Sveučilište u Zagrebu
Fakultet elektrotehnike i računarstva

Petar Dučić

Seminarski rad iz predmeta «Prevođenje programskih jezika»

Zadatak broj 51
(teža inačica)

Zagreb, siječanj 2009.

Seminarski rad iz predmeta Prevođenje programskih jezika

Student: Petar Dučić
Matični broj studenta: 0036427438

Zadatak broj 51, teža inačica:
Programski ostvariti generator tablice LALR parsera. Ulaz u program je datoteka slobodnog formata koja sadrži zadanu gramatiku. S obzirom na složenost izračunavanja SLIJEDI i ZAPOČINJE skupova, radi jednostavnijeg ostvarenja generatora, ulazna datoteka treba sadržavati SLIJEDI i ZAPOČINJE skupove za sve nezavršne znakove.***

****NAPOMENA : zbog preglednosti, datoteke u kojima su zadane produkcije, odnosno 'ZAPOCINJE' skupovi, učitavaju se odvojeno. (uz suglasnost Mr. sc. Daniela Skrobe)*

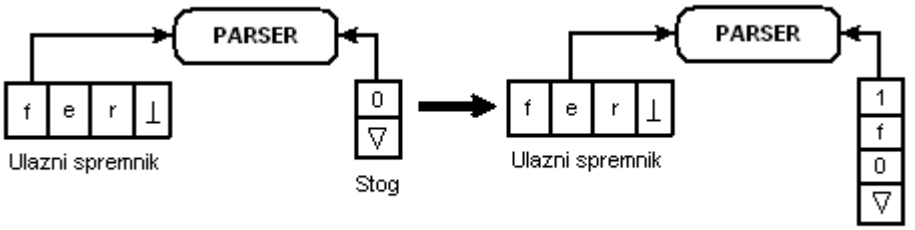
1. UVOD

LALR (Look- Ahead LR) je postupak izgradnje tablice LR parsera. LR(k) parsiranje, gdje je k>=0 je parsiranje od dna prema vrhu. Znak 'L' označava da se niz čita slijeva nadesno, dok znak 'R' označava da se sintaksno stablo gradi postupkom generiranja niza zamjenom krajnje desnog nezavršnog znaka. Broj 'k' označava da se redukcija primjenjuje na temelju pročitanih svih znakova koji se generiraju iz znakova desne strane produkcije, te još *k* sljedećih znakova ulaznog niza.

Razvijeno je više postupaka koji grade LR parser izravno na temelju produkcija zadane gramatike. Koristi se više postupaka različite složenosti. Ja ću nabrojati one osnovne: SLR (Simple LR), zatim kanonski LR algoritam i zadnji, ali ne i posljednji: LALR postupak, kojem ćemo se detaljnije posvetiti. Za nas je važan i kanonski LR parser, koji obuhvaća LR(1) klasu jezika, jer se LALR postupak izgradnje tablice zasniva na spomenutom, kanonskom LR parseru.

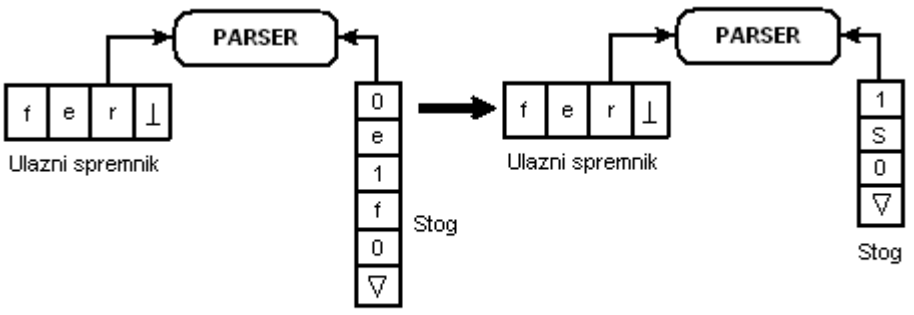
1.1.RAD LR PARSERA

LR parser koristi tehniku parsiranja *Pomakni/Reduciraj*. To znači da parser iz trenutnog ulaznog i samo jednog znaka na vrhu stoga, odnosno stanja parsera, može odrediti koja će se akcija obaviti. Konkretno, za akciju *Reduciraj*, nije potrebno čitati više znakova sa stoga ne bi li se odredila točna redukcija. Parserom se upravlja primjenom dviju tablica, tablica *Akcija* i tablica *Novo Stanje*. Redovi tih tablica određuju stanja parsera, dok stupci određuju završne, odnosno nezavršne znakove gramatike.



