­Simulator LR(1) parsera

Ivan Karačić

0036432354

Sadržaj

[1. Uvod 1](#_Toc219277777)

[2. Parsiranje 2](#_Toc219277778)

[2.1. Općenito o parserima i parsiranju 2](#_Toc219277779)

[2.1.1. Parsiranje od vrha prema dnu 3](#_Toc219277780)

[2.1.2. Parsiranje od dna prema vrhu 3](#_Toc219277781)

[2.2. LR(k) parser 3](#_Toc219277782)

[2.2.1. LR stavka 4](#_Toc219277783)

[2.3. LR(1) parser 5](#_Toc219277784)

[2.3.1. Izgradnja LR(1) parsera 5](#_Toc219277785)

[3. Simulator LR(1) parsera 7](#_Toc219277786)

[3.1. Generiranje pseudokoda 7](#_Toc219277787)

[3.2. Programsko rješenje 7](#_Toc219277788)

[3.3. Upute za rad sa simulatorom 8](#_Toc219277789)

[3.3.1. Parsiranje niza 8](#_Toc219277790)

[3.3.2. Primjer parsiranja niza 9](#_Toc219277791)

[3.3.3. Primjer rada simulatora 11](#_Toc219277792)

[4. Zaključak 13](#_Toc219277793)

# Uvod

Zbog sve veće upotrebe osobnih računala u raznim ljudskim djelatnostima, kao primjerice pretraživanje baza podataka, kupovina preko računala, obrada digitalnih slika i videosnimki stvorila se potreba za izgradnjom raznoraznih sustava. Problem pri tomu predstavlja korisnikovo nepoznavanje najosnovnijih programskih paradigmi. Postoje veliki broj sustava koji su koncipirani upravo za korisnički rad tj. oni prirodni jezik, kojim se koristi čovjek, pretvore u neki oblik razumljiv računalu koji nakon toga obavlja određene akcije (primjerice, pretraživanje globalne mreže *Internet*). Taj korak prema razumljivosti velikoj većini ljudi doveo je do problema kako određene dijelove prirodnog jezika prebaciti u strojni jezik. Rješenje tom problemu je *parser* tj. proces zvan *parsiranje*

# Parsiranje

## Općenito o parserima i parsiranju

Parsiranje je postupak prepoznavanja niza i gradnja generativnog stabla na temelju zadanih produkcija kontekstno neovisne gramatike. Dani postupak omogućavaju nam *parseri*. Parseri raščlanjuju ulazne podatke na manje, prema određenom skupu pravila koji opisuju njegovu strukturu. Većina podataka tako se smanjuje do određene razine.

Tako, primjerice, telefonski oblika (00385) 01 1234-123 broj možemo rastaviti na *pozivniBrojDržave, pozivniBrojGrada, brojDijelaGrada(prefiks), brojKuće*(sufiks). Sljedeća gramatika definira izgradnju telefonskih brojeva:

<brojTelefona> ( <pozivniBrojDržave> ) <pozivniBrojGrada> <brojDijelaGrada> – <brojKuće>

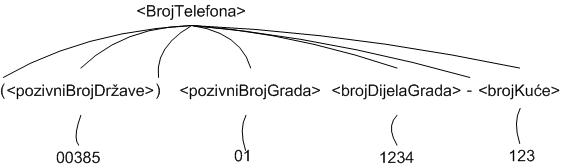
<pozivniBrojGrada> prirodniBroj

<brojDijelaGrada> prirodniBroj

<brojKuće> prirodniBroj

Svako pravilo gramatike, produkcija, opisuje kompoziciju simbola. Znak „ “ se interpretira kao „ kompozicija je simbola“. Prvu produkciju, dakle čitamo: „Broj telefona kompozicija je pozivnog broja države, pozivnog broja grada, broja dijela grada i broja kuće“. Sve ostale produkcije čitamo: „Pozivni broj države, pozivni broj grada, broj dijela grada i broj kuće kompozicija su prirodnih brojeva“.

