarrow a & x \leftarrow (& x \rightarrow \perp \\ a \rightarrow x & (\leftarrow x & x \rightarrow) \\ & & \nabla \leftarrow x \end{array}$$

Znak a je oznaka završnog znaka varijable.

- 5) Nadalje, definiraju se slijedeće relacije prednosti:

$$\begin{array}{lll} (\leftrightarrow) & \nabla \leftarrow (& \nabla \leftarrow a \\ (\leftarrow (& a \rightarrow \perp &) \rightarrow \perp \\ (\leftarrow a & a \rightarrow) &) \rightarrow) \end{array}$$

Znak a je oznaka završnog znaka varijable.

- a) Operator potenciranja \uparrow je najveće prednosti i desno je asocijativan.
- b) Operatori množenja $*$ i dijeljenja $/$ jednake su prednosti i lijevo su asocijativni. Njihova vrijednost je manja od prednosti operatora \uparrow , a veća je od prednosti operatora $+$ i $-$.
- c) Operatori zbrajanja i oduzimanja jednake su prednosti, lijevo su asocijativni i imaju manju prednosti od operatora \uparrow , $*$ i $/$.

³ Algoritam je preuzet iz Srbljić, Siniša : Prevođenje programskih jezika, 1. izdanje, Element, Zagreb, 2007. ; 135. – 136. str.

Na temelju zadanih prednosti operatora i navedenih pravila, izgradi se slijedeća tablica relacija prednosti:

	+	-	*	/	↑	a	()	⊥
+	→	→	←	←	←	←	←	→	→
-	→	→	←	←	←	←	←	→	→
*	→	→	→	→	←	←	←	→	→
/	→	→	→	→	←	←	←	→	→
↑	→	→	→	→	←	←	←	→	→
a	→	→	→	→	→			→	→
(←	←	←	←	←	←	←	↔	
)	→	→	→	→	→			→	→
∇	←	←	←	←	←	←	←		

1.7. Algoritam parsiranja tehnikom prednosti operatora⁴

Parsiranje niza w tehnikom prednosti operatora izvodi se primjenom slijedećeg algoritma:

Postavi KAZALJKU da pokazuje na krajnje lijevi znak niza $w⊥$;

```

dok(1)
{
    ako ( ( ZnakNaVrhuStoga==∇ ) && ( Ulaz==⊥ ) )
        Prihvati();
    inače
    {
        //Neka je x završni znak koji je najbliži vrhu stoga
        //Neka je y znak na koji pokazuje KAZALJKA

        ako ((x ← y) || (x ← y))          //Pomakni
        {
            StaviNaStog(y);
            Pomakni KAZALJKU na slijedeći znak ulaznog niza;
        }
        inače ako (x → y)                  //Reduciraj
        {
            UzmiSVrhaStoga(1 znak);
            dok ( !( ZnakNaVrhuStoga ← UzetiZnak ) )
                UzmiSVrhaStoga(1 znak);
        }
        inače
            Odbaci();
    }
}

```

⁴ Algoritam je preuzet iz Srbljić, Siniša : Prevođenje programskih jezika, 1. izdanje, Element, Zagreb, 2007. ; 137. str.

2. Ostvarenje

Zadatak je programski ostvariti parser koji parsira tehnikom prednosti operatora. Programsko rješenje pisano je u Python 3.1.3 programskom jeziku, a sastoji se od simulatora parsera i dodatnih funkcija za čitanje tekstualnih datoteka koje sadrže gramatiku, tablicu prednosti i ulazne nizove. Program *parser.py* je program koji samo parsira šest ulaznih nizova bez gradnje generativnog stabla. Program *parser_generativno_stablo.py* uzima 3 ispravna niza, parsira ih i gradi pripadajuće generativno stablo.

Koncept rada parsera temelji se na simulatoru koji simulira rad parsera. Simulator učitava tablicu prednosti operatora iz tekstualne datoteke *tablicaPrednosti.txt*, završne znakove i produkcije operatorske gramatike iz tekstualne datoteke *gramatika.txt*. Ulazni niz prima preko tekstualnih datoteka *UlazniNiz.txt*, *UlazniNiz2.txt*, *UlazniNiz3.txt*, *UlazniNiz4.txt*, *UlazniNiz5.txt* i *UlazniNiz6.txt*. U ulaznim nizovima *UlazniNiz.txt*, *UlazniNiz2.txt* i *UlazniNiz3.txt* i *UlazniNiz4.txt* zadani su ispravni nizovi, a u nizovima *UlazniNiz5.txt* i *UlazniNiz6.txt* zadani su pogrešni nizovi.

2.1. Tablica prednosti operatora

Tablica prednosti operatora (Slika 1.) sastoji se od završnih znakova abecede, oznake dna stoga ('@') i oznake kraja ulaznog niza ('\$'). Radi jednostavnosti u programu se koristi slijedeći zapis oznaka prednosti:

Relacija	Značenje
$a < b$	Prednost a je manja od prednosti b
$a = b$	a i b su jednake prednosti
$a > b$	Prednost a je veća od prednosti b

Tablica Prednosti				
	a	+	*	\$
a	_	>	>	>
+	<	>	<	>
*	<	>	>	>
@	<	<	<	_

Slika 1. Primjer tablice prednosti operatora.

