

ZAVOD ZA ELEKTRONIKU, MIKROELEKTRONIKU, RAČUNALNE I INTELIGENTNE SUSTAVE  
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA  
SVEUČILIŠTE U ZAGREBU

Melita Mihaljević

**Seminarski rad iz predmeta  
Automati, formalni jezici i jezični procesori I**

Zadatak broj 47

Zagreb, 2007.

## **Seminarski rad iz predmeta Automati, formalni jezici i jezični procesori I**

**Student:** Melita Mihaljević

**Matični broj studenta:** 0036411392

**Zadatak broj 47:**

Implementirati na računalu program koji za zadanu kontekstno neovisnu gramatiku gradi odgovarajući potisni automat ( $L(M) = L(G)$ ), prema algoritmu datom na predavanju (u udžbeniku).

# Sadržaj

1. Uvod.....	4
2. Opis algoritma.....	5
2.1. Svođenje produkcija gramatike na Greibachov normalni oblik.....	5
2.2. Pretvorba gramatike u potisni automat.....	5
2.3. Simulacija PA.....	6
3. Programsko rješenje.....	7
3.1. Svođenje produkcija gramatike na Greibachov normalni oblik.....	8
3.2. Konstrukcija PA.....	9
3.3. Simulacija PA.....	10
4. Primjeri.....	10
4.1. Primjer 1.....	10
4.2. Primjer 2.....	12
5. Zaključak.....	16
6. Dodatci.....	17
6.1. Popis klasa i pripadnih metoda.....	17
6.2. Hijerarhijska struktura klasa.....	18
7. Literatura.....	19

## 1. Uvod

Izgradnja potisnog automata iz kontekstno neovisne gramatike moguća je zbog činjenice da kontekstno neovisna gramatika generira jedan od kontekstno neovisnih jezika, te isto tako za bilo koji kontekstno neovisan jezik  $L$  postoji nedeterministički potisni automat  $PA M$  koji prihvaća praznim stogom  $N(M)=L$ . Pretpostavka je da prazni niz nije element jezika  $L$ .

Za kontekstno neovisnu gramatiku moguće je sagraditi potisni automat  $PA$  koji prihvaća praznim stogom, a prihvaća jezik koji zadana kontekstno neovisna gramatika generira.

Potisni automat se iz kontekstno neovisne gramatike gradi na sljedeći način:

Neka je zadana gramatika  $G = (V, T, P, S)$  koja generira kontekstno neovisan jezik  $L$ , a koja ima produkcije u Greibachovom normalnom obliku (sve produkcije su oblika  $A \rightarrow a\beta$ , gdje je  $A$  element nezavršnih znakova  $V$ , a je element završnih znakova  $T$ , a  $\beta$  je iz skupa nezavršnih znakova, konstruira se  $PA$ :

$$M = (\{q\}, \Sigma, \Gamma, \delta, q, S, \emptyset)$$

- a)  $PA$  ima samo jedno stanje  $q$  koje je ujedno i početno stanje
- b) skup znakova stoga  $\Gamma$  jednak je skupu nezavršnih znakova gramatike  $V$
- c) skup ulaznih znakova  $\Sigma$  jednak je skupu završnih znakova gramatike  $T$
- d) početni znak stoga jest početni nezavršni znak  $S$
- e) skup prihvatljivih stanja je prazan skup
- f)  $PA M$  prihvaća praznim stogom

Funkcije prijelaza zadaju se na sljedeći način:

$$\delta(q, a, A) \text{ sadrži } (q, \gamma) \text{ ako i samo ako postoji produkcija } A \rightarrow a\gamma$$

$PA M$  simulira postupak generiranja niza zamjenom krajnje lijevog nezavršnog znaka.

Dokaz da  $PA M$  prihvaća niz  $x$  praznim stogom ako i samo ako gramatika  $G$  generira niz  $x$  provodi se matematičkom indukcijom, te je moguće dokazati:

$PA M$  prolazi slijedom prijelaza  $(q, x, S) \rightarrow (q, \varepsilon, \alpha)$  ako i samo ako gramatika generira niz  $x\alpha$  postupkom zamjene krajnjeg lijevog nezavršnog znaka.

- i) Za  $\alpha$  prazan niz
- ii)  $(q, x, S) \rightarrow (q, \varepsilon, \varepsilon)$  ako i samo ako  $S \rightarrow x$

## 2. Opis algoritma

Rješenje zadatka sastoji se od tri dijela:

1. Svođenje produkcija gramatike na Greibachov normalni oblik
2. Pretvorba gramatike u potisni automat
3. Simulacija potisnog automata izgrađenog iz zadane gramatike

### 2.1. Svođenje produkcija gramatike na Greibachov normalni oblik

Svođenje produkcija gramatike na Greibachov normalni oblik ostvareno je nizom algoritama koji omogućuju da se gramatika ne sastoji od beskorisnih znakova, jediničnih produkcija, te epsilon produkcija, što omogućuje zadavanje kontekstno neovine gramatike u bilo kojem obliku (tj.  $S \rightarrow \beta$ , pri čemu je  $\beta$  niz koji se sastoji od završnih i nezavršnih znakova, ili od znaka epsilon)

Algoritmi za odbacivanje mrtvih znakova i odbacivanje nedohvatljivih znakova, te algoritmi za odbacivanje epsilon produkcija i jediničnih produkcija istovjetni su algoritmima opisanim u udžbeniku, te se izvode točno određenim redoslijedom. Najprije se izvodi algoritam odbacivanja mrtvih znakova, zatim nedohvatljivih, te zatim algoritmi odbacivanja epsilon produkcija i jediničnih produkcija. Odabir redoslijeda izvođenja posljednja dva algoritma nebitan je za krajnji oblik produkcija.