Nakon definiranja sintakse podataka pomoću gramatičkih pravila, parser može koristiti danu gramatiku za postupak parsiranja, tj. raščlanjivanju ulaznih podataka na manje podatke. Kao izlaz parser daje stablo. Stablo nastalo parsiranjem opisuje i hijerarhijsku strukturu unešenog niza. Tako se u našem slučaju generira stablo:



Slika Generirano stablo

Parsiranje je postupak u kojem se traži simbol u gramatici koji podudara sa znakom u ulaznom nizu. Rezultirajuće stablo je preslika gramatičkih simbola na ulazne podatke.

Iako se parseri najviše koriste pri izgradnji jezičnih procesora, oni svoju primjenu nalaze i u rutinskim programskim zadacima poput iščitavanja teksta iz formatiranih datoteka, ispitivanju ispravnosti određenih podataka (poput e-mail adresa u digitalnim formularima) i slično. Zapravo je potreba za parserima prati razvoj tehnologije koja sve više postaje informacijski orijentirana.

Postoje dvije vrste parsiranja:

* Parsiranje od vrha prema dnu
* Parsiranje od dna prema vrhu

Ukratko ćemo opisati obadva postupka kako bismo bolje razumjeli način rada LR(1) parsera.

### Parsiranje od vrha prema dnu

Parsiranje od vrha prema dnu započinje gradnju generativnog stabla vrhom generativnog stabla koji je označen početnim nezavršnim znak gramatike. Gradnja stabla nastavlja se primjenom produkcija gramatike na čvorove generativnog stabla koji su označeni nezavršnim znakovima gramatike. Ako je oznaka čvora jednaka lijevoj strani produkcije, onda se ta produkcija primjeni na promatrani čvor. Gradnja generativnog stabla nastavlja se sve dok ima čvorova označenih nezavršnim znakovima gramatike. Izgradi li se generativno stablo koje ima listove označene završnim znakovima niza *w*, niz *w* jest u zadanom jeziku

Tijekom parsiranja znakovi se čitaju slijedno slijeva nadesno, te se na temelju pročitanog znaka i znakova lijeve i desne strane produkcije primjeni odgovarajuća produkcija gramatike. Učinkovitost procesa parsiranja ovisi o preciznosti kojom je moguće odrediti koju produkciju je potrebno primijeniti.

### Parsiranje od dna prema vrhu

Parsiranje od dna prema vrhu započinje gradnju generativno stabla listovima. Listovi stabla označeni su završnim znakovima gramatike. Gradnja stabla nastavlja se primjenom desnih strana produkcija gramatike na prethodno izgrađene čvorove. Budući da se u svakom koraku smanjuje (reducira) ili ostaje isti broj znakova u međunizu, produkcije se u postupku parsiranja od dna prema vrhu nazivaju *redukcije*. Gradnja generativnog stabla nastavlja se do korijena stabla. Ako se izgradi generativno stablo koje ima korijen označen početnim nezavršnim znakom gramatike, niz  *w*  je u jeziku koji generira zadana gramatika. Znakovi niza *w* čitaju se slijedno slijeva nadesno. Na temelju pročitanih znakova izabere se odgovarajuća redukcija. Učinkovitost postupka parsiranja ovisi o preciznosti kojom je moguće odrediti koju je redukciju potrebno primijeniti u danom koraku parsiranja.

