Sveučilište u Zagrebu

Fakultet elektrotehnike i računarstva

Zoran Horvat

**1. seminarski rad - Samostalni studentski projekt**

“**Prevođenje programskih jezika**”

Zadatak broj 2037

Zagreb, prosinac 2010.

**Druga domaća zadaća iz predmeta “Uvod u teoriju računarstva”**

**Student:** Zoran Horvat

**Matični broj studenta:** 0036447803

**Zadatak broj 2037:** Pronaći i proučiti 3 alata slična po namjeni alatu YACC, odnosno alata za generiranje sintaksnih analizatora. Odabrani alati mogu generirati kod sintaksnog analizatora u bilo kojem programskom jeziku. Opisati i usporediti mogućnosti odabranih alata. Odabrati

proizvoljnu ili konstruirati novu gramatiku jednostavnog programskog jezika te prikazati

korištenje odabranih alata generiranjem sintaksnih analizatora za istu odabranu gramatiku.

Usporediti rad generiranih sintaksnih analizatora. Samostalno definirati ulazni niz (tablicu

uniformnih znakova) koji će se prenijeti sintaksnim analizatorima - nije potrebno ostvarivati i

leksičku analizu. Uz dokumentaciju priložiti datoteke s definicijom gramatike kao ulaze u

alate, odabrane alate i generirane parsere.

# Uvod – Generator sintaksnog analizatora

Generator sintaksnog analizatora je alat koji generira kod sintaksnog analizatora za zadani formalni opis nekog jezika. Uobičajen oblik alata je takav da na ulazu prima gramatiku koja opisuje jezik nad kojim se analiza provodi, koja je obicno predstavljena u BNF notaciji, a na svom izlazu daje izvorni kod sintaksnog analizatora koji je obicno dio većeg projekta, najčešće je dio jezičnog procesora koji prevodi programski jezik.

Za potrebe ovog seminara korišteni su slijedeći alati – *UltraGram, Grammatica, TinyPG.* Upravo ti alati su odabrani jer su pisani prvenstveno za operacijski sustav *Windows* te zbog jednostavnosti korištenja, inteligentnog korisničkog sučelja, vrlo dobre funkcionalnosti i određenih dodatnih mogućnosti, o čemu će više biti u nastavku. Korišteni alati su priloženi uz seminar, kao i korištene definicije gramatika, te izvorni kodovi generiranih sintaksnih analizatora dobivenih korištenjem priloženih alata.

Jednostavna gramatika koja će se koristiti za ispitivanje funkcionalnosti je slijedeća:

<S> → a <E> a <E> → <S> e

<S> → b <E> b <E> → e

<S> → a <E> b

<S> → b <E> a

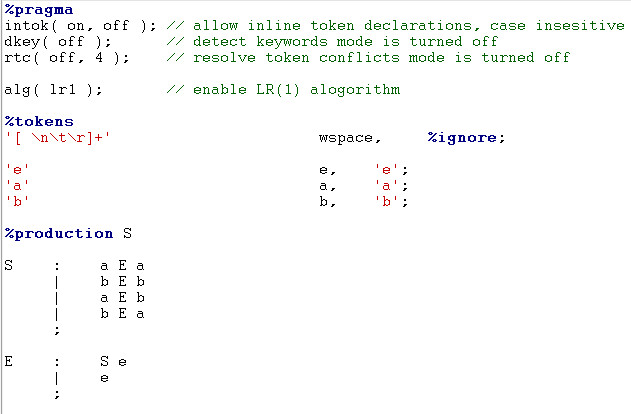
# Alati

U ovom poglavlju će biti opisane mogućnosti pojedinog korištenog alata te će biti opisan način uporabe glavnih funkcionalnosti. Slijedi opis alata *UltraGram*, *Grammatica*  i *TinyPG*.

## UltraGram

Prilikom pokretanja programa, kako bismo bilo što mogli činiti, moramo napraviti novi projekt ili učitati već postojeći. Obje opcije možemo naći u izborniku *File* na izborničkoj traci. Kad je stvoren novi projekt, kreirati će se dvije datoteke – *.grm* i *.txt*. U prvoj je sadržana definicija ulaza, gramatike, u generator sintaksnog analizatora. Druga datoteka se koristi za testiranje generiranog sintaksnog analizatora i u nju je upisan ulazni niz korišten za testiranje.

