0 1 2 3

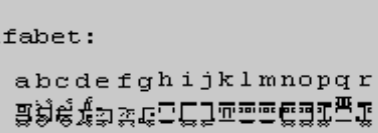
Zdravko Dovedan

FORMALNI JEZICI

• sintaksna analiza

Alfabet:

a b c d e f g h i j k l m n o p q r



Rječnik:

dcba

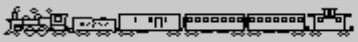
gfe

ihkj

nmlo

rqp

Rečenica:



$E \rightarrow T \mid T+E$

$T \rightarrow F \mid F*T$

$F \rightarrow (E) \mid a$

w = (a)

(q, 1, ε, E$)
- ⊢(q,1,E1, T$)
- ⊢(q,1,E1T1, F$)
- ⊢(q,1,E1T1F1, (E)$)
- ⊢(q,2,E1T1F1(, E)$)
- ⊢(q,2,E1T1F1(E1, T)$)
- ⊢(q,2,E1T1F1(E1T1, F)$)
- ⊢(q,2,E1T1F1(E1T1F1, (E))$)
- ⊢(b,2,E1T1F1(E1T1F1, (E))$)
- ⊢(q,2,E1T1F1(E1T1F2, a)$)
- ⊢(q,3,E1T1F1(E1T1F2a,)$)
- ⊢(q,4,E1T1F1(E1T1F2a), $)
- ⊢(t,4,E1T1F1(E1T1F2a), ε)

E⇒T⇒F⇒(E)⇒(T)⇒(F)⇒(a)

Sadržaj

Predgovor

Prošlo je četvrt stoljeća otkad sam prvi put saznao da je sve što nas okružuje jezik! Da, upravo tako. Naime, prema definiciji formalnog jezika, to je bilo koji skup, pa i prazan skup, a sve čime smo okruženi jest dio nekog skupa, a koliko tek ima praznih skupova! (Ima li?)

Teorija formalnih jezika rođena je u drugoj polovici pedesetih godina dvadesetoga stoljeća u ranim radovima Noama Chomskog. Temeljila se na odgovarajućem modelu prirodnih jezika, kao što je npr. engleski.

Uskoro potom veliki poticaj daljnjem razvoju teorije formalnih jezika dala je uporaba beskontekstnih gramatika i onoga što se danas naziva "Backus-Naurova forma" (BNF). Prvi je put bila upotrijebljena u defininiciji jezika ALGOL 60, 1963. godine.

Na početku se teorija formalnih jezika više bavila specificiranjem skupova i određenim formalizmima, kao što su regularni izrazi i gramatike, beskontekstne gramatike itd.

Daljnji razvoj teorije formalnih jezika išao je u nekoliko pravaca. Osim dijela teorije koji je bio bliži matematici, sve više se razvijala teorija koja je nalazila primjene u računarskim znanostima:

- definiranju jezika za programiranje,
- definiranju prevodilaca jezika za programiranje,
- teoriji programiranja,
- teoriji modela i struktura podataka,
- teoriji baza podataka, itd.

Danas je značenje teorije formalnih jezika višestruko. Fundamentalni koncepti teorije formalnih jezika nalaze svoju primjenu u nekoliko područja; osim u računarskim znanostima još i u umjetnoj inteligenciji (razumijevanju prirodnih jezika), prepoznavanju oblika, kemiji, biologiji itd.

Poslije četvrt stoljeća bavljenja teorijom formalnih jezika i njezinom praktičnom primjenom u definiciji i izradbi prevodilaca jezika za programiranje, poslije jednako toliko predavanja na dodiplomskim studijima, te poslije velikog broja realiziranih algoritama teorije formalnih jezika i prevodilaca, posebno predprocesora, te nekoliko desetaka diplomskih radova iz područja formalnih jezika, pred vama je jedna od knjiga koja sve to objedinjuje.

Teorija formalnih jezika ima više aspekata. Sigurno da će se pristup u njezinom izučavanju razlikovati na prirodoslovno-matematičkom fakultetu i filozofskom fakultetu. Mi smo, može se tako reći, više koristili teorijske rezultate koji su nastali izučavanjem formalnih jezika i pokušali ukazati na moguće praktične primjene. Ako smo ponegdje uveli teorem, nije nas interesirao njegov dokaz, već više njegove posljedice.

Pred vama je prva od dvije knjige koje se odnose na teoriju formalnih jezika, s podnaslovom **Sintaksna analiza**. Druga je knjiga, s podnaslovom **Prevođenje**, u pripremi i bit će objavljena početkom 2004. godine.

U uvodnom poglavlju, *Osnove*, dan je kratki repetitorij onih dijelova matematike koji su neophodni za potpuno razumijevanje ostalih poglavlja, a to su: skupovi, relacije, funkcije, matematička logika i grafovi.

Prvo poglavlje, *Uvod u teoriju formalnih jezika*, sadrži osnove teorije formalnih jezika, pojam znaka i niza znakova, definiciju formalnog jezika, te regularne skupove i izraze kao formalizme za specificiranje regularnih jezika.

U drugom poglavlju, *Gramatike*, definiraju se gramatike kao jedan od sustava za specificiranje i generiranje jezika. Dana je klasifikacija gramatika i načini prikaza gramatika. Veći dio poglavlja odnosi se na beskontekstne gramatike i jezike, kao uvod u preostala poglavlja gdje ćemo se također pretežno baviti beskontekstnim jezicima.

Treće se poglavlje, *Izvođenje i transformiranje gramatika*, također bavi gramatikama, posebno opisom kako izvesti inicijalnu gramatiku danog jezika, a potom je transformirati u podesniji oblik ili zadanu formu.

Četvrto je poglavlje, *Uvod u teoriju automata*, posvećeno drugom načinu prikaza, generiranja i prepoznavanja jezika – automatima, s posebnim osvrtom na konačne i stogovne automate. Također je prikazano kako se konačni automati mogu korisno upotrijebiti u izvođenju regularnih gramatika.

Peto je poglavlje, *Općeniti postupci sintaksne analize*, kao što slijedi iz naslova, posvećeno problemu sintaksne analize. Obrađena su dva najpoznatija općenita postupka nedeterminističke sintaksne analize, silazna i uzlazna analiza, te dva tablična postupka: Cocke-Younger-Kasamijev (CYK) i Earleyjev postupak sintaksne analize.

Šesto je poglavlje, *Jednoprolazna sintaksna analiza*, nastavak opisa postupaka sintaksne analize. Tu je dano nekoliko postupaka deterministične sintaksne analize: silazni, uzlazni i tablični.

Završno, sedmo poglavlje, *Jezici sa svojstvima*, predstavlja sintezu moga dugogodišnjeg rada na razvoju posebnog opisa jezika i učinkovitoga deterministič kog postupka sintaksne analize. Definicija jezika sa svojstvima obuhvaća sve tipove jezika (od tipa 3 do tipa 0) i jezike za programiranje.

U dodatku su dani programi napisani u Turbo Pascalu koji pokazuju ustroj pojedinih postupaka opisanih u knjizi: Sintaksna analiza jezika BNF, višeprolazne sintaksne analize (silazna, uzlazna i Cocke-Younger-Kasamijev algoritam) i predikatna sintaksna analiza kao primjer jedne jednoprolazne sintaksne analize. Osim toga što programi prikazuju kako se pojedini algoritmi mogu ustrojiti u Turbo Pascalu i predstavljaju simulaciju nekih postupaka sintaksne analize, istodobno mogu poslužiti u njihovom izučavanju.

U svim je poglavljima dano puno primjera koji upotpunjuju teorijska razmatranja, posebno pojedine definicije i algoritme. Na kraju poglavlja su pitanja i zadaci.

Knjiga je namijenjena, u prvom redu, studentima smjera opće informatologije (i smjera lingvistike) na Filozofskom fakultetu Sveučilišta u Zagrebu kao temeljna literatura u izučavanju predmeta *Formalni jezici i prevodioci*. Sadržaj knjige, odabir pojedinih tema, primjeri i programi rezultat su mojih najnovijih istraživanja u promicanju i modernizaciji nastavnih sadržaja predmeta Formalni jezici i prevodioci u okviru projekta ***Tempus***.

Siguran sam da će knjiga s ovakvim sadržajem zainteresirati studente sličnih usmjerenja, inženjere informatike i računarskih znanosti, napredne srednjoškolce i mnoge druge samouke informatičare. Vjerujem da će programi dani u prilozima biti dovoljna motivacija znatiželjnijim čitateljima da ih u prvom koraku usavrše, a potom pristupe realizaciji nekih drugih algoritama danih u knjizi.

Na kraju zahvaljujem svima onima koji su na bilo koji način utjecali na sadržaj ove knjige, a posebno gospodinu **Miroslavu Zečeviću** koji me je prvi, davne 1977. godine, "zarazio" teorijom formalnih jezika, ukazao na literaturu i dao temeljne smjernice u toj disciplini.

Neizmjerna je moja zahvala i **Georgesu Moustakiju** koji je svojom prekrasnom poezijom i šansonama uvijek bio i ostao u pozadini svih mojih radova! Mislim da ćete u prijevodima nekoliko njegovih šansona, danih na početku svakog poglavlja, prepoznati nešto posebno (i možda otkriti tajnu vezu šansone sa sadržajem poglavlja u kojem je dana, jer od mnogih Moustakijevih šansona nisam nimalo slučajno izabrao baš te!).

Na kraju, bit ću vam zahvalan ako pažljivo pročitate ovu knjigu, prihvatite barem jedan njezin djelić i primijenite u svojoj praksi. Ako vam se, ipak, sve ovo učini prilično zamršenim, nerazumljivim ili neupotrebljivim, nemojte očajavati, ima i ljepših stvari u životu. Svi vi, a posebno oni kojima se svide (samo) šansone, možete mi se slobodno javiti na e-mail adresu:

zdovedan@hotmail.com.

Bit će mi posebno zadovoljstvo pokloniti vam CD sa svojim interpretacijama tih šansona (dakako, na francuskom!).

U Zagrebu, prosinca 2002. godine

A u t o r

0.

OSNOVE

prolazi, prolazi vrijeme
nema ga više puno

ipak još uvijek živim
ipak još uvijek vodim ljubav
dođe mi čak i da zasviram
na svojoj gitari
o dječaku kakav sam bio
o dječaku kakvog sam stvorio

prolazi, prolazi vrijeme
nema ga više puno

dok sam pjevao
dok sam te volio
dok sam sanjao
bilo je još vremena

prekasno je
il est trop tard

(georges moustaki/
zdravko dovedan)

dok sam spavao
dok sam sanjao
kazaljke su se okretale
prekasno je
moje djetinjstvo je tako daleko
već je sutra

prolazi, prolazi vrijeme
nema ga više puno

dok sam te volio
dok sam te imao
ljubav je otišla
prekasno je
bila si tako lijepa
sam sam u svom krevetu

prolazi, prolazi vrijeme
nema ga više puno

dok sam pjevao
svojoj dragoj slobodi
drugi su je u lance okovali
prekasno je
neki su se čak i borili
ja to nikada nisam znao

Izučavanje teorije formalnih jezika zahtijeva strog, formalni pristup. Također pretpostavlja solidno predznanje iz matematike, posebno teorije skupova, grafova i matematičke logike. Sve to, veoma sažeto, dano je u ovom uvodnom poglavlju.

0.1 SKUPOVI

Skup intuitivno shvaćamo kao kolekciju elemenata (ili članova) koji posjeduju izvjesna svojstva. Na primjer, skup dana u tjednu sačinjavaju sljedeći elementi: `ponedjeljak, utorak, srijeda, četvrtak, petak, subota, nedjelja`.

Ako je α element skupa `S`, piše se $\alpha \in S$ i čita "α je element skupa `S`", a ako nije, piše se $\alpha \notin S$ i čita "α nije element skupa `S`". Na primjer, $petak \in S$, $svibanj \notin S$, gdje je `S` skup dana u tjednu.

Elementi skupa mogu biti jedinke, koje predstavljaju sami sebe, ili neki drugi skupovi. Prikazuje se samo jedno pojavljivanje nekog elementa u skupu. Redoslijed pisanja elemenata skupa nije bitan. Primjeri:

$\{a,b,c\} = \{a,c,b\} = \{b,a,c\} = \{b,c,a\} = \{c,a,b\} = \{c,b,a\}$
$\{a,a\} = \{a\}$
$\{a,\{b\}\} \neq \{a,b\}$
$\{a\} \neq \{\{a\}\}$

Ako skup sadrži konačan broj elemenata, naziva se konačnim, a ako ima beskonačno mnogo elemenata, onda je beskonačan. Definira se i prazan skup, skup koji ne sadrži nijedan element. Označavat ćemo ga s $\varnothing$. Također se definira i prebrojiv skup, a to je ili konačan skup ili beskonačan skup u kojem su elementi uređeni tako da se uvijek može odrediti sljedeći, odnosno, kaže se da tada postoji bijekcija sa skupa prirodnih brojeva na taj skup.

Podskupovi

Do pojma podskupa dolazi se promatranjem dijela nekog skupa. Kaže se da je `B` podskup skupa `A` ako je svaki element `x` iz `B` ujedno i element skupa `A`. U tom slučaju piše se

$B \subseteq A$

Na primjer, skup

`P={ponedjeljak, srijeda, petak}`

podskup je skupa S svih dana tjedna, tj. $P \subseteq S$. Piše se $B \subset A$ i kaže da je B pravi podskup skupa A ako u A postoji najmanje jedan element koji nije u B. Ako se želi istaknuti da B, u krajnjem slučaju, može biti cijeli A, piše se $B \subseteq A$. Partitivni skup nekog skupa A, označen s $\mathcal{P}(A)$, jest skup svih skupova koji se mogu izgraditi od elemenata skupa A, uključujući sam skup A i prazan skup. Ako skup A sadrži n elemenata, partitivni skup $\mathcal{P}(A)$ sadržavat će n^2 elemenata (skupova).

Zadavanje skupova

Skup se S smatra zadanim ako je nedvosmisleno rečeno, objašnjeno ili specificirano, što su elementi tog skupa. Zadati skup S znači dati ograničenje, propis ili svojstvo kojim su u potpunosti određeni svi elementi skupa.

U nekim se slučajevima elementi skupa jednostavno navedu između vitičastih zagrada. Na primjer:

$S=\{1,3,a,d\}$

Često je nemoguće nabrojiti sve elemente skupa koji se mogu zadati nekim propisom, kao na primjer kada se govori o skupu svih parnih brojeva. Premda je taj skup određen, nije moguće navesti sve njegove elemente, pa se piše

$S=\{2,4,6,8,10,12,...\}$

Točkice naznačuju sve ostale elemente iz S koji se eventualno ne mogu navesti. U takvim je slučajevima prikladno skup zadati na sljedeći način:

$S=\{x|\ P(x)\}$ ili $S=\{x:\ P(x)\}$

što se čita: "Skup S sadrži one elemente x za koje je ispunjeno svojstvo (ili predikat) P(x)". Na primjer, ako je N skup prirodnih brojeva, može se napisati

$S=\{x|\ \text{"x je paran"}\}$ ili $S=\{x:\ x=2n,\ n \in N\}$

Operacije sa skupovima

Postoji nekoliko operacija sa skupovima koje se mogu koristiti pri izgrađivanju novih skupova. Neka su A i B skupovi. Unija od A i B, napisana kao $A \cup B$, jest skup koji sadrži sve elemente skupa A zajedno sa svim elementima skupa B:

$A \cup B=\{x \mid x \in A \text{ ili } x \in B\}$

Presjek skupova A i B, $A \cap B$, skup je elemenata sadržanih i u A i u B:

$A \cap B=\{x \mid x \in A \text{ i } x \in B\}$

Razlika skupova A i B, $A\setminus B$, skup je elemenata koji pripadaju skupu A, a nisu u B:

$$A\setminus B=\{x \mid x\in A \text{ i } x\notin B\}$$

0.2 RELACIJE

Izravni ili Kartezijev produkt dvaju skupova A i B je skup:

$$A\times B = \{(a,b) \mid a\in A,\ b\in B\}$$

Element tako nastalog skupa, (a,b), naziva se uređeni par. Na primjer, ako je $A=\{a,b\}$, $B=\{0,1\}$, Kartezijev produkt $A\times B$ jest skup

$$\{(a,0),\ (a,1),\ (b,0),\ (b,1)\}$$

Prva komponenta bilo kojeg para mora biti iz A, druga iz B. Zbog toga $(0,a)$ nije element skupa $A\times B$.

Relacija, označimo je s ρ, jest bilo koji podskup Kartezijevog produkta skupova. Na primjer, ako je $N=\{1,2\}$, $M=\{0,1,2,3,4,5\}$, relacija ρ, $\rho\subseteq N\times M$, može biti

$$\rho = \{(1,0),\ (1,1),\ (1,2),\ (1,3),\ (2,0),\ (2,1),\ (2,2)\}$$

Primijetiti da relacija ρ u ovom primjeru označuje svojstvo parova (x,y), $x\in N$, $y\in M$, da im je zbroj manji ili jednak 4. Isto svojstvo može se napisati kao $x\rho y$ što se čita "x je u relaciji ρ sa y" ili "između x i y postoji relacija ρ".

0.3 FUNKCIJE

Funkcija (preslikavanje, transformacija) f iz skupa A u skup B jest relacija iz A u B takva da, ako su (a,b) i (a,c) u f, vrijedi $b=c$.

Ako je (a,b) u f često se piše $b=f(a)$. Kaže se da je $f(a)$ definirano ako postoji b u B tako da je (a,b) u f. Ako je $f(a)$ definirano za sve a iz A, kaže se da je f potpuno preslikavanje sa A u B. Ako to nije ispunjeno, f je djelomično preslikavanje iz A u B. U oba slučaja piše se

$$f\colon A \to B$$

i kaže da je A domena, a B kodomena funkcije f.

0.4 MATEMATIČKA LOGIKA

Dobro poznavanje elemenata matematičke logike osnovni je uvjet shvaćanja problema formalnih jezika. Ovo potpoglavlje predstavlja kratki repetitorij matematičke logike.

Elementi algebre sudova

Algebra sudova osnova je drugih dijelova matematičke logike, prijeko potrebna za njihovo razumijevanje. Izgrađuje se na isti način kao i mnogobrojne matematičke teorije. Za osnovne pojmove uzima se neka klasa objekata, te neka svojstva, odnosi i operacije nad tim objektima. Ti se osnovni pojmovi smatraju polaznim, i za njih nije potrebno dati nikakvu definiciju. Osnovni pojmovi najčešće se objašnjavaju primjerima. Osnovni objekt kojeg proučava algebra sudova je elementarni sud. Na primjer, elementarni sudovi su:

"Broj 100 djeljiv je s 4" "Danas je lijepo vrijeme"
"Štef nema djevojku" "Mjesec je veći od Zemlje"
"17 je prost broj"

Svi elementarni sudovi moraju imati jedno i samo jedno svojstvo:

"biti istinit" ili "biti lažan"

Na primjer, sud "7 je veće od 5" je istinit, a sud "8 je prost broj" nije istinit. U daljnjem tekstu elementarne sudove označivat ćemo malim slovima `a`, `b`, `c`, itd.

Za primjene u programiranju posebno su važni elementarni sudovi nazvani relacijski izrazi. Općenito relacijski izraz sadrži dva izraza istog tipa (na primjer, dva aritmetička izraza) između kojih je napisana jedna od relacija:

=	jednako	≠	različito
<	manje	≤	manje ili jednako
>	veće	≥	veće ili jednako

Na primjer, relacijski izraz (elementarni sud) `"9>5"` čitat ćemo `"9` je veće od `5"`.

Logičke operacije

Od elementarnih sudova moguće je, uz pomoć nekoliko logičkih operacija, graditi složene sudove. Jasno je da će i složeni sud biti istinit ili lažan. U daljnjem tekstu govorit će se o njegovoj "vrijednosti istinitosti", koja će biti označena s `T` za istinito, i s `F` za lažno.

Vrijednost istinitosti složenog suda ovisi o istinitosti sudova od kojih je složeni sud izgrađen. Postoji nekoliko unarnih i binarnih logičkih operacija (logičkih veznika ili konektiva). Ovdje su opisane samo one osnovne; negacija, kao unarna logička operacija, te binarne logičke operacije:

- konjunkcija
- disjunkcija
- implikacija

Negacija

Negacija je unarna logička operacija i ujedno najjednostavnija operacija algebre sudova. U govornom jeziku odgovara joj približno riječ "ne". Ako negaciju označimo s "¬a", njezino djelovanje na sud a dano je u sljedećoj tablici:

a	¬a
F	T
T	F

Ako je a neki sud, na primjer "5 je djelitelj od 7", ¬a novi je sud "5 nije djelitelj od 7". Sud ¬a čita se "ne a" ili "non a".

Konjunkcija

Ako su a i b sudovi, onda je a∧b novi složeni sud ili konjunkcija sudova a i b. Znak "∧" čita se "i" ili "et". Djelovanje operacije konjunkcije definirano je sljedećom tablicom:

a	b	a∧b
F	F	F
F	T	F
T	F	F
T	T	T

Treba zapamtiti da će složeni sud a∧b biti istinit onda i samo onda ako su i a i b istiniti. Operacija konjunkcije po smislu približno odgovara vezniku "i". Na primjer, elementarni sudovi "6 je djeljivo sa 3", "10 je veće od 5" istiniti su, pa je istinita i njihova konjunkcija "6 je djeljivo sa 3 i 10 je veće od 5". Međutim, elementarni sudovi "3 je djeljivo sa 7" i "3 je veće od 7" lažni su, pa je lažan i složeni sud "3 je djeljivo sa 7 i 3 je veće od 7".