## LR(k) parser

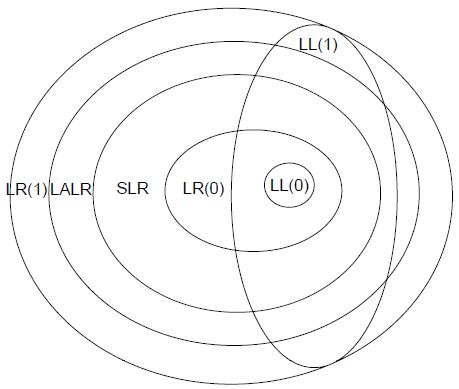
LR(k) parsiranje je parsiranje od dna prema vrhu, gdje je k≥0. Iz oznake parsera zaključujemo sljedeće:

* Znak L označava da se niz čita slijeva nadesno.
* Znak R označava da se stablo gradi obrnutim postupkom generiranja niza zamjenom krajnjeg desnog nezavršnog znaka.
* Broj k označava da se redukcija primjenjuje na temelju pročitanih svih znakova koji se generiraju iz znakova desne strane produkcije i još sljedećih k znakova ulaznog niza

Kao što je navedeno LR(k) parsiranje je parsiranje od dna prema vrhu te se pri tom postupku učitava veći broj znakova. To omogućava primjenu LR(k) parsera na široku klasu jezika.

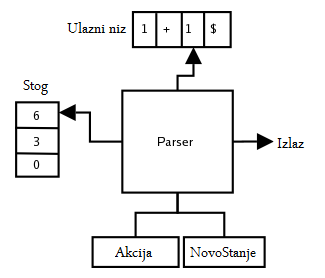
LR parseri nisu jednostavni za izgraditi te su se iz tog razloga razvili određeni postupci koji ga izgrađuju na osnovu zadanih produkcija zadane gramatike:

* SLR ( Simple LR) – jednostavan, ali nije primjenjiv na velikom skupu jezika
* Kanonski LR – složeniji , obuhvaća skup jezika LR(1)
* LALR (Look Ahead LR) – jednostavniji za programsku izvedbu, širi skup jezika u odnosu SLR parser, ali manji u odnosu na kanonski LR parser



Slika Veličine podržanih skupova pojedinih LR parsera

Pri LR parsiranju koristimo tehniku *Pomakni-Reduciraj*. Parserom se u tom slučaju upravlja s dvije tablice: *Akcija* i *NovoStanje*. Stanja u tablici *Akcija* predstavljaju stanja parsera. Njih dobivamo kodiranjem pojedinih znakova gramatike LR parsera. To stanje se, zajedno s pročitanim znak na ulazu, u proširenoj akciji *Pomakni(Stanje)* stavlja na stog te se na osnovu njega određuju daljnje akcije. Ukoliko se na stog stavi nezavršni znak sljedeće stanje se određuje na osnovu drugog dijela tablice, *NovoStanje.* Ta se tablica isključivo koristi tijekom izvođenja akcije *Reduciraj*.



Slika Arhitektura LR parsera

### LR stavka

*LR stavka* je produkcija gramatike koja ima oznaku točke na proizvoljnom mjestu svoje desne strane, uključujući krajnje lijevo krajnje desno mjesto. U slučaju ε-produkcije A ε, LR stavka A •

Gramatika produkcije:

S A a

A b A|a

Ima sljedeću LR stavku:

S •A a A • b A A • a

S A• a A b • A A a •

S A a • A b A •

Ako je oznaka točke na krajnje desnom mjestu desne strane produkcije, onda se kaže da je LR stavka *potpuna*. Navedena gramatika ima tri potpune LR stavke: S A a •, A b A •, A a •.

## LR(1) parser

Oznakama LR stavki dodaju se izračunati skupovi {a1,a2,…an}. Proširena LR stavka je LR(1) stavka. Parser koji se gradi primjenom LR(1) stavki je LR(1) parser, jer odlučuje o primjeni akcije na temelju znakova *w* koji se generiraju iz znakova desne strane produkcije i dodatno još jednog znaka koji slijedi znakove niza *w*. Gramatiku za koju je moguće izgraditi LR(1) parser je LR(1) gramatika. LR(1) stavka zapisuje se na sljedeći način:

A α • β, {a1,a2,…an}

gdje je {a1,a2,…an} skup završnih znakova gramatike uključujući i oznaku kraja niza, u ovom slučaju Ł.