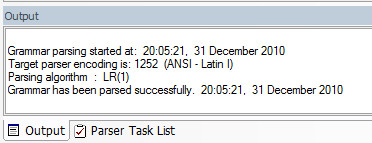
Definicija ulaza spomenuta u uvodu se u *.grm* datoteci zadaje se kao što je prikazano na slici 1.



Slika 1. Definicija ulaza u generator sintaksnog analizatora programa *UltraGram*

Za potrebe naše definicije koristi se nekoliko sekcija definicije. Sekcije *%pragma*, *%tokens* i *%productions*. U prvoj sekciji, nakon linije *%pragma* se zadaju pojedine direktive vezane uz procesuiranje gramatike i niza koji se testira. Posebno je zanimljiva direktiva *alg().* Među zagradama te opcije zadajemo kojim algoritmom će se vršiti parsiranje. Program podržava parsiranje LR(1), LALR(1) i GLR algoritmom. Ukoliko se eksplicitno ne zada koji će se algoritam koristiti, parsiranje se vrši LALR(1) algoritmom. Nakon linije *%tokens* zadajemo završne znakove gramatike te nakon linije *%productions* zadajemo produkcije gramatike. Produkcije gramatike zadaju se u obliku *<lijeva\_strana> : <desna\_strana>;* te za isti nezavršni znak se produkcije odvajaju znakom ”|” (AltGr + W), kao što možemo viđeti na slici 1.

Da bi se moglo nastaviti s radom programa, potrebno je provjeriti gramatiku pritiskom na tipku F7 ili izborom sa izborničke trake *Build -> Compile.* Nakon što je uspješno prošla provjera gramatike iz *.grm* datoteke, viđeti ćemo istu poruku o tome u konzoli na dnu prozora programa, kako je prikazano na slici 2.