Disjunkcija

Operacija disjunkcije najčešće se označuje sa "∨" i čita "ili". Treba napomenuti da veznik "ili" u mnogim jezicima ima dva različita značenja. U jednom slučaju radi se o tzv. "isključnom", u drugom o "neisključnom" vezniku "ili". Razlika među njima pokazana je u sljedećem primjeru:

Ako su a i b dva lažna suda, lažan je i složeni sud a∨b. Ako je a istinito, a b lažno, ili a lažno, a b istinito, istinit je i složeni sud a∨b. Što je, međutim, s vrijednošću istinitosti suda a∨b ako su i a i b istiniti? U prvom slučaju, ako se složena izjava smatra istinitom, govori se u neisključnoj, u drugom o isključnoj disjunkciji.

U matematičkoj logici operacija disjunkcije odgovara neisključnom vezniku "ili". Iz prethodnih razmatranja slijedi definicija: disjunkcija sudova a i b, napisana kao a∨b, složen je sud koji je lažan onda i samo onda ako su i a i b lažni. Iz te definicije slijedi tablica:

a	b	a∨b
F	F	F
F	T	T
T	F	T
T	T	T

Implikacija

Implikacija se obično označuje sa "⇒" i definira na sljedeći način: ako su a i b dva suda, a⇒b (čita se "a implicira b") složeni je sud koji je istinit uvijek osim ako je a istinito, a b lažno. Tablica istinitosti implikacije dana je sa:

a	b	a⇒b
F	F	T
F	T	T
T	F	F
T	T	T

Operacija implikacije odgovara, u određenom smislu, veznicima "ako..., onda...". Sud "ako je a, onda je b" može se shvatiti kao ¬a∨b. U većini slučajeva pogodnije je sud a⇒b zamijeniti sudom ¬(a∧¬b). Takvo značenje implikacije ističe činjenicu da pri istinitosti a⇒b ne može nastupiti slučaj da je a istinito, a b lažno. To odgovara važnom svojstvu implikacije da istina ne može implicirati neistinu. Na kraju treba napomenuti da se sud a⇒b ne podudara uvijek po smislu sa sudom "ako je a, onda je b". Otuda, da bi se izbjegli eventualni nesporazumi, složeni je sud a⇒b bolje čitati "a implicira b".

Logički izrazi

Uporabom navedenih logičkih operacija i uvođenjem zagrada mogu se, kao i u algebri, graditi razni složeni sudovi ili logički izrazi. Na primjer:

(a∨b)∧c (a∧b)⇒(c∨d) ¬a∨(b⇒c)

Logičke varijable koje se pojavljuju u izrazima mogu biti elementarni sudovi, na primjer "x<6", ili nosioci logičkih vrijednosti dobivenih kao rezultat prethodnog izračunavanja (nekog logičkog izraza). Također se može pojaviti logička konstanta F, čija je vrijednost istinitosti uvijek "laž", i logička konstanta T čija je vrijednost istinitosti uvijek "istina". Logički izrazi koji sadrže samo operacije negacije, konjunkcije i disjunkcije, te zagrade, nazivaju se Booleove formule. Za Booleove formule vrijede sljedeći zakoni:

1) *Zakon komutacije*:
$x \vee y \equiv y \vee x$
$x \wedge y \equiv y \wedge x$

2) *Zakon asocijacije*:
$x \vee (y \vee z) \equiv (x \vee y) \vee z$
$x \wedge (y \wedge z) \equiv (x \wedge y) \wedge z$

3) *Zakon idempotentnosti*:
$x \vee x \equiv x$
$x \wedge x \equiv x$

4) *Zakon distribucije*:
$x \vee (y \wedge z) \equiv (x \vee y) \wedge (x \vee z)$
$x \wedge (y \vee z) \equiv (x \wedge y) \vee (x \wedge z)$

5) *De Morganov zakon*:
$\neg(x \vee y) \equiv \neg x \wedge \neg y$
$\neg(x \wedge y) \equiv \neg x \vee \neg y$

6) *Zakon dvostruke negacije*:
$\neg\neg x \equiv x$

Posebno je česta uporaba de Morganovog zakona u programiranju.

0.5 GRAFOVI

Graf je apstraktni matematički objekat. No, uobičajeno je da se njegov geometrijski prikaz - lik sastavljen od točaka i crta koje ih spajaju - naziva graf.

Grafovima (stablima) je moguće opisati mnoge strukture u računarskim znanostima. Ovdje je izložen koncept teorije grafova koji ima posebne primjene u teoriji formalnih jezika.

Definicija grafa

Neformalno, grafovi su likovi sastavljeni od točaka od kojih su neke (dvije po dvije) spojene krivuljama.

Postoji nekoliko definicija grafa. Ovdje su dane dvije, ona kojom se pojam grafa povezuje s pojmom binarne relacije, i definicija prema C. Bergeu.

Definicija 0.1
Neka je $X=\{x_1, x_2, \ldots, x_n\}$ neprazan skup i ρ binarna relacija u X, $\rho \subseteq X \times X$. Uređeni se par $\Gamma=(X,\rho)$ naziva graf, elementi skupa X čvorovi, a elementi skupa ρ grane grafa.

Na primjer, ako je $X_1=\{1,2,3,4\}$, a $\rho_1=\{(1,1),(1,2),(2,3),(2,4),(3,4),(4,3)\}$, primjer grafa je $\Gamma_1=(X_1,\rho_1)$.

Slijedi definicija grafa prema C. Bergeu:

Definicija 0.2
Neka je $X=\{x_1,x_2,...,x_n\}$ neprazan skup i $P(X)$ njegov partitivni skup. Definira se preslikavanje δ sa X u $P(X)$, tj. za svaki $x_i \in X$ je $\delta(x_i) \in P(X)$. Graf je definiran skupom X i preslikavanjem δ, tj. graf je uređeni par $\Delta=(X,\delta)$.

Na primjer, graf Δ iz prethodnog primjera može se definirati kao $\Delta=(X,\delta)$, gdje je $X=X_1$, a funkcija δ definirana je sa:

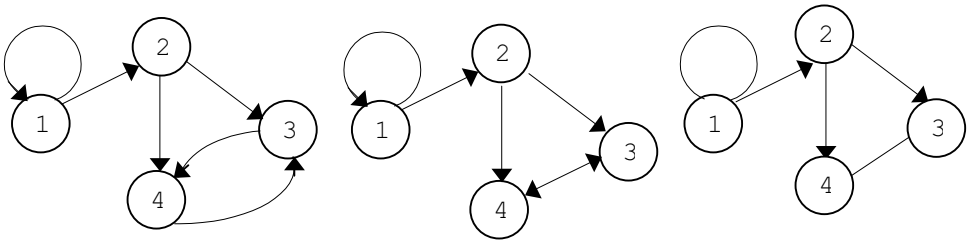
$\delta(1) = \{1,2\}$ $\quad \delta(2) = \{3,4\}$
$\delta(3) = \{4\}$ $\quad \delta(4) = \{3\}$

Prikaz grafa

Graf može biti prikazan crtežom ili matricom susjedstva. Prikaz grafa crtežom je na sljedeći način: Najprije se nacrtaju čvorovi x_1, ..., x_n kao točke ili kružnice u ravnini (ili prostoru). Njihov razmak i pozicija potpuno su proizvoljni. Potom se spoje čvorovi za koje vrijedi $(x_i,x_j) \in \rho$. Čvor x_i, prikazan kružnicom ili točkom, spaja se sa x_j neprekinutom crtom i usmjerava strjelicom od x_i k x_j. Grana (x_i,x_j) izlazi iz čvora x_i i ulazi u čvor x_j. Ako $(x_i,x_j) \notin \rho$, čvorovi x_i i x_j nisu izravno povezani na crtežu.

Ako je $(x_i,x_i) \in \rho$, čvor x_i treba spojiti sa samim sobom. Takva se grana naziva petlja. Smjer petlje nema posebnog značenja pa može biti izostavljen.

Ako su čvorovi x_i i x_j povezani s dvije grane, (x_i,x_j) i (x_j,x_i), na crtežu se povlače dvije crte, ili jedna crta obostrano usmjerena, ili jedna crta bez oznake usmjerenja. Na primjer, graf $\Gamma_1=(X_1,\rho_1)$, gdje su X_1 i ρ_1 kao što je ranije definirano, može biti prikazan na iduće načine:



Osim crtežom, graf može biti prikazan kvadratnom matricom čiji je red jednak broju čvorova. Takva se matrica naziva matrica susjedstva ili Booleova matrica (jer su joj elementi 0 ili 1). Element a_{ij} na presjeku i-tog retka i

j-tog stupca u toj matrici, jednak je 1, ako je $(x_i, x_j) \in \rho$, odnosno 0 ako to nije ispunjeno. Na primjer, graf Γ_1 može biti prikazan sljedećom matricom susjedstva:

	1	2	3	4
1	1	1	0	0
2	0	0	1	1
3	0	0	0	1
4	0	0	1	0

Putovi u grafu

Najprije dajemo definiciju puta u grafu potom još nekoliko važnih definicija:

Definicija 0.3
Niz čvorova $(x_0, \ldots, x_n)$, $n \geq 1$, jest put duljine n od čvora x_0 do čvora x_n ako postoji grana koja izlazi iz čvora x_{i-1} i ulazi u čvor x_i, za $1 \leq i \leq n$. Ako između čvorova a i b grafa Γ postoji put duljine k, $k \geq 1$, tada je a prethodnik od b, odnosno, b je slijednik od a. Ako je $k=1$, a je izravni prethodnik od b. Tada je b izravni slijednik od a.

Definicija 0.4
Kružni ili zatvoreni put jest put $(x_0, \ldots, x_n)$ u kojem je $x_n = x_0$.

Definicija 0.5
Graf $\Gamma = (X, \rho)$ jest povezan ako postoji put od a do b za sve parove različitih čvorova.

Definicija 0.6
Ulazni stupanj čvora x grafa $\Gamma = (X, \rho)$, $x \in X$, jest broj grana koje ulaze u x. Izlazni stupanj jest broj grana koje izlaze iz čvora x.

Stablo

Na kraju ovoga poglavlja dajemo još nekoliko definicija koje se odnose na posebnu skupinu grafova - stabla.

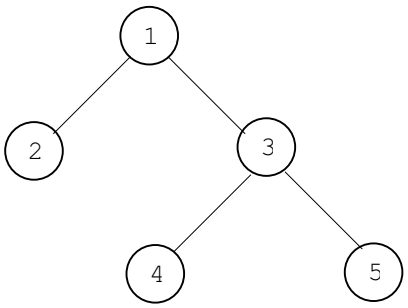
Definicija 0.7
Stablo, označimo ga s τ, jest povezani graf $\Gamma = (X, \rho)$ s $n \geq 1$ čvorova i $m = n - 1$ grana.

Definicija 0.8
Korijen stabla jest čvor x sa svojstvima:

- ulazni stupanj od x jednak je 0,
- svi ostali čvorovi stabla imaju ulazni stupanj jednak 1, i
- svaki je čvor iz X slijednik od x.

List stabla τ jest čvor z sa svojstvom da mu je izlazni stupanj jednak 0.

Sljedeći graf je primjer stabla:



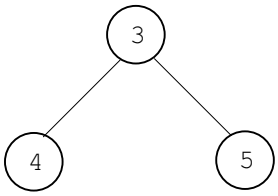
Korijen je označen s 1. Čvorovi 2 i 3 izravni su slijednici od 1, a čvor 3 izravni prethodnik čvorovima 4 i 5.

Ako je τ stablo, iz njegove definicije slijedi da ne sadrži petlje i da postoji jedinstveni put od korijena do svakog njegova čvora.

Definicija 0.9
Podstablo stabla $\tau=(X,\rho)$ jest stablo $\tau'=(X',\rho')$ takvo da:

- X' nije prazan i sadržan je u X, tj. $X'\subseteq X$,
- $\rho'=(X'\times X')\cap\rho$ i
- nijedan čvor iz $X\backslash X'$ nije slijednik čvora X'.

Na kraju, evo primjera podstabla stabla iz prethodnog primjera:



1.

UVOD U TEORIJU FORMALNIH JEZIKA

prepustit ćemo se životu
da bismo bili slobodni
bez planova i navika
o njemu ćemo moći sniti

priđi tu sam
samo tebe čekam
sve je moguće
sve je dopušteno

dođi slušaj ove riječi
što trepere na zidovima u svibnju
što svjedoče nam o istini
da se sve jednoga dana može izmijeniti

priđi tu sam
samo tebe čekam
sve je moguće
sve je dopušteno

vrijeme življenja
le temps de vivre

(georges moustaki/
zdravko dovedan)

Primarni će predmet našeg bavljenja biti skupovi čiji su elementi znakovi i nizovi znakova. Ovdje se uvode osnovne definicije i terminologija koji se odnose na njih. Potom je dana definicija formalnog jezika i ukratko opisani regularni skupovi i izrazi koji su imali posebno značenje u razvoju teorije formalnih jezika.

1.1 ZNAKOVI I NIZOVI ZNAKOVA

Znak je jedinstven, nedjeljiv element, kao što su: a, b, ..., A, B, ..., 0, 1, ..., +, −, *, itd. Alfabet je konačan skup znakova. Najčešće ćemo ga označivati sa $\mathcal{A}$. Na primjer, $\mathcal{A}=\{0,1\}$ jest alfabet, poznat kao *binarni alfabet*.

Ako se znakovi alfabeta $\mathcal{A}$ poredaju jedan do drugog dobije se niz znakova (engl. *string*) ili "povorka", ili "nizanica". Na primjer, nizovi znakova nad binarnim alfabetom su: 00, 01, 10, 11, 000, 001, 010, 011, itd. Operacija dopisivanja znaka iza znaka, ili niza iza niza, naziva se nastavljanje ili konkatenacija nizova.

Duljina niza znakova jest broj znakova sadržanih u nizu znakova. Na primjer, niz je znakova 010101 duljine 6. Često se duljina niza znakova x označuje s $d(x)$ ili $|x|$. Na primjer, $|000|=3$. Niz znakova aaaa...a, sačinjen od n jednakih znakova, piše se kao a^n. Na primjer, umjesto 000 može se napisati 0^3.

Dva su niza x i y, $x=a_1...a_n$ i $y=b_1...b_m$, jednaka, tj. $x=y$, ako su jednake duljine, $n=m$, i ako je $a_i=b_i$ za $i=1, ..., n$.

Neka su x i y nizovi znakova nad alfabetom $\mathcal{A}$. Kaže se da je x prefiks ("početak"), a y sufiks ("dočetak") niza xy i da je y podniz niza xyz (x kao prefiks i y kao sufiks niza xy istodobno su i njegovi podnizovi). Ako je $x\neq y$ i x je prefiks (sufiks) niza y, kaže se da je x svojstveni prefiks (sufiks) od y.

Definira se i niz znakova koji ne sadrži nijedan element. Naziva se prazan niz. Označivat ćemo ga s ε ili λ. Duljina praznog niza jednaka je 0, $|\varepsilon|=0$, odnosno, ako je a bilo koji znak, vrijedi $a^0=\varepsilon$. Ako je $x=a_1...a_n$ niz, obrnuti niz (ili "reverzni" niz) jest x^R, $x^R=a_n...a_1$.

Skup svih nizova znakova koji se mogu izgraditi nad alfabetom $\mathcal{A}$, uključujući i prazan niz ε i sam alfabet, označivat ćemo sa $\mathcal{A}^*$. Sa $\mathcal{A}^+$ označivat ćemo skup $\mathcal{A}^*\setminus\{\varepsilon\}$. Primijetiti da su $\mathcal{A}^*$ i $\mathcal{A}^+$ beskonačni ali prebrojivi skupovi! Sa $\mathcal{A}^{*k}$ označivat ćemo konačan skup (podskup od $\mathcal{A}^*$) svih nizova znakova nad $\mathcal{A}$ duljine od 0 do k.

Primjer 1.1
Nad binarnim alfabetom definirani su skupovi nizova znakova:

$\mathcal{A}^{\star}$ = {ε,0,1,00,01,10,11,000,001,010,011,100,... }
$\mathcal{A}^{\star 2}$ = {ε,0,1,00,01,10,11}

1.2 DEFINICIJA FORMALNOG JEZIKA

Slijedi definicija formalnog jezika:

Definicija 1.1
Ako je $\mathcal{A}$ alfabet i $\mathcal{A}^{\star}$ skup svih nizova znakova nad $\mathcal{A}$, jezik $\mathcal{L}$ nad alfabetom $\mathcal{A}$ jest bilo koji podskup od $\mathcal{A}^{\star}$, tj.

$$\mathcal{L} \subseteq \mathcal{A}^{\star}$$

Često se piše $\mathcal{L}(\mathcal{A})$ da se naznači definiranost nekog jezika $\mathcal{L}$ nad alfabetom $\mathcal{A}$. Nizovi znakova koji čine elemente jezika nazivamo rečenice. Dakle, jezik je skup rečenica.

Primjer 1.2
Evo nekoliko primjera jezika definiranih nad binarnim alfabetom:

$\mathcal{L}_1$ = {0,1} $\mathcal{L}_2$ = {00,01,10,11}
$\mathcal{L}_3$ = {000,001,010,011,100,101,110,111}
$\mathcal{L}_4$ = {0,010,01010,0101010,010101010,... }
$\mathcal{L}_5 = \mathcal{A}^{\star}$ = {ε,0,1,00,01,10,11,000,001,010,011,100,... }

Iz definicije formalnog jezika slijedi da je to skup, pa se u posebnom slučaju mogu definirati dva jezika, $\mathcal{L}_1 = \varnothing$ i $\mathcal{L}_2 = \{\varepsilon\}$.

Definicija 1.2
Za jezik $\mathcal{L}$ u kojem za sve njegove rečenice ω vrijedi da nisu svojstveni prefiks (sufiks) ni jednoj rečenici x, $x \in \mathcal{L}$ i $x \neq \omega$, kaže se da ima svojstvo prefiksa (sufiksa).

Primjer 1.3
Jezik $\{0^n | \ n \geq 0\}$ nema svojstvo prefiksa jer je, na primjer, niz 0 svojstveni prefiks za sve rečenice (nizove) $0^n, n>1$, dok jezik $\{0^n 1 | \ n \geq 0\}$ ima svojstvo prefiksa, jer se ne može naći ni jedna rečenica koja bi bila svojstveni prefiks nekoj drugoj rečenici, ali nema svojstvo sufiksa.

Operacije nad jezicima

S obzirom na to da je jezik skup, primjenom poznatih operacija nad skupovima mogu se iz definiranih graditi novi jezici.

Elementi jezika su nizovi znakova pa se može definirati i operacija nastavljanja.

Definicija 1.3
Ako su $\mathcal{L}_1$ i $\mathcal{L}_2$ jezici, $\mathcal{L}_1 \subseteq \mathcal{A}_1^*$ i $\mathcal{L}_2 \subseteq \mathcal{A}_2^*$, tada je $\mathcal{L}_1\mathcal{L}_2$ nastavljanje (ulančavanje ili konkatenacija) ili produkt jezika $\mathcal{L}_1$ i $\mathcal{L}_2$:

$$\mathcal{L}_1\mathcal{L}_2 = \{xy \mid x \in \mathcal{L}_1,\ y \in \mathcal{L}_2\}$$

Primjer 1.4
Ako je $\mathcal{L}_1 = \{0,1\}$ i $\mathcal{L}_2 = \{00,01,10,11\}$, tada je:

$$\mathcal{L}_3 = \mathcal{L}_1\mathcal{L}_2 = \{000,001,010,011,100,101,110,111\}$$

Definicija 1.4
Zatvarač jezika $\mathcal{L}$, označen s $\mathcal{L}^*$, definiran je sa:

(1) $\mathcal{L}^0 = \{\varepsilon\}$ (2) $\mathcal{L}^n = \mathcal{L}\,\mathcal{L}^{n-1}$, $n \geq 1$ (3) $\mathcal{L}^* = \bigcup_{n \geq 0} \mathcal{L}^n$

Pozitivni zatvarač od $\mathcal{L}$, označen sa $\mathcal{L}^+$, jest $\mathcal{L}^* \setminus \{\varepsilon\}$. Primijetiti da je $\mathcal{L}^+ = \mathcal{L}\,\mathcal{L}^* = \mathcal{L}^*\mathcal{L}$.

Simboli i nizovi simbola

Često se promatraju nizovi znakova konačne duljine koji se mogu smatrati jedinstvenom, nedjeljivom cjelinom. Takvi nizovi znakova nazivaju se simboli ili riječi.

Primjer 1.5
Nad alfabetom

$$\mathcal{A} = \{a,b,c,d,e,f\}$$

može se definirati simbole, nizove jednakih znakova duljine 2: aa, bb, cc, dd, ee, ff.