U prvom stupcu tablice prednosti operatora nalaze se završni znakovi gramatike i oznaka dna stoga ('@'). U drugom retku, odmah ispod naziva „Tablica Prednosti“ nalaze se završni znakovi gramatike i oznaka kraja niza ('\$'). Svi znakovi su odvojeni razmakom, a u drugom retku na samom početku stavljena su tri razmaka zbog lakšeg dohvaćanja tablice. Prednosti koje se zapisuju određene su prvim znakovima u redcima, nakon drugog retka, i znakovima iz drugog retka (Slika 2.)

Tablica Prednosti				
a	+	*	\$	
a	_	>	>	>
+	<	>	<	>
*	<	>	>	>
@	<	<	<	_

Slika 2. Određenje zapisa prednosti.

Tablica prednosti učitava se pomoću funkcije *procitaj_tablicu()*, koja kao ulazni parametar prima naziv tekstualne datoteke koja sadrži tablicu prednosti zadane gramatike.

```
def procitaj_tablicu(path):
    rjecnik = {}
    datoteka = open(path, 'r')
    linije = datoteka.read().split('\n')
    for i in range(1, len(linije[1:]) + 1):
        for brojac in range(1, len(linije[2])):
            if ((linije[i][brojac] != ' ') and (linije[i][0] != ' ')):
                rjecnik[linije[i][0], linije[1][brojac]] = linije[i][brojac]
    return rjecnik
```

Funkcija procitaj_tablicu.

Funkcija *procitaj_tablicu()* čita tekstualnu datoteku u kojoj je napisana tablica prednosti i sprema zapise tablice prednosti u rječnik⁵, a svaki ključ određuje jedan par znakova. Zapis u rječnik je slijedeći:

$$rječnik [(x, y)] = z$$

pri tome *x* označava jedan od znakova stoga (znakovi u prvom stupcu zapisa tablice prednosti), *y* označava znak ulaznog niza (znakovi u drugom retku zapisa tablice prednosti) a *z* je određeni znak prednosti ('<', '>' ili '=').

Primjer zapisa (Slika 2.) je slijedeći:

$$rječnik [(+, *)] = < .$$

Unutar ovog projekta postoje dvije tablice prednosti za dvije gramatike: *tablicaPrednosti.txt* i *tablicaPrednosti2.txt*. Zapis tablica je slijedeći:

<i>tablicaPrednosti.txt</i>				
Tablica Prednosti				
a	+	*	\$	
a	_	>	>	>
+	<	>	<	>
*	<	>	>	>
@	<	<	<	_

⁵ Dictionary

tablicaPrednosti 2.txt										
Tablica Prednosti										
	+	-	*	/		a	()	\$	
+	>	>	<	<	<	<	<	>	>	
-	>	>	<	<	<	<	<	>	>	
*	>	>	>	>	<	<	<	>	>	
/	>	>	>	>	<	<	<	>	>	
	>	>	>	>	<	<	<	>	>	
a	>	>	>	>	>			>	>	
(<	<	<	<	<	<	<	=		
)	>	>	>	>	>			>		
@	<	<	<	<	<	<	<			

2.2. Gramatika

Gramatika je zapisana u tekstualnoj datoteci *gramatika.txt* koja sadrži redom završne znakove, parove znakova za koje vrijedi svojstvo *Reduciran znakom* i produkcije gramatike.

```
Gramatika
a + *
E$,T+,T$,P+,P*,P$,a+,a*,a$
1 E->E+T
2 E->T
3 T->T*P
4 T->P
5 P->a
```

Slika 3. Primjer tekstualne datoteke gramatike.

U prvom retku zapisani su završni znakovi, odnosno operatori gramatike. Ispod toga u drugom retku zapisani su skupovi znakova za koje vrijedi relacija *Reduciran Znakom* i to na slijedeći način:

ako vrijedi *ReduciranZnakom*(A, x) onda je zapis u datoteci: Ax .

Zapisi su odvojeni zarezima da bi se lakše dohvatili u program.

Nakon relacija *Reduciran Znakom* slijede prijelazi gramatike. Dohvaćanje operatora, relacija *Reduciran Znakom* i prijelaza vrši se pomoću nekoliko funkcija koje dohvaćaju iz datoteke. Funkcija *citaj_zavrsne()* prima datoteku *gramatika.txt*, a vraća listu završnih znakova, funkcija *citaj_reduciranZnakom()* prima datoteku *gramatika.txt*, a vraća listu relacija *Reduciran Znakom*, a funkcija *citaj_prijelaze()* prima također datoteku *gramatika.txt*, a vraća listu prijelaza.

```
def citaj_zavrsne(path):
    lista = []
    temp = open(path, 'r')
    tmp = temp.read().split('\n')
    for i in range(0, len(tmp[1])):
        if(tmp[1][i] != ' '):
            lista.append(tmp[1][i])
    return lista
```

Funkcija citaj_zavrsne.

```
def citaj_reduciranZnakom(path):
    temp = open(path, 'r')
    tmp = temp.read().split('\n')
    return tmp[2].split(',')
```

Funkcija citaj_reduciranZnakom.

```
def citaj_prijelaze(path):
    lista = []
    temp = open(path, 'r')
    tmp = temp.read().split('\n')
    for i in range(3, len(tmp[1:]) + 1):
        lista.append(tmp[i][2:])
    return lista
```

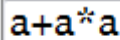
Funkcija citaj_prijelaze.