Slika 1.1. Pomakni: ukoliko za ulazni znak 'f' i stanje parsera '0' imamo akciju Pomakni(1)

Tablica *Akcija* popunjena je akcijama *Pomakni(stanje),* koje na stog stavljaju ulazni znak i stanje LR parsera(slika 1.1), akcijama *Prihvati,* koje prihvaćaju niz, te akcijama *Reduciraj(produkcija)* (slika 1.2), koje znakove sa vrha stoga zamjenjuju lijevom stranom produkcije i stanjem koje se izračunava iz tablice *Novo Stanje.* Ta tablica sadrži isključivo *Stavi(stanje)* akcije, koje za stanje, koje je ostalo na vrhu stoga nakon redukcije, i nezavršni znak s lijeve strane produkcije određuju novo stanje parsera. To bi, ukratko, bio postupak parsiranja LR parserom. Međutim, moj zadatak u ovom seminarskom radu jest izgraditi upravo jedan ovakav par tablica na temelju zadanih produkcija i 'ZAPOČINJE' skupova za svaki nezavršni znak.



Slika 1.2. Redukcija: ukoliko za ulazni znak 'r' i stanje '0' imamo akciju Reduciraj(S->fe), te u tablici Novo stanje za nezavršni znak 'S' i stanje '0' akciju Stavi(1)

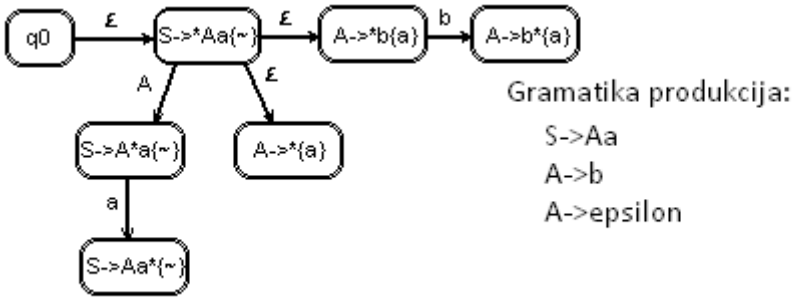
1.2.KONSTRUKCIJA TABLICE KANONSKOG LR PARSERA

LR stavke je produkcija gramatike koja ima oznaku točke na proizvoljnom mjestu svoje desne strane uključujući krajnje lijevo i krajnje desno mjesto. U slučaju *epsilon* produkcije , S->ε, LR stavka jest S->*. Ukoliko je oznaka točke na krajnje desnom mjestu, kažemo da je stavka potpuna.
Primjer stavki za produkciju S->ABc:

S->*ABc
S->A*Bc
S->AB*c
S->ABc* (potpuna stavka)

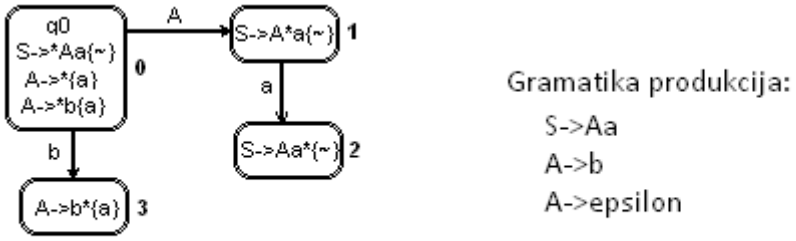
LR(1) stavka zapisuje se na sljedeći način: S->α*β{a,b}, gdje je {a,b} skup završnih znakova gramatike, uključujući i oznaku kraja niza.
Neka je G=(V,T,P,S) kontekstno neovisna gramatika za koju vrijedi da početni nezavršni znak S nije na desnoj strani nijedne produkcije. Generiranje tablice kanonskog LR parsera provodi se kroz niz koraka: konstrukcija ε-NKA, iz ε-NKA gradnja DKA te iz DKA definiranje tablice.
Prvi korak u gradnji tablice kanonskog parsera je izgradnja ε-NKA automata. Gradi se na slijedeći način:

1. Skup stanja je skup svih LR(1) stavki u uniji sa početnim stanjem q_0.
2. Skup ulaznih znakova je unija skupova nezavršnih i završnih znakova gramatike.
3. Sva stanja su u skupu dozvoljenih stanja.
4. Funkcije prijelaza δ definiraju se na slijedeći način:
   a)δ(q_0, ε)=S->*α{~}, gdje je S->α produkcija gramatike, a znak '~' oznaka kraja niza Ovaj prijelaz omogućuje početak rada nedeterminističkog stroja.
   b)δ((A->α*Xβ{a,b,c}),X)=A-> αX*β{a,b,c}
   c) δ((A->α*Xβ{a,b,c}), ε)=X->*γ{Y}, gdje je X-> γ produkcija gramatike, a Y skup svih završnih znakova *p* odnosno oznaka kraja niza, za koje vrijedi:
      1.Znak *p* je element skupa ZAPOČINJE(β)
      2. Ako je iz znakova niza β generirati prazan niz, onda je *p* iz skupa {a,b,c}



Slika 1.3. Primjer epsilon-NKA za zadanu gramatiku produkcija. Znak '~' označava kraj niza

Iz opisanog ε-NKA gradimo DKA. Ostvarivanje DKA iz postojećeg ε-NKA je trivijalno i neće se objašnjavati u ovom seminaru.



Slika 1.4. Konstrukcija DKA za zadanu gramatiku produkcija. Znak '~' označava kraj niza

Dobiveni DKA koristimo dalje za popunjavanje tablica *Akcija* i *Novo Stanje:*

1. Gramatika se proširuje novim početnim stanjem S' i produkcijom S'->S, te se za dobivenu gramatiku izgradi pripadajući DKA'.

2. Tablica *Akcija* popunjava se na slijedeći način:
   - Ako je LR(1) stavka A->α*aβ{a,b,c} u stanju *x* i ako je δ(x,a)=*y* prijelaz našeg DKA', onda za završni znak *a* i stanje *x* definiramo akciju *Pomakni(y).* Konkretno, za primjer sa slike 1.4., za stanje DKA' *S->A*a{~}* i završni znak *a* definira se akcija *Pomakni(S->Aa*{~}).*
   - Ako je LR(1) stavka A->α*{a,b,c} u stanju *x*, tada za sve završne znakove *a,b,c* i stanje *x* definiramo akciju *Reduciraj(A->α).* Akcija redukcije se ne primjenjuje za početni nezavršni znak gramatike S'.
   - Ako je LR(1) stavka *S'->S*{~}* u stanju *x*, tada za oznaku kraja niza ~ i stanje *x* definiramo akciju *Prihvati().*

3. Tablica *Novo Stanje* popunjava se:
   - Ako je δ(x,A)=y definiran prijelaz DKA, tada za nezavršni znak *A* i stanje *x* definiramo akciju *Stavi(y).*

4. Svi ostali, 'nepopunjeni' elementi tablice označavaju akciju *Odbaci().*

5. Početno stanje označeno je LR(1) stavkom *S'->*S{~}*

Stanje	**Akcija(a)**	**Akcija(b)**	**Akcija(~)**	**Novo Stanje(S)**	**Novo Stanje(A)**
0	Reduciraj(A->)	Pomakni(3)			Stavi(1)
1	Pomakni(2)				
2			Prihvati()		
3	Reduciraj(A->b)				

Tablica 1.1. Tablica kanonskog LR parsera za primjer DKA sa slike 1.4.

1.3.KONSTRUKCIJA LALR TABLICE PARSERA

Osnovna ideja LALR postupka gradnje LR tablice je grupiranje stanja označenih istim LR(0) stavkama u jedinstveno stanje.
Na primjer, ako je stanje *x* DKA kanonskog parsera definirano kao:

X->x*Y{a,b},

a stanje *y* definirano:

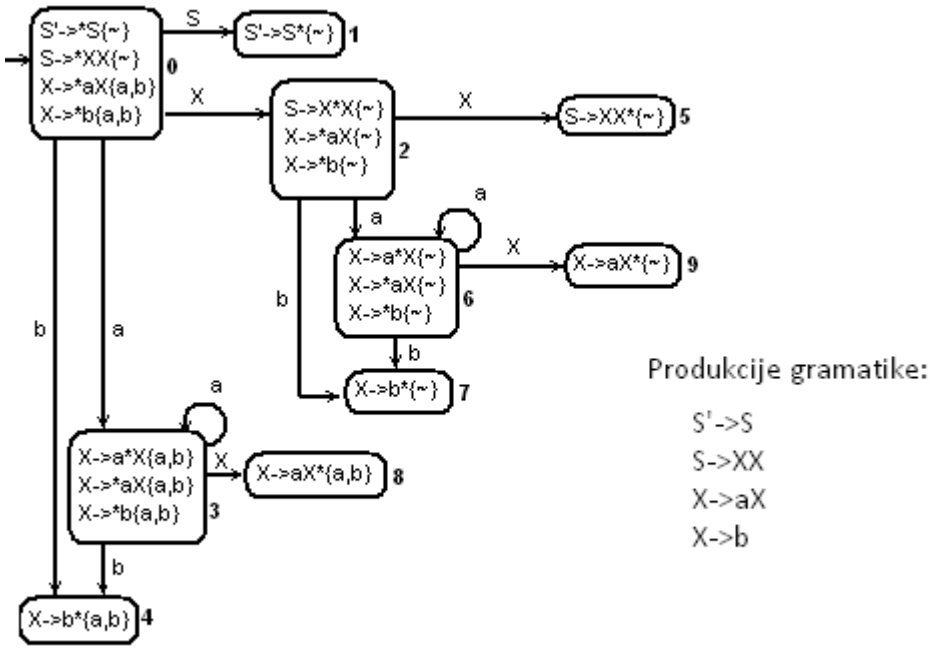
X->x*Y{a,c},

grupiranjem stanja *x* i *y* u jedinstveno stanje *xy* koje je označeno slijedećom stavkom:

X->x*Y{a,b,c}, jer je {a,b} U {a,c} = {a,b,c}.

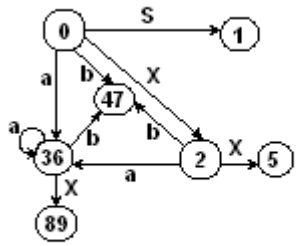
Kao rezultat dobivamo DKA sa manje stanja, a samim time i manju tablicu LR parsera. Međutim, to može dovesti do *Reduciraj/Reduciraj* proturječnosti. To znači, da grupiranjem dva stanja koja sadrže različite stavke, a skupovi završnih znakova za koje se provode akcije *Reduciraj* su im identični, dobivamo proturječje, gdje ne znamo koju redukciju provesti. Konkretno, ako u stanju *x* imamo stavku X->a*{b,c}, a u stanju *y* stavku Y->a*{b,c}, te izgradnjom LALR parsera udružimo ta dva stanja, za završne znakove *b* i *c* ne znamo koju redukciju provesti, *X->a* ili *Y->a*.

Zaključak bi bio da se LALR parser može koristiti za parsiranje više jezika od SLR, a opet manje od kanonskog parsera. LALR parser je popularan upravo zbog tog omjera veličine tablice koju generira i broja gramatika koje može parsirati. Koristi ga i program *Yacc (Yet another compiler- compiler),* koji na temelju zadanih produkcija gramatike gradi sintaksni analizator za zadani jezik.



Slika 1.5. DKA kanonskog parsera

Primjerice, za produkcije gramatike sa slike 1.5. gradi se DKA oblika:



Slika 1.6. Primjer DKA LALR parsera zasnovanog na kanonskom parseru sa Slike 1.5.

Iz dobivenog DKA, tablica se gradi na identičan način kao pri izgradnji tablice kanonskog parsera:

Stanje	Akcija(a)	Akcija(b)	Akcija(~)	Novo Stanje(S)	Novo Stanje(X)
0	Pomakni(36)	Pomakni(47)		Stavi(1)	Stavi(2)
1			Prihvati()		
2	Pomakni(36)	Pomakni(47)			Stavi(5)
36	Pomakni(36)	Pomakni(47)			Stavi(89)
47	Reduciraj(X->b)	Reduciraj(X->b)	Reduciraj(X->b)		
5			Reduciraj(S->XX)		
89	Reduciraj(X->aX)	Reduciraj(X->aX)	Reduciraj(X->aX)		