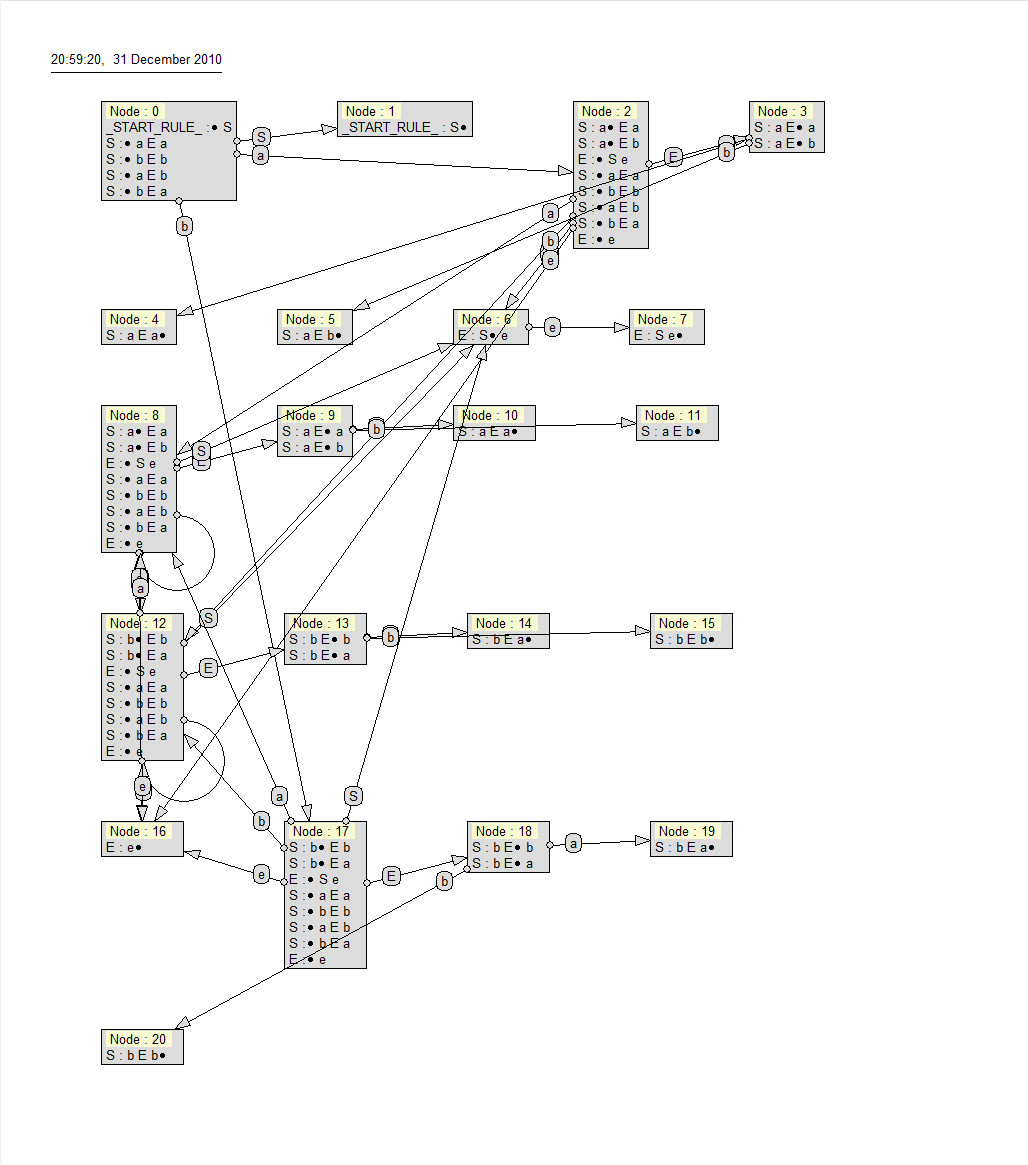


Slika 2. Konzola programa *UltraGram*

Slijedeće što želimo učiniti jest generirati kod sintaksnog analizatora. Program *UltraGram* pruža mogućnost generiranja izlaza u četiri različita programska jezika – *Java*, *C++*, *C#* i *VB.NET*. To možemo učiniti pritiskom na *Build -> Generate* te odabirom željenog izlaznog programskog jezika. Za potrebe ovog primjera, odabran je *VB.NET*. Generirani izlaz je priložen u mapi gđe se nalazi i instalacijska datoteka programa *UltraGram*. Sintaksni analizator temeljen na toj gramatici se sastoji od dvije datoteke, jedna sa klasom koja sadrži glavnu funkcionalnost analizatora, i većim dijelom je jednaka za svaku gramatiku, te druga koja sadrži podatkovne strukture i članove specifične za našu gramatiku.

Prije ispisa generiranog analizatora u datoteke, bilo bi dobro testirati funkcionalnost istog. To se može učiniti tako da dodamo novu *.txt* datoteku za testiranje (ukoliko vec jedna nije dodana prilikom kreiranja novog projekta) pritiskom na *File -> New* te u nju zapišemo naš ulazni niz, nakon čega pritiskom na gumb *Start debugger* iz alatne trake pokrecemo sintaksnu analizu ulaznog niza. Ukoliko je provjera prošla uspješno, u konzoli će se pojaviti poruka o tome. Ukoliko nije, ispisat će se poruka o neuspješnosti provjere te će u nizu biti obilježeno na kojem dijelu je greška nastala. Također je moguće *debuggiranje* korak po korak (uniformni znak po znak) kao u bilo kojem naprednijem razvojnom okruženju nekog programskog jezika.

Prilično korisna funkcionalnost alata jest generiranje jednostavnog i razumljivog grafa DKA iz zadane gramatike. Crtanje automata se pokrece odabirom *Build -> Build DFA Table* ili, naravno, odabirom iste opcije iz alatne trake. Konkretno za našu gramatiku generirani automat izgleda kao što je prikazan na slici 3.