Skup svih simbola definiran nad alfabetom $\mathcal{A}$ označivat ćemo s $\mathcal{V}$ i nazivati rječnik. Budući je $\mathcal{V} \subseteq \mathcal{A}^*$, zaključujemo da je $\mathcal{V}$ jezik. U posebnom se slučaju i alfabet može smatrati skupom simbola duljine 1, pa ćemo ponekad umjesto "znak" reći "simbol" (dakako, obrnuto ne vrijedi!).

Primjer 1.6
Nad alfabetom $\mathcal{A} = \{I,V,X,L,C,D,M\}$ može se definirati rječnik

$$\mathcal{V} = \{I,\ V,\ IV,\ X,\ IX,\ L,\ XL,\ C,\ XC,\ D,\ CD,\ M,\ CM\}$$

Primjer 1.7

Nad alfabetom $\mathcal{A}=\{-,.\}$ može se definirati rječnik Morseovih simbola:

A	B	C	D	E	F	G	H
.-	-...	-.-.	-..	.	..-.	--.	....
I	**J**	**K**	**L**	**M**	**N**	**O**	**P**
..	.---	-.-	.-..	--	-.	---	.--.
Q	**R**	**S**	**T**	**U**	**V**	**W**	**X**
--.-	.-.	...	-	..-	.--	-..-	-.--
Y	**Z**	**0**	**1**	**2**	**3**	**4**	**5**
...-	--..	-----	.----	..---	...--	....-	.....
6	**7**	**8**	**9**	**/**	**.**	**,**	**+**
-....	--...	---..	----.	-..-.	.-.-.-	--..--	.-.-.
?	**:**	**;**					
..--..	---...	-.-.-.					

Primjer rečenice: `... --- ...` (SOS)

Primjer 1.8

Alfabet:

a b c d e f g h i j k l m n o p q r

Rječnik:

dcba

gfe

ihkj

nmlo

rqp

Rečenica:

Definiranje jezika

Jezik je definiran kao podskup skupa svih nizova znakova nad alfabetom. Najčešće je to dosta velik ili beskonačan skup. Zbog toga bi teško bilo i nepraktično definirati ga navodeći eksplicitno sve njegove elemente.

Postoji nekoliko metoda za specificiranje skupa koji čini jezik. Jedna metoda koristi formalizam *regularnih skupova i regularnih izraza*. Druga metoda koristi generativni sustav nazvan *gramatika*. Svaka rečenica jezika može se izvesti koristeći pravila gramatike (nazvana "produkcije").

Treća metoda pripada klasi automata. Koristi posebnu proceduru koja poslije konačno mnogo izračunavanja i dosezanja konačnog stanja "izbacuje" rečenicu jezika. Takvi automati nazivaju se *generatori*.

1.3 REGULARNI SKUPOVI I IZRAZI

Regularni skupovi jesu klasa jezika, tzv. *regularnih jezika*. Imali su posebno značenje u početnom razvoju teorije formalnih jezika, krajem pedesetih godina dvadesetog stoljeća. Ovdje ćemo pokazati kako se teorija regularnih skupova i izraza može primijeniti u specifikaciji jezika.

Definicija 1.5
Neka je $\mathcal{A}$ alfabet. Regularni skup (regularni jezik) nad $\mathcal{A}$ definiran je rekurzivno na sljedeći način:

1) $\varnothing$ (prazan skup) je regularni skup nad $\mathcal{A}$.
2) $\{\varepsilon\}$ je regularni skup nad $\mathcal{A}$.
3) $\{a\}$ je regularni skup nad $\mathcal{A}$, za sve a iz $\mathcal{A}$.
4) Ako su P i Q regularni skupovi nad $\mathcal{A}$, regularni skupovi su i:
   a) $P \cup Q$
   b) PQ
   c) P^*
5) Ništa drugo nije regularni skup.

Dakle, podskup od $\mathcal{A}^*$ jest regularan ako i samo ako je $\varnothing$, $\{\varepsilon\}$ ili $\{a\}$, za neki a u $\mathcal{A}$, ili se može dobiti iz njih konačnim brojem primjene operacija unije, nastavljanja i operacije *.

Definicija 1.6
Regularni izrazi na $\mathcal{A}$ označuju određene regularne skupove. Definirani su, također rekurzivno, na sljedeći način:

1) $\varnothing$ je regularni izraz koji označuje regularni skup $\varnothing$.
2) ε je regularni izraz koji označuje regularni skup $\{\varepsilon\}$.
3) a iz $\mathcal{A}$ je regularni izraz koji označuje regularni skup $\{a\}$.
4) Ako su p i q regularni izrazi koji označuju regularne skupove P i Q, redom, tada su regularni izrazi i:

   a) $(p+q)$ je regularni izraz koji označuje regularni skup $P \cup Q$.
   b) (pq) je regularni izraz koji označuje regularni skup PQ.
   c) $(p)^*$ je regularni izraz koji označuje regularni skup P^*.

5) Više ništa drugo nije regularni izraz.

Ako je pp^* regularni izraz, može se napisati kao p^+. Ako ne postoji dvoznačnost u nekom regularnom izrazu, suvišne se zagrade mogu izbaciti. Može se zamisliti da operacija * (i $^+$) ima najviši proritet, potom operacija nastavljanja i na kraju operacija $+$.

Primjer 1.9
Regularni izraz $101(101)^*$ opisuje skup (jezik) koji sadrži niz 101 ili nizove dobivene konkatenacijom niza 101:

$$\mathcal{L} = \{101, 101101, 101101101, 101101101101, \ldots\}$$

Primjer 1.10
Regularni izraz $(0(1)^*+0)$ može se napisati kao 01^*+0. Evo još nekoliko primjera regularnih izraza i odgovarajućih regularnih skupova:

1) 01 označuje $\{01\}$
2) 0^* označuje $\{0\}^*$
3) $(0+1)^*$ označuje $\{0,1\}^*$
4) $(0+1)^*11$ označuje skup svih nizova koji sadrže znakove 0 i 1, ali uvijek završavaju s 11.
5) $(a+b)(a+b+0+1)^*$ označuje skup nizova iz $\{a,b,0,1\}^*$ koji počinju s a ili b.
6) $(00+11)^*(01+10)(00+11)^*\ (01+10)\ (00+11)^*$ označuje skup nizova koji sadrže znakove 0 i 1, ali uvijek paran broj 0 i paran broj 1.

Lako se može pokazati da se za svaki regularni skup može pronaći bar jedan regularni izraz koji ga označuje. I obrnuto, za svaki regularni izraz može se naći regularni skup kojeg taj izraz označuje.

Nažalost, za svaki regularni skup postoji beskonačan broj regularnih izraza koji ga označuju. Reći ćemo da su dva regularna izraza jednaka (=) ako označuju isti regularni skup. Ako su α, β i γ regularni izrazi, tada vrijede sljedeća algebarska svojstva:

1) $\alpha+\beta = \beta+\alpha$	2) $\varnothing^* = \varepsilon$	3) $\alpha+(\beta+\gamma) = (\alpha+\beta)+\gamma$
4) $\alpha(\beta\gamma) = (\alpha\beta)\gamma$	5) $\alpha(\beta+\gamma) = \alpha\beta+\alpha\gamma$	6) $(\alpha+\beta)\gamma = \alpha\gamma+\beta\gamma$
7) $\alpha\varepsilon = \varepsilon\alpha = \alpha$	8) $\varnothing\alpha = \alpha\varnothing = \varnothing$	9) $\alpha^* = \alpha+\alpha^*$
10) $(\alpha^*)^* = \alpha^*$	11) $\alpha+\alpha = \alpha$	12) $\alpha+\varnothing = \alpha$

Pitanja i zadaci

1) Jesu li nizovi "6" i "2+2+2" jednaki?

2) Dan je alfabet $\mathcal{A}=\{a,(,),+,*\}$. Definirajte nekoliko jezika nad $\mathcal{A}^*$.

3) Ako su $\mathcal{L}_1=\varnothing$ i $\mathcal{L}_2=\{\varepsilon\}$ dva jezika, je li $\mathcal{L}_1=\mathcal{L}$?

4) Dan je alfabet $\mathcal{A}=\{x,y\}$. Napišite sve elemente jezika $\mathcal{L}$ definiranog nad $\mathcal{A}$ koji će imati svojstvo da sadrži nizove duljine do 4.

5) Dan je alfabet $\mathcal{A}=\{a,b\}$. Definirajte jezik $\mathcal{L}$ nad $\mathcal{A}$ u kojem će rečenice x biti maksimalne duljine 6 i imat će svojstvo $x=x^R$. Je li prazan niz ε u jeziku $\mathcal{L}$?

6) Jesu li "prometni znakovi" znakovi ili simboli?

7) Napišite nekoliko rečenica (elemenata skupa) kojeg označuju sljedeći regularni izrazi:

   a) a^+
   b) $0(a+b)^*1$
   c) $(01)^+$
   d) $(bb+a)^*(aa+b)^*$
   e) $(a+ab+aab)^*(a+aa)$
   f) $01+(1+00)1^*0$
   g) $((x+y)(x+y))^*((a+b)(a+b)(a+b))^*$
   h) $pjeva(m+š+mo+te+ju)$

8) Napišite regularne izraze nad alfabetom {a,b} koji će označivati sljedeće regularne skupove:

   a) Nizove koji moraju započeti s a.
   b) Nizove koji moraju završavati s b.
   c) Nizove koji moraju sadržavati barem po jedan par aa i bb.
   d) Nizove koji sadrže aba kao podniz.

9) Napišite regularni izraz koji će označivati promjenu nekoliko izabranih imenica hrvatskog jezika u svim padežima jednine i množine.

10) Napišite regularne izraze koji označuju:

   a) Sve prirodne brojeve.
   b) Sve parne prirodne brojeve.
   c) Sve neparne prirodne brojeve.
   d) Sve prirodne brojeve djeljive s pet.

2.

GRAMATIKE

šansona tek obična pjesma
za sva godišnja doba
o vremenu što prolazi
šansona koju zviždimo za sebe
koju pjevušimo u pola glasa
ili koju prihvaća gomila

šansona tek obična pjesma
za sva godišnja doba
pomalo dosadna muzika
šansona koju znam napamet
koju najčešće pjevam
kada me obuzmu misli sjetne

(šansona) o ljubavi i prijateljstvima

pjesme
chansons

(georges moustaki/
sanja seljan)

šansona o ljubavi i prijateljstvima
šansona o starim putnicima
o otrcanim refrenima
šansona o ulicama i pločnicima
izgubljena ili nađena
na obalama Seine

šansona što živi u mome sjećanju
što silazi na moju gitaru
i svira mi onu pjesmu malu
šansona sa stolnjaka od papira
šansona što budi snove
muzike bezazlene

šansona o ljubavi i kajanju
šansona što mami suze
brbljivice iz kolibice
šansona za Serge ili Edith
stara ili još neobjavljena
u svakom slučaju poznata

U ovom su poglavlju opisane gramatike – veoma značajni formalizmi za specificiranje jezika. Veći dio poglavlja posvećen je beskontekstnim gramatikama i jezicima.

2.1. DEFINICIJA GRAMATIKE

Gramatike su jedna od najznačajnijih klasa generatora jezika. Ovdje će biti razmatrane klase gramatika koje se ponekad nazivaju gramatike Chomskog ili gramatike fraznih struktura.

Gramatika jezika $\mathcal{L}$ koristi dva disjunktna skupa znakova. To su skup neterminalnih znakova, često označen s $\mathcal{N}$, i skup terminalnih znakova (alfabet), označen s $\mathcal{T}$. Skup terminalnih znakova osnova je za tvorbu rečenica jezika. Neterminalni znakovi nisu dio jezika. Koriste se u definiranju pravila za generiranje rečenica jezika.

Glavni dio gramatike jest konačan skup tvorbenih pravila ili produkcija, označenih s $\mathcal{P}$, kojima je opisan način generiranja rečenica jezika. Produkcija je par nizova znakova iz skupa (Kartezijevog produkta)

$$(\mathcal{N}\cup\mathcal{T})^{*}\,\mathcal{N}\,(\mathcal{N}\cup\mathcal{T})^{*}\times(\mathcal{N}\cup\mathcal{T})^{*}$$

Prva komponenta je bilo koji niz koji sadrži najmanje jedan neterminalni simbol. Druga komponenta može biti bilo što, pa i prazan niz. Jezik definiran gramatikom je skup nizova znakova koji se sastoje samo od terminalnih znakova i mogu biti izvedeni počevši s jednim posebnim znakom iz $\mathcal{N}$, najčešće označenim sa $\mathcal{S}$. Prije nego što to pokažemo, evo definicije gramatike.

Definicija 2.1
Gramatika je četvorka $\mathcal{G}=(\mathcal{N},\mathcal{T},\mathcal{P},\mathcal{S})$, gdje su:

$\mathcal{N}$ konačan skup neterminalnih znakova,

$\mathcal{T}$ konačan skup terminalnih znakova (alfabet) uz uvjet da je $\mathcal{T}\cap\mathcal{N}=\emptyset$,

$\mathcal{P}$ konačan skup parova nizova:

$$\{(\alpha,\beta):\ \alpha=\alpha_1\gamma\alpha_2;\ \ \alpha_1,\alpha_2,\beta\in(\mathcal{N}\cup\mathcal{T})^{*},\ \ \gamma\in\mathcal{N}\}$$

(niz α je iz $(\mathcal{N}\cup\mathcal{T})^{+}$ i mora sadržati bar jedan znak iz skupa $\mathcal{N}$),

$\mathcal{S}$ poseban znak iz $\mathcal{N}$, $\mathcal{S}\in\mathcal{N}$, nazvan početni znak (ili *početni simbol*).

Element (α,β) iz $\mathcal{P}$ piše se $\alpha\rightarrow\beta$ i naziva produkcija. Simbol "$\rightarrow$" čita se "producira", "može biti zamijenjeno s" ili "preobličuje se u".

Primjer 2.1
Primjer gramatike je $\mathcal{G}=(\mathcal{N},\mathcal{T},\mathcal{P},\mathcal{S})$, gdje su:

$\mathcal{N} = \{A, S\}$
$\mathcal{T} = \{0, 1\}$
$\mathcal{P} = \{(S, 0A1), (0A, 00A1), (A, \varepsilon)\}$

Početni simbol je S. Skup produkcija $\mathcal{P}$ možemo napisati i kao

$$\begin{aligned} S &\rightarrow 0A1 \\ 0A &\rightarrow 00A1 \\ A &\rightarrow \varepsilon \end{aligned}$$

Ako $\mathcal{P}$ u nekoj gramatici sadrži produkcije:

$$\alpha \rightarrow \beta_1 \quad \ldots \quad \alpha \rightarrow \beta_n$$

piše se

$$\alpha \rightarrow \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$$

Znak "|" čita se "ili". β_i su alternative za α. Ako u $\mathcal{P}$ postoji produkcija oblika

$$\alpha \rightarrow \varepsilon \mid \beta \mid \beta\beta \mid \beta\beta\beta \mid \ldots$$

piše se $\alpha \rightarrow \{\beta\}$.

Vitičaste zagrade omeđuju niz koji može biti izostavljen ili napisan jedanput, dvaput, triput, itd. Produkcija oblika:

$$\alpha \rightarrow \varepsilon \mid \beta$$

piše se $\alpha\rightarrow[\beta]$.

U daljnjem ćemo tekstu neterminale označivati velikim slovima engl. abacede. Terminali će biti mala slova engl. abecede i ostali znakovi (brojke, +, -, *, /, (,), itd.). Neterminal na početku prve produkcije bit će početni simbol.

Gramatika kao generator jezika

Gramatika $\mathcal{G}=(\mathcal{N},\mathcal{T},\mathcal{P},\mathcal{S})$ generira jezik na sljedeći način: Krene se od početnog simbola $\mathcal{S}$ i njegovih produkcija, koje se općenito mogu napisati kao

$$\mathcal{S} \rightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n, \qquad \alpha_i \in (\mathcal{N}\cup\mathcal{T})^*, \; i \geq 1$$

i izabere se, proizvoljno, jedna alternativa za S. Na primjer, $S \rightarrow \alpha_1$. Ako je $\alpha_1 \in \mathcal{T}^*$, dobiven je jedan niz iz jezika $\mathcal{L}(\mathcal{T})$, a ako je $\alpha_1 \in (\mathcal{N} \cup \mathcal{T})^+$, bilo koji podniz od α_1, koji se pojavljuje na lijevoj strani u skupu produkcija, treba zamijeniti jednom od njegovih alternativa. Postupak se ponavlja u novo dobivenom nizu sve dok se ne dobije niz iz $\mathcal{T}^*$, a to je ujedno i niz iz $\mathcal{L}(\mathcal{T})$, jezika kojeg gramatika $\mathcal{G}$ generira. Iscrpljujući sve mogućnosti takvog postupka na kraju bi se dobili svi nizovi koji tvore jezik $\mathcal{L}(\mathcal{T})$. Umjesto oznake $\mathcal{L}(\mathcal{T})$ često se koristi notacija $\mathcal{L}(\mathcal{G})$, čita se "jezik generiran gramatikom $\mathcal{G}$".

Primjer 2.2
Gramatika $\mathcal{G}=(\mathcal{N},\mathcal{T},\mathcal{P},S)$, gdje je $\mathcal{P}$:

$S \rightarrow 0S \mid 1$

generira jezik $\mathcal{L}(\mathcal{G})=\{1,01,001,0001,...\}=\{0^n1: n\geq 0\}$. Na primjer, ako se za S izabere alternativa $0S$ pa se S u tom nizu zamijeni s $0S$, dobije se niz $00S$. Ako se potom za S izabere druga alternativa, konačno se dobije niz 001, jedan element jezika $\mathcal{L}(\mathcal{G})$.

Poslije neformalnog opisa načina generiranja rečenica jezika primjenom produkcija gramatike, slijede definicije kojima se uvodi notacija i relacije kojima je u potpunosti određeno izvođenje rečenica jezika.

Definicija 2.2
Rečenična forma gramatike $\mathcal{G}=(\mathcal{N},\mathcal{T},\mathcal{P},S)$ definirana je rekurzivno:

1) Početni znak je rečenična forma.
2) Ako je $\alpha\delta\gamma$, gdje su $\alpha,\gamma \in (\mathcal{N} \cup \mathcal{T})^*$, rečenična forma i $\delta \rightarrow \beta$ produkcija u $\mathcal{P}$, tada je $\alpha\beta\gamma$ također rečenična forma.

Rečenična forma koja ne sadrži nijedan neterminal naziva se rečenica.

Definicija 2.3
Nad skupom $(\mathcal{N} \cup \mathcal{T})^*$ gramatike $\mathcal{G}=(\mathcal{N},\mathcal{T},\mathcal{P},S)$ definira se relacija $\Rightarrow$, čita se "izravno izvodi", na sljedeći način: Ako je $\alpha\delta\gamma$ niz iz $(\mathcal{N} \cup \mathcal{T})^*$ i $\delta \rightarrow \beta$ produkcija iz $\mathcal{P}$, tada

$$\alpha\delta\gamma \Rightarrow \alpha\beta\gamma$$

Ako za $\alpha_0, \alpha_1, ..., \alpha_n, \alpha_i \in (\mathcal{N} \cup \mathcal{T})^*$, $n \geq 1$, vrijedi

$$\alpha_0 \Rightarrow \alpha_1 \Rightarrow \ldots \Rightarrow \alpha_n$$

tada je $\alpha_0 \overset{n}{\Rightarrow} \alpha_n$ niz izvođenja duljine n. Općenito se piše

$$\alpha_0 \overset{*}{\Rightarrow} \alpha_n,\ n\geq 0,\quad \alpha_0 \overset{+}{\Rightarrow} \alpha_n,\ n>0$$

i kaže da α_0 izvodi α_n.

Shodno dvjema prethodnim definicijama jezik generiran gramatikom $\mathcal{G}$ može se napisati kao

$$\mathcal{L}(\mathcal{G}) = \{\omega \in \mathcal{T}^* : \mathcal{S}^* \Rightarrow \omega\}$$

što čitamo: "Jezik $\mathcal{L}$ generiran gramatikom $\mathcal{G}$ jest skup rečenica dobivenih nizom svih mogućih izvođenja krenuvši od početnog simbola $\mathcal{S}$".

Primjer 2.3
Gramatika s produkcijama:

$$\begin{aligned} E &\rightarrow T \mid T+E \\ T &\rightarrow F \mid T*F \\ F &\rightarrow a \mid (E) \end{aligned}$$

generira jednostavne aritmetičke izraze. Ako je E početni simbol, rečenične forme su:

E T F a T+E T*F+E (E) a+a a*a (a)

Primjer izvođenja:

$$E \Rightarrow T+E \Rightarrow T*F+E \Rightarrow F*F+E \Rightarrow a*F+E \Rightarrow a*a+E \Rightarrow a*a+T \Rightarrow a*a+F \Rightarrow a*a+a$$

što se može napisati i kao

$$E \overset{8}{\Rightarrow} a*a+a \quad \text{ili} \quad E \overset{+}{\Rightarrow} a*a+a$$

Niz $a*a+a$ je jedna rečenica jezika koji gramatika s danim produkcijama generira.

Klasifikacija gramatika

Gramatike se mogu klasificirati prema obliku svojih produkcija. Najčešća je klasifikacija dana u sljedećoj definiciji.