### Izgradnja LR(1) parsera

LR(1) parser izgrađujemo u tri koraka:

U prvom koraku izgrađujemo ε-NKA, iz kojeg se gradi DKA. Pomoću tog DKA se na kraju izgrađuje tablica LR(1) parsera:

U prvom koraku izgrađujemo ε-NKA

1. Skup stanja je skup svih LR(1) stavki gramatike G uvećan početnim stanje q0. Osim početnog stanja q0 sva ostala stanja su LR(1) stavke.
2. Skup ulaznih znakova je unija skupova završnih i nezavršnih znakova gramatike, Σ=TυV
3. Budući da su sva stanja valjana LR(1) stavka ulaznog niza γ, onda su sva stanja skupu dozvoljenih stanja, F = Q
4. Funkcija prijelaza δ definira se na sljedeći način:
   1. δ(q0,ε) = {S •α,{ Ł} | S α je produkcija gramatike G}

δ ((A α•Χβ, {a1,a2,…an}),Χ) = {A αΧ•β, {a1,a2,…an }}

* 1. Ako automat pročita znak X, onda se oznaka točke pomakne iza tog znaka. Znak X je završni ili nezavršni znak gramatike
  2. δ ((A α•Bβ, {a1,a2,…an}),ε) = {B γ•,T|B γ je produkcija gramatike G}

Ako je nezavršni znak B desno od točke, onda ε-prijelaz pokreće analizu tog znaka. T je skup svih završnih znakova b ili oznaka kraja niza Ł za koje vrijedi :

* Znak b započinje niza β odnosno *bєZAPOČINJE(*β*).*
* Ako je moguće iz znakova niza β generirati prazni niz ε,odnosno β ε

U drugom koraku se iz danog ε-NKA gradi DKA. Taj korak nećemo posebno opisivati. Cijeli postupak je opisan u knjizi *Uvod u teoriju računarstva, Siniša Srbljić.*

Treći korak, izgradnja LR(1) tablice, odvija se u sljedećim koracima:

1. Gramatika G proširi se novim početnim stanjem S' i produkcijom S' S. Na temelju preuređene gramatike G' izgradi se DKA koji prihvaća skup živih prefiksa gramatike G'. Neka je skup stanja DKA jednak Q = {0, 1, 2, .., n}.
2. Reci tablice *Akcija* i *NovoStanje* označavaju se stanjima DKA. Stupci tablice *Akcija* označavaju se završnim znakovima gramatike i oznakom kraja niza Ł, a stupci tablice *NovoStanje* označavaju se nezavršniim znakovima gramatike
3. Tablica *Akcija* popunjava se na sljedeći način:
   1. Ako je LR(1) stavka A α•Bβ, {a1,a2,…an} u stanju *s* i ako je δ(s,a) = t prijelaz DKA, onda se za završni znak gramatike *a* i stanje *s* definira *Akcija*[s,a] = *Pomakni(t).* Akcija *Pomakni(t)* stavlja stanje *t* na vrh stoga.
   2. Ako je LR(1) stavka A α•, {a1,a2,…an} u stanju *s*, onda se za sve završne znakove *a* koji su iz skupa {a1,a2,…an} i stanje *s* definira *Akcija*[s,a] = *Reduciraj(A α).* Akcija redukcije se ne primjenjuje za početni nezavršni znak gramatike S'.
   3. Ako je LR(1) stavka S' S•,{Ł} u stanju *s*, onda se za oznaku kraja niza Ł i stanje *s* definira *Akcija[s, Ł] = Prihvati().*
4. Tablica *NovoStanje* popunjava se na sljedeći način:
   1. Ako je δ(s, A) = t prijelaz DKA, Onda se za nezavršni znak *A* i stanje *s* definira *NovoStanje[s, A] = Stavi(t).*
5. Svi elementi za koje nije definirana akcija u koracima 3. I 4. Označavaju akciju *Odbaci()*, odnosno niz se ne prihvaća.
6. Početno stanje označeno je LR(1) stavkom S' S,{Ł}