Unutar ovog projekta postoje dvije gramatike: *gramatika.txt* i *gramatika2.txt*. Zapis gramatika je slijedeći:

<i>gramatika.txt</i>	<i>gramatika2.txt</i>
Gramatika	Gramatika 2
a + *	+ * a ()
E\$,T+,T\$,P+,P*,P\$,a+,a*,a\$	E\$,T+,T),T\$,P+,P*,P),a+,a*,a)
1 E→E+T	1 E→E+T
2 E→T	2 E→T
3 T→T*P	3 T→T*P
4 T→P	4 T→P
5 P→a	5 P→(E)
	6 P→a

2.3. Parser

Parser je napravljen funkcijom *simulator()* kao simulator parsera. Funkcija parsera prima tekstualnu datoteku u kojoj je bez razmaka u jednom retku zapisan niz koji se želi parsirati.



Slika 4. Primjer tekstualne datoteke ulaznog niza.

Parser koristi algoritam parsiranja opisan sljedećim pseudokodom⁶:

Postavi KAZALJKU da pokazuje na krajnje lijevi znak niza w^{\perp} ;

```
dok(1)
{
    ako ( ( ZnakNaVrhuStoga== $\nabla$  ) && ( Ulaz== $\perp$  ) )
        Prihvati();
    inače
    {
        //Neka je x završni znak koji je najbliži vrhu stoga
        //Neka je y znak na koji pokazuje KAZALJKA

        ako (( $x \leftarrow y$ ) || ( $x \leftarrow y$ ))          //Pomakni
        {
            StaviNaStog(y);
            Pomakni KAZALJKU na sljedeći znak ulaznog niza;
        }
        inače ako ( $x \rightarrow y$ )                    //Reduciraj
        {
            UzmiSVrhaStoga(1 znak);
            dok ( !( ZnakNaVrhuStoga  $\leftarrow$  UzetiZnak ) )
                UzmiSVrhaStoga(1 znak);
        }
        inače
            Odbaci();
    }
}
```

Opisani algoritam ostvaren je na sljedeći način:

```
while(zastavica):
    if( (stog[-1] == '@') and (ulazni[kazaljka] == '$') ):
        print("Akcija prihvati() ")
        print( "Ulazni niz " + str(ulazni) + " se prihvaca" )
        zastavica = 0
        break
    else:
        if( (rjecnik_prednosti[(stog[-1], ulazni[kazaljka])] == '<') or
            (rjecnik_prednosti[(stog[-1], ulazni[kazaljka])] == '=') ):
            print("Ulazni znak: " + str(ulazni[kazaljka]))
            print("Akcija pomakni() ")
            print("Stog = " + str(stog))
```

⁶ Algoritam je preuzet iz Sribljčić, Siniša : Prevođenje programskih jezika, 1. izdanje, Element, Zagreb, 2007. ; 137. str.

```

stog.append(ulazni[kazaljka])

kazaljka += 1

elif(rjecnik_prednosti[(stog[-1], ulazni[kazaljka])] == '>' ):
    print("Ulazni znak: " + str(ulazni[kazaljka]))
    print("Akcija reduciraj()")
    print("Stog = " + str(stog))

    znakSaVrhaStoga = stog.pop()
    print("Znak sa vrha stoga = " + str(znakSaVrhaStoga))
    duljina_uzorka = len(uzorakZaZamjenu)

    while( rjecnik_prednosti[stog[-1], znakSaVrhaStoga] != '<' ):
        znakSaVrhaStoga = stog.pop()
        print("Izbacujem: " + str(znakSaVrhaStoga))
    else:
        print("Ulazni niz " + str(ulazni) + " se odbacuje.")
        zastavica = 0
        break

```

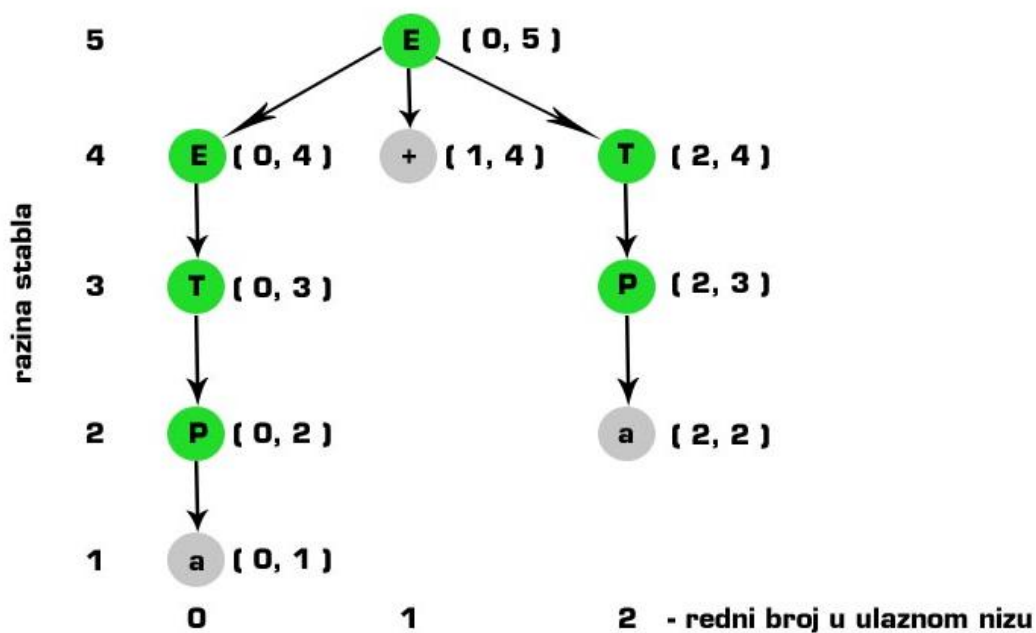
Program provjerava ako je ulaz završni znak ulaznog niza '\$' i ako je na stogu oznaka dna stoga '@' to znači da je u prihvatljivom stanju. Ako nije kraj ulaznog niza, provjerava se ulazni znak manje prednosti od znaka na vrhu stoga, ako je onda je redukcija. Provjera se vrši pristupanjem u rječnik u kojem se nalaze prednosti operatora. Za ključ pri pristupu uzimaju se znak sa stoga i ulazni znak. Ako nije nijedna prednost određena s tim znakovima dogodila se pogreška u ulaznom nizu i niz se odbacuje.