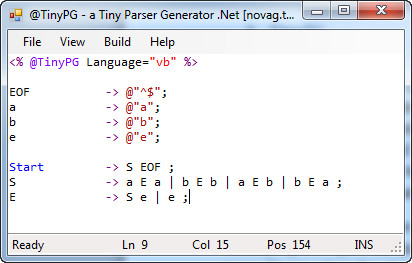
Slika 3. Deterministički konačni automat generiran pomoću alata *UltraGram,* na temelju zadane gramatike.

Postoji još panel koji pruža uvid u stablo parsiranja niza koji se testira nakon pritiska *Debug* gumba. Naravno, tu su još i standardne opcije koje svaki program za uređivanje teksta ima (*Print preview, Save, Copy*, ...) i one neće biti detaljnije obrađene.

## TinyPG

*TinyPG* ili *Tiny Parser Generator*, imenom odgovara implementaciji. Najjednostavniji je od sva tri, a i mnogo jednostavniji od većine generatora sintaksnog analizatora. Izgledom korisničkog sučelja podsjeća na neki jednostaviniji program za uređivanje teksta, na primjer *Notepad*. Nepostojanje naprednijeg grafičkog sučelja odmah nakon pokretanja mu je najveća razlika između tri programa. Funkcionalnošću i lakoćom uporabe lako parira programima *UltraGram* i *Grammatica*.

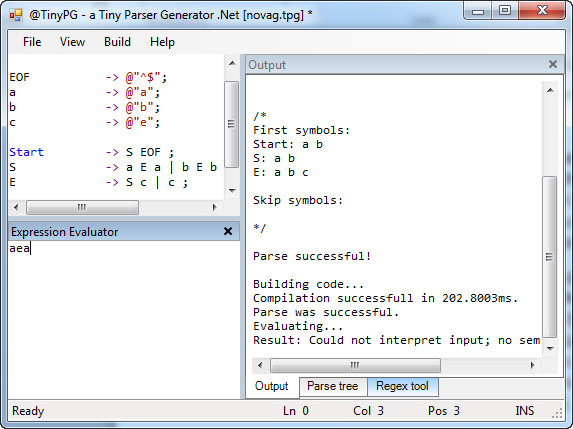
Gramatika iz koje će se generirati kod sintaksnog analizatora unosi se u (prilično veliko pri prvom pokretanju) polje za unos teksta. Sintaksa potrebna za ispravan unos gramatike je vrlo jednostavna. U prvoj liniji se zadaje izlazni jezik, jezik u kojem će kod sintaksnog analizatora biti generiran. Dalje se zadaju završni znakovi gramatike kao regularni izrazi te produkcije gramatike u EBNF sintaksi. Unos naše gramatike i izgled jednostavnog sučelja prikazan je na slici 4.



Slika 4. Korisničko sučelje i unešena gramatika u programu *TinyPG*

Program nudi generiranje koda sintaksnih analizatora u jezicima *C#* i *VB*. Podržani algoritmi su LL(1) i LL(k). Pretpostavljeni korišteni algoritam je LL(1) i korištenje istog će rezultirati ljepšim kodom, dok će LL(k) pojednostaviti pisanje gramatike. Za potrebe uratka pretpostavljeni algoritam je sasvim dovoljan. Produkcije gramatike se zadaju u obliku *<lijeva\_strana> -> <desna\_strana>.* Također, za isti nezavršni znak se produkcije mogu odvajati znakom “|” (AltGr + W). Riječ *Start* je u ovom programu ključna riječ i označava početnu produkciju gramatike. Iskoristili smo ju za prijelaz u početnu produkciju S radi lakšeg dodavanja novih produkcija.

Da bi se generirao kod sintaksnog analizatora potrebno je odabrati sa izborničke trake *Build -> Generate* ili pritisnuti tipku F6. Ukoliko želimo unutar *TinyPG* isprobati rad našeg sintaksnog analizatora, pritisne se CTRL + E ili odabere *View -> Expression evaluator* kako bismo pozvali panel za upisivanje izraza te upisali niz koji želimo testirati. Ukoliko je uneseni niz sintaksno ispravan i generiranje koda prošlo uspješno, dobiti ćemo poruku *“Parse was successful.”* nakon *poruke “Compilation successfull in ...”* u konzoli programa kako je prikazano na slici 5.