# Simulator LR(1) parsera

## Generiranje pseudokoda

Zadatak ovog projekta je programski ostvariti simulator LR(1) parsera. Na osnovu rada LR(1) napisali smo sljedeći pseudokod:

1. Učitaj tablicu prijelaza i tablicu produkcija
2. Pročitaj znak na kojem se nalazi glava za ulazni niza i stanje na vrhu stoga
3. Pronađi *akciju* za dani ulazni znak i stanje na stogu
   1. Ukoliko je učitana akcija P prihvati niz
   2. Ukoliko akcija ne postoji odbaci niz i prekini parsiranje
   3. Ukoliko akcija postoji
      1. Pročitaj o kojoj se akciji radi ( r ili p )
      2. Ukoliko je akcija p
         1. Spremi učitani znak na stog
         2. Spremi stanja koji se nalazi u akciji (npr. akcija r7, stanje je 7) na stog
         3. Pomakni glavu za jedno mjesto u desno
         4. Vrati se na korak 1.
      3. Ukoliko je akcija r
         1. Učitaj odgovarajuću redukciju (npr. redukcija r1) u memoriju
         2. Izvadi sa stoga dvostruko veći broj elemenata od znakova s desne ¸ strane produkcije
         3. Pročitaj stanje na vrhu stoga i spremi ga u *privremenoStanje*
         4. Spremi nezavršni znak s lijeve strane produkcije na stog
         5. Pronađi akciju za *privremenoStanje* i nezavršni znak na stogu te to stanje (npr. s6) spremi ga na stog
         6. Vrati se na korak 1.

## Programsko rješenje

U programskom rješenju kojeg smo izradili u jeziku C# slijedili smo dani pseudokod te smo s nekoliko if-else naredbi ostvarili cijeli sustav. Način rada cijelog simulatora te upute kako ga koristi opisat ćemo u sljedećem poglavlju, dok ćemo ovdje samo opisati funkcionalnost pojedinih dijelova programa.

Simulator je izgrađen od ukupno četiri funkcije:

* *UcitajTablice()*
* *UcitajProdukcije()*
* *StvoriDKA()*
* *obradaUlaznogNiza()*

**UcitajTablice()** i **UcitajProdukcije()** su funkcije koje se pozivaju prilikom svakog novog zahtjeva za parsiranje. *UcitajTablice()* učitava tekstualnu datoteku u kojoj je definirana tablica prijelaza te simulatoru prosljeđuje putanju kojom se dolazi do datoteke s tablicom prijelaza. *UcitajProdukcije()* obavlja identični zadatak kao i *UcitajTablice()* s jedinom razlikom što ona simulatoru vraća putanju prema tekstualnoj datoteci koja sadrži popis produkcija dane gramatike.

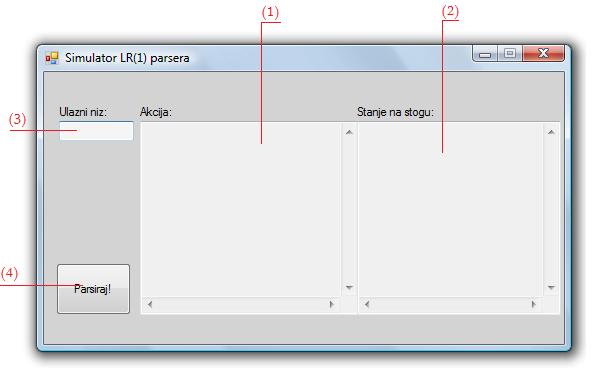
***StvoriDKA()*** funkcija uzima putanje pojedinih tablica te ih obrađuje.Prema pravilu koje smo naveli prije tablica prijelaza je podijeljena u dva dijela. U prvom dijelu nalaze se akcije za svaki pojedini učitani znak, dok su u drugom dijelu tablice smještene oznake novih stanja. Prilikom učitavanja tablice ona se, znak po znak, kopira u dvodimenzionalno polje *formiranaTablica*  te se kasnije radnje simulatora baziraju samo na čitanju akcija u polju. Ukoliko se nije postavila putanja prema tablici prijelaza *StovriDKA()* javlja grešku. Ukoliko je tablica prijelaza uspješno stvorena stvara se i tablica produkcija. Kao i u prvom koraku stvaranja tablice prijelaza i ovdje se tekstualna datoteka sprema u polje, samo u ovom slučaju jednodimenzionalno. Ukoliko dođe do greške javit će se poruka o grešci.