2.4. Parser s gradnjom generativnog stabla

Gradnja generativnog stabla vrši se pomoću relacije *Reduciran Znakom*. Uzorak se uzima sa stoga i ako vrijedi *Reduciran Znakom* za znak uzet sa stoga i ulazni znak onda se vrši zamjena znaka uzetog sa stoga. Stablo se sprema u rječnik sa ključevima rednog broj u ulaznom nizu i razine stabla.

$$rjecnik_stablo(kazaljka, razina) = x$$

Kazaljka označava redni broj znaka u ulaznom nizu, razina označava razinu čvora u stablu, a x predstavlja završni znak, ako se radi o listu, ili nezavršni znak ako se radi o čvoru. Za gradnju stabla koristi se pomoćni stog na koji se spremaju sve zamjene lijevim stranama produkcije da bi se kasnije sve moglo spremiti u rječnik stabla. Razine idu rastućim brojevima od 1 do n odozdo prema gore, a kazaljka ide od 0 do broja znakova u ulaznom nizu, bez znaka koji označava kraj ulaznog niza '\$'. Na slici 5. prikazano je generativno stablo za ulazni niz 'a+a'. Sivi krugovi predstavljaju listove, a zeleni čvorove.



Slika 5. Primjer generativnog stabla ulaznog niza 'a+a'.

Izgled rječnika za primjer 'a+a' bio bi:

rjecnik_stablo (0, 1) = 'a'	rjecnik_stablo (1, 4) = '+'	rjecnik_stablo (2, 2) = 'a'
rjecnik_stablo (0, 2) = 'P'		rjecnik_stablo (2, 3) = 'P'
rjecnik_stablo (0, 3) = 'T'		rjecnik_stablo (2, 4) = 'T'
rjecnik_stablo (0, 4) = 'E'		
rjecnik_stablo (0, 5) = 'E'		

U programu se dobije ispis rječnika iz kojeg se dalje jednostavno konstruira generativno stablo. Ispis za ulazni niz 'a+a' prikazan je na slici 6.

```

Ulazni niz ['a', '+', 'a', '$'] se prihvaca
Rjecnik koji sadrzi generativno stablo = {(0, 1): 'a', (2, 2): 'a', (1, 4): '+',
(0, 2): 'P', (2, 3): 'P', (0, 4): 'E', (0, 5): 'E', (0, 3): 'T', (2, 4): 'T'}
>>> |

```

Slika 6. Primjer ispisa rječnika ulaznog niza 'a+a'.

Kod koji konstruira generativno stablo je kod parsera s dodatnim funkcijama, petljama i uvjetima za izgradnju generativnog stabla. Izgradnja generativnog stabla ostvaruje se na slijedeći način:


```

while(zastavica):
    if( (stog[-1] == '@') and (ulazni[kazaljka] == '$') ):
        #prihvati
        print( "Ulazni niz " + str(ulazni) + " se prihvaca" )
        zastavica = 0

        break

    else:
        if( (rjecnik_prednosti[(stog[-1], ulazni[kazaljka])] == '<') or
            (rjecnik_prednosti[(stog[-1], ulazni[kazaljka])] == '=') ):
            print("Ulazni znak: " + str(ulazni[kazaljka]))
            print("Akcija pomakni()")
            print("Stog = " + str(stog))
            if(ulazni[kazaljka] != '$'):
                stog.append(ulazni[kazaljka])
            print("Stog = " + str(stog))
            kazaljka += 1

        elif( rjecnik_prednosti[(stog[-1], ulazni[kazaljka])] == '>' ):
            print("Ulazni znak: " + str(ulazni[kazaljka]))
            print("Akcija reduciraj!")
            print("Stog = " + str(stog))

            znakSaVrhaStoga = stog.pop()
            while( rjecnik_prednosti[stog[-1], znakSaVrhaStoga] != '<' ):
                znakSaVrhaStoga = stog.pop()
                uzorak.append(znakSaVrhaStoga)
                print("izbacujem: " + str(znakSaVrhaStoga))
            print("Znak sa vrha stoga = " + str(znakSaVrhaStoga))
            uzorak.append(ulazni[kazaljka-1])
            stogStablo.append(uzorak[-1])

        while((str(uzorak[-1]) + str(ulazni[kazaljka])) in reduciranZnakom) and
            (ulazni[kazaljka] != '$')):
            for i in prijelazi:
                if(ulazni[kazaljka] in i):
                    for s in prijelazi:
                        if(uzorak[-1] == s[3]):
                            uzorak[-1] = s[0]
                            stogStablo.append(s[0])

        if(ulazni[kazaljka] == '$'):
            for s in prijelazi:
                duljina = len(s[3:])
                tmp = ''.join(str(n) for n in uzorak[-duljina:])
                if(uzorak[-2] in s):
                    while(tmp not in s):
                        for x in prijelazi:
                            if(uzorak[-1] == x[3]):
                                uzorak[-1] = x[0]
                                stogStablo.append(x[0])
                                tmp = ''.join(str(n) for n in uzorak[-duljina:])
                            elif(uzorak[-1] == x[-1]):
                                tmp2 = ''.join(str(n) for n in uzorak[-duljina:])
                                if(tmp2 == x[-duljina:]):
                                    for i in duljina:
                                        uzorak.pop()
                                        uzorak.append(x[0])
                                tmp = tmp2
                        else:
                            tmp = tmp + x[0]
            for i in range(len(stog)-1):
                stog.pop()

        uzorak.append(ulazni[kazaljka])
        duljina = len(stogStablo)