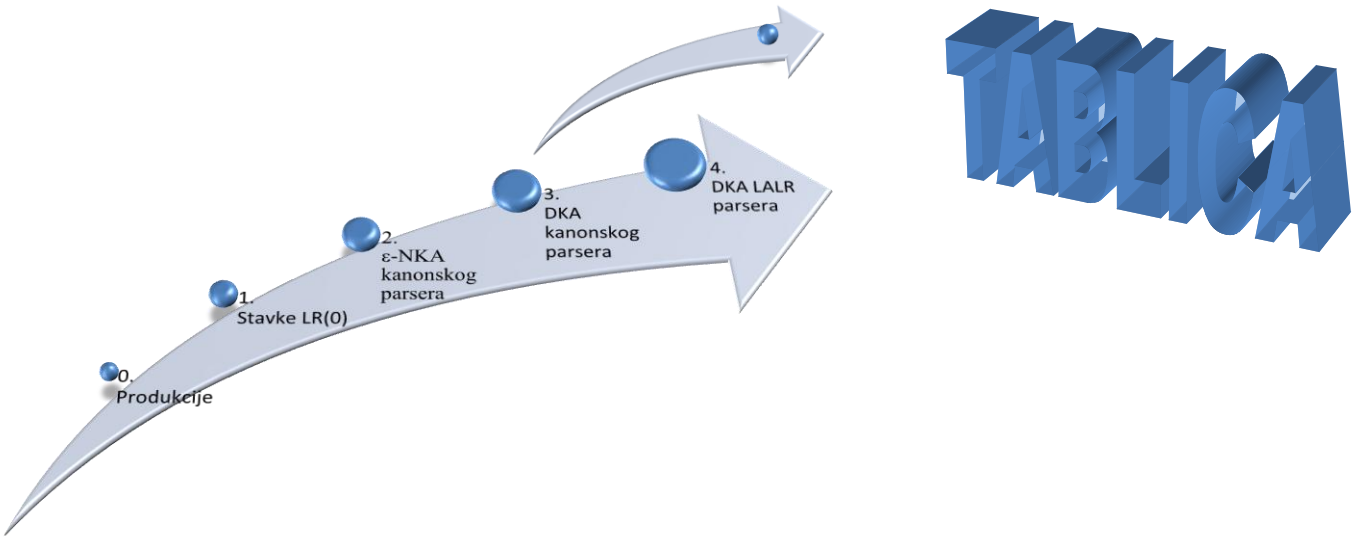
Tablica 1.2. Tablica LALR parsera za gramatiku produkcija sa Slike 1.5.

To bi bilo ukratko o LALR postupku generiranja tablice LR parsera. U nastavku će biti objašnjeno kako je programski ostvaren taj postupak, što je ujedno i glavni zadatak ovog seminarskog rada.

2. OSTVARENJE

Generator tablice LALR parsera programski je ostvaren pomoću jezika C# (C sharp). Razlog odabira baš ovog jezika jest njegova široka primjenjivost, te prilična intuitivnost pri učenju i shvaćanju. Moram priznat da u početku rada projekta nisam bio nimalo upoznat sa ovih jezikom, što će možda iskusno oko primijetiti iz koda.

Program prolazi kroz određene faze generiranja tablice. Prvo se iz zadanih produkcija izračunavaju LR(0) stavke. Zatim se izračunavaju prijelazi ε-NKA kanonskog parsera. Iz prijelaza ε-NKA se potom izvode prijelazi DKA. Spajaju se stanja koja sadrže iste LR(0) stavke, te se iz dobivenih stanja i prijelaza generira tražena LALR tablica. Cijeli je ovaj postupak detaljno i postepeno objašnjen u uvodnom dijelu seminara.



Slika 2.1. Postupak izgradnje tablice LALR/kanonskog parsera.

Pošto je tablica LALR parsera generirana pomoću DKA kanonskog parsera, ostavio sam mogućnost generiranja tablice kanonskog parsera, što se može primijetiti iz Slike 2.1. Stavke LR(0) ostvarene su pomoću 'string'-ova međusobno povezanih u listu, dok su ε-NKA i DKA(obje vrste) ostvareni kao struktura od dva 'string'-a, te jednog 'charactera'. Prvi 'string' označava početno, drugi 'string' odredišno stanje, dok 'character' označava prijelaz.