Slika 5. Testiranje sintaksnog analizatora uz potvrdu u konzoli.

Razlog upozorenju u zadnjoj liniji ispisa jest taj što ne postoji implementirana logika iza naših pravila produkcija.

Slijedeće što želimo učiniti je pogledati kod generiranog sintaksnog analizatora. Prikaz koda je omogućen pritiskom na *Build -> View parser code.* Izvorni kod će se otvoriti programom za uređivanje teksta pretpostavljenim za otvaranje *.vb* datoteka u ovom slučaju budući da se radi o kodu u programskom jeziku VisualBasic. Drugi slučaj je da smo generirali kod u C#, tada bi se radilo o izlaznoj datoteci s nastavkom *.cs.*

Također kao i prethodni alati, *TinyPG* pritiskom na *Build -> Generate & Run* daje mogućnost uvida u stablo parsiranja testiranog izraza upisanog u *Expression Evaluator* panel. Zanimljiva mogućnost alata je *Regex tool*, do kojeg se dolazi pritiskom CTRL + R i omogućava lakše rukovanje regularnim izrazima u većim projektima. Već su upisani neki popularni regularni izraz te je također moguće dodavati nove koji bi mogli biti ponovo korišteni i moguće je brzo i lako pretraživanje istih kako bi se došlo do željenog izraza.

Nedostatak alata je nemogućnost prikaza automata, bilo DKA ili NKA, iz zadane gramatike.

## Grammatica

*Grammatica* je alat koji se koristi iz komandne linije, što znači da nema grafičko sučelje. Prilično je jednostavan za korištenje jer uz njega dolazi i detaljna dokumentacija.

Kao ulaz prima *.grammar* datoteku te za uspješno generiranje sintaksnog analizatora iz nje potrebno je da se sastoji od tri dijela – dio koji počinje linijom *%header%,* dio s početkom *%tokens%* i dio koji počinje sa *%productions%.* U prvom dijelu dolaze različiti podaci o gramatici, najbitniji je podatak o korištenom algoritmu. Trenutno aplikacija podržava samo LL(k) algoritam pa je to i jedina moguća opcija zasad. U *%tokens%* dijelu zadajemo završne znakove gramatike, a u *%productions%* produkcije gramatike. Produkcije se zadaju u obliku *<lijeva\_strana> = <desna\_strana>,* te se za isti nezavršni znak produkcije odvajaju znakom “|” (CTRL + W). Na izlazu daje izvorni kod sintaksnog analizatora u programskom jeziku Java, VB ili C# (ovisi o opciji iz komandne linije) razlomljen u četiri datoteke.

Generiranje koda sintaksnog analizatora pokrece se u konzoli iz mape gđe je instalirana *Grammatica* naredbom:

*java –jar lib/grammatica-1.5.jar <put\_do\_.grammar\_datoteke> --javaoutput <ime\_nove\_mape>*

Primjer se nalazi na slici 6.



Slika 6. Primjer pokretanja naredbe za generiranje izvornog koda sintaksnog analizatora.

Takva naredba će kreirati novu mapu s imenom *test* te u nju izgenerirati četiri nove datoteke u kojima su sadržani podatkovni članovi, strukture potrebni za rad sintaksnog analizatora te osnovni dio generiranog sintaksnog analizatora. Ukoliko generiranje koda prođe bez greške pojaviti će se poruka u konzoli kao što je prikazano na slici 7.



Slika 7. Uspješno izvršeno generiranje izvornog koda sintaksnog analizatora.

Moguće je generirati kod još i u C# i VisualBasic tako da se riječ *javaoutput* zamijeni sa *csoutput* ili *vboutput*. Također postoji opcija *--parse* kojom se može testirati generirani sintaksni analizator.

Nedostatak grafičkog sučelja čini ovaj alat puno težim za korištenje i razumijevanje od ostala dva alata, unatoč tome što pruža dobru funkcionalnost generiranja koda sintaksnog analizatora. Doduše, dobra dokumentacija dijelom pokriva ovaj nedostatak.