```

```

        if(ulazni[kazaljka] != '$'):
            for i in range(len(stogStablo)):
                znak = stogStablo.pop()
                rjecnik_stablo[(kazaljka-1, duljina-i)] = znak
                nivo.append(duljina-i)
                for s in prijelazi:
                    if((str(znak) + str(ulazni[kazaljka])) == s[3:5]):
                        rjecnik_stablo[(kazaljka, duljina-i)] = ulazni[kazaljka]
                        maks = duljina - i
            else:
                for i in range(len(stogStablo)):
                    znak = stogStablo.pop()
                    rjecnik_stablo[(kazaljka-1, maks-i)] = znak
                    nivo.append(maks-i)
                    for s in prijelazi:
                        if((str(znak) + str(ulazni[kazaljka])) == s[3:5]):
                            rjecnik_stablo[(kazaljka, maks-i)] = ulazni[kazaljka]
        else:
            #odbaci
            print("Ulazni niz " + str(ulazni) + " se odbacuje.")
            zastavica = 0
            stog_stablo = []

            break

uzorak.pop()

maks = max(nivo)
if(len(uzorak) == 3):
    maks +=1
duljina_uzorka = len(uzorak)
while((len(uzorak) > 1) and (uzorak[-1] != 'E')):
    for s in prijelazi:
        duljina = len(s[3:])
        tmp = ''.join(str(n) for n in uzorak[-duljina:])
        if(tmp == s[3:]):
            for j in range(duljina):
                uzorak.pop()
            uzorak.append(s[0])
            rjecnik_stablo[(duljina_uzorka - duljina, maks)] = s[0]
            stogStablo.append(s[0])
            maks += 1
            if (len(uzorak) >= duljina):
                tmp = ''.join(str(n) for n in uzorak[-duljina:])
            elif((len(uzorak) == 1) and (str(uzorak[-1]) != 'E')):
                for x in prijelazi:
                    if(str(uzorak[-1]) == x[3]):
                        uzorak[-1] = x[0]
                        rjecnik_stablo[(duljina_uzorka - duljina, maks)] = x[0]
print("Rjecnik koji sadrzi generativno stablo = " + str(rjecnik_stablo))

```

Unutar programa provjerava se relacija *Reduciran Znakom*, ako zadovoljavaju 2 znaka relaciju, vrši se zamjena desne strane produkcije sa lijevom stranom produkcije. Svaka zamjena sprema se na stog *stogStablo* i kasnije se pomoću njega spremaju čvorovi stabla u ispravnom redoslijedu u rječnik stabla nazvan *rjecnik_stablo*. U rječniku *rjecnik_stablo* spremljeno je cijelo stablo sa prije opisanim ključevima. Na taj način na kraju rada programa imamo spremljeno stablo koje je lako dohvatljivo dalje u programu.

2.5. Izvođenje programa

Za izvođenje programa potrebno je pokrenuti datoteku *parser.py* ili *parser_generativno_stablo.py* pomoću ugrađenog *Python IDLE*. Za provjeru rada parsera (*parser.py*) koristi se 6 ulaznih nizova od kojih 2 ulazna niza nisu ispravna, a ostalih 4 jesu. Pošto se koriste dvije gramatike (*gramatika.txt* i *gramatika2.txt*) isto kao i dvije tablice prednosti, za sve nizove možemo koristiti *gramatiku2.txt* jer je ona proširena sa '(' i ')' i pokriva prijelaze i operatore *gramatike.txt* kao što možemo koristiti i tablicu prednosti *tablicaPrednosti.txt*. Ulazni nizovi *UlazniNiz5.txt* i *UlazniNiz6.txt* nisu ispravni i njih parser odbacuje, a ostali nizovi su ispravni i njih parser prihvata.

Datoteka	Ulazni niz	Ispravnost
<i>UlazniNiz.txt</i>	a+a*a	Ispravan
<i>UlazniNiz2.txt</i>	a*a	Ispravan
<i>UlazniNiz3.txt</i>	a+a	Ispravan
<i>UlazniNiz4.txt</i>	a*(a+a)	Ispravan
<i>UlazniNiz5.txt</i>	a*aa	Neispravan
<i>UlazniNiz6.txt</i>	aaa	Neispravan