Algoritam za pretvorbu produkcija gramatike u Greibachov normalni oblik ponešto se razlikuje od algoritma navedenog u udžbeniku. Algoritam naveden u udžbeniku ne koristi se zbog kompleksnosti izvedbe. Kao pomoć izvedenom algoritmu koristi se algoritam rješavanja krajnje lijevog nezavršnog znaka također prikazan u udžbeniku,

Produkcije u Greibachovom obliku su oblika:  $S \rightarrow a\beta$  gdje je  $\beta$  niz nezavršnih znakova ili prazan niz. Algoritam se izvodi na sljedeći način:

1. produkcije koje su oblika  $A \rightarrow a\beta$  ili  $A \rightarrow a$  ( $\beta$  je skup nezavršnih znakova) su u Greibachovom obliku i dodaju se u listu produkcija u Greibachovom obliku
2. za produkcije oblika  $A \rightarrow \beta$  primjenjuje se algoritam rješavanja krajnje lijevog nezavršnog znaka. Nakon primjene algoritma rješavanja krajnje lijevog nezavršnog znaka produkcije su u jednom od dva moguća oblika:  $A \rightarrow a\beta$  pa se dodaju u listu produkcija u Greibachovom obliku, ili  $A \rightarrow a\gamma$ , gdje je  $\gamma$  skup završnih i nezavršnih znakova
3. za sve produkcije oblika  $A \rightarrow a\gamma$  svi završni znakovi zamijene se nezavršnim znakom  $[b]$ , te se dodaju produkcije  $[b] \rightarrow b$ .

Nakon trećeg koraka sve produkcije su u Greibachovom normalnom obliku, pogodnom za pretvorbu u potisni automat.

### 2.2. Pretvorba gramatike u potisni automat

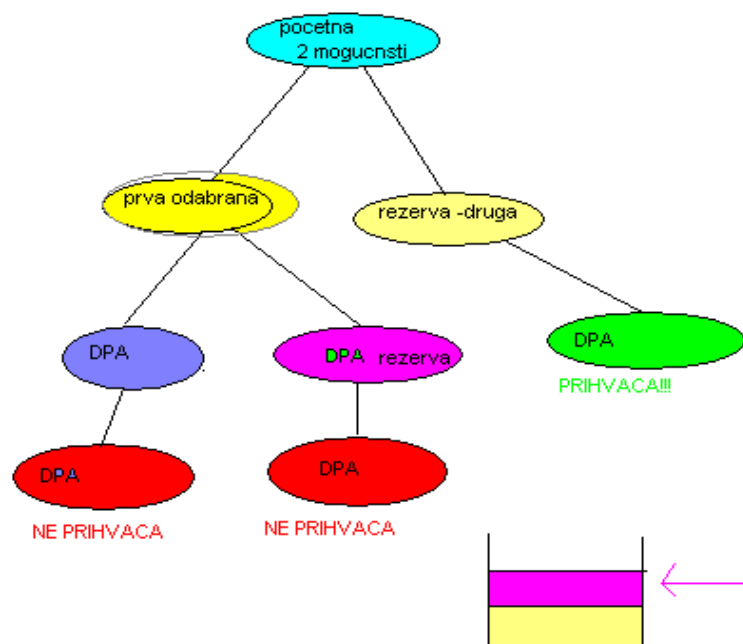
U drugom dijelu vrši se pretvorba kontekstno neovisne gramatike u potisni automat PA M koji prihvaća praznim stogom. Kao što je navedeno u tekstu zadatka PA se gradi prema algoritmu prikazanom na predavanju (u udžbeniku), također, objašnjenom u uvodnom poglavlju.

### 2.3. Simulacija PA

Kako se gradnjom potisnog automata iz kontekstno neovisne gramatike stvara nedeterministički potisni automat, njegova simulacija je riješena na sljedeći način:

1. Za svaki učitani znak i za svako trenutno stanje na vrhu stoga provjeri da li postoji mogućnost odabira između dvije ili više akcija stoga. Idi na korak 3.
2. Ako postoji mogućnost odabira između dvije ili više akcija stoga, odaberi prvu prijeđi na korak 3 . Ostale mogućnosti spremi na rezervni stog akcija, na ostale rezervne stogove spremi trenutni niz, trenutno stanje stoga i znak koji je učitani
3. Pokušaj simulirati DPA dok ne učitaj cijeli niz (uvijek provjeravaj korak 1). Ukoliko se nakon učitanih niza stog uspio isprazniti niz se prihvaća i zaustavlja se simulacija. Ukoliko se stog nije ispraznio, ili ne postoje daljnji prijelazi, trenutni DPA ne prihvaća niz. Sa rezervnih stogova skidaju se informacije o novom DPA kojeg treba simulirati. Ako je rezervni stog niza prazan nema se što učitati, te se tek tada donosi odluka da zadani PA ne prihvaća niz.

Ovim algoritmom ostvarena je struktura stabla pri simuliranju NPA. Algoritam je objašnjen na sljedećoj slici:



Slika 1: Simulacija potisnog automata

Kao što prikazuje Slika 1, u trenutku kada imamo mogućnost odabira dvije akcije koje će se izvesti nad stogom, odaberemo prvu a drugu stavimo na rezervni stog. U sljedećem koraku imamo opet mogućnost odabira dvije akcije stoga pa odabiremo prvu, a drugu stavljamo na rezervni stog. Nakon toga do kraja simuliramo DPA i nakon što prvi DPA ne prihvaća skidamo sa rezervnog stoga elemente koji sadrže informaciju o novom DPA

(drugi odabir akcije stoga PA kojeg simuliramo (ljubičasti)), nakon toga simuliramo DPA do kraja. Kako niti ovaj DPA ne prihvaća moramo se vratiti prema vrhu stabla na prvo gornje mjesto grananja. Skidamo sa rezervnog stoga informacije o novom DPA (žuti), simuliramo do kraja DPA te ovaj DPA prihvaća i niz se prihvaća. Ako bilo koji DPA prihvati niz, NPA prihvaća niz i simulacija se zaustavlja. U slučaju da niti jedan DPA ne prihvati niz se ne prihvaća i simulacija se zaustavlja.