***obradaUlaznoNiza()*** je funkcija koja koristi podatke dobivene nakon stvaranja DKA automata. Nakon što se učita znak sa ulaznog niza ispituje se postoji li akcija za dani ulazni zna. Ukoliko ne postoji javlja se greška i prekida se proces parsiranja. Ukoliko postoji akcija čita se prvi znak akcije te se na temelju toga odlučuje je li riječ o akciji stavljanja na stog ili reduciranju elemenata na stogu.

Prilikom stavljanja novog znaka na stog pokazivač na trenutni znak (varijabla *trenutniZnak*) se pomiče na sljedeći znak. Ukoliko se, pak, radi o redukciji elemenata na stogu pokazivač se neće pomaknuti. Na kraju obadva slučaja ispisat će se rezultat akcije u obliku *izvršena akcija*  i *stanje na stogu.*

## Upute za rad sa simulatorom

Osnovna funkcionalnost opisana je slikom 4. Na njoj vidimo kako u simulatoru imamo dva polja za ispis. Jedno nam pokazuje koju akciju trenutno izvršavamo (1), a drugo trenutno stanje na stogu u nakon obavljanja samo akcije (2). Uz to je još postavljeno polje za unos niza koji će se parsirati(3) te gumb kojim pokrećemo simulator(4).



Slika . Izgled simulatora

### Parsiranje niza

Kako biste uspješno koristili simulator potrebno je prvo unijeti ulazni niz te nakon toga,pritiskom na gumb „Parsiraj!“ odabrati, prvo, tablicu prijelaza i zatim, drugo, tablicu produkcija. Nakon toga će se obaviti parsiranje niza te će vas simulator obavijestiti o uspješnosti parsiranja.

Prvo ćemo pokazati jedan primjer parsiranja bez simulatora, a zatim ćemo isto to učiniti pomoću simulatora kako bi se uvjerili u ispravnost istog.

### Primjer parsiranja niza

Tablica prijelaza koju ćemo u obadva primjera koristi je sljedeća:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Stanje | Akcija | | | Novo Stanje | | |
| b | c | Ł | <S> | <A> | <B> |
| 0 | p1 |  |  |  |  |  |
| 1 | p2 |  |  |  | s3 |  |
| 2 | p5 |  |  |  |  | s6 |
| 3 | p4 |  |  |  |  | s7 |
| 4 |  |  | r3 |  |  |  |
| 5 |  | r3 |  |  |  |  |
| 6 |  | p8 |  |  |  |  |
| 7 |  |  | r1,P |  |  |  |
| 8 | r2 |  |  |  |  |  |