Pri pokretanju programa *parser.py*, program direktno dohvaća redom datoteke i za svaku pokreće parsiranje. Na kraju svakog parsiranja ispiše da li je niz prihvaćen ili odbijen. Ispis za neke ispravne nizove dobiven pokretanjem *parser.py* u *Python Shell-u* je sljedeći:

```
Ulazni niz 1
Ulazni niz : a+a*a
Ulazni znak: a
Akcija pomakni()
Stog = ['@']
Ulazni znak: +
Akcija reduciraj()
Stog = ['@', 'a']
Znak sa vrha stoga = a
Ulazni znak: +
Akcija pomakni()
Stog = ['@']
Ulazni znak: a
Akcija pomakni()
Stog = ['@', '+']
Ulazni znak: *
Akcija reduciraj()
Stog = ['@', '+', 'a']
Znak sa vrha stoga = a
Ulazni znak: *
Akcija pomakni()
Stog = ['@', '+']
Ulazni znak: a
Akcija pomakni()
Stog = ['@', '+', '*']
Ulazni znak: $
Akcija reduciraj()
Stog = ['@', '+', '*', 'a']
Znak sa vrha stoga = a
Ulazni znak: $
Akcija reduciraj()
Stog = ['@', '+', '*']
Znak sa vrha stoga = *
Ulazni znak: $
Akcija reduciraj()
Stog = ['@', '+']
Znak sa vrha stoga = +
Akcija prihvati()
Ulazni niz ['a', '+', 'a', '*', 'a', '$'] se prihvaca

Ulazni niz 2
Ulazni niz : a*a
Ulazni znak: a
Akcija pomakni()
Stog = ['@']
Ulazni znak: *
Akcija reduciraj()
Stog = ['@', 'a']
Znak sa vrha stoga = a
Ulazni znak: *
Akcija pomakni()
Stog = ['@']
Ulazni znak: a
Akcija pomakni()
Stog = ['@', '*']
Ulazni znak: $
Akcija reduciraj()
Stog = ['@', '*', 'a']
Znak sa vrha stoga = a
Ulazni znak: $
Akcija reduciraj()
Stog = ['@', '*']
Znak sa vrha stoga = *
Akcija prihvati()
Ulazni niz ['a', '*', 'a', '$'] se prihvaca
```

Ispis za neispravne nizove dobiven pokretanjem datoteke *parser.py* u *Python Shell-u* je sljedeći:

```
Ulazni niz 5
Ulazni niz : a*aa
Ulazni znak: a
Akcija pomakni()
Stog = ['@']
Ulazni znak: *
Akcija reduciraj()
Stog = ['@', 'a']
Znak sa vrha stoga = a
Ulazni znak: *
Akcija pomakni()
Stog = ['@']
Ulazni znak: a
Akcija pomakni()
Stog = ['@', '*']
Ulazni niz ['a', '*', 'a', 'a', '$'] se odbacuje.

Ulazni niz 6
Ulazni niz : aaa
Ulazni znak: a
Akcija pomakni()
Stog = ['@']
Ulazni niz ['a', 'a', 'a', '$'] se odbacuje.
```

Pokretanjem datoteke *parser_generativno_stablo.py* parsiraju se 3 ispravna niza i gradi se za svakog od njih pripadajuće generativno stablo spremljeno u rječnik. Ispisom rječnika u kojem se nalazi stablo možemo konstruirati stablo.

U parsiranju s izgradnjom generativnog stabla koriste se sljedeći nizovi:

Datoteka	Ulazni niz	Ispravnost
<i>UlazniNiz.txt</i>	a+a*a	Ispravan
<i>UlazniNiz2.txt</i>	a*a	Ispravan
<i>UlazniNiz3.txt</i>	a+a	Ispravan

Ispis tijeka programa i pripadajućeg generativnog stabla je sljedeći:

```
Python Shell

File Edit Shell Debug Options Windows Help

Ulazni niz 1

Ulazni niz : a+a*a
Ulazni znak: a
Akcija pomakni()
Stog = ['@']
Stog = ['@', 'a']
Ulazni znak: +
Akcija reduciraj!
Stog = ['@', 'a']
Znak sa vrha stoga = a
Ulazni znak: +
Akcija pomakni()
Stog = ['@']
Stog = ['@', '+']
Ulazni znak: a
Akcija pomakni()
Stog = ['@', '+']
Stog = ['@', '+', 'a']
Ulazni znak: *
Akcija reduciraj!
Stog = ['@', '+', 'a']
Znak sa vrha stoga = a
Ulazni znak: *
Akcija pomakni()
Stog = ['@', '+']
Stog = ['@', '+', '*']
Ulazni znak: a
Akcija pomakni()
Stog = ['@', '+', '*']
Stog = ['@', '+', '*', 'a']
Ulazni znak: $
Akcija reduciraj!
Stog = ['@', '+', '*', 'a']
Znak sa vrha stoga = a
Ulazni niz ['a', '+', 'a', '*', 'a', '$'] se prihvaca
Rjecnik koji sadrzi generativno stablo = {(0, 1): 'a', (2, 5): 'E', (3, 3): '*',
(2, 2): 'P', (1, 4): '+', (2, 4): 'T', (2, 3): 'T', (2, 1): 'a', (4, 3): 'P', (0,
4): 'E', (4, 2): 'a', (0, 3): 'T', (0, 2): 'P'}
```

*Ispis tijeka programa i generativnog stabla za ulazni niz 'a+a*a'.*

```
Python Shell

File Edit Shell Debug Options Windows Help

Ulazni niz 2

Ulazni niz : a*a
Ulazni znak: a
Akcija pomakni()
Stog = ['@']
Stog = ['@', 'a']
Ulazni znak: *
Akcija reduciraj!
Stog = ['@', 'a']
Znak sa vrha stoga = a
Ulazni znak: *
Akcija pomakni()
Stog = ['@']
Stog = ['@', '*']
Ulazni znak: a
Akcija pomakni()
Stog = ['@', '*']
Stog = ['@', '*', 'a']
Ulazni znak: $
Akcija reduciraj!
Stog = ['@', '*', 'a']
Znak sa vrha stoga = a
Ulazni niz ['a', '*', 'a', '$'] se prihvaca
Rjecnik koji sadrzi generativno stablo = {(0, 1): 'a', (1, 3): '*', (0, 4): 'T',
(2, 3): 'P', (2, 2): 'a', (0, 5): 'E', (0, 3): 'T', (0, 2): 'P'}
```