### 3. Programsko rješenje

Za programski jezik odabran je Python, prvenstveno zbog toga što je Python objektno - orijentiran i dijelom funkcijski programski jezik jednostavne sintakse. Odabran je programski jezik koji je objektno orijentiran zbog toga što je program dizajniran na objektno orijentiran način: moguće je koristiti pojedine dijelove programa neovisno o drugim dijelovima, smanjeno je nepotrebno ponavljanje koda, te je putem sučelja objekta ostvarena mogućnost korištenja metoda jedino imajući saznanja o tome što pojedina metoda radi, što prima kao argumente i što vraća kao argumente. Hijerarhijska struktura klasa prikazana je u drugom dodatku na kraju dokumenta.

Svaka klasa u programskom jeziku Python ima metodu `__init__` koja je konstruktor klase. Svaka metoda se kreira na način:

```
def ime_metode(self, argumenti):  
    tijelo metode
```

Za pristup pojedinoj metodi klase potrebno je postojanje reference na objekt ,a to je parametar `self`. Parametar `self` je parametar koji se uvijek nalazi u popisu argumenata pojedine metode. `self` parametar također omogućava korištenje objekata u više različitih metoda. Kod poziva metode ne navodi se parametar `self`. Primjer kreiranja instanci neke klase i pozivanje metoda je sljedeći:

```
ime_instance=ime_klase()  
foo= ime_instance.ime_metode(argumenti)
```

U programskom jeziku Python dopušteno je višestruko nasljeđivanje, točnije nasljeđivanje više od jedne klase jednostavnim navođenjem klasa od kojih definirana klasa nasljeđuje metode i svojstva:

```
class nasljedjena (prva_klasa, druga_klasa):  
    def __init__(self, argumenti):  
        tijelo metode  
    def metoda1 (self, argumenti):  
        tijelo metode  
    ....
```

Moguće je naslijeđene klase koristiti u izvornom obliku, a moguće ih je “prepraviti” da rade na način koji predviđa zadana klasa.

Kako je u tri dijela objašnjeno rješenje problema zadatka, na isti način ukratko su navedeni dijelovi programa.

### 3.1. Svođenje produkcija gramatike na Greibachov normalni oblik:

U ovom dijelu program se sastoji od niza klasa i metoda u klasama kojima se opisuju elementi gramatike (nezavršni znakovi, završni znakovi te skup produkcija). Nezavršni znakovi kojim se zadaju produkcije su isključivo velika slova abecede. Završni znakovi su mala slova abecede (u slučaju da se niz koji se ispituje sastoji od brojeva potrebno ga je zadati slovima koji imaju jednako značenje kao i zadani brojevi, npr 1=j).

Produkcije gramatike su smještene u internoj strukturi programskog jezika Python – rječnik (eng. *dictionary, mapping*) koji služi kao hash tablica gdje su ključevi nezavršni znakovi gramatike, s pripadnim desnim stranama produkcije.

Produkcije gramatike zadaju se u obliku S->aA datotekom u kojoj je je svaka produkcija navedena u jednoj liniji datoteke.

U nastavku teksta ukratko su objašnjene pojedine klase, a cjelovit popis klasa i pripadnih metoda dan je u tablici na kraju dokumenta.

Klasama `završni_znakovi` i `nezavršni_znakovi` definirane su metode za kreiranje listi završnih i nezavršnih znakova, metode koje provjeravaju jesu li pojedini znakovi u listi završnih znakova, odnosno u listi nezavršnih znakova, te ispisuju pojedine liste. Ekvivalentne metode u obje klase imaju jednaka imena radi jednostavnijeg korištenja.

Klasa produkcije koja je naslijeđena klasa klase nezavršni znakovi sadrži metode koje omogućuju dodavanje produkcija gramatike u rječnik `produkcije`. Dodavanje produkcija u rječnik produkcije obavlja se pomoću metode `dodaj_produkciju` (koja prima argumente nezavršni znak i desnu stranu produkcije, a vraća rječnik `produkcije`) na sljedeći način:

```
produkcije= {}
...
self.produkcije[nezavršni]=[]
...
self.produkcije[nezavršni].append(desna_strana_produkcije)
```

Kao što se vidi u gornjem primjeru prvo je potrebno instancirati rječnik podataka, zatim omogućiti da za svaki nezavršni znak može postojati više desnih strana produkcije koje dodajemo u listu, koja je instancirana u drugom redu. Nakon toga, ugrađenom metodom `append` dodajemo desnu stranu produkcije u rječnik podataka.

Ostale metode koje sadrži klasa produkcije su metode za brisanje produkcija, ispis desnih strana te ispis produkcija gramatike u obliku : { 'nezavršni\_znak':['aB','BC','\$'],...} koji je ekvivalentan produkcijama gramatike u obliku: nezavršni\_znak->aB|BC|\$

Sljedeća klasa `izraz` omogućava da se iz produkcija oblika A->bC koje su pročitane iz datoteke generiraju produkcije u rječnik podataka. Klasa `izraz` sadrži jednu metodu: `dodaj_u_produkciju` koja učitani niz u obliku A->bc razdvaja na lijevu stranu produkcije (nezavršni znak) i na desnu stranu produkcije, te poziva metodu klase produkcije koja gradi rječnik podataka. Za svaku pročitani desnu stranu produkcije poziva metodu klase `završni_znakovi` koja dodaje završne znakove u listu završnih, te isto tako poziva metodu klase `nezavršni_znakovi` koja dodaje lijeve strane produkcije u listu nezavršnih znakova.