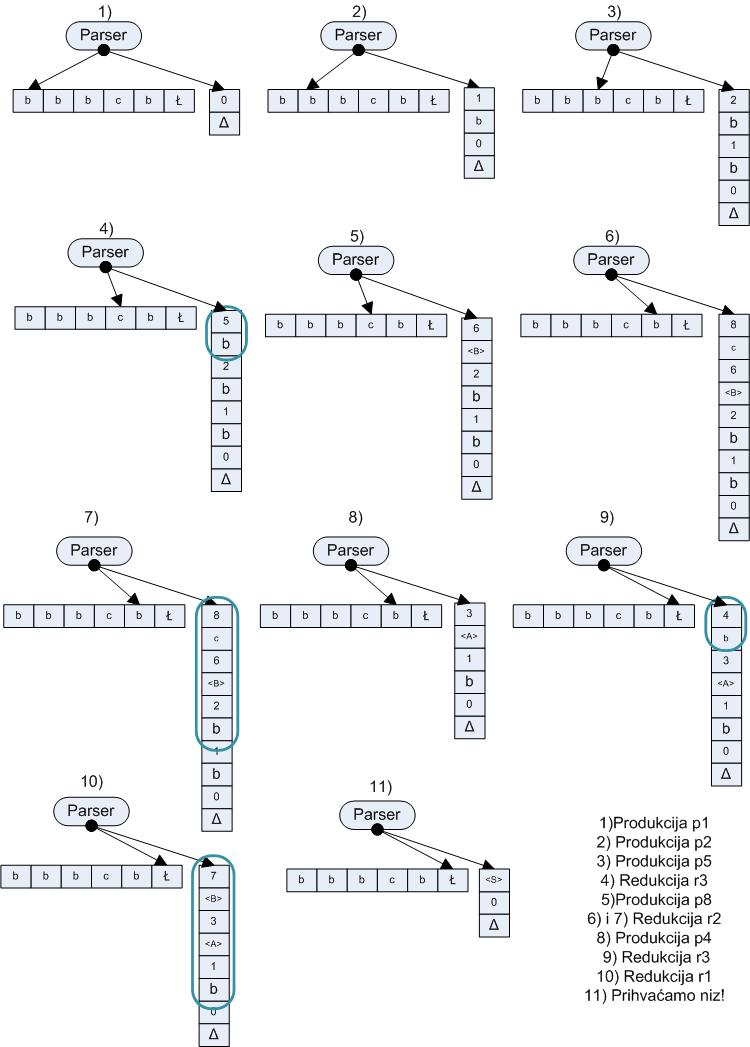
Produkcije koje koristimo su:

r1 = *Reduciraj* ( <S> b <A> <B>)

r2 = *Reduciraj* ( <A> b <B> c )

r3 = *Reduciraj* ( <B> b )

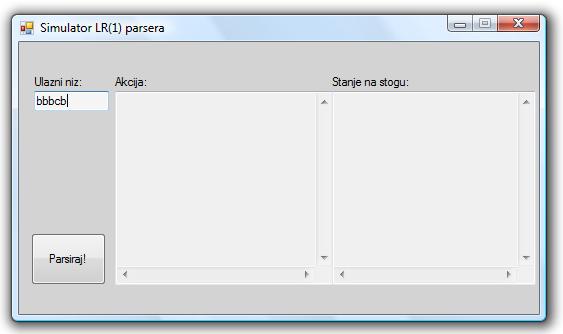
Niz koji ćemo parsirati ( a koji je ujedno i jedini niz koji ova gramatika može generirati ☺) je ***bbbcb***