Definicija 2.4
Za gramatiku $\mathcal{G}=(\mathcal{N},\mathcal{T},\mathcal{P},\mathcal{S})$ kaže se da je:

1) Linearna zdesna ili tipa 3 ako je svaka produkcija iz $\mathcal{P}$ oblika

$A \rightarrow xB$ ili $A \rightarrow x$ $\quad A,B\in\mathcal{N},\ x\in\mathcal{T}^*$

Linearna slijeva ako je svaka produkcija iz $\mathcal{P}$ oblika

$A \rightarrow Bx$ ili $A \rightarrow x$ $\quad A,B\in\mathcal{N},\ x\in\mathcal{T}^*$

Gramatika linearna zdesna naziva se regularna gramatika ako je svaka produkcija oblika

$A \rightarrow aB$ ili $A \rightarrow a$ $\quad A,B\in\mathcal{N},\ a\in\mathcal{T}$

i jedino je dopuštena produkcija $\mathcal{S}\rightarrow\varepsilon$, ali se tada $\mathcal{S}$ ne smije pojavljivati niti u jednoj alternativi ostalih produkcija.

2) Beskontekstna ili tipa 2 ako je svaka produkcija iz $\mathcal{P}$ oblika:

$A \rightarrow \alpha$ $\quad A\in\mathcal{N},\ \alpha\in(\mathcal{N}\cup\mathcal{T})^*$

3) Kontekstna ili tipa 1 ako je svaka produkcija iz $\mathcal{P}$ oblika

$\alpha \rightarrow \beta$

uz uvjet da je $|\alpha|\leq|\beta|$

4) Bez ograničenja ili tipa 0 ako produkcije ne zadovoljavaju nijedno od navedenih ograničenja.

Jezik je bez ograničenja ako je generiran gramatikom tipa 0, kontekstan ako je generiran gramatikom tipa 1, beskontekstan ako je generiran gramatikom tipa 2 i regularan ako je generiran gramatikom tipa 3. Četiri tipa gramatika i jezika uvedenih prethodnom definicijom nazivaju se hijerarhija Chomskog.

Svaka regularna gramatika istodobno je beskontekstna, beskontekstna bez ε–produkcija je kontekstna i, konačno, kontekstna gramatika je istodobno gramatika bez ograničenja. Ako s $\mathcal{L}_i$ označimo jezik tipa i, vrijedi $\mathcal{L}_{i+1}\subseteq\mathcal{L}_i$, $0\leq i<3$. Regularne gramatike generiraju najjednostavnije jezike koji mogu biti generirani regularnim izrazima. Idući prema gramatikama tipa 0 jezici su sve složeniji.

Primjer 2.4
Na idućoj slici prikazana je silueta jednog objekta aproksimirana gramatikama tipa 3, 2 i 1, a tek u potpunosti opisana gramatikom tipa 0:

Primjer 2.5
Evo primjera regularne, beskontekstne i kontekstne gramatike, te gramatike bez ograničenja:

1) Regularna:

$S \rightarrow aS \mid bS \mid \varepsilon$

generira jezik $\mathcal{L}=\{a,b\}^*$ tj. regularni skup označen regularnim izrazom $(a+b)^*$

2) Beskontekstna:

$S \rightarrow aSb \mid ab$

generira jezik $\mathcal{L}=\{a^n b^n : n \geq 1\}$.

3) Kontekstna:

$S \rightarrow aSBC \mid abc$
$bB \rightarrow bb$
$cC \rightarrow cc$
$CB \rightarrow BC$
$bC \rightarrow bc$

generira jezik $\mathcal{L}=\{a^n b^n c^n : n \geq 1\}$.

4) I, na kraju, primjer gramatike bez ograničenja:

$$
\begin{aligned}
S &\rightarrow CD \\
C &\rightarrow aCA \mid bCB \mid \varepsilon \\
AD &\rightarrow aD \\
BD &\rightarrow bD \\
Aa &\rightarrow aA \\
Ab &\rightarrow bA \\
Ba &\rightarrow aB \\
Bb &\rightarrow bB \\
D &\rightarrow \varepsilon
\end{aligned}
$$

generira jezik $\mathcal{L}=\{\omega\omega:\ \omega\in\{a,b\}^{*}\}$

Beskontekstni jezici

Od četiri klase gramatika u hijerarhiji Chomskog beskontekstne su najvažnije za primjene u jezicima za programiranje, posebno u definiranju njihove osnovne sintaksne strukture. Osim toga, beskontekstne gramatike koriste se kao osnova u raznim shemama za specificiranje prevođenja. U ovom i sljedećem poglavlju dane su definicije i transformacije beskontekstnih gramatika na kojima će se temeljiti proučavanje sintaksne analize beskontekstnih jezika.

Definicija 2.5
Za beskontekstnu gramatiku $\mathcal{G}=(\mathcal{N},\mathcal{T},\mathcal{P},S)$ kaže se da je:

1) Rekurzivna slijeva ako $\mathcal{P}$ sadrži produkciju oblika

$$A \rightarrow A\alpha \qquad \alpha\in(\mathcal{N}\cup\mathcal{T})^{+}$$

2) Rekurzivna zdesna ako $\mathcal{P}$ sadrži produkciju oblika

$$A \rightarrow \alpha A \qquad \alpha\in(\mathcal{N}\cup\mathcal{T})^{+}$$

3) Rekurzivna ako $\mathcal{P}$ sadrži produkciju oblika

$$A \rightarrow \alpha A\beta \qquad \alpha,\beta\in(\mathcal{N}\cup\mathcal{T})^{+}$$

ili ako je istodobno rekurzivna slijeva i zdesna, ili ako nije rekurzivna, ali postoji implicitna rekurzija:

$$A \overset{+}{\Rightarrow} \alpha A\beta \qquad \alpha,\beta\in(\mathcal{N}\cup\mathcal{T})^{+}$$

Gramatika rekurzivna slijeva, zdesna, odnosno, općenito rekurzivna, generira beskonačan jezik.

Primjer 2.6
Regularna gramatika iz prethodnog primjera:

$S \rightarrow aS \mid bS \mid \varepsilon$

rekurzivna je zdesna, a beskontekstna gramatika iz istog primjera:

$S \rightarrow aSb \mid ab$

jest rekurzivna. Gramatika s produkcijama:

$E \rightarrow E+E \mid E*E \mid (E) \mid a$

rekurzivna je "sa svih strana", tj. samo se kaže da je rekurzivna.

Stabla izvođenja

Moguće je imati nekoliko ekvivalentnih izvođenja u danoj gramatici, u smislu da sva izvođenja koriste iste produkcije na istom mjestu, ali različitim redom. Za gramatike bez restrikcija teško se može definirati kada su dva izvođenja ekvivalentna, ali u slučaju beskontekstnih gramatika to je moguće i čini se pomoću tzv. "stabla izvođenja".

Stablo izvođenja beskontekstne gramatike $\mathcal{G}=(\mathcal{N},\mathcal{T},\mathcal{P},\mathcal{S})$ označeno je uređeno stablo u kojem su čvorovi označeni znakovima iz $\mathcal{N}\cup\mathcal{T}\cup\{\varepsilon\}$. Ako je neki unutarnji čvor označen s A, a njegovi direktni slijednici su $X_1, X_2, \ldots, X_n$, tada je $A \rightarrow X_1X_2\ldots X_{n-1}X_n$ produkcija u $\mathcal{P}$.

Definicija 2.6
Označeno uređeno stablo $\mathcal{D}$ je <u>stablo izvođenja</u> (ili stablo sintaksne analize) beskontekstne gramatike $\mathcal{G}(\mathcal{S}) = (\mathcal{N},\mathcal{T},\mathcal{P},\mathcal{S})$ ako vrijedi:

1) Korijen od $\mathcal{D}$ označen je sa A.

2) Ako su $\mathcal{D}_1, \ldots, \mathcal{D}_k$ podstabla direktnih slijednika korijena i korijen od $\mathcal{D}_i$ je označen sa X_i, tada je

   $\mathcal{S} \rightarrow X_1 \ldots X_k$

   produkcija u $\mathcal{P}$. $\mathcal{D}_i$ mora biti stablo izvođenja za

   $\mathcal{G}(X_i) = (\mathcal{N},\mathcal{T},\mathcal{P},X_i)$

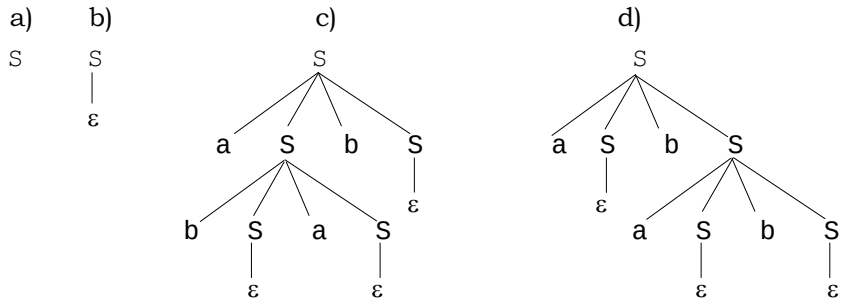
   ako je X_i neterminal, odnosno $\mathcal{D}_i$ je čvor (list) označen sa X_i ako je X_i terminal.

3) Inače, ako je $\mathcal{D}_1$ jedino podstablo korijena $\mathcal{D}$ i korijen od $\mathcal{D}_1$ označen je sa ε, tada je $\mathcal{S}\rightarrow\varepsilon$ produkcija u $\mathcal{P}$.

Primjer 2.7
Primjeri stabala izvođenja gramatike $\mathcal{G}$, definirane sa

$$S \rightarrow aSbS \mid bSaS \mid \varepsilon$$

dani su na sljedećoj slici:



Definicija 2.7
Granica stabla izvođenja jest niz koji se dobije nastavljanjem oznaka listova (u uređenju slijeva nadesno).

Primjer 2.8
Granice stabala izvođenja iz prethodnog primjera jesu:

a) S b) ε c) abab d) abab

Definicija 2.8
Neka je $\mathcal{G}=(\mathcal{N},\mathcal{T},\mathcal{P},\mathcal{S})$ beskontekstna gramatika. Pisat ćemo

$$\alpha \underset{lm}{\Rightarrow} \beta$$

ako je $\alpha=\omega A\gamma$, $\beta=\omega\delta\gamma$, $\omega\in\mathcal{T}^*$, $\delta\in(\mathcal{N}\cup\mathcal{T})^*$ i $A\rightarrow\delta$ je produkcija u $\mathcal{P}$. Odnosno, rečenična forma β dobivena je zamjenom prvog neterminalnog znaka slijeva u rečeničnoj formi α. Izvođenje

$$\alpha_0 \overset{*}{\underset{lm}{\Rightarrow}} \alpha_1 \overset{*}{\underset{lm}{\Rightarrow}} \ldots \overset{*}{\underset{lm}{\Rightarrow}} \alpha_n$$

je krajnje izvođenje slijeva za α_n iz α_0, u gramatici $\mathcal{G}$. Ako je

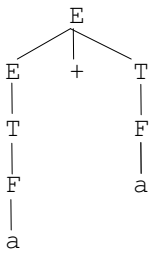
$$\mathcal{S} \overset{*}{\underset{lm}{\Rightarrow}} \alpha$$

α se naziva lijeva rečenična forma. Definira se izvođenje zdesna i krajnje izvođenje zdesna, analogno izvođenju slijeva. Na svim mjestima umjesto "lijevo" treba stajati "desno", a oznaka "lm" prelazi u "rm".

Primjer 2.9
Neka je $\mathcal{G}$ gramatika s produkcijama

$$E \rightarrow E+T \mid T \qquad T \rightarrow T*F \mid F \qquad F \rightarrow (E) \mid a$$

gdje je E početni simbol. Stablo izvođenja prikazano na sljedećoj slici predstavlja deset ekvivalentnih izvođenja rečenice a+a.



Krajnje izvođenje slijeva je:

$$E \Rightarrow E+T \Rightarrow T+T \Rightarrow F+T \Rightarrow a+T \Rightarrow a+F \Rightarrow a+a$$

a krajnje izvođenje zdesna:

$$E \Rightarrow E+T \Rightarrow E+F \Rightarrow E+a \Rightarrow T+a \Rightarrow F+a \Rightarrow a+a$$

Definicija 2.9
Kaže se da je beskontekstna gramatika $\mathcal{G}$ dvoznačna ako postoji najmanje jedna rečenica ω u $\mathcal{L}(\mathcal{G})$ za koju postoji više od jednog različitog stabla izvođenja s granicom ω. To je ekvivalentno tvrdnji da je $\mathcal{G}$ dvoznačna gramatika ako postoji rečenica ω u $\mathcal{L}(\mathcal{G})$ s dva ili više različitih krajnjih izvođenja slijeva (ili zdesna).

Primjer 2.10
Neka je $\mathcal{G}$ gramatika s produkcijama

$$E \rightarrow E+E \mid E*E \mid (E) \mid a$$

$\mathcal{G}$ je dvoznačna. Na primjer, rečenica a*a+a može se dobiti s dva različita niza izvođenja slijeva:

$$1)\ E \Rightarrow E*E \Rightarrow a*E \Rightarrow a*E+E \Rightarrow a*a+E \Rightarrow a*a+a$$
$$2)\ E \Rightarrow E+E \Rightarrow E*E+E \Rightarrow a*E+E \Rightarrow a*a+E \Rightarrow a*a+a$$

2.2 PRIKAZ GRAMATIKA

U prethodnim definicijama i primjerima razlikovali smo neterminalne i terminalne simbole prema vrsti znakova: velika slova bila su rezervirana za neterminale, a mala slova i ostali znakovi za terminale. Takvim dogovorom nije bilo neophodno uvijek posebno navoditi skupove neterminala i terminala. Bilo je dovoljno napisati produkcije i zadati početni simbol. Na taj način zadana je sintaksa jezika. Dva su najčešća načina prikaza gramatika: Backus-Naurovom formom i sintaksnim dijagramima.

Backus-Naurova forma (BNF)

Formalizam pisanja produkcija (ili "pravila zamjenjivanja") kojim smo dosad zadavali produkcije gramatike poznat je kao Backus-Naurova forma ili BNF. Prvi je put bio primijenjen u definiciji jezika ALGOL 60, 1963. godine. Često ćemo u praksi susresti drugi oblik pisanja BNF, prema sljedećim pravilima:

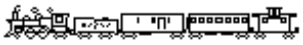
1) Neterminalni simboli pišu se između znakova “<” i “>”.
2) Umjesto “→” koristi se simbol “::=” i čita “definirano je kao”. Pišući neterminale između znakova “<” i “>” moguće je izborom njihovih imena uvesti “značenje” u produkcije, jer će nas imena podsjećati na vrstu rečenica koja će se generirati u nekom podjeziku.

Primjer 2.11

Sljedeća gramatika generira “jezik kaubojskih vlakova”:

```
<kaubojski vlak> ::= <lokomotiva><poštanska kola>
                     <dodatak>
<dodatak>        ::= {<putnička kola>}<kola za konje>|ε
<lokomotiva>     ::=
<poštanska kola> ::=
<putnička kola>  ::=
<kola za konje>  ::=
```

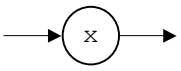
Primjeri rečenica:



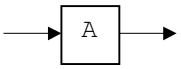
Sintaksni dijagrami

Produkcije gramatike G mogu biti prikazane i u obliku koji se naziva sintaksni dijagram. Sve je veća prisutnost sintaksnih dijagrama u novijoj literaturi, prije svega što se njihovom uporabom bolje uočava struktura jezika. Pravila konstruiranja sintaksnih dijagrama su sljedeća:

1) Terminalni simbol x prikazan je kao



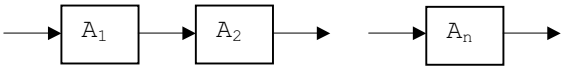
2) Neterminalni simbol A prikazan je kao



3) Produkcija oblika

$$A \rightarrow A_1 A_2 \ \ldots \ A_n \qquad A_i \in (\mathcal{N} \cup \mathcal{T})$$

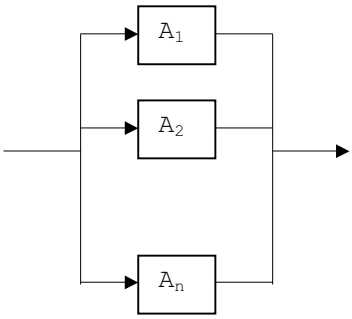
predstavlja se dijagramom



4) Produkcija oblika

$$A \rightarrow A_1 | A_2 | \ldots | A_n \qquad A_i \in (\mathcal{N} \cup \mathcal{T})$$

prikazuje se dijagramom

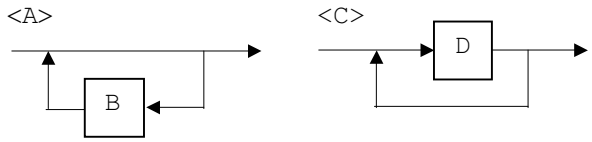


gdje je svaki A_i prikazan prema pravilima od (1) do (4). Ako je $A_i = \varepsilon$, ta se alternativa prikazuje punom crtom.

5) Produkcije oblika

$$A \rightarrow \{B\} \quad \text{i} \quad C \rightarrow [D] \qquad B, D \in (\mathcal{N} \cup \mathcal{T})$$

prikazuju se dijagramima:

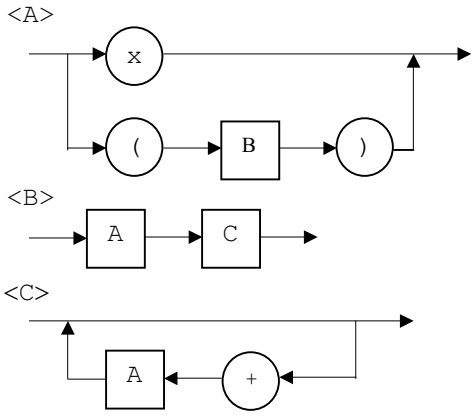


gdje su B i D prikazani dijagramima prema pravilima (1) do (4).

Primjer 2.12
Gramatika $\mathcal{G}$ s produkcijama $\mathcal{P}$

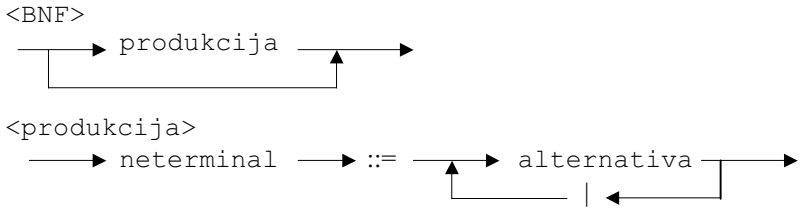
A → x | (B) B → AC C → {+A}

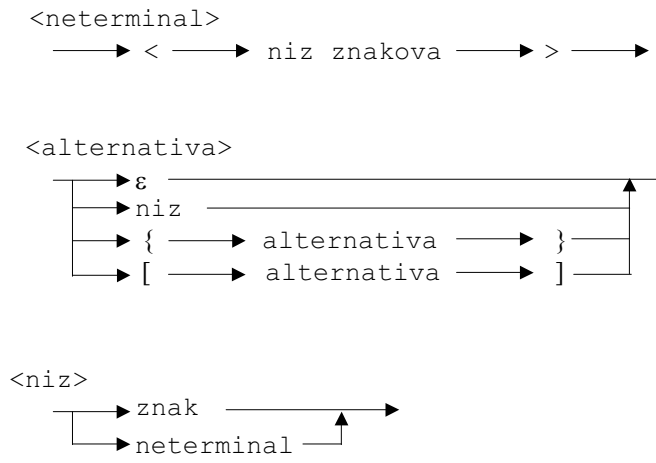
generira jezik $\mathcal{L}(\mathcal{G})$={x, (x), ((x)), . . . , (x+x), . . .}. Primjenjujući pravila (1) do (5), dobili bismo dijagrame



Često se u praksi pojednostavljuje pisanje sintaksnih dijagrama, posebno ako je nedvojbena razlika u pisanju terminala i neterminala. Na primjer, neterminali su riječi napisane malim slovima, a terminali su riječi napisane velikim slovima ili su to brojevi i ostali znakovi.

Primjer 2.13
Formalizam pisanja produkcija u BNF-u također je jezik! Evo beskontekstne gramatike jezika BNF prikazane u pojednostavljenom zapisu sintaksnih dijagrama:





gdje je <znak> znak alfabeta, a <niz znakova> izabrani skup znakova (slova, brojke).

Pitanja i zadaci

1) Dane su dvije gramatike, $\mathcal{G}_1=(\mathcal{N}_1,\mathcal{T}_1,\mathcal{P}_1,\mathcal{S}_1)$ i $\mathcal{G}_2=(\mathcal{N}_2,\mathcal{T}_2,\mathcal{P}_2,\mathcal{S}_2)$, $\mathcal{N}_1\cap\mathcal{N}_2=\varnothing$. Definirajte gramatiku $\mathcal{G}$ koja generira jezik

$$\mathcal{L}(\mathcal{G})=\{\omega_1\omega_2:\ \omega_1\in\mathcal{L}(\mathcal{G}_1),\ \omega_2\in\mathcal{L}(\mathcal{G}_2)\}$$

2) Definirajte regularne gramatike koje generiraju jezik označen sljedećim regularnim izrazima:

   a) $(0)^+(1)^+$
   b) $((00+11+(01+10)(00+11)^*(01+10))^*$
   c) $01(((ab)^*+ccc)^*+x)^*z$

3) Definirajte gramatike jezika:

   a) $\mathcal{L}_1 = \{1,2,3,4,5,6,\ ...\}$
   b) $\mathcal{L}_2 = \{2,4,6,8,10,12,\ ...\}$
   c) $\mathcal{L}_3 = \{1,3,5,7,9,11,\ ...\}$
   d) $\mathcal{L}_4 = \{4,8,12,16,20,\ ...\}$
   e) $\mathcal{L}_5 = \{3,6,9,12,15,18,...\}$
   f) $\mathcal{L}_6 = \{9,18,27,36,45,...\}$

4) Definirajte gramatiku jezika palindroma (nizova x koji imaju svojstvo da je $x=x^R$) nad alfabetom $\mathcal{A}=\{a,b,c\}$.