Sljedećih nekoliko klasa implementiraju algoritme za odbacivanje beskorisnih znakova, rješavanje epsilon produkcije te jediničnih produkcija.

Klasa `zivi_znakovi` metodom `izbaci_mrtve_znakove`, te pomoćnim metodama koje provjeravaju

da li se neki znak nalazi u listi živih implementiraju algoritam za odbacivanje mrtvih znakova. Nakon izvršavanja algoritma u riječniku produkcije nalaze se samo produkcije koje sadrže žive znakove.

Klasa `dohvatljivi_znakovi` metodom `izbaci_nedohvatljive_znakove` te isto tako pomoćnim metodama (slične metodama u klasi `zivi_znakovi`), implementira algoritam odbacivanja mrtvih znakova. Nakon izvršenja algoritma u riječniku produkcije nalaze se produkcije koje u sebi ne sadrže beskorisne znakove.

Ostale dvije klase koje su građom vrlo slične prethodno objašnjenim klasama su: `epsilon_produkcije` sa pripadnom metodom `riješ_epsilon` (implementira osnovna dva koraka) i `izbaci_epsilon` (odbacuju se na kraju epsilon produkcije koje smo prethodnim algoritmom riješili), te klasa `jedinicne_produkcije` sa pripadnom metodom `izbaci_jedinicne`.

Nakon sređivanja produkcija gramatika prethodnim metodama, pristupa se pretvorbi produkcija u Greibachov normalni oblik.

Klasa `greibach` sadrži dvije metode: metodu `krajnje_lijevi` koja implementira algoritam rješavanja krajnje lijevog nezavršnog znaka, te metodu `greibachov_oblik` koja implementira objašnjen algoritam pretvorbe produkcija u Greibachov oblik.

Metodom u `greibacha` klase `greibachov_oblik`, generiraju se dva skupa produkcija koja su u Greibachovom obliku. Skup produkcija u svom ispisu razlikuje se od standardnog oblika Greibachovih produkcija zato što ne koristi mogućnost pretvorbe završnih znakova koji se ne nalaze na krajnje desnom mjestu lijeve produkcije u nezavršne znakove u obliku  $X_i$ , nego ih se ostavlja u zadanom obliku te se u skup produkcija dodaju dodatne produkcije oblika `['završni']->'završni'` u kojima završni znak "ide" sam u sebe (postao je nezavršni znak izgledom jednak završnom znaku). Oblik je uveden radi jednostavnije konstrukcije potisnog automata.

Nakon pozivanja metode `greibachov_oblik` sve produkcije su u Greibachovom normalnom obliku te je moguće konstruirati potisni automat na temelju zadanih produkcija.

### 3.2. Konstrukcija PA

Konačan skup ulaznih znakova ostvaren je klasom `skup_ulaznih` koja je nasljeđena klasa od klase `greibachov_oblik`. Klasa sadrži metodu za dodavanje u skup ulaznih znakova (ostvarena strukturom liste koja je u Pythonu također primitivni tip podataka), koji je jednak skupu završnih znakova gramatike, metodu za ispitivanje je li znak u skupu ulaznih, te metodu koja omogućava ispisivanje liste ulaznih znakova.

Konačan skup znakova stoga ostvaren je metodom `listu_stoga` ostvarenom u klasi `radni_stog`.

Funkcija `prijelaza_prijelaz_PA` ostvarena je riječnikom (dictionaryem) kojemu su ključevi trenutno učitani znak i trenutni znak na stogu, a pridružena im je akcija koja se odvija nad stogom. Programski ostvarena funkcija `prijelaza` sadrži samo dva ulazna elementa (znak niza i znak stoga) radi jednostavnosti i činjenice da potisni automat ima samo jedno stanje koje ne utječe na akciju nad stogom. Ispis funkcije `prijelaza` je u standardnom obliku, npr.  $\delta(q,a,S)=AB$ . Funkcija `prijelaza` ostvarena je klasom `prijelazi_PA` koja sadrži metode: `generiraj_prijelaze` koja na temelju produkcija u Greibachovom obliku generira funkciju `prijelaza_prijelaz_PA`. Kako za pojedini ulazni znak i znak stoga može postojati više mogućih akcija koje se trebaju obaviti na stogu (ovdje dolazi do izražaja nedeterminizam PA), akcije se pri generiranju dodaju u listu. Prvo je potrebno inicijalizirati riječnik, pa zatim listu, te tada dodavati u listu.

To činimo na sljedeći način:

```
prijelaz_PA={}
self.prijelaz_PA[self.znak, self.trenutnoStog]=[]
self.prijelaz_PA[self.znak, self.trenutnoStog].append(self.akcija_stog)
```

Ako ispišemo produkcije gramatike koji je dan u dodatnom primjeru izbacivanja beskorisnih znakova iz gramatike, epsilon produkcija i jediničnih produkcija, koji je dan u cijelosti u odlomku Primjeri, izgledati će :

```
prod {'A': ['bcA', '$'], 'C': ['eA', '$'], 'B': ['CD', 'ad'], 'E': ['ed',
'ac'], 'D': ['cDAwB', 'bDaE'], 'S': ['bAbE', 'aABc']}
```

Druga metoda je funkcija\_prijelaza koja ispisuje funkcije prijelaza u standardnom obliku npr.  $\&(q,a,S)=AB$ , a koristi se podacima riječnika prijelaz\_PA. Ispis funkcije prijelaza dan je u primjerima.