*Ispis tijeka programa i generativnog stabla za ulazni niz 'a*a'.*

```
Python Shell

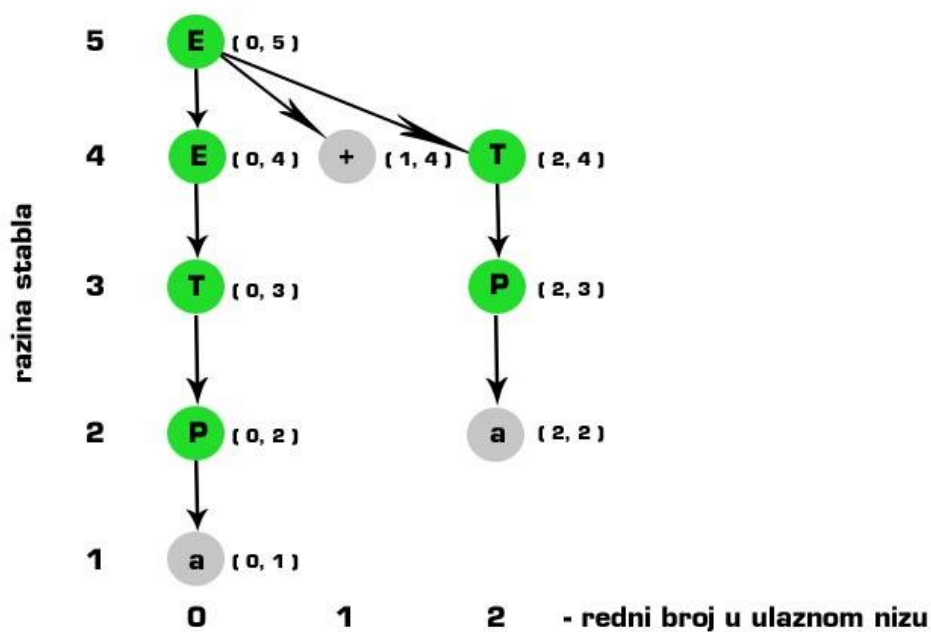
File Edit Shell Debug Options Windows Help

Ulazni niz 3

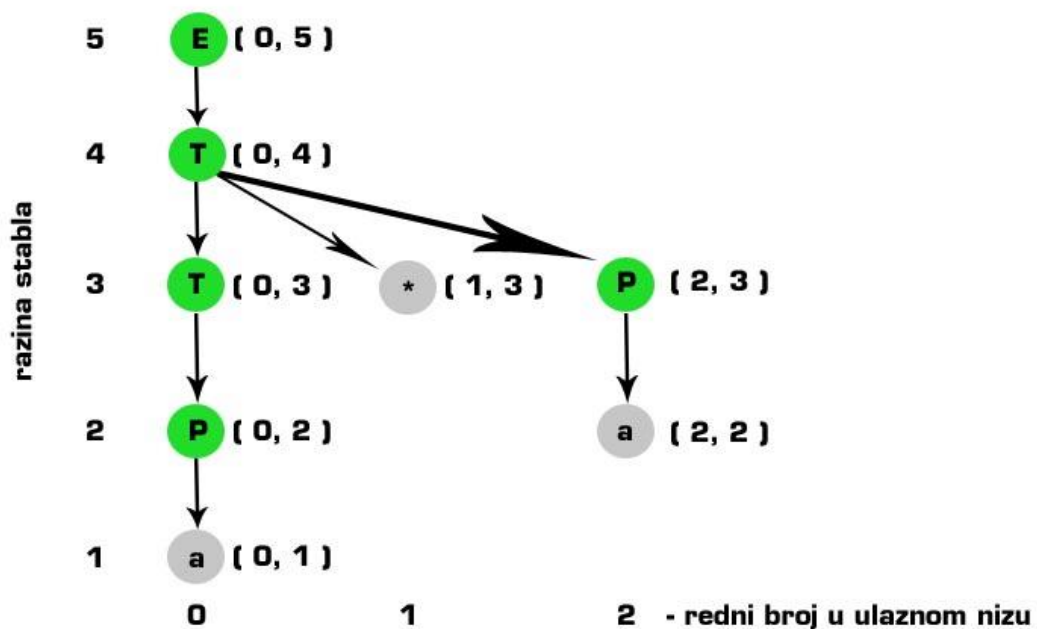
Ulazni niz : a+a
Ulazni znak: a
Akcija pomakni()
Stog = ['@']
Stog = ['@', 'a']
Ulazni znak: +
Akcija reduciraj!
Stog = ['@', 'a']
Znak sa vrha stoga = a
Ulazni znak: +
Akcija pomakni()
Stog = ['@']
Stog = ['@', '+']
Ulazni znak: a
Akcija pomakni()
Stog = ['@', '+']
Stog = ['@', '+', 'a']
Ulazni znak: $
Akcija reduciraj!
Stog = ['@', '+', 'a']
Znak sa vrha stoga = a
Ulazni niz ['a', '+', 'a', '$'] se prihvaca
Rjecnik koji sadrzi generativno stablo = {(0, 1): 'a', (2, 2): 'a', (1, 4): '+',
(0, 2): 'P', (2, 3): 'P', (0, 4): 'E', (0, 5): 'E', (0, 3): 'T', (2, 4): 'T'}
```

Ispis tijeka programa i generativnog stabla za ulazni niz 'a+a'.