5) Definirajte gramatiku jezika rimskih brojeva i, ii, iii, iv, v, ..., mmmcmxcviii, mmmcmxcix.

6) Definirajte gramatiku jezika:

   a) $\mathcal{L}_1=\{xx^R : x\in\{0,1,a,b\}^+\}$, gdje je x^R reverzni niz niza x. Na primjer, u jeziku su: 00, 11, aa, bb, ..., 0a1bb1a0, itd.
   b) $\mathcal{L}_2=\{a^n b^{2n} : n\geq 1\}$
   c) $\mathcal{L}_3=\{a^n b^m c^m d^n : m\geq 1, n\geq 1\}$

7) Definirajte gramatiku jezika prijestupnih godina, od 2000. do 9996. godine. (Pazite, godine koje su djeljive sa 100 moraju biti djeljive i s 400 da bi bile prijestupne! Godina 2000. je bila prijestupna, ali godine 2100., 2200. i 2300. nisu itd.)

8) Definirajte gramatiku jezika datuma napisanih u obliku DD.MM.GGGG, gdje su:

   DD - dan (01, 02, ..., 31)
   MM - mjesec (01, 02, ..., 12)
   GGGG - godina (2000 do 9999)

   Na primjer, u jeziku su 01.01.2000 i 29.02.9996, a nisu 29.02.2100, 01.13.2000, itd.

9) Dana je gramatika:

   $S \rightarrow aB \mid bA$
   $A \rightarrow a \mid aS \mid bAA$
   $B \rightarrow b \mid bS \mid aBB$

   Za niz aaabbabbba treba naći:

   a) krajnje izvođenje slijeva
   b) krajnje izvođenje zdesna
   c) stablo izvođenja

10) Je li gramatika iz primjera 2.9 dvoznačna?

11) Definirajte gramatiku logičkih izraza u kojima će "varijable" biti `f`, `t`, `x` i `y`, "logičke operacije" `#` negacija, `$` disjunkcija i `&` konjunkcija. Izraz može sadržavati i zagrade.

12) Analizirajte popis literature u nekoj knjizi ili stručnom časopisu i probajte definirati gramatiku pravila njihova pisanja (ako postoje!). Produkcije gramatike prikažite sintaksnim dijagramima u kojima će neterminali biti riječi hrvatskoga jezika (na primjer, `<ime>`, `<izdavač>` itd.).

3.

IZVOĐENJE I TRANSFORMIRANJE GRAMATIKA

ako prođu kraj vas dobro ih pogledajte
i kao oni budite ludi i kao oni budite pijani
jer njihova jedina ludost je želja da budu slobodni

jedina im je
životna filozofija
"imamo čitav život za zabavljanje
imamo cijelu smrt za odmaranje"

i oni će ostariti, neka ostanu to što jesu
sa starim utopijama na čudne načine
ljubavnika, poeta i onih što spjevaju pjesme

jedina im je životna filozofija
"imamo čitav život za zabavljanje
imamo cijelu smrt za odmaranje"

filozofija
la philosophie
(georges moustaki/
dubravka stojnić)

to je zgodno društvo veseljaka
koji lijegaju u zoru i kasno ustaju
misleći samo na ljubav i sviranje gitare

jedina im je životna filozofija
"imamo čitav život za zabavljanje
imamo cijelu smrt za odmaranje"

ne rade ništa drugo do slavljenja svakoga trenutka
pozdrava punom mjesecu, proslave proljeća
pa im i ne preostaje vremena za rad

jedina im je životna filozofija
"imamo čitav život za zabavljanje
imamo cijelu smrt za odmaranje"

i često se prepoznam u njima
kao i oni tratio sam život na svim vjetrovima
i govorim si da su mi to braća ili djeca

jedina im je životna filozofija
"imamo čitav život za zabavljanje
imamo cijelu smrt za odmaranje"

I ovo je poglavlje posvećeno gramatikama. Prvi, "empirijski" dio, bavi se pokušajem dobivanja odgovora na pitanje: Kako za dani jezik izvesti odgovarajuću gramatiku? U drugom dijelu poglavlja, polazeći od pretpostavke da smo izveli kakvu-takvu gramatiku nekog jezika, dani su postupci transformiranja gramatike u jednostavniji oblik, a ponekad i u zadani oblik ili "formu".

3.1 IZVOĐENJE GRAMATIKA

Vidjeli smo da je za danu gramatiku moguće promatrati sve nizove izvođenja i dobiti skup svih rečenica jezika kojeg generira. Međutim, pitanje je kako se za dani jezik može definirati (izvesti) odgovarajuća gramatika i je li ona jedinstvena. Nažalost, ne postoji gotov "recept" koji bi propisao kako za dani jezik izvesti gramatiku.

Ponekad ćemo u nekom jeziku prepoznati podjezik za koji znamo gramatiku pa neće biti problema da se dodaju produkcije koje će generirati zadani jezik. Na primjer, ako smo definirali gramatiku prirodnih brojeva, neće biti problema da definiramo gramatiku parnih ili neparnih prirodnih brojeva, ili, ako smo definirali gramatiku cijelih brojeva, neće biti problema da definiramo gramatiku realnih brojeva napisanih u normalnom ili eksponencijalnom obliku kao što je, na primjer, u Pascalu.

Primjer 3.1

Izvedimo gramatiku jezika $\{a^n b^n : n \geq 1\}$. Prvo, ab je u jeziku, pa se može napisati:

$S \rightarrow ab$

Sljedeća je rečenica aabb, gdje smo sa ab označili prvu rečenicu. Ta se rečenica dobila dopisujući na početak prve rečenice znak a, na kraj znak b. Treća je rečenica aaabbb, dobila se istom operacijom, ali primijenjenoj na drugu rečenicu, itd. Dakle, ako je α rečenica, tada je i aαb također rečenica, a to je "klasični" primjer rekurzije, pa je gramatika danog jezika:

$S \rightarrow ab \mid aSb$

Primjer 3.2

No, izvođenje gramatike jezika $\{a^n b^n c^n : n \geq 1\}$ sigurno je složenije. Započet ćemo najjednostavnijom rečenicom: abc koja nam daje produkciju:

(1) $S \rightarrow abc$

Iduća je rečenica aabbcc. Ne možemo je izravno dobiti iz prve. Ideja je da se krene slijeva i ispred prvog b u abc umetne ponovo abc, pa imamo aabcbc, a to je nova alternativa:

(2)S → aSQ

Iz niza izvođenja

S ⇒ aSQ ⇒ aabcQ

bolje se vidi značenje novo uvedenog simbola Q. On mora biti zamijenjen na neki način s bc, a to se može postići ovako:

cQ → Qc
bQc → bbcc

Dakle, gramatika s produkcijama:

S → abc | aSQ
bQc → bbcc
cQ → Qc

generira jezik $\{a^n b^n c^n : n \geq 1\}$. Na primjer:

S ⇒ aSQ ⇒ aaSQQ ⇒ aaaSQQQ ⇒ aaaabcQQQ
⇒ aaaabQcQQ ⇒ aaaabbccQQ ⇒ aaaabbcQcQ
⇒ aaaabbQccQ ⇒ aaaabbbcccQ ⇒ aaaabbbccQc
⇒ aaaabbbcQcc ⇒ aaaabbbQccc ⇒ aaaabbbbcccc

Primjer 3.3
Izvedimo gramatiku jezika prirodnih brojeva {1,2,3,...}. Prvo, 1, 2, ..., 9 su u jeziku:

N → B
B → 1| 2| 3| 4| 5| 6| 7| 8| 9

Drugo, ako prirodni broj ima više od jedne znamenke, mora započeti s B, potom slijedi niz znamenki, npr. C

N → BC
C → B| 0| CC| ε

Primijetiti da se umjesto ove dvije produkcije u proširenom BNF-u može napisati:

N → B{C}
C → B| 0

Sada ne bi bio problem izvesti gramatike jezika parnih ili neparnih prirodnih brojeva, što vam ostavljamo za vježbu.

Primjer 3.4
Izvedimo gramatiku jezika {4,8,12,16,20,...}, tj. jezika koji sadrži prirodne brojeve djeljive s četiri. U matematici smo naučili da su to brojevi 4 i 8, odnosno, ako broj ima više od jedne znamenke, djeljiv je s četiri ako je broj sačinjen od njegove dvije posljednje znamenke djeljiv s četiri ili se završava s 00. Na primjer, u jeziku su nizovi 20, 1004, 12345678900 ili 101010196. Dakle, na najvišoj razini, možemo definirati gramatiku:

S → 4| 8| D| NE

gdje je N prirodni broj kako je definirano u primjeru 3.3, D predstavlja sve dvoznamenkaste brojeve djeljive sa četiri:

D → PX| QY
P → 1| 3| 5| 7| 9
X → 2| 6
Q → 2| 4| 6| 8
Y → 0| 4| 8

a E je:

E → 0Y| D

na primjer, rečenica 1004 može se izvesti u nizu izvođenja:

S ⇒ NE ⇒ BCE ⇒ 10E ⇒ 100Y ⇒ 1004

Primjer 3.5
A sada pokušajmo izvesti gramatiku jezika {3,6,9,12,15,18,...}, tj. jezika koji sadrži prirodne brojeve djeljive s 3. Znamo da je broj djeljiv s 3 ako je zbroj njegovih znamenki djeljiv s 3. Ali, kako u produkcijama beskontekstne gramatike zadovoljiti taj uvjet? Je li to moguće, kad znamo da u gramatici nisu definirane nikakve operacije?!

Dakako, iz promatranja beskonačnog skupa jezika brojeva djeljivih s 3 ne bismo mogli doći do rješenja. No, pokušajmo razmišljati ovako: ako je broj djeljiv s 3, zbroj ostataka dijeljenja svih njegovih znamenki s 3 jednak je nula ili je neki broj koji je također djeljiv s 3. To, na primjer, znači da ako broj započinje s 1, 4 ili 7 (ostatak dijeljenja s 3 jednak je 1), drugi dio broja mora sadržavati znamenku čiji je ostatak dijeljenja s tri jednak 2 (a to su 2, 5 ili 8) ili dvije znamenke čiji je ostatak dijeljenja jednak 1. Na primjer, brojevi 18, 111 ili 147 djeljivi su s 3. Dakle, na najvišoj razini, možemo definirati gramatiku:

S → AB| BA| C

gdje A predstavlja niz brojki od kojih je zbroj ostataka dijeljenja s 3 jednak 1, B niz brojki od kojih je zbroj ostataka dijeljenja s 3 jednak 2, a C niz brojeva koji počinju s 3, 6, ili 9. Dalje je:

A → 1| 4| 7
B → 2| 5| 8| BX| AA
A → 1| 4| 7| AX| BB

Na kraju, kompletna gramatika koja generira brojeve djeljive s 3 je:

S → AB| BA| C
A → 1| 4| 7| AX| BB
B → 2| 5| 8| BX| AA
C → Y| CX| CS
X → 0| Y| XS
Y → 3| 6| 9

Na primjer, rečenice 1200, 2322543 i 111 može se izvesti u nizu izvođenja:

S ⇒ AB ⇒ 1B ⇒ 1BX ⇒ 1BXX ⇒ 12XX ⇒ 120X ⇒ 1200
S ⇒ BA ⇒ BXA ⇒ 2XA ⇒ 2YA ⇒ 23A ⇒ 23BB ⇒ 232B
⇒ 232AA ⇒ 232BBA ⇒ 2322BA ⇒ 23225A ⇒ 23225AX
⇒ 2322254X ⇒ 232254Y ⇒ 2322543
S ⇒ AB ⇒ 1B ⇒ 1AA ⇒ 11A ⇒ 111

3.2. TRANSFORMIRANJE GRAMATIKA

U primjeru 2.9 dana je kontekstna gramatika koja generira isti jezik kao gramatika iz primjera 2.10. Uspoređujući te dvije gramatike sigurno ćemo zaključiti da nisu nimalo slične, ali su ekvivalentne! Što to znači?

Definicija 3.1
Kaže se da su dvije gramatike, $\mathcal{G}_1$ i $\mathcal{G}_2$, ekvivalentne ako je ispunjen uvjet:

$$\mathcal{L}(\mathcal{G}_1) = \mathcal{L}(\mathcal{G}_2)$$

Iz ove definicije slijedi zaključak da općenito skupovi neterminalnih znakova, produkcija i početni znak gramatika $\mathcal{G}_1$ i $\mathcal{G}_2$ mogu biti različiti. Isti je samo skup terminalnih znakova (alfabet). Ekvivalentnost gramatika jedini je uvjet koji mora biti ispunjen pri transformiranju jedne gramatike u drugu.

Kao što je već rečeno, ne postoje algoritmi niti nešto slično što bi moglo pomoći u automatskom izvođenju gramatike danog jezika. Najčešće će se do "prave" gramatike doći u nekoliko iteracija, podijelivši prije toga jezik na nekoliko disjunktnih skupova (podjezika) i potom čineći uniju dobivenih gramatika. Pri tome je korisno znati koje su transformacije dopuštene da bi se sačuvalo prvobitno značenje.

Transformacije imaju i šire značenje. Ponekad će se koristiti da bi se, na primjer, eliminirale rekurzije slijeva, izbacile prazne produkcije, ili, općenito, gramatika svela na neku formu ili tip. Kao što ćemo vidjeti u idućim poglavljima, posebno je važno gramatike svesti na odgovarajuće forme podesne za primjenu postupaka sintaksne analize. Ovdje su opisani mnogi postupci transformacija gramatika. Počinjemo s dva načina izravnog transformiranja gramatike, a to su supstitucija i faktorizacija.

Supstitucija

Neka je dana gramatika $G_1=(N_1, T, P_1, S)$, gdje je P

$$S \rightarrow 0A1$$
$$0A \rightarrow 00A1$$
$$A \rightarrow \varepsilon$$

Pokažimo na primjeru ove gramatike kako se supstitucijom može izvesti ekvivalentna gramatika. Prije toga treba uočiti da G_1 generira jezik

$$L(G_1)=\{01, 0011, 000111, \ldots\} = \{0^n1^n : n\geq 1\}$$

Najprije možemo uvesti novi neterminal, na primjer B, i produkciju

$$B \rightarrow 0A$$

pa se G_1 transformira u G_2 u kojoj je skup produkcija:

$$S \rightarrow B1$$
$$B \rightarrow 0B1$$
$$B \rightarrow 0A$$
$$A \rightarrow \varepsilon$$

Dakle, svako pojavljivanje niza 0A zamijenjeno je s B i dodana je produkcija B→0A. Primijetiti da je gramatika G_2 beskontekstna, dok je G_1 gramatika tipa 0.

Dalje se, s obzirom da je A→ε jedina alternativa za A, G_2 može transformirati u gramatiku G_3 s produkcijama:

$$S \rightarrow B1$$
$$B \rightarrow 0B1 \mid 0$$

To nije sve. Promatrajući jezik koji gramatike $\mathcal{G}_1$, $\mathcal{G}_2$ i $\mathcal{G}_3$ generiraju, može se izvesti gramatika $\mathcal{G}_4$ (primjer 3.1, samo umjesto terminala a i b imamo 0 i 1) s produkcijama:

$S \rightarrow 01 \mid 0S1$

Faktorizacija

Pokažimo postupak faktorizacije na primjeru gramatike $\mathcal{G}_4$. Obje alternative za S i počinju i završavaju istim znakom, s 0, odnosno s 1. Zbog toga se faktorizacija može sprovesti slijeva ili zdesna. Na primjer, zdesna bi bilo:

$S \rightarrow (0 \mid 0S)\ 1$

Ako se dio u zagradi (zagrade ovdje imaju ulogu metasimbola) zamijeni novim neterminalom, na primjer X, dobilo bi se

$S \rightarrow X1$
$X \rightarrow 0 \mid 0S$

Dalje se produkcija za X može faktorizirati slijeva:

$X \rightarrow 0\ (\varepsilon \mid S)$

pa se uvođenjem smjene, na primjer $Y \rightarrow \varepsilon|S$, dobiva gramatika $\mathcal{G}_5$ ekvivalentna svim prethodnim gramatikama:

$S \rightarrow X1$
$X \rightarrow 0Y$
$Y \rightarrow \varepsilon \mid S$

Izbacivanje neupotrebljivih simbola

Ponekad beskontekstna gramatika može sadržavati neupotrebljive neterminale, terminale ili, općenito, produkcije. Na primjer, gramatika s produkcijama:

$S \rightarrow aba$
$A \rightarrow c$

generira jezik {aba}. U izvođenju tog jezika neterminal A neće nikada biti upotrijebljen, niti znak c, pa se izbacivanjem produkcije za A dobiva ekvivalentna gramatika.

Definicija 3.2
Kaže se da je simbol $X \in \mathcal{N} \cup \mathcal{T}$ <u>neupotrebljiv</u> u beskontekstnoj gramatici $\mathcal{G}=(\mathcal{N}, \mathcal{T}, \mathcal{P}, \mathcal{S})$ ako ne postoji izvođenje $\mathcal{S}^* \Rightarrow wXy^* \Rightarrow wxy$, gdje su $w, x, y \in \mathcal{T}^*$.

Da bi se utvrdilo je li neterminal A neupotrebljiv u gramatici $\mathcal{G}$, treba prvo znati je li $\mathcal{L}(\mathcal{G})$ neprazan skup. Za to će nam trebati sljedeći algoritam:

Algoritam 3.1
Je li $\mathcal{L}(\mathcal{G})$ neprazan?

Ulaz
Beskontekstna gramatika $\mathcal{G}=(\mathcal{N}, \mathcal{T}, \mathcal{P}, \mathcal{S})$.

Izlaz
"da" ako je $\mathcal{L}(\mathcal{G})\neq\emptyset$, "ne" inače.

Postupak
Izgrade se skupovi $\mathcal{N}_0$, $\mathcal{N}_1$, ... rekurzivno, kao što slijedi:

(1) Neka je $\mathcal{N}_0=\emptyset$ i $i=1$.
(2) Neka je $\mathcal{N}_i=\{A: A\rightarrow\alpha$ je u $\mathcal{P}$, $\alpha\in(\mathcal{N}_{i-1}\cup\mathcal{T})^*\}\cup\mathcal{N}_{i-1}$.
(3) Ako je $\mathcal{N}_i\neq\mathcal{N}_{i-1}$, tada $i=i+1$, ide se na korak (2). Inače, $\mathcal{N}_e=\mathcal{N}_i$.
(4) Ako je $\mathcal{S}\in\mathcal{N}_e$, ispisati "da", inače ispisati "ne".

Budući da je $\mathcal{N}_e\subseteq\mathcal{N}$, postupak će se završiti poslije najviše $n+1$ koraka.

Osim provjere je li $\mathcal{L}(\mathcal{G})$ neprazan treba provjeriti sadrži li gramatika $\mathcal{G}$ i nedokučive simbole.

Definicija 3.3
Kaže se da je simbol $X\in\mathcal{N}\cup\mathcal{T}$ nedokučiv u beskontekstnoj gramatici $\mathcal{G}=(\mathcal{N}, \mathcal{T}, \mathcal{P}, \mathcal{S})$ ako se X ne pojavljuje niti u jednoj rečeničnoj formi.

Algoritam 3.2
Izbacivanje nedokučivih simbola.

Ulaz
Beskontekstna gramatika, $\mathcal{G}=(\mathcal{N}, \mathcal{T}, \mathcal{P}, \mathcal{S})$.

Izlaz
Ekvivalentna gramatika $\mathcal{G}'=(\mathcal{N}', \mathcal{T}', \mathcal{P}', \mathcal{S})$ tako da je $\mathcal{L}(\mathcal{G})=\mathcal{L}(\mathcal{G}')$ i za svaki $X\in\mathcal{N}'\cup\mathcal{T}'$ postoje $\alpha,\beta\in(\mathcal{N}'\cup\mathcal{T}')^*$ tako da je $\mathcal{S}^*\Rightarrow\alpha X\beta$ u $\mathcal{G}'$.

Postupak
(1) $V_0=\{\mathcal{S}\}$, $i=1$.
(2) $V_i=\{X: (A\rightarrow\alpha X\beta)\in\mathcal{P} \wedge A\in V_{i-1}\}\cup V_{i-1}$.
(3) Ako je $V_i\neq V_{i-1}$, $i=i+1$ i ide se na korak (2). Inače:

$\mathcal{N}' = V_i \cap \mathcal{N}$
$\mathcal{T}' = V_i \cap \mathcal{T}$
$\mathcal{P}'$ će biti one produkcije iz $\mathcal{P}$ koje sadrže samo simbole iz V_i
$\mathcal{G}' = (\mathcal{N}', \mathcal{T}', \mathcal{P}', \mathcal{S})$

Sada se može može dati algoritam za izbacivanje neupotrebljivih simbola.