### 3.3. Simulacija PA

Simulacija potisnog automata sastoji se od klase simulacija\_PA koja se sastoji od dvije metode: DPA\_simulacija i NPA\_simulacija. U metodi DPA\_simulacija počinje se sa simulacijom determinističkog potisnog automata. Ukoliko za učitani znak niza i znak sa vrha stoga postoje dvije ili više mogućnosti prijelaza izabire se prva mogućnost, a podaci o ostalim mogućnostima prijelaza spremaju se na stog rezervna (ostvaren podatkovnom strukturom liste). Na stog se sprema uređena trojka [niz, stog, akcija] koja sadrži podatke o sljedećem DPA kojeg treba simulirati. Ukoliko prethodni DPA ne prihvaća zadani niz poziva se metoda NPA\_simulacija. U njoj se skidaju sa stoga informacije o sljedećem DPA, ukoliko u tom DPA dođe do "grananja" spremaju se podaci na listu. Nakon svake simulacije DPA provjerava se rezervni stog i simulacija se izvršava dok ima elementa na stogu ili dok DPA ne prihvati niz. U slučaju da DPA prihvati niz, izgrađeni potisni automat prihvaća niz i simulacija se završava. Ako se rezervni stog isprazni, a posljednji DPA nije prihvatio niz, izgrađeni potisni automat ne prihvaća niz.

## 4. Primjeri

U nastavku su dana 3 primjera gramatike za koje se grade pripadni potisni automati. Na CD-u koji je priložen uz dokumentaciju nalazi se uz kod programa i 2 datoteke s ispitnim primjerima te datoteka s pripadnim nizovima za testiranje.

### 4.1. Primjer 1

Primjer 1. dan je samo kao primjer izbacivanja beskorisnih znakova, epsilon i jediničnih produkcija iz zadane gramatike, te pretvorba u Greibachov normalni oblik i konstrukcija potisnog automata. Primjer je isproban tijekom testiranja programa te nije zadan posebnom datotekom, ali je dan ispis rezultata.

Neka su zadane produkcije gramatike:

```
S->bAbE    B->DC    D->cDAwB
S-> aABc    B->ad    D->bDae
A-> bcA    C ->eA    E->ed
```

```
A> $          C->$          E->ac
```

Nakon izvršavanja programa pozivanjem odgovarajućih metoda ispis je sljedeći:

```
gizmo@mogwai:~$ python testing.py
produkcije: {'A': ['bcA', '$'], 'C': ['eA', '$'], 'B': ['CD', 'ad'], 'E':
['ed', 'ac'], 'D': ['cDAwB', 'bDaE'], 'S': ['bAbE', 'aABc']}
zivi {'A': ['bcA', '$'], 'C': ['eA', '$'], 'B': ['ad'], 'E': ['ed', 'ac'],
'S': ['bAbE', 'aABc']}
dohvatljivi {'A': ['bcA', '$'], 'B': ['ad'], 'E': ['ed', 'ac'], 'S':
['bAbE', 'aABc']}
bez epsilon: {'A': ['bcA', 'bc'], 'B': ['ad'], 'E': ['ed', 'ac'], 'S':
['bAbE', 'aABc', 'bbE', 'aBc']}
bez jedinичnih {'A': ['bcA', 'bc'], 'B': ['ad'], 'E': ['ed', 'ac'], 'S':
['bAbE', 'aABc', 'bbE', 'aBc']}
```

Posljednja produkcija je produkcija koja je spremna na pretvorbu u Greibachov normalni oblik produkcija koji izgleda:

```
greibach {'A': ['bcA', 'bc'], 'c': ['c'], 'B': ['ad'], 'E': ['ed', 'ac'],
'd': ['d'], 'S': ['bAbE', 'aABc', 'bbE', 'aBc'], 'b': ['b']}
```

Na temelju produkcija u Greibachovom normalnom obliku kreiramo potisni automat za zadanu gramatiku. Najprije generiramo listu ulaznih znakova, listu znakova stoga, funkciju prijelaza,...:

```
lista_ul ['b', 'c', 'a', 'd', 'e']
lista_stoga: ['A', 'd', 'B', 'E', 'b', 'S', 'c']
FUNKCIJA PRIJELAZA PA
&(q, b , A )= cA
&(q, b , A )= c
&(q, a , B )= d
&(q, e , E )= d
&(q, a , E )= c
&(q, b , S )= AbE
&(q, a , S )= ABc
&(q, b , S )= bE
&(q, a , S )= Bc
&(q, b , b )= $
&(q, c , c )= $
&(q, d , d )= $
```

```
funkcija {'d', 'd'): ['$'], ('b', 'A'): ['cA', 'c'], ('e', 'E'): ['d'],
('a', 'S'): ['ABc', 'Bc'], ('b', 'S'): ['AbE', 'bE'], ('a', 'E'): ['c'],
('a', 'B'): ['d'], ('c', 'c'): ['$'], ('b', 'b'): ['$']}
```

Kao što je objašnjeno u tekstu ona je samo funkcija učitanoj znaku niza i trenutnog znaka na stogu (jer potisni automat ima samo jedno stanje o kojem ne ovisi akcija koja će se

izvršiti nad stogom). Ovdje je, također moguće primijetiti nedeterminizam izgrađenog potisnog automata. Za pojedine znakove niza i za znak sa vrha stoga moguća je više od jedne akcije, npr. za učitani znak 'a', ako se na stogu nalazi 'S'.