2.1. SUČELJE PROGRAMA

Slika 2.2. 'Otvori' dijalog

Ulazne jedinice u program jesu popis produkcija gramatike, te 'ZAPOČINJE' skupovi za sve nezavršne znakove gramatike.
Produkcije se zadaju pomoću **.txt** datoteke proizvoljnog imena. Produkcije se zapisuju u obliku 'S->aA', **bez razmaka** između znakova, te svaka nova produkcija u novi red. Nezavršni znakovi gramatike označuju se jednim velikim slovom, dok se završni označavaju malim slovom abecede.
Nezavršni znak lijeve strane produkcije koja je **prva** zapisana u nizu produkcija (u prvom redu), biti će uzet kao početni znak gramatike. Epsilon produkcije se zapisuju u obliku 'A->'. Također treba paziti na produkcije koje nemaju riješenu lijevu rekurziju, tipa $A->A\alpha$.
Na sva ova ograničenja treba posebno paziti, jer će u suprotnom rezultirati greškom/netočnim rješenjem.
Datoteke se zapisanim produkcijama i 'ZAPOČINJE' skupovima se otvaraju se pritiskom na tipku 'Otvori' u alatnoj traci, te dalje pritiskom na odgovarajuću tipku, što je prikazano na Slici 2.2.
'ZAPOČINJE' skup svih nezavršnih znakova zadaje se obliku 'S={a,b}', gdje su znakovi *a* i *b* elementi skupa 'ZAPOČINJE' nezavršnog znaka *S*. Također se zapisuju bez razmaka, te svaki skup u posebnom redu. Bitno je napomenuti da je potrebno zadati 'ZAPOČINJE' skup za svaki nezavršni znak, jer će se u suprotnom dogoditi greška.
Kada ste učitali željene datoteke, potrebno je odabrati želite li generirati tablicu LALR ili kanonskog parsera. Pritiskom na tipku 'Generiraj!' pokreće se rad programa. Kao rezultat rada, generira se tablica odgovarajućeg parsera, što je prikazano na Slici 2.3.

STANJE	AKCIJA(~)	AKCIJA(a)	AKCIJA(b)	NOVO STANJE(Z)	NOVO STANJE(S)	NOVO STANJE(X)
[0]		Pomakni 2	Pomakni 6		Stavi 4	Stavi 1
[1]		Pomakni 2	Pomakni 6			Stavi 5
[2]		Pomakni 2	Pomakni 6			Stavi 8
[4]	Prihvati					
[5]	Reduciraj S->XX					
[6]	Reduciraj X->b	Reduciraj X->b	Reduciraj X->b			
[8]	Reduciraj X->aX	Reduciraj X->aX	Reduciraj X->aX			

Slika 2.3. Primjer tablice LALR parsera, za produkcije gramatike sa Slike 1.5.

Valja napomenuti da su stanja ispremiješana u odnosu na teorijsko rješavanje zadatka u uvodnom dijelu seminara. Još treba spomenuti, da stanje, koje nastaje pri spajanju stanja kod generiranja LALR DKA, dobiva ime po stanju sa manjim brojem u imenu. Konkretno, na Slici 2.3. vidimo da su stanja *2* i *3* spojena u zajedničko stanje, koje se zove '2'. Također, isto

vrijedi i za stanja *6* i *7*, te stanja *8* i *9*. Na Slici 2.4. vidimo tablicu kanonskog parsera, na kojoj su prisutna sva stanja, te se jasno vidi razlika između tablice sa Slike 2.3.

Slika 2.4. Tablica kanonskog parsera, za produkcije gramatike sa Slike 1.5.

Nakon generiranja odgovarajuće tablice, prilično se mijenja izgled programa. Pošto sam još uvijek podosta neiskusan programer, a vrijeme mi je sve samo ne saveznik, program je (trenutno) samo za jednokratnu upotrebu. Naime, nakon generiranja tablice, potrebno je izaći iz programa te ponovno ući u isti ukoliko želimo generirati još koju tablicu. Zbog toga se tipka 'Generiraj!' pretvara u tipku 'Izađi', a dijalog za otvaranje datoteka postaje neupotrebljiv. Pri radu programa, generiraju se određene datoteke, radi kontrole i boljeg razumijevanja tijeka izvođenja.