Algoritam 3.3
Izbacivanje neupotrebljivih simbola.

Ulaz
Beskontekstna gramatika $\mathcal{G}=(\mathcal{N}, \mathcal{T}, \mathcal{P}, \mathcal{S})$, tako da je $\mathcal{L}(\mathcal{G}) \neq \emptyset$.

Izlaz
Ekvivalentna gramatika $\mathcal{G}'=(\mathcal{N}', \mathcal{T}', \mathcal{P}', \mathcal{S})$, tako da je $\mathcal{L}(\mathcal{G})=\mathcal{L}(\mathcal{G}')$, i ne postoji nijedan neupotrebljiv simbol u $\mathcal{N}' \cup \mathcal{T}'$.

Postupak
(1) Upotrijebiti algoritam 3.1 da bi se dobio skup N_e. Tada je gramatika $\mathcal{G}_1=(\mathcal{N} \cap N_e, \mathcal{T}, \mathcal{P}_1, \mathcal{S})$, gdje $\mathcal{P}_1$ sadži samo one produkcije iz $\mathcal{P}$ koje uključuju simbole iz $N_e \cup \mathcal{T}$.
(2) Primijeniti algoritam 3.2 na $\mathcal{G}_1$ da bi se dobilo $\mathcal{G}'=(\mathcal{N}', \mathcal{T}', \mathcal{P}', \mathcal{S})$.

Korak (1) algoritma 3.3 izbacuje sve neterminale iz $\mathcal{G}$ koji ne mogu izvesti niz terminala. Potom se u koraku (2) izbacuju svi nedokučivi simboli. Svaki simbol X u rezultirajućoj gramatici mora se pojaviti najmanje jedanput u izvođenju $\mathcal{S} \Rightarrow^* wXy \Rightarrow^* wxy$. Da smo najprije upotrijebili algoritam 3.2, potom algoritam 3.1, ne bismo uvijek dobili konačnu gramatiku bez neupotrebljivih simbola.

Primjer 3.6
Neka je $\mathcal{G}=(\{S,A,B\},\{a,b\},\mathcal{P},S)$, gdje je skup produkcija $\mathcal{P}$:

$$S \rightarrow a|A \qquad A \rightarrow AB \qquad B \rightarrow b$$

Primijenimo algoritam 3.3 na tu gramatiku. U koraku (1) je $N_e=\{a,b\}$, tako da je $\mathcal{G}_1=(\{S,B\},\{a,b\},\{S \rightarrow a, B \rightarrow b\},S)$. Dalje, primjenom algoritma 3.2, dobili bismo $V_2=V_1=\{S,a\}$. Dakle, $\mathcal{G}'=(\{S\},\{a\},\{S \rightarrow a\},S)$.

Da smo prvo upotrijebili algoritam 3.2 na gramatiku $\mathcal{G}$, zaključili bismo da su svi simboli dokučivi, pa bi gramatika ostala nepromijenjena. Primjenom potom algoritma 3.1, dobili bismo $N_e=\{S,B\}$, tako da bi rezultirajuća gramatika bila $\mathcal{G}_1$, a ne $\mathcal{G}'$.

Izbacivanje ε-produkcija

Često će biti potrebno transformirati izvornu gramatiku koja ima ε-produkcije u ekvivalentnu gramatiku bez njih. Dakako, ako je $\varepsilon \in \mathcal{L}(\mathcal{G})$, tada mora postojati produkcija $\mathcal{S} \rightarrow \varepsilon$.

Definicija 3.4
Kaže se da je gramatika $\mathcal{G}=(\mathcal{N}, \mathcal{T}, \mathcal{P}, \mathcal{S})$ bez ε-produkcija ako

(1) $\mathcal{P}$ ne sadrži ε-produkcije, ili
(2) Postoji samo jedna ε-produkcija $\mathcal{S} \rightarrow \varepsilon$ i $\mathcal{S}$ se ne pojavljuje niti u jednoj alternativi (desnoj strani) ostalih produkcija iz $\mathcal{P}$.

Algoritam 3.4
Izbacivanje ε-produkcija.

Ulaz
Beskontekstna gramatika $\mathcal{G}=(\mathcal{N}, \mathcal{T}, \mathcal{P}, \mathcal{S})$.

Izlaz
Ekvivalentna gramatika $\mathcal{G}'=(\mathcal{N}', \mathcal{T}, \mathcal{P}', \mathcal{S}')$.

Postupak
(1) Izgraditi $N_e=\{A: A \in \mathcal{N} \text{ i } A^+ \Rightarrow \varepsilon\}$ slično kao u algoritmu 3.1.

(2) Neka je $\mathcal{P}'$ skup produkcija izgrađen na sljedeći način:

(a) Ako je $A \rightarrow \alpha_0 B_1 \alpha_1 B_2 \ldots B_k \alpha_k$ u P, $k \geq 0$, i za $1 \leq i \leq k$ svaki B_i je u N_e, ali nijedan simbol iz α_j nije u N_e, $0 \leq j \leq k$, dodati u $\mathcal{P}'$ sve produkcije oblika:

$$A \rightarrow \alpha_1 X_1 \alpha_1 \ldots B_k \alpha_k$$

gdje je X_i ili B_i ili ε, bez dodavanja $A \rightarrow \varepsilon$ u $\mathcal{P}'$ (to će se dogoditi ako su svi $\alpha_i=\varepsilon$).

(b) Ako je $\mathcal{S} \in N_e$, dodati u $\mathcal{P}'$ produkcije:

$$\mathcal{S}' \rightarrow \varepsilon | \mathcal{S}$$

gdje je $\mathcal{S}'$ novi (početni) simbol i $\mathcal{N}'=\mathcal{N} \cup \{\mathcal{S}'\}$. Inače je $\mathcal{N}'=\mathcal{N}$ i $\mathcal{S}'=\mathcal{S}$.

(3) Nova ekvivalentna gramatika je $\mathcal{G}'=(\mathcal{N}', \mathcal{T}, \mathcal{P}', \mathcal{S}')$.

Primjer 3.7
Primijenimo algoritam 3.4. na gramatiku $\mathcal{G}$ s produkcijama:

S → aSbS | bSaS | ε

Konačno će se dobiti gramatika $\mathcal{G}'$ s produkcijama:

S'→S| ε S →aSbS| bSaS| aSb| abS| ab| bSa| baS| ba

Izbacivanje jediničnih produkcija

Korisna transformacija gramatika jest izbacivanje produkcija oblika $A \to B$ ili tzv. jediničnih produkcija.

Algoritam 3.5
Izbacivanje jediničnih produkcija.

Ulaz
Beskontekstna gramatika $\mathcal{G}=(\mathcal{N}, \mathcal{T}, \mathcal{P}, \mathcal{S})$ bez ε–produkcija.

Izlaz
Ekvivalentna beskontekstna gramatika $\mathcal{G}'$ bez jediničnih produkcija i bez ε–produkcija.

Postupak
(1) Izgraditi za svaki A iz $\mathcal{N}$ skup $N_A=\{B : A \overset{*}{\Rightarrow} B\}$ na sljedeći način:

(a) $N_0=\{A\}$, $i=1$.
(b) $N_i=\{C : (B \to C) \in \mathcal{P} \wedge B \in N_{i-1}\} \cup N_{i-1}$.
(c) Ako je $N_i \neq N_{i-1}$, $i=i+1$ i ide se na korak (b). Inače, $N_A=N_i$.

(2) Izgraditi $\mathcal{P}'$ tako da ako je $B \to \alpha$ u $\mathcal{P}$ i nije jedinična produkcija, produkcijama $\mathcal{P}'$ dodati sve $A \to \alpha$ za koje je $B \in N_A$.

(3) Nova ekvivalentna gramatika je $\mathcal{G}' = (\mathcal{N}, \mathcal{T}, \mathcal{P}', \mathcal{S})$.

Primjer 3.8
Primijenimo algor itam 3.5. na gramatiku $\mathcal{G}$ s produkcijama

$$
\begin{aligned}
E &\to E+T \mid T \\
T &\to T*F \mid F \\
F &\to (E) \mid a
\end{aligned}
$$

U koraku (1) je $N_E=\{E, T, F\}$, $N_T=\{T, F\}$, i $N_F=\{F\}$. Poslije koraka (2), $\mathcal{P}'$ postaje:

$$
\begin{aligned}
E &\to E+T \mid T*F \mid (E) \mid a \\
T &\to T*F \mid (E) \mid a \\
F &\to (E) \mid a
\end{aligned}
$$

Definicija 3.5
Beskontekstna gramatika $\mathcal{G}$ je svojstvena (engl. *proper*) ako nema produk-cija $A \to \varepsilon$ i nema petlji, tj. ne postoji niz izvođenja $A \overset{+}{\Rightarrow} A$.

Chomskyjeva normalna forma (CNF)

Dosad smo se bavili transformacijama gramatika kojima smo iz nekih razloga "popravljali" prvobitno definiranu gramatiku dobijajući ekvivalentne gramatike bez praznih ili jediničnih produkcija, ili izbacujući suvišne produkcije. Sada ćemo definirati posebnu formu gramatika poznatu kao "Chomskyjeva normalna forma" (CNF), koja je posebno važna za primjene nekih postupaka sintaksne analize danih u petom poglavlju ove knjige.

Definicija 3.6
Kaže se da je gramatika $\mathcal{G}=(\mathcal{N},\mathcal{T},\mathcal{P},\mathcal{S})$ u Chomskyjevoj normalnoj formi (CNF) ako je svaka produkcija u $\mathcal{P}$ oblika:

(1) $A\rightarrow BC$, $A,B,C\in\mathcal{N}$, ili
(2) $A\rightarrow a$, $a\in\mathcal{T}$, ili
(3) Ako je ε u jeziku $\mathcal{L}(\mathcal{G})$, tada je $\mathcal{S}\rightarrow\varepsilon$ i $\mathcal{S}$ se ne pojavljuje niti u jednoj alternativi (desnoj strani) ostalih produkcija iz $\mathcal{P}$.

Može se dokazati da svaka beskontekstna gramatika ima ekvivalentnu gramatiku u CNF. Za takvu je transformaciju dobrodošao sljedeći algoritam:

Algoritam 3.6
Konverzija u CNF.

Ulaz
Svojstvena beskontekstna gramatika $\mathcal{G}=(\mathcal{N},\mathcal{T},\mathcal{P},\mathcal{S})$.

Izlaz
Ekvivalentna gramatika $\mathcal{G}'=(\mathcal{N}',\mathcal{T},\mathcal{P}',\mathcal{S})$ u CNF.

Postupak
Skup produkcija $\mathcal{P}'$ izgraditi na sljedeći način:
(1) Dodati sve produkcije iz $\mathcal{P}$ oblika $A\rightarrow a$ u $\mathcal{P}'$.
(2) Dodati sve produkcije iz $\mathcal{P}$ oblika $A\rightarrow BC$ u $\mathcal{P}'$.
(3) Ako je $\mathcal{S}\rightarrow\varepsilon$ u $\mathcal{P}$ dodati $\mathcal{S}\rightarrow\varepsilon$ skupu produkcija $\mathcal{P}'$.
(4) Za sve produkcije oblika $A\rightarrow X_1\ldots X_k$ iz $\mathcal{P}$, $k>2$, iz $\mathcal{P}$ dodati u $\mathcal{P}'$ sljedeći skup produkcija. Neka X'_i stoji za X_i, $X_i\in\mathcal{N}\cup\mathcal{T}$, i X'_i je novi neterminal ako je $X_i\in\mathcal{T}$. Preuređene produkcije su:

$$
\begin{aligned}
A &\rightarrow X'_1\langle X_2\ldots X_k\rangle\\
\langle X_2\ldots X_k\rangle &\rightarrow X'_2\langle X_3\ldots X_k\rangle\\
&\ldots\\
\langle X_{k-2}\ldots X_k\rangle &\rightarrow X'_{k-2}\langle X_{k-1}X_k\rangle\\
\langle X_{k-1}\ldots X_k\rangle &\rightarrow X'_{k-1}X'_k
\end{aligned}
$$

gdje je svaki $\langle X_i\ldots X_k\rangle$ novi neterminal.

(5) Za sve produkcije oblika $A \to X_1X_2$ iz $\mathcal{P}$, gdje je $X_1 \in \mathcal{T} \vee X_2 \in \mathcal{T}$, dodati u $\mathcal{P}'$ produkciju $A \to X'_1X'_2$.

(6) Za svaki neterminal oblika a', uvedenog u koracima (4) i (5), dodati u produkciju $a' \to a$. Konačno, skup neterminala $\mathcal{N}'$ je skup $\mathcal{N}$ kojem su dodani svi novouvedeni neterminali, pa je gramatika u CNF.

Primjer 3.9
Primijenimo algoritam 3.6 na gramatiku $\mathcal{G}$ s produkcijama

```
S → aAB | BA
A → BBB | a
B → AS  | b
```

Najprije se, prema koracima (1) i (2) u $\mathcal{P}'$ mogu prenijeti produkcije:

```
S → BA
A → a
B → AS | b
```

Zatim zamjenjujemo S→aAB s S→a'<AB> i <AB> s <AB>→AB. Potom je zamijenjeno A→BBB s A→B<BB> i <BB> s <BB>→BB. Konačno, u $\mathcal{P}'$ se dodaje produkcija a'→a, pa je rezultirajuća gramatika sa skupom produkcija P':

```
   S → a'<AB> | BA
   A → B<BB>  | a
   B → AS     | b
<AB> → AB
<BB> → BB
  a' → a
```

Ako uvedemo supstituciju X za a', Y za <AB> i Z za <BB>, s ciljem da neterminale prikažemo samo jednim slovom, dobili bismo pregledniju gramatiku:

```
S → XY | BA
A → BZ | a
B → AS | b
Y → AB
Z → BB
X → a
```

Eliminiranje rekurzija slijeva

Transformacija gramatike koja je rekurzivna slijeva u ekvivalentnu gramatiku koja to nije, česta je u teoriji formalnih jezika, posebno u teoriji sintaksne analize.

Teorem 3.1
Svaki beskontekstni jezik ima gramatiku nerekurzivnu slijeva.

Dokaz teorema 3.1 nije važan za naše bavljenje teorijom formalnih jezika. Važno je samo upamtiti tu činjenicu. Također je korisno znati i sljedeći teorem.

Teorem 3.2
Ako je $\mathcal{G}$ gramatika $\mathcal{G}=(\mathcal{N},\mathcal{T},\mathcal{P},\mathcal{S})$ u kojoj su:

$$A \rightarrow A\alpha_1|\ldots|A\alpha_m|\beta_1|\ldots|\beta_n$$

sve A-produkcije u $\mathcal{P}$ i nijedan β_i ne počinje s A, tada je $\mathcal{G}'$ gramatika ekvivalentna gramatici $\mathcal{G}$, $\mathcal{G}'=(\mathcal{N}'\cup\{A'\},\mathcal{T},\mathcal{P}',\mathcal{S})$, gdje je A' novi neterminal, a $\mathcal{P}'$ je dobiveno iz $\mathcal{P}$ u kojem su produkcije zamijenjene sa:

$$A \rightarrow \beta_1|\ldots|\beta_n|\beta_1A'|\ldots|\beta_nA'$$
$$A' \rightarrow \alpha_1|\ldots|\alpha_m|\alpha_1A'|\ldots|\alpha_mA'$$

Primjer 3.9
Primijenimo transformaciju iz teorema 3.2. na gramatiku $\mathcal{G}$ s produkcijama:

```
E → E+T | T
T → T*F | F
F → (E) | a
```

Dobit ćemo gramatiku $\mathcal{G}'$ s produkcijama:

```
E  → T   | TE'
E' → +T  | +TE'
T  → F   | FT'
T' → *F  | *FT'
F  → (E) | a
```

odnosno, ako E' zamijenimo s X, a T' s Y, imamo:

```
E → T   | TX
X → +T  | +TX
T → F   | FY
Y → *F  | *FY
F → (E) | a
```

Algoritam 3.7
Eliminiranje rekurzija slijeva.

Ulaz
Svojstvena beskontekstna gramatika $\mathcal{G}=(\mathcal{N},\mathcal{T},\mathcal{P},\mathcal{S})$.

Izlaz
Ekvivalentna gramatika $\mathcal{G}'=(\mathcal{N}',\mathcal{T},\mathcal{P}',\mathcal{S})$ nerekurzivna slijeva.

Postupak
Teorem 3.2 je temeljni za eliminiranje rekurzija slijeva. Skup produkcija $\mathcal{P}'$ izgraditi na sljedeći način:

(1) Neka je $\mathcal{N}=\{A_1, \ldots, A_n\}$. Treba transformirati $\mathcal{G}$ tako da ako je $A_i \rightarrow \alpha$ produkcija, tada α počinje ili terminalom ili neterminalom A_j tako da je $j>i$. Postaviti $i=1$.
(2) Neka je A_i-produkcija oblika $A_i \rightarrow A_i\alpha_1|\ldots|A_i\alpha_m|\beta_1|\ldots|\beta_p$ gdje nijedan β_j ne počinje s A_k ako je $k\leq i$. (Uvijek će to biti moguće.) Zamijeniti A_i-produkcije sa:

$$A_i \rightarrow \beta_1|\ldots|\beta_p\ |\beta_1A'_i|\ldots|\beta_pA'_i$$
$$A'_i \rightarrow \alpha_1|\ldots|\alpha_m|\alpha_1A'_i|\ldots|\alpha_mA'_i$$

gdje je A'_i novi neterminal. Sada sve A_i-produkcije počinju terminalom ili s A_k, $k>i$.

(3) Ako je $i=n$, $\mathcal{G}'$ će biti rezultirajuća gramatika i prekida se daljnje izvođenje postupka. Inače, $i=i+1$ i $j=1$.
(4) Zamijeniti sve produkcije oblika $A_i \rightarrow A_j\alpha$, gdje je $A_j \rightarrow \beta_1|\ldots|\beta_m$, produkcijama $A_i \rightarrow \beta_1\alpha|\ldots|\beta_m\alpha$. Ako je A_j-produkcija počinjala terminalom ili s A_k, $k>j$, isto svojstvo će sada imati A_i-produkcija.
(5) Ako je $j=i-1$, ide se na korak (2). Inače, $j=j+1$ i ide se na korak (4).

Primjer 3.10
Primijenimo algoritam 3.7 na gramatiku $\mathcal{G}$ s produkcijama:

$$A \rightarrow BC \mid a$$
$$B \rightarrow CA \mid Ab$$
$$C \rightarrow AB \mid CC \mid a$$

Odmah uočavamo da je C-produkcija rekurzivna slijeva, dok A i B imaju skrivene rekurzije, također slijeva. Neka je $A_1=A$, $A_2=B$ i $A_3=C$. Prikazana je promjena gramatika poslije svake primjene koraka (2) ili (4), ali samo nove produkcije neterminala čije su produkcije promijenjene:

(2), i=1: nema promjene

(4), i=2, j=1: B → CA|BCb|ab

(2), i=2: B → CA|ab|CAB'|abB'
B'→ CbB'|Cb

(4), i=3, j=1: C → BCB|aB|CC|a

(4), i=3, j=2: C → CACB|abCB|CAB'CB|abB'CB|aB|CC|a

(2), i=3: C → abCB|abB'CB|aB|a|abCBC'|abB'CBC'|aBC'|aC'
C'→ ACBC'|AB'CBC'|CC'|ACB|AB'CB|C

Konačno, uvodeći smjenu X za B' i Y za C', rezultirajuća gramatika bez rekurzija slijeva ima produkcije:

A → BC | a
B → CA | ab | CAX | abX
X → CbX | Cb
C → abCB | abXCB | aB | a | abCBY | abXCBY | aBY | aY
Y → ACBY | AXCBY | CY | ACB | AXCB | C

Greibachova normalna forma (GNF)

Završavamo ovo poglavlje s još jednom formom beskontekstnih gramatika, tzv. "Greibachovom normalnom formom" (GNF). Karakteristika je te forme da svaka produkcija mora početi terminalom (iz čega slijedi da ne može biti rekurzivna slijeva). Kao što ćemo vidjeti, prvi preduvjet transformacije gramatike u tu formu jest da gramatika ne smije biti rekurzivna slijeva, pa već imamo primjenu algoritma 3.7. Najprije definicija Greibachove normalne forme:

Definicija 3.7
Kaže se da je gramatika $\mathcal{G}=(\mathcal{N}, \mathcal{T}, \mathcal{P}, \mathcal{S})$ u Greibachovoj normalnoj formi (GNF) ako je $\mathcal{G}$ bez ε–produkcija i ako je svaka produkcija iz $\mathcal{P}$ oblika:

$A \rightarrow a\alpha \quad a \in T, \alpha \in N^*$

Ako gramatika nije rekurzivna slijeva, moguće je naći linearno uređenje označeno s < na skupu neterminala $\mathcal{N}$ tako da ako je $A \rightarrow B\alpha$ produkcija u $\mathcal{P}$, vrijedi A<B.

Algoritam 3.8
Konverzija gramatike u Greibachovu normalnu formu (GNF).