## 4.2. Primjer 2

Gramatika koja generira nizove oblika  $ab^jcd^kde^i$ . Produkcije zadane gramatike su sljedeće:

S→aAe	B→bBd	C→\$
A→bBd	B→cC	
A→aAe	C→cC	

Nakon što je program pozvan generirana je gramatika i pretvorena u normalan Greibachov oblik su sljedeći:

produkcije {'A': ['bBd', 'aAe'], 'S': ['aAe'], 'B': ['bBd', 'cC'], 'C': ['\$', 'cC']} zivi {'A': ['bBd', 'aAe'], 'S': ['aAe'], 'B': ['bBd', 'cC'], 'C': ['\$', 'cC']} dohvatljivi {'A': ['bBd', 'aAe'], 'S': ['aAe'], 'B': ['bBd', 'cC'], 'C': ['\$', 'cC']} epsilon {'A': ['bBd', 'aAe'], 'S': ['aAe'], 'B': ['bBd', 'cC', 'c'], 'C': ['cC', 'c']} jedinичne {'A': ['bBd', 'aAe'], 'S': ['aAe'], 'B': ['bBd', 'cC', 'c'], 'C': ['cC', 'c']} greibach {'A': ['bBd', 'aAe'], 'C': ['cC', 'c'], 'B': ['bBd', 'cC', 'c'], 'e': ['e'], 'd': ['d'], 'S': ['aAe']}
---

Uneseni niz: "aabcdee" izgrađeni potisni automat prihvaća na sljedeći način :

POČETAK ['S'] NIZ aabcdee za ulazni znak a za znak na stogu S = ['Ae'] STOG: ['e', 'A'] za ulazni znak a za znak na stogu A = ['Ae'] STOG: ['e', 'e', 'A'] za ulazni znak b za znak na stogu A = ['Bd'] STOG: ['e', 'e', 'd', 'B'] za ulazni znak c za znak na stogu B = ['C', '\$'] STOG: ['e', 'e', 'd', 'C'] nema definiranog prijelaza DPA SE NE PRIHVACA, POKUSAVAM DALJE SIMULIRATI NPA NA REZERVNOM STOGU: ['dee', ['e', 'e', 'd', 'C'], '\$'] STOG: ['e', 'e', 'd'] ZNAK d akcija d
--

```
STOG ['e', 'e'] ZNAK e akcija e
STOG ['e'] ZNAK e akcija e
STOG [] DPA SE PRIHVACA
```

Uneseni niz: "aabceded" izgrađeni potisni automat ne prihvaća :

```
POCETAK ['S']
NIZ aabceded
za ulazni znak a za znak na stogu S = ['Ae']
STOG: ['e', 'A']
za ulazni znak a za znak na stogu A = ['Ae']
STOG: ['e', 'e', 'A']
za ulazni znak b za znak na stogu A = ['Bd']
STOG: ['e', 'e', 'd', 'B']
za ulazni znak c za znak na stogu B = ['C', '$']
STOG: ['e', 'e', 'd', 'C']
nema definiranog prijelaza
DPA SE NE PRIHVACA, POKUSAVAM DALJE SIMULIRATI
NPA
NA REZERVNOM STOGU: ['ded', ['e', 'e', 'd', 'C'], '$']
STOG: ['e', 'e', 'd']
ZNAK d akcija d
STOG ['e', 'e'] ZNAK e akcija e
STOG ['e'] NPA SE NE PRIHVACA
```

### 4.3. Primjer 3

Za primjer pretvorbe gramatike u potisni automat uzeta je gramatika koja generira nizove  $\{0,1,2\}$  u kojoj nema uzastopnog ponavljanja niza 01 (zadatak preuzet iz auditornih vježbi uz dodane mrtve znakove, nedohvatljive, epsilon produkcije te jedinične produkcije):

Produkcije gramatike su:

```
S->nA|jS|dS|$|D
A->nA|jB|dS|$|E
B->nC|jS|dS|$
C->nA|dS|$
D->E
E->DF
F->njd
```

uz to dodani su mrtvi znakovi: D i E te nedohvatljiv znak F

Nakon što pozovemo program ispisi sređivanja gramatike su sljedeći:

```
produkcije {'A': ['nA', 'jB', 'dS', '$', 'ED'], 'C': ['nA', 'dS', '$'], 'B':
['nC', 'jS', 'dS', '$'], 'E': ['DF'], 'D': ['E'], 'F': ['njd'], 'S': ['nA',
'jS', 'dS', '$']}
zivi {'A': ['nA', 'jB', 'dS', '$'], 'C': ['nA', 'dS', '$'], 'B': ['nC',
'jS', 'dS', '$'], 'F': ['njd'], 'S': ['nA', 'jS', 'dS', '$']}
```

```
dohvatljivi {'A': ['nA', 'jB', 'dS', '$'], 'C': ['nA', 'dS', '$'], 'B':
['nC', 'jS', 'dS', '$'], 'S': ['nA', 'jS', 'dS', '$']}

epsilon {'A': ['nA', 'jB', 'dS', 'n', 'j', 'd'], 'C': ['nA', 'dS', 'n',
'd'], 'B': ['nC', 'jS', 'dS', 'n', 'j', 'd'], 'S': ['nA', 'jS', 'dS', 'n',
'j', 'd']}

jedinicne {'A': ['nA', 'jB', 'dS', 'n', 'j', 'd'], 'C': ['nA', 'dS', 'n',
'd'], 'B': ['nC', 'jS', 'dS', 'n', 'j', 'd'], 'S': ['nA', 'jS', 'dS', 'n',
'j', 'd']}

greibach {'A': ['nA', 'jB', 'dS', 'n', 'j', 'd'], 'C': ['nA', 'dS', 'n',
'd'], 'B': ['nC', 'jS', 'dS', 'n', 'j', 'd'], 'S': ['nA', 'jS', 'dS', 'n',
'j', 'd']}
```