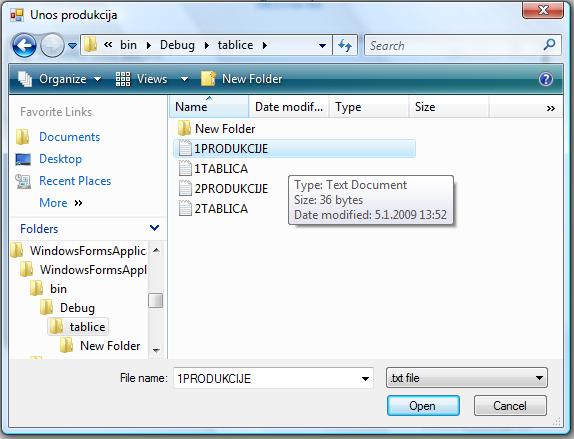
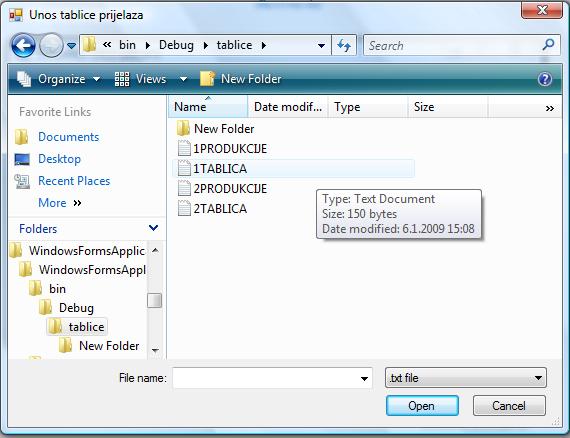
Slika Parsiranje niza

### Primjer rada simulatora

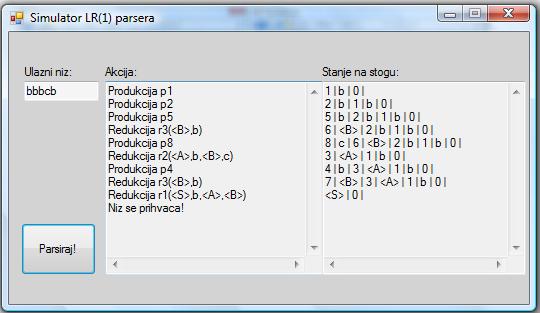
Nakon što smo unijeli niz koji želimo parsirati i kliknuli na gumb „Parsiraj“(slika 6), otvorila su nam se, jedan za drugim, dva prozora. U prvom biramo tablicu prijelaza (slika 7), dok u drugom odabiremo odgovarajuću tablicu s produkcijama (slika 8). Nakon toga će simulator ispisivati pojedina akcije parsiranja i izvijestiti nas o uspješnosti (slika 9) ili neuspješnosti (slika 10) parsiranja.



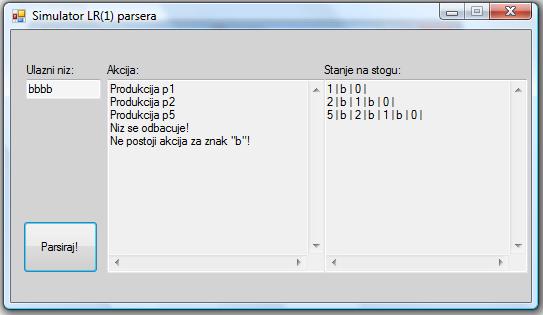
Slika Unos niza



Slika Unos tablice prijelaza Slika Unos produkcija



Slika Uspješno parsiranje



Slika Neuspješno parsiranje i dojava greške

Simulator u slučaju neuspješnog parsiranja javlja sljedeće greške:

* Ne postoji akcija za znak „x“!
* Učitani znak „x“ ne postoji u gramatici
* Greška s tablicama ( ukoliko je učitana pogrešna tablica)
* Greška na stogu ( ukoliko se pokuša skinuti više znakova nego što je moguće)

**Važno: potrebno je pridržavanje spomenutih koraka kako bi se omogućilo uspješno parsiranje!**

# Zaključak

LR(k) porodica parsera predstavlja moćan alat pri izgradnji jezičnih procesora. Relativno složena, programska, izvedbom rezultira širokim skupom podržanih jezika omogućava stvaranje sustava čija je primjena sve veća i sve bliža prirodnom jeziku. LR(1) parser kao jedan od najsloženijih parsera obuhvaća jedan od najvećih skupova jezika. U ovom projektu izgradili smo simulator jednog LR(1) parsera te smo se upoznali s njegovim načinom rada.