Ulaz
Svojstvena beskontekstna gramatika $\mathcal{G}=(\mathcal{N}, \mathcal{T}, \mathcal{P}, \mathcal{S})$ koja nije rekurzivna slijeva.

Izlaz
Ekvivalentna gramatika $\mathcal{G}'=(\mathcal{N}',\mathcal{T},\mathcal{P}',\mathcal{S})$ u Greibachovoj normalnoj formi.

Postupak

(1) Linearno urediti skup neterminala, $\mathcal{N}=\{A_1,\ldots,A_n\}$ tako da vrijedi: $A_1<A_2<\ldots<A_n$.

(2) `i = n-1`

(3) Ako je i=0 ide se na korak (5). Inače, zamijeniti svaku produkciju oblika $A_i\rightarrow A_j\alpha$, $j>i$, s $A_i\rightarrow\beta_1\alpha|\ldots|\beta_m\alpha$, gdje je $A_j\rightarrow\beta_1|\ldots|\beta_m$, tj. β_k su sve alternative od A_j koje počinju terminalom.

(4) `i = i-1` i vraća se na korak (3).

(5) Sada sve produkcije (osim $\mathcal{S}\rightarrow\varepsilon$, ako postoji) počinju terminalom. U svakoj produkciji, $A\rightarrow aX_1\ldots X_k$, treba zamijeniti X_j, ako je terminal, s X'_j, gdje je X'_j novi neterminal.

(6) Za sve X'_j uvedene u koraku (5) dodati produkciju $X'_j\rightarrow X_j$.

Primjer 3.11
Transformirajmo rezultirajuću gramatiku iz primjera 3.9 u GNF:

```
E → T   | TX
X → +T  | +TX
T → F   | FY
Y → *F  | *FY
F → (E) | a
```

Prvo uvedimo linearno uređenje na skupu neterminala: `X<E<Y<T<F`. Neterminal `F` je najveći u tom uređenju i sve njegove alternative počinju terminalom. `T` je sljedeći simbol s produkcijama $T\rightarrow F|FY$, tako da se `F` supstituira sa svojim alternativama, pa se dobije:

```
T → (E)|a|(E)Y|aY
```

Sada je `Y` na redu, ali sve njegove alternative počinju terminalom pa nema promjena. Slijedi zamjena `E`-produkcija sa:

```
E → (E)| a| (E)Y| aY| (E)X| aX| (E)YX| aYX
```

X-produkcije započinju terminalom pa nije potrebna transformacija. Koraci (5) i (6) uvode novi neterminal)' i produkciju:

)' →)

Na svim mjestima pojavljivanja terminala) treba stajati)', no mi ćemo, opet radi preglednosti, umjesto) pisati Z, pa je rezultirajuća gramatika u Greibachovoj normalnoj formi:

E → (EZ| a | (EZY | aY | (EZX | aX | (EZYX | aYX
X → +T | +TX
T → (EZ | a | (EZY | aY
Y → *F | *FY
F → (E) | a
Z →)

Neželjena posljedica transformacije gramatike u GNF, što je vidljivo i iz prethodnog primjera, jest veliki broj produkcija, pa je takva gramatika nepreglednija od svojeg originala.

Pitanja i zadaci

1) Transformirajte sintaksne dijagrame iz primjera 2.12 u jednostavniji oblik.

2) Pokažite da se gramatika G s produkcijama

S → B1 B → 0B1 | 0

može transformirati u ekvivalentnu gramatiku s produkcijama

S → 01 | 0S1

3) Nađite gramatiku bez ε–produkcija ekvivalentnu gramatici:

S → ABC A → BB | ε
B → CC | a C → AA | b

4) Nađite svojstvenu gramatiku ekvivalentnu gramatici:

S → A | B A → C | D B → D | E
C → S | a | ε D → S | b E → S | c | ε

5) Transformirajte sljedeće gramatike u Chomskyevu normalnu formu:

a) S → 0S1 | 01
b) S → aB | bA
   A → aS | bAA | a
   B → bS | aBB | b

6) Transformirajte iduću gramatiku u Greibachovu normalnu formu:

S → Ba | Ab A → Sa | AAb | a B → Sb | BBa | b

4.

UVOD U TEORIJU AUTOMATA

sunce se popelo visoko
lagano k'o na jutro uskrsno
a ja ležim ispružen
u svojoj mreži za spavanje

to traje iz godine u godinu
takav je moj znak u zodijaku
možda sam stvarno rođen
u svojoj mreži za spavanje

ponekad zaželim raditi
ali tu je lijenčina u meni što u
protunapad kreće
i u mrežu za spavanje
jastuk mi podmeće

inače vidjevši druge kako rade
dobro shvaćam da ih to uništava
a ja čelična sam zdravlja
u svojoj mreži za spavanje

niti mi je hladno niti vruće
niti sam gladan niti žedan
vjetar mi lagano mrsi kosu
i miluje kožu

novca ipak treba naći
no ja sam strašno snalažljiv
i pustim da mi plate za patent
moje mreže za spavanje

to je izmišljena mreža za uživanje
udobna kao Cadillac
gotovo jedan dom jedno ljubavno
gnijezdo
ta moja mreža za spavanje

čim se u zraku osjeti
slatki afrodizijački miris
može se vidjeti potrgano lišće
u mojoj mreži za spavanje

ali iako ima mjesta za jednog
kada smo dvoje sve se mijenja i kida
kad sve uzmem u obzir
jednako nam je dobro
na travi

u svojoj mreži za spavanje
dans mon hamac
(georges moustaki/ sanja seljan)

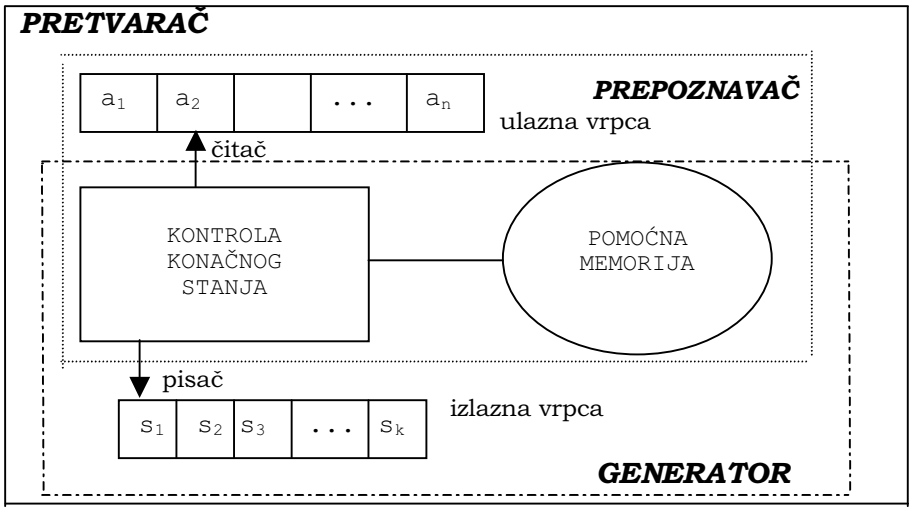
Osim gramatika, uvodimo automate kao važnu klasu generatora jezika, posebno pogodnih za implementaciju na računalima.

4.1 UVOD

Teorija je konačnih automata koristan instrument za razvoj sustava s konačnim brojem stanja čije mnogobrojne primjene nalazimo i u informatici. Programi, kao što je npr. tekstovni editor, često su načinjeni kao sustavi s konačnim brojem stanja. Na primjer, računalo se zasebno također može promatrati kao sustav s konačno mnogo stanja. Upravljačka jedinica, memorija i vanjska memorija nalaze se teoretski u svakom trenutku u jednom od vrlo velikog broja stanja, ali još uvijek u konačnom skupu stanja.

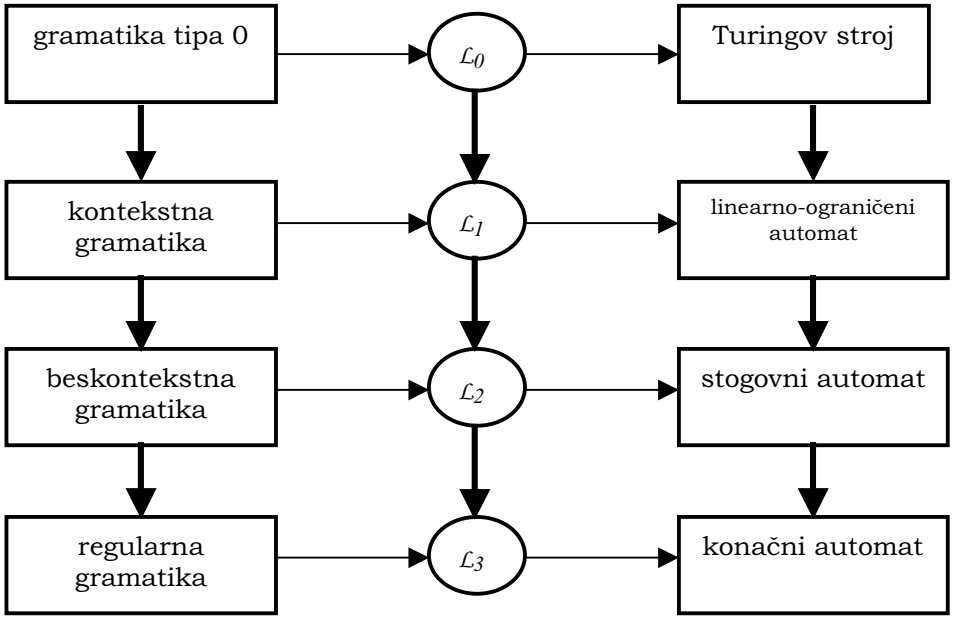
Iz svakodnevnog je života upravljački mehanizam dizala još jedan dobar primjer sustava s konačno mnogo stanja. Prirodnost koncepta sustava s konačno mnogo stanja je razlog primjene tih sustava u različitim područjima, pa je i to vjerojatno najvažniji razlog njihova proučavanja.

I mi ćemo proučiti još jednu njihovu primjenu, a to je u programima za postupke sintaksne analize jezika ili u tzv. "prepoznavateljima" jezika. Opći model automata dan je na sljedećem crtežu. Automat koji sadrži sve navedene 'dijelove' naziva se pretvarač, automat bez ulazne vrpce je generator, a automat bez izlazne vrpce je prepoznavač.



Sl. 4.1 – *Opći model automata.*

U našim će razmatranjima automati najčešće imati ulogu prepoznavača. Ovisno o jeziku koji se prepoznaje, odnosno o tipu gramatike, postoje i vrste prepoznavača dane na sljedećem crtežu.



Sl. 4.2 - *Chomskyjeva hijerarhija gramatika, njihovi odgovarajući jezici i prepoznavači.*

S obzirom da su regularni i beskontekstni jezici predmet našeg bavljenja, u nastavku ćemo detaljnije opisati konačne i stogovne automate.

4.2 KONAČNI AUTOMAT

Konačni automat nema pomoćne memorije. To je matematički model sustava koji se nalazi u jednom od mnogih konačnih stanja. Stanje sustava sadrži obavijesti koje su dobivene na temelju dotadašnjih podataka i koje su potrebne da bi se odredila reakcija sustava iz idućih podataka. Drugim riječima, radi se o prijelaznim stanjima koje izvodi dani ulazni znak iz danog alfabeta Σ pod utjecajem funkcije prijelaza. Svaki se ulazni znak može nalaziti samo u jednom prijelaznom stanju, pri čemu je dopušten povratak na prethodno stanje.

Definicija 4.1
Konačni automat je uređena petorka

$$\mathcal{M}=(Q,\Sigma,\delta,q_0,F)$$

gdje su:

Q	konačni skup stanja
Σ	alfabet
δ	funkcija prijelaza, definirana kao $\delta: Q\times\Sigma\rightarrow\mathcal{P}(Q)$ gdje je $\mathcal{P}(Q)$ particija od Q
q_0	početno stanje, $q_0\in Q$
F	skup završnih stanja, $F\subseteq Q$

Primjer 4.1

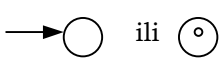
$Q=\{1,2,3,4\}$ $\Sigma=\{a,b,c,d\}$ $q_0=1$ $F=\{4\}$
$\delta(1,a)=\{1\}$ $\delta(1,b)=\{2\}$ $\delta(1,c)=\{3\}$
$\delta(2,c)=\{4\}$ $\delta(3,d)=\{4\}$ $\delta(4,a)=\{4\}$

Funkcija prijelaza može se prikazati tablično. Tada se kaže da je to tablica prijelaza. Redovi tablice predstavljaju stanja, a stupci su označeni znakovima iz alfabeta i predstavljaju prijelaze. Na mjestu u redu označenom s q, $q\in Q$, i stupcu označenom s x, $x\in\Sigma$, upisan je skup narednih stanja (bez vitičastih zagrada) ako je funkcija $\delta(q,x)$ definirana, odnosno nije ništa upisano ako $\delta(q,x)$ nije definirano.

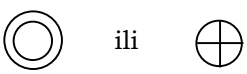
Primjer 4.2
Funkcija prijelaza iz primjera 4.1 može se prikazati kao tablica prijelaza:

δ	a	b	c	d
1	1	2	3	
2			4	
3				4
4	4			

Iz definicije funkcije prijelaza zaključujemo da je to označeni, usmjereni graf čiji čvorovi odgovaraju stanjima automata, a grane su označene znakovima iz alfabeta. Ako, dakle, funkciju prijelaza prikažemo kao graf, dobivamo dijagram prijelaza. Označimo li u tom dijagramu početno stanje i skup završnih stanja, dobivamo konačni automat prikazan grafički. Početno stanje se označuje s:

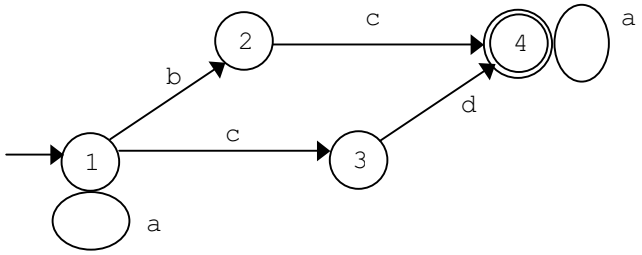


a završno s:



Ako za znak a iz alfabeta postoji prijelaz iz stanja q u stanje p, tada će grana u dijagramu prijelaza koja spaja q i p, s početkom u q i završetkom u p, biti označena s a.

Primjer 4.3
Konačni automat iz primjera 4.1 sada se može prikazati dijagramom prijelaza:



Početno stanje je 1, a završno 4.

Dakle, konačni automat operira čineći niz pomicanja. Ako se radi o prepoznavaču, pomicanje je određeno tekućim stanjem kontrole konačnog stanja i tekućim znakom na koji je postavljen čitač ulazne vrpce. Prelaskom u iduće stanje pomiče se i čitač udesno za jedan znak.

Da bismo znali daljnje "ponašanje" konačnog automata, moramo znati tekuće stanje kontrole konačnog stanja i preostali niz znakova koji se sastoji od tekućeg znaka na koji je pozicioniran čitač i niza znakova na ulaznoj vrpci desno od tekućeg znaka. Te dvije obavijesti predstavljaju konfiguraciju automata, preciznije danu sljedećom definicijom:

Definicija 4.2
Ako je $\mathcal{M}=(Q,\Sigma,\delta,q_0,F)$ konačni automat, par $(q,w)\in Q\times\Sigma^*$ naziva se konfiguracija automata $\mathcal{M}$. (q_0,w) je početna konfiguracija, a (q,ε), $q\in F$, jest završna (prihvatljiva) konfiguracija.

Pomak u $\mathcal{M}$ prikazan je binarnom relacijom $\vdash$ na konfiguracijama. Ako $\delta(q,a)$ sadrži q', tada vrijedi:

$(q,aw) \vdash (q',w)$ za sve $w\in\Sigma^*$

Ako postoje konfiguracije $C_0, C_1, C_2, \ldots, C_k$, tako da je

$C_i \vdash C_{i+1} \quad 0\le i<k,$

tada je

$C_0 \vdash^k C_k$

niz pomaka duljine k. Ako nije bitan broj pomaka, pišemo:

$C \vdash^* C'$

što ima značenje

$C \vdash^k C' \quad k \geq 0$

a ako je postojao najmanje jedan pomak

$C \vdash^+ C' \quad k > 0$

Definicija 4.3
Kaže se da je ulazni niz w prihvatljiv s $\mathcal{M}$ ako

$(q_0, w) \vdash^* (q, \varepsilon)$, za neki $q \in F$

Jezik definiran (generiran) s $\mathcal{M}$, $\mathcal{L}(\mathcal{M})$, jest skup nizova prihvatljivih s $\mathcal{M}$, tj.

$$\mathcal{L}(\mathcal{M}) = \{w: \ w \in \Sigma^*, \ (q_0, w) \vdash^* (q, \varepsilon), \ q \in F\}$$

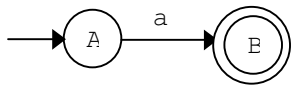
Konačni automati generiraju i prihvaćaju regularne jezike.

Izvođenje regularnih gramatika

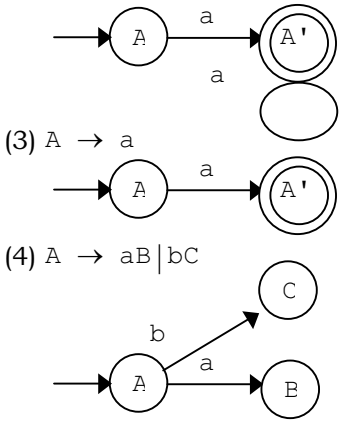
Ako konačni automati generiraju regularne jezike, onda postoji ekvivalentnost konačnih automata i gramatika tipa 3, tj. za konačni automat $\mathcal{M}$ postoji bar jedna gramatika tipa 3 tako da je $\mathcal{L}(\mathcal{M}) = \mathcal{L}(\mathcal{G})$, i obrnuto, za danu gramatiku $\mathcal{G}$ tipa 3 postoji konačni automat $\mathcal{M}$ tako da je $\mathcal{L}(\mathcal{G}) = \mathcal{L}(\mathcal{M})$. U oba ćemo slučaja reći da su $\mathcal{G}$ i $\mathcal{M}$ *ekvivalentni*. Pojednostavljeno, tada će postojati ekvivalentnost između produkcija gramatike i funkcije (tablice ili dijagrama) prijelaza konačnog automata.

Ako su zadane produkcije regularne gramatike, problem izvođenja tablice (dijagrama) prijelaza ekvivalentnog konačnog automata dan je s nekoliko trivijalnih pravila:

(1) $A \rightarrow aB \mid a$



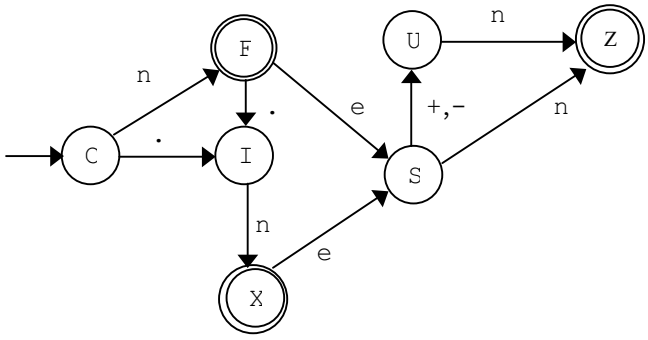
(2) $A \rightarrow aA \mid a$ (rekurzija)



Primjer 4.4
Regularnoj gramatici s produkcijama:

C → n|nF|.I F → .I|eS I → n|nX
X → eS S → n|+U|-U U → n

ekvivalentan je konačni automat definiran dijagramom prijelaza:



Pretpostavimo sada da trebamo izvesti gramatiku regularnog jezika $\mathcal{L}$ nad alfabetom $\mathcal{A}=\{a_1, a_2, \ldots, a_n\}$, znajući svojstva njegovih rečenica. Definirat ćemo konačni automat dijagramom (tablicom) prijelaza u kojem će prijelazi (terminali) biti označeni znakovima alfabeta jezika $\mathcal{L}$, $a_1, a_2, \ldots, a_n$, a stanja (neterminali) velikim slovima engleskog alfabeta, S_0, S_1,..., S_k:

		a_1	a_2	...	a_j	...	a_n
→	S_0	$\delta(S_0, a_1)$	$\delta(S_0, a_2)$				$\delta(S_0, a_2)$
	S_1	$\delta(S_1, a_1)$	$\delta(S_1, a_2)$				
	.						
⊗	S_i	$\delta(S_i, a_1)$	$\delta(S_i, a_2)$		$\delta(S_i, a_j)$		$\delta(S_i, a_n)$
	.						
⊗	S_k	$\delta(S_k, a_1)$	$\delta(S_k, a_2)$		$\delta(S_k, a_j)$		$\delta(S_k, a_n)$

Simbol → označuje početno stanje, a simbol ⊗ završno stanje. Pravila izvođenja regularne gramatike iz ove tablice su sljedeća:

(1) Početni simbol označen je sa S_0.
(2) Alternative produkcije koja započinje početnim simbolom bit će:

$$S_0 \rightarrow a_j \delta(S_0, a_j)$$

za sve parove (S_0, a_j) za koje je definiran prijelaz u naredno stanje.
(3) Napisati ostale produkcije za stanja (neterminali u produkcijama gramatike) S_1 do S_k. Ako je S_i konačno stanje, produkijama za S_i dodati praznu alternativu.