Izgrađeni potisni automat je :

```
lista ulaznih: ['n', 'j', 'd']
['A', 'C', 'B', 'S']
lista_stoga: ['A', 'C', 'B', 'S']
FUNKCIJA PRIJELAZA PA
&(q, n , A )= A
&(q, j , A )= B
&(q, d , A )= S
&(q, n , A )= $
&(q, j , A )= $
&(q, d , A )= $
&(q, n , C )= A
&(q, d , C )= S
&(q, n , C )= $
&(q, d , C )= $
&(q, n , B )= C
&(q, j , B )= S
&(q, d , B )= S
&(q, n , B )= $
&(q, j , B )= $
&(q, d , B )= $
&(q, n , S )= A
&(q, j , S )= S
&(q, d , S )= S
&(q, n , S )= $
&(q, j , S )= $
&(q, d , S )= $
```

Kada se za jednak ulazni niz kao u drugom primjeru simulira treći primjer rezultat simulacije je sljedeći (primjer unošenja niza znakova koji nisu u skupu ulaznih znakova):

```
POCETAK ['S']
```

NIZ abcdee

znak a nije u skupu ulaznih

Te se simulacija zaustavlja.

za niz "njdnjd" (012012):

POCETAK ['S']

NIZ njdnjd

za ulazni znak n za znak na stogu S = ['A', '\$']

STOG: ['A']

za ulazni znak j za znak na stogu A = ['B', '\$']

STOG: ['B']

za ulazni znak d za znak na stogu B = ['S', '\$']

STOG: ['S']

za ulazni znak n za znak na stogu S = ['A', '\$']

STOG: ['A']

za ulazni znak j za znak na stogu A = ['B', '\$']

STOG: ['B']

za ulazni znak d za znak na stogu B = ['S', '\$']

STOG: ['S']

NPA SE NE PRIHVACA

## 5. Zaključak

Za potrebe prihvaćanja kontekstno neovisnih jezika gradi se potisni automat. Ponekad je jednostavnije napisati gramatiku koja generira određene jezike te gramatiku pretvoriti u potisni automat jednostavnim algoritmom, nego "napamet" graditi potisni automat.

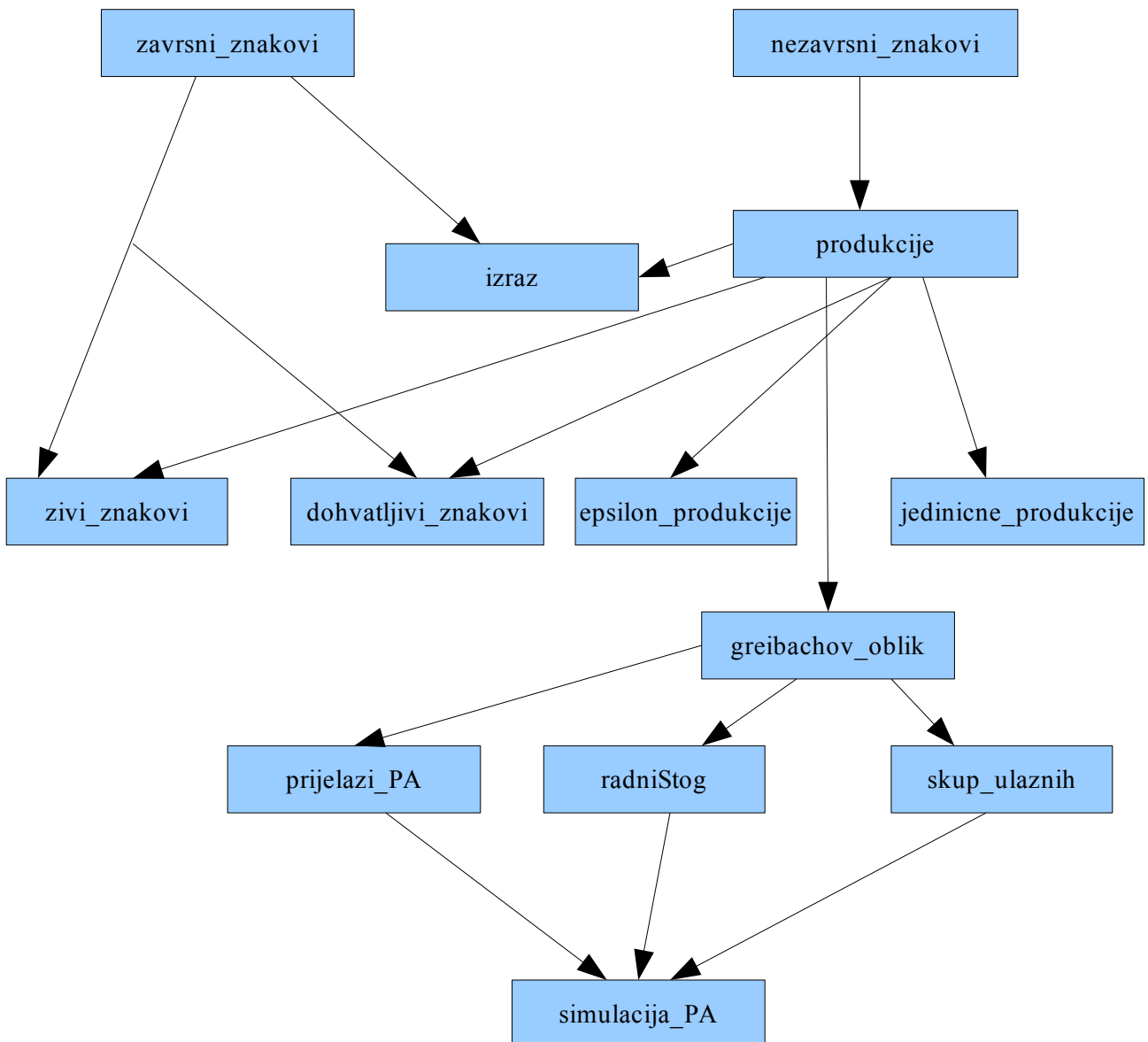
Svođenje gramatike na Greibachov normalni oblik može biti kompliciranije nego izgradnja samog potisnog automata, ali zato možemo biti sigurni da smo ukoliko smo dobro generirali gramatiku obuhvatili sve moguće slučajeve određenog problema za razliku ako PA gradimo iz glave. U ovom projektu objašnjeni su svi koraci koje je potrebno napraviti da bi se izgradio potisni automat iz kontekstno neovisne gramatike, te je simulacijom pokazana da potisni automat izgrađen iz kontekstno neovisne gramatike prema zadanim pravilima prihvaća jezik ako i samo ako ga zadana gramatika generira.