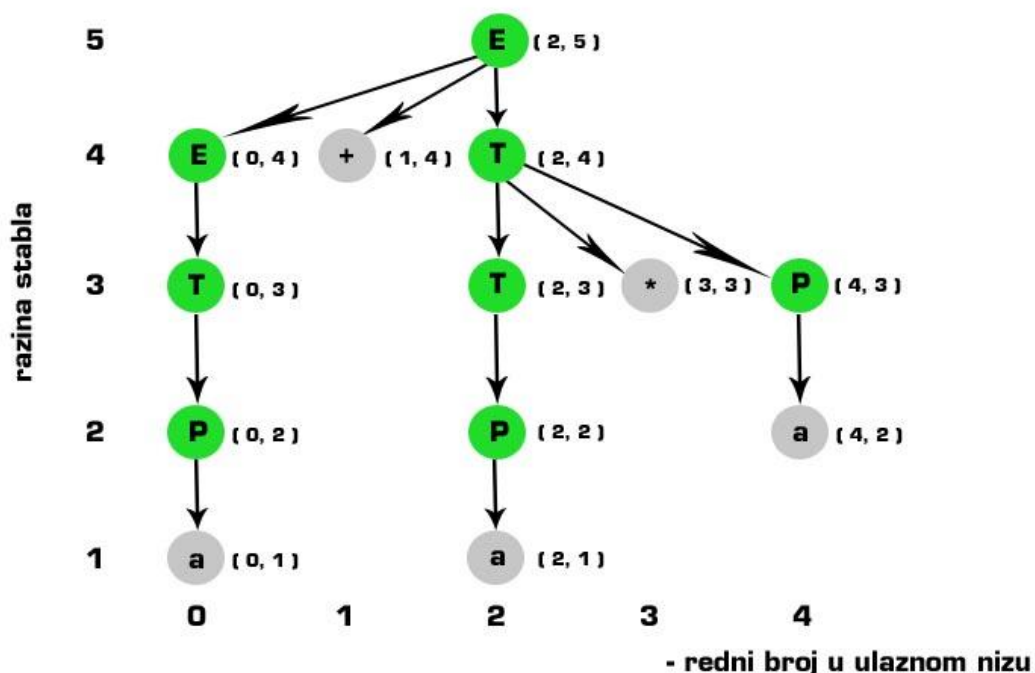
Izgrađena generativna stabla ulaznih nizova 'a+a', 'a*a' i 'a+a*a' sa svojim poveznicama sa rječnikom stabla bila bi slijedeća:



Generativno stablo ulaznog niza 'a+a'



Generativno stablo ulaznog niza 'a*a'



*Generativno stablo ulaznog niza 'a+a*a'*

S lijeve strane na grafu označene su razine brojkama. Ispod svih znakova označeni su indeksi svakog znaka u ulaznoj datoteci. Brojeve u zagradi s desne strane svakog čvora i lista su ključevi za taj element u rječniku stabla. Zeleni krug predstavlja čvor unutar stabla i u njemu se nalaze nezavršni znakovi gramatike, a sivi krug predstavlja listove stabla i u njemu se nalaze završni znakovi gramatike.

3. Zaključak

Parser s prednosti operatora parsira ulazne nizove pomoću tablice prednosti operatora i gramatike. Parsiranje prednosti operatora vrši se tako da se uspoređuje završni znak na vrhu stoga i znak u ulaznom nizu i na temelju relacije prednosti za ta dva znaka odlučuje o primjeni akcije *Pomakni* ili *Reduciraj*. Programsko rješenje je napisano u programskom jeziku Python 3.1.3 i sve programske datoteke imaju nastavak '.py'. Ovo programsko rješenje moglo bi se dalje koristiti u programima koji trebaju odrediti da li je ispravno zadan niz u ovisnosti o prednosti operatora. Kod se lako može implementirati kao dodatni modul i možemo ga koristiti pomoću import u *Python* programskom jeziku.

U načinu konstrukcije rješenja parsera pomoć je nađena u konstruiranju LR(1) parsera iz druge laboratorijske vježbe. Logika korištena u drugoj laboratorijskoj vježbi slična je i logici rješavanja ovog problemskog zadatka. Problemi su bili nedostatak literature i nedostatak informacija na internetu. Do rješenja se došlo pomoću olovke, papira i modeliranje samog koda u početku. Parser bez generativnog stabla nije bio problem, jer on ima način rada koji se lako implementira. Izgradnja generativnog stabla je bila poprilično zahtjevan zadatak i tu se

trebalo puno više vremena i truda uložiti. Pojavljuju se određeni problemi s gradnjom generativnog stabla nekih nizova. Problem stoji zbog toga što nema strogo definiranog načina gradnje generativnog stabla. Ako se pokrije jedna situacija u jednom nizu pri zamjeni desne strane produkcije lijevom stranom, nastaje problem pri zamjeni u drugoj situaciji ili u drugom nizu. Problem se može riješiti tako da se nađe pravilan algoritam gradnje generativnog stabla koji bi vrijedio za svaku situaciju i pokrio svaki mogući problem. Rješenje problema izgradnje samog generativnog stabla bilo bi puno lakše da postoji opširnija literatura i više primjera.

4. Literatura

1. Srbljić, Siniša : „Prevođenje programskih jezika“, 1. izdanje, Element, Zagreb, 2007.
2. Srbljić, Siniša : „Jezični procesori 1“, 2. izdanje, Element, Zagreb, 2002.
3. Python v3.0.1 documentation, <http://docs.python.org/release/3.0.1/>