Ime datoteke	Faza u kojoj se generira	Kratki opis sadržaja
STAVKE.txt	**1**	Sve stavke LR(0). Pojam stavki objašnjen je u uvodnom dijelu seminara.
epsilon-NKA.txt	**2**	Svi prijelazi ε-NKA u obliku '*stanje1 prijelaz stanje2*'. Princip izgradnje prijelaza također objašnjen u uvodu.
DKA.txt	**3**	Svi prijelazi DKA kanonskog parsera, u obliku istom kao ε-NKA.
STANJA.txt	**3**	Sadrži sve LR(1) stavke sa prefiksom koji određuje u kojem se stanju nalaze.
BROJAC.txt	**4**	Pomoćna datoteka u kojoj su navedene sve stavke svakog stanja, te broj istih po stanju. Uz pomoć ove datoteke obavlja se usporedba stanja DKA kanonskog parsera.
DKA_LALR.txt* **	**4**	Svi prijelazi DKA LALR parsera, zapisani u istom obliku kao DKA i ε-NKA.
STANJA_LALR.txt* **	**4**	Sve LR(1) stavke sa prefiksom stanja u kojem se nalaze.

Tablica 2.1. Datoteke generirane tokom izvođenja.

****Ukoliko generiramo tablicu kanonskog parsera, ove se datoteke neće generirati. Također, ukoliko su tablice kanonskog i LALR parsera jednake, ove datoteke će biti identične 'DKA.txt' odnosno 'STANJA.txt'*

3. ZAKLJUČAK

LALR postupak generiranja LR tablice je u potpunosti egzaktan algoritam. Točno se zna kako se radi, te nema odstupanja kao ni u jednom matematičkom procesu. Već smo ranije učili taj postupak, tako da me ovaj seminar sa te strane nije mogao puno tome naučiti. Međutim, s druge strane, sa strane samog programerskog posla, sam naučio prilično mnogo. Što i nije za čuđenje, pošto mi je ovo bio daleko najveći projekt što sam morao sam iznjedriti.
Prije svega, valja napomenuti da ovaj zadatak, ma koliko možda lagano izgledao na papiru, zahtjeva prilično kompleksno razmišljanje. Tako da sam znao po par dana mozgati oko jednog problema i pokušavajući ga 'pobijediti'. Mislim da se kompleksnost razmišljanja u nekim trenutcima može zaključiti i iz koda, čiji neki dijelovi i meni samom izgledaju nevjerojatno i neshvatljivo, već samo par sati nakon što sam ih pisao.
To je bila prva lekcija, gdje sam shvatio da moj budući posao programera iziskuje više živaca nego posao jedne tete u vrtiću. Na trenutke. Međutim, osjećaj da ste uspjeli nešto konstruirati iz ničega, te taj vaš produkt funkcionira, opravdava sve te trenutke živčanih slomova. Vjerujem da sa veličinom i važnosti programa u zajednici, pa na kraju krajeva i financijskom nagradom, taj osjećaj raste eksponencijalno.
Naučio sam, zatim, veliku važnost dobre organizacije. U početku rada projekta, razdijelio sam si poslove, te si postavio rokove do kojih bi isti trebali biti završeni. Mada, istini za volju, i uz takav 'savršen' plan, prilično mi je nedostajalo vremena za finaliziranje i finese.
Također, valja napomenuti da sam se uz ovaj projekt naučio i osnovama C#, koji se u novije doba postavlja kao svojevrstan standard u pisanju programa. Sa ponuđenim opcijama i funkcijama, te iznimno laganim povezivanjem između klasa, prilično je intuitivan za shvaćanje, što je i jedan od razloga zašto je toliko sveprisutan u zadnje vrijeme.
Međutim, otkrio sam i neke loše strane ovoga posla, koje su na kraju krajeva i opće poznate. Ali kad čovjek nešto osjeti na vlastitoj koži, odnosno vratu, očima ili zglobovima, tek onda počne neke stvari shvaćati ozbiljno. Naravno, radi se o stalnom sjedenju, te svakodnevnom i maratonskom gledanju u ekran. Posljedice se osjećaju vrlo brzo, a o onim dugoročnim je bolje niti ne razmišljati. No, zato i postoje raznorazne tjelesne aktivnosti.
I tako sve sjeda na svoje mjesto... U ovom slučaju, ja za računalo.