## 6. Dodatci

### 6.1. Popis klasa i pripadnih metoda

Klasa	Metoda	Kratak opis
završni_znakovi	dodaj_string	Za zadani niz znakova u listu dodaje one koji su završni
	dodaj_sign	Dodaje samo jedan znak u listu završnih znakova
	check_sign	Provjerava da li je znak u listi završnih znakova
	ispisi	Ispisuje listu završnih znakova
nezavršni_znakovi	dodaj_string	Za zadani niz znakova u listu dodaje one koji su nezavršni
	dodaj_sign	Dodaje samo jedan znak u listu nezavršnih znakova
	check_sign	Provjerava da li je znak u listi nezavršnih znakova
	ispis	Ispisuje listu nezavršnih znakova
produkcije	dodaj_produkciju	Za zadani nezavršni znak i desnu stranu dodaje u produkciju
	obrisi_produkciju	Za zadani nezavršni znak briše produkciju
	obrisi_desnu_stranu ispisi	Za zadani nezavršni znak i desnu stranu briše desnu stranu zadane produkcije Ispisuje produkcije
izraz	dodaj_u_produkciju	Iz stringa stvara riječnik produkcija
zivi_znakovi	dodaj_ziv	Dodaje znak u listu živih
	is_ziv	Provjerava da li je znak u listi živih
	provjeri_listu	Ispisuje listu živih znakova
dohvatljivi_znakovi	izbaci_mrtve_znakove	Implementacija algoritma za izbacivanje mrtvih znakova
	dodaj_dohvatljivi	Dodaje znak u listu dohvatljivih
	is_dohvatljiv	Provjerava da li je znak u listi dohvatljivih
	provjeri_listu	Ispisuje listu dohvatljivih znakova
epsilon_produkcije	izbaci_nedohvatljive_znakove	Implementacija algoritma za izbacivanje nedohvatljivih znakova
	is_epsilon	Provjerava da li je zadana produkcija epsilon
	riješ_epsilon	Implementacija algoritma bez brisanja epsilon produkcija
jedinicne_produkcije	brisanje_epsilon_produkcija_nakon_primjene_algoritma	Brisanje epsilon produkcija nakon primjene algoritma
	is_jedinicna	Provjerava da li je produkcija jedinična
	izbaci_jedinicne	Implementacija algoritma za izbacivanje jediničnih produkcija
greibachov_oblik	krajnje_lijevi	Implementacija algoritma za rješavanje krajnje lijevog nezavršnog znaka
	u_greibacha listu_ulaznih	Primjena koraka 2 i 3 za pretvorbu u Greibachov normalni oblik Gradi listu ulaznih znakova
skup_ulaznih	check_znak	Provjerava nalazi li se znak u skupu ulaznih
	ispisi	Ispisuje listu ulaznih znakova
radniStog	listu_stoga	Gradi listu znakova stoga
	check_znak	Provjerava da li je znak u listi znakova stoga
	dodaj_naStog	Dodaje niz znakova na stog
	ispisi_radni	Ispisuje izgled stoga
	skini	Skida jedan element sa stoga
	akcija_stog trenutno_stog	Provodi akciju stoga na temelju funkcije prijelaza PA Ispisuje trenutni znak na vrhu stoga
prijelazi_PA	generiraj_prijelaze	Na temelju produkcija u Greibachovom obliku generira funkciju prijelaza
simulacija_PA	funkcija_prijelaza	Ispisuje generiranu funkciju prijelaza
	DPA_simulacija	Ako izgrađeni PA nije NPA simulira DPA
	NPA_simulacija	Ako je NPA, nakon odabira DPA prelazi na NPA simulaciju

## 6.2. Hijerarhijska struktura klasa



## 7. Literatura

1. S. Srbljić, *Jezični procesori 1*, Element, Zagreb, 2000.
2. Wikipedia, URL: [http://en.wikipedia.org/wiki/Python\\_\(programming\\_language\)](http://en.wikipedia.org/wiki/Python_(programming_language)), (2.1.2007)
3. Funkcijsko programiranje u Pythonu,  
URL: <http://fly.srk.fer.hr/~ipozgaj/etc/tekstovi/fpup.html> (30.12.2006)
4. Introduction to OOP with Python,  
URL: <http://www.voidspace.org.uk/python/articles/OOP.shtml#the-init-method>  
(4.1.2007)