U nastavku dajemo nekoliko primjera izvođenja gramatika jezika prirodnih brojeva i podskupa prirodnih brojeva s određenim svojstvima.

Primjer 4.5 *Jezik prirodnih brojeva*
Tablica prijelaza konačnog automata koji prepoznaje jezik prirodnih brojeva dana je s:

		0	1	2	3	4	5	6	7	8	9
→	N		A	A	A	A	A	A	A	A	A
⊗	A	A	A	A	A	A	A	A	A	A	A

iz čega slijede produkcije ekvivalentne regularne gramatike:

N → 1A| 2A| 3A| 4A| 5A| 6A| 7A| 8A| 9A
A → 0A| 1A| 2A| 3A| 4A| 5A| 6A| 7A| 8A| 9A| ε

što se može transformirati u ekvivalentnu beskontektnu gramatiku:

N → XA
A → 0A | N | ε
X → 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Primjer 4.6 *Jezik prirodnih brojeva djeljivih s 4*
Promatrajući prirodne brojeve djeljive s 4, a to su 4, 8, 12, ..., 96, 100, 104, 108, 112, 116, zaključit ćemo da su 4 i 8 djeljivi s 4, a ako je broj veći od 10, djeljiv je s 4 ako je broj sačinjen od dvije posljednje znamenke djeljiv s 4 ili ako su dvije posljednje znamenke 00. Dalje ćemo zaključiti, promatrajući dvoznamenkaste brojeve djeljive s 4, da ako je pretposljednja znamenka parna ili 0, posljednja mora biti 0, 4 ili 8, a ako je pretposljednja znamenka neparna, posljednja mora biti 2 ili 6. Jezik prirodnih brojeva djeljivih s 4 definiran je konačnim automatom u kojem je tablica prijelaza dana s:

		0	1	2	3	4	5	6	7	8	9
→	A		C	D	C	B	C	D	C	B	C
⊗	B	B	C	D	C	B	C	D	C	B	C
	C	D	C	B	C	D	C	B	C	D	C
	D	B	C	D	C	B	C	D	C	B	C

Sada nije problem izvesti produkcije gramatike jezika prirodnih brojeva djeljivih s 4:

$$A \to 1C \mid 2D \mid 3C \mid 4B \mid 5C \mid 6D \mid 7C \mid 8B \mid 9C$$
$$B \to 0B \mid 1C \mid 2D \mid 3C \mid 4B \mid 5C \mid 6D \mid 7C \mid 8B \mid 9C \mid \varepsilon$$
$$C \to 0D \mid 1C \mid 2B \mid 3C \mid 4D \mid 5C \mid 6B \mid 7C \mid 8D \mid 9C$$
$$D \to 0B \mid 1C \mid 2D \mid 3C \mid 4B \mid 5C \mid 6D \mid 7C \mid 8B \mid 9C$$

Poslije faktorizacija i supstitucija na kraju dobivamo pregledniju ekvivalentnu gramatiku:

$$A \to XC \mid YD \mid ZB$$
$$X \to 1 \mid 3 \mid 5 \mid 7 \mid 9$$
$$Y \to 2 \mid 6$$
$$Z \to 4 \mid 8$$
$$B \to 0B \mid A \mid \varepsilon$$
$$C \to 0D \mid XC \mid YB \mid ZD$$
$$D \to 0B \mid A$$

Primjer 4.7 *Jezik prirodnih brojeva djeljivih s 3 i 6*
Prirodni je broj djeljiv s 3 (v. primjer 3.5) ako mu je zbroj znamenaka djeljiv s 3. I ovaj, na prvi pogled kontekstni aspekt, može biti riješen izvođenjem regularne gramatike, odnosno najprije definiranjem tablice prijelaza konačnog automata. Ako promatramo brojeve djeljive s tri, 3, 6 i 9 su prva tri broja koja su djeljiva s 3. Potom može slijediti 0, 3, 6 ili 9. Ali, što ako broj počinje s 1 ili 2, ili ako počinje s 3, 6 ili 9, a potom slijedi 1, 2 ili možda 8? Pažljivom analizom svih brojeva djeljivih s 3 zaključit ćemo da, na primjer, ako broj sadrži brojku 1, 4 ili 7, mora sadržavati ili još dvaput brojku 1, 4 ili 7 ili brojku 2, 5 ili 8, tako da se ukupnim zbrojem tih znamenki dobije broj koji je djeljiv s 3. Pritom eventualno umetnute 0 ne utječu na ukupni zbroj. Sada nije problem definirati tablicu prijelaza koja sve to obuhvaća:

		0	1	2	3	4	5	6	7	8	9
→	A		C	D	B	C	D	B	C	D	B
⊗	B	B	C	D	B	C	D	B	C	D	B
	C	C	D	B	C	D	B	C	D	B	C
	D	D	E	C	D	E	C	D	E	C	D
⊗	E	E	C	D	E	C	D	E	C	D	E

Na primjer, za ulazni niz w=2220 imamo niz pomaka:

(A, 2220) •• (D, 220) •• (C, 20) •• (B, 0) •• (B, ε)

Sada se iz tablice prijelaza dobiju produkcije ekvivalentne gramatike:

A → 1C|2D|3B|4C|5D|6B|7C|8D|9B
B → 0B|1C|2D|3B|4C|5D|6B|7C|8D|9B|ε
C → 0C|1D|2B|3C|4D|5B|6C|7D|8B|9C
D → 0D|1E|2C|3D|4E|5C|6D|7E|8C|9D
E → 0E|1C|2D|3E|4C|5D|6E|7C|8D|9E|ε

Poslije nekoliko faktorizacija i supstitucija, ova se regularna gramatika može transformirati u ekvivalentnu beskontekstnu gramatiku:

A → XC | YD | ZB
C → 0C | XD | YB | ZC
E → 0E | XC | YD | ZE | ε
Y → 2 | 5 | 8

B → 0B | A | ε
D → 0D | XE | YC | ZD
X → 1 | 4 | 7
Z → 3 | 6 | 9

Ako bismo sada iz ove gramatike pokušali izvesti gramatiku prirodnih brojeva djeljivih sa 6, a znamo da su to parni brojevi djeljivi s tri, prilično bismo se namučili. Vratimo li se konačnom automatu, problem ima jednostavno rješenje: treba umjesto završnih stanja B i E uvesti dva nova završna stanja, npr. F i G, kojima će se osigurati da je broj djeljiv s tri i da je paran (tj. završava s 0, 2, 4, 6 ili 8):

Q={A,B,C,D,E,F,G} Σ={0,1,2,3,4,5,6,7,8,9} q_0=A $\mathcal{F}$={F,G}

Evo tablice prijelaza konačnog automata koji generira prirodne brojeve djeljive sa 6:

		0	1	2	3	4	5	6	7	8	9
→	A		C	D	B	C	D	F	C	D	B
	B	B	C	D	B	C	D	F	C	D	B
	C	C	D	F	C	D	B	C	D	F	C
	D	D	E	C	D	G	C	D	E	C	D
	E	G	C	D	E	C	D	G	C	D	E
⊗	F	F	C	D	B	C	D	F	C	D	B
⊗	G	G	C	D	E	C	D	G	C	D	E

pa sada nije problem napisati produkcije gramatike jezika prirodnih brojeva djeljivih sa 6, a potom ih transformirati u konačan oblik (beskontekstnu gramatiku):

A → XC | YD | ZB | 6F
C → 0C | XD | YB | ZC
E → 0G | XC | YD | ZE | 6G

B → 0B | A
D → 0D | XE | YC | ZD
F → 0F | A | ε

```
G → 0G | XC | YD | ZE | ε
X → 1  | 4 | 7
Y → 2  | 5 | 8
Z → 3  | 9
```

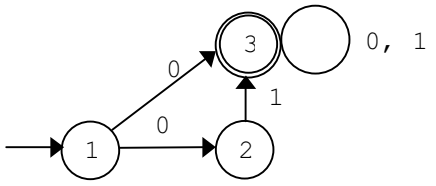
Nedeterministički konačni automat

Iz definicije konačnog automata i funkcije prijelaza slijedi da je moguće $\delta(q, a)=\{q', q''\}$, gdje su q, q' i q'' iz F, $a \in \Sigma$, tj. moguć je pomak iz konfiguracije $(q, a\omega)$ u stanje q' ili q'', pa se kaže da je, općenito konačni automat nedeterministički.

Definicija 4.4
Ako je $M=(Q, \Sigma, \delta, q_0, F)$ konačni automat, kažemo da je M deterministički ako za $\delta(q, a)=q'$ i $\delta(q, a)=q''$ slijedi da je $q'=q''$.

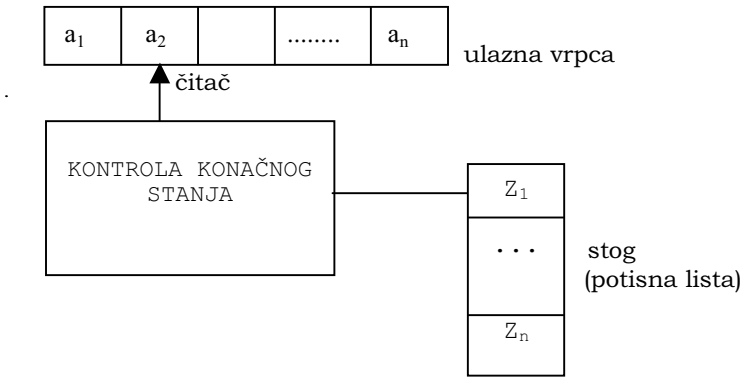
Svi su konačni automati iz dosadašnjih primjera deterministički.

Primjer 4.8
Primjer nedeterminističkog konačnog automata:



4.3 STOGOVNI AUTOMAT

Automat koji ima pomoćnu memoriju sa strukturom stoga (stack) naziva se stogovni automat, odnosno stogovni prepoznavač ako je bez izlazne vrpce, sl. 4.3.



Sl. 4.3 – *Stogovni prepoznavač.*

Definicija 4.5
Stogovni automat je uređena sedmorka $\mathcal{P}=(Q,\Sigma,\Gamma,\delta,q_0,Z_0,F)$, gdje su:

- Q konačan skup stanja (kontrole konačnog stanja)
- Σ ulazni alfabet
- Γ alfabet znakova stoga (potisne liste)
- δ funkcija prijelaza, definirana kao

$$\delta: Q\times(\Sigma\cup\{\varepsilon\})\times\Gamma \rightarrow Q\times\Gamma^*$$

- q_0 početno stanje, $q_0\in Q$
- Z_0 početni znak potisne liste, $Z_0\in\Gamma$
- F skup završnih stanja, $F\subseteq Q$

Definicija 4.6
Konfiguracija stogovnog automata $\mathcal{P}$ jest (q,w,α) iz $Q\times\Sigma^*\times\Gamma^*$, gdje su:

- q tekuće stanje
- w preostali dio ulaznog niza
- α niz znakova koji predstavlja sadržaj potisne liste; vrh je prvi znak niza

Početna konfiguracija je (q_0,w,Z_0), a završna konfiguracija (q,ε,α), $q\in F$, $\alpha\in\Gamma^*$.

Definicija 4.7
Pomak stogovnog automata $\mathcal{P}$ jest relacija $\vdash_P$ (ili samo $\vdash$ ako se $\mathcal{P}$ podrazumijeva):

$$(q,aw,Z\alpha) \vdash (q',w,\gamma\alpha)$$

ako $\delta(q,a,Z)$ sadrži (q',γ) za $q\in Q$, $a\in\Sigma\cup\{\varepsilon\}$, $w\in\Sigma^*$, $Z\in\Gamma$. Kaže se da je ulazni niz ω prihvatljiv s $\mathcal{P}$ ako

$$(q_0,\omega,Z_0) \vdash^*(q,\varepsilon,\alpha)$$

Definicija 4.8
Jezik definiran s $\mathcal{P}$, označen s $L(\mathcal{P})$, jest skup nizova ω prihvatljivih s $\mathcal{P}$. To je općenito beskontekstni jezik:

$$L(\mathcal{P})=\{\omega:\ \omega\in\Sigma^*\ \wedge\ (q_0,\omega,Z_0)\vdash^*(q,\varepsilon,\alpha),\ q\in F,\ \alpha\in\Gamma^*\}$$

Primjer 4.9
Stogovni automat koji prepoznaje beskontekstni jezik $L=\{0^n1^n:\ n\geq1\}$ jest $\mathcal{P}=(\{q_0,q_1,q_2\},\{0,1\},\{\$,0\},\delta,q_0,\$,\{q_0\})$, gdje je funkcija prijelaza δ definirana sa:

$\delta(q_0,0,\$) = \{(q_1,0\$)\}$ $\quad \delta(q_1,0,0) = \{(q_1,00)\}$
$\delta(q_1,1,0) = \{(q_2,\varepsilon)\}$ $\quad \delta(q_2,1,0) = \{(q_2,\varepsilon)\}$
$\delta(q_2,\varepsilon,\$) = \{(q_0,\varepsilon)\}$

Provjerimo je li ulazni niz ω=000111 prihvatljiv:

$$\begin{aligned}(q_0,000111,\$) &\vdash (q_1,00111,0\$)\\ &\vdash (q_1,0111,\ 00\$)\\ &\vdash (q_1,111,\ \ 000\$)\\ &\vdash (q_2,11,\ \ \ 00\$)\\ &\vdash (q_2,1,\ \ \ \ 0\$)\\ &\vdash (q_2,\varepsilon,\ \ \ \ \$)\\ &\vdash (q_0,\varepsilon,\ \ \ \ \varepsilon)\end{aligned}$$

Dakle, niz 000111 je prihvatljiv.

Općenito je stogovni automat nedeterministički. U tom slučaju, da bi automat prihvatio ulazni niz, mora proći kroz sve moguće prijelaze dok ne dosegne završno stanje. Na primjer, konstruirajmo automat koji će prepoznavati beskontekstni jezik $\mathcal{L}=\{ww^R:\ w\in\{a,b\}^+\}$. To će biti stogovni automat $\mathcal{P}=(\{q_0,q_1,q_2\}\{a,b\}\{Z,a,b\},\delta,q_0,Z,\{q_2\})$, gdje je:

(1) $\delta(q_0,a,Z)=\{(q_0,aZ)\}$
(2) $\delta(q_0,b,Z)=\{(q_0,bZ)\}$
(3) $\delta(q_0,a,a)=\{(q_0,aa),(q_1,\varepsilon)\}$
(4) $\delta(q_0,a,b)=\{(q_0,ab)\}$
(5) $\delta(q_0,b,a)=\{(q_0,ba)\}$
(6) $\delta(q_0,b,b)=\{(q_0,bb),(q_1,\varepsilon)\}$
(7) $\delta(q_1,a,a)=\{(q_1,\varepsilon)\}$
(8) $\delta(q_1,b,b)=\{(q_1,\varepsilon)\}$
(9) $\delta(q_1,\varepsilon,Z)=\{(q_2,\varepsilon)\}$

P će inicijalno prenijeti dio svog ulaza u stog, prema pravilima (1), (2), (4) i (5) i prvim alternativama pravila (3) i (6). Nedeterminizam je u pravilima (3) i (6). Na primjer, ako je ulazni niz abba, $\mathcal{P}$ može načiniti sljedeće nizove premještanja:

$$\begin{aligned}(1)\ (q_0,abba,Z) &\vdash (q_0,bba,aZ)\\ &\vdash (q_0,ba,\ baZ)\\ &\vdash (q_0,a,\ \ bbaZ)\\ &\vdash (q_0,\varepsilon,\ \ abbaZ)\end{aligned}$$

$$\begin{aligned}(2)\ (q_0,abba,Z) &\vdash (q_0,bba,aZ)\\ &\vdash (q_0,ba,\ baZ)\\ &\vdash (q_1,a,\ \ aZ)\\ &\vdash (q_1,\varepsilon,\ \ Z)\\ &\vdash (q_2,\varepsilon,\ \ \varepsilon)\end{aligned}$$

Budući da niz premještanja (2) okončava u završnoj konfiguraciji, odnosno u završnom stanju q_2, $\mathcal{P}$ prihvaća ulazni niz abba.

Definicija 4.9
Prošireni stogovni automat je uređena sedmorka $\mathcal{P}=(Q,\Sigma,\Gamma,\delta,q_0,Z_0,F)$, gdje je funkcija prijelaza definirana kao $\delta: Q\times(\Sigma\cup\{\varepsilon\})\times\Gamma^* \rightarrow Q\times\Gamma^*$. Značenje ostalih simbola je nepromijenjeno.

Primjer 4.10
Definirajmo prošireni stogovni automat jezika $\mathcal{L}=\{ww^R: w\in\{a,b\}^*\}$. To će biti $\mathcal{P}=(\{p,q\}\{a,b\}\{a,b,S,Z\},\delta,q,Z,\{p\})$, gdje je funkcija prijelaza:

$$
\begin{aligned}
&(1)\ \delta(q,a,\varepsilon) &&= \{(q,a)\}\\
&(2)\ \delta(q,b,\varepsilon) &&= \{(q,b)\}\\
&(3)\ \delta(q,\varepsilon,\varepsilon) &&= \{(q,S)\}\\
&(4)\ \delta(q,\varepsilon,aSa) &&= \{(q,S)\}\\
&(5)\ \delta(q,\varepsilon,bSb) &&= \{(q,S)\}\\
&(6)\ \delta(q,\varepsilon,SZ) &&= \{(p,\varepsilon)\}
\end{aligned}
$$

Za ulazni niz aabbaa $\mathcal{P}$ će načiniti sljedeći niz premještanja:

$$
\begin{aligned}
(q,aabbaa,Z) &\vdash (q,abbaa,aZ)\\
&\vdash (q,bbaa,\ aaZ)\\
&\vdash (q,baa,\ \ baaZ)\\
&\vdash (q,baa,\ \ SbaaZ)\\
&\vdash (q,aa,\ \ \ bSbaaZ)\\
&\vdash (q,aa,\ \ \ SaaZ)\\
&\vdash (q,a,\ \ \ \ aSaaZ)\\
&\vdash (q,a,\ \ \ \ SaZ)\\
&\vdash (q,\varepsilon,\ \ \ aSaZ)\\
&\vdash (q,\varepsilon,\ \ \ SZ)\\
&\vdash (q,\varepsilon,\ \ \ \varepsilon)
\end{aligned}
$$

Najprije se prednji dio ulaznog niza premješta u stog. Potom se oznaka sredine, S, postavlja na vrh stoga. $\mathcal{P}$ zatim smješta sljedeći simbol ulaznog niza u stog i zamjenjuje aSa ili bSb sa S u listi. $\mathcal{P}$ nastavlja na isti način sve dok se ne iskoristi cijeli ulazni niz. Ako je SZ ostalo u listi, P briše SZ i unosi završno stanje.

Definicija 4.10
Neka je $\mathcal{P}=(Q,\Sigma,\Gamma,\delta,q_0,Z_0,F)$ stogovni ili prošireni stogovni automat. Kaže se da je niz $\omega\in\Sigma^*$ prihvatljiv automatom $\mathcal{P}$ s praznom potisnom listom ako je $(q_0,\omega,Z_0)^+\vdash(q,\varepsilon,\varepsilon)$ za neki $q\in Q$. Skup prihvatljivih nizova označujemo s $\mathcal{L}_e(\mathcal{P})$.

Teorem 4.1
Neka je $\mathcal{G}=(\mathcal{N},\mathcal{T},\mathcal{P},\mathcal{S})$ beskontekstna gramatika. Iz $\mathcal{G}$ se može definirati stogovni automat $\mathcal{R}$ tako da je $\mathcal{L}_e(\mathcal{R}) = \mathcal{L}(\mathcal{G})$.

Dakako, važno je upamtiti što teorem 4.1 tvrdi, dok nas njegov dokaz ne zanima. Dalje, valja primijetiti da za automat s praznom stogovnom listom ne trebamo imati nijedno završno stanje, jer su prazna potisna lista i prazan neprihvaćeni dio ulaznog niza uvjeti okončanja postupka prihvaćanja, bez obzira u kojem se stanju kontrola završnog stanja zatekla.

Primjer 4.11
Definirajmo stogovni automat $\mathcal{R}$ tako da je $\mathcal{L}_e(\mathcal{R})=\mathcal{L}(\mathcal{G})$, gdje je $\mathcal{G}$ gramatika aritmetičkih izraza:

$$\begin{array}{lll} E \rightarrow T & | & T+E \\ T \rightarrow F & | & T*F \\ F \rightarrow (E) & | & a \end{array}$$

Stogovni automat će biti $\mathcal{R}=(\{q\},\Sigma,\Gamma,\delta,q,Z,\varnothing)$, gdje je skup završnih stanja prazan, a funkcija je prijelaza δ definirana sa:

$$\begin{array}{ll} (1)\ \delta(q,\varepsilon,E) & = \{(q,E+T),(q,T)\} \\ (2)\ \delta(q,\varepsilon,T) & = \{(q,T*F),(q,F)\} \\ (3)\ \delta(q,\varepsilon,F) & = \{(q,(E)),(q,a)\} \\ (4)\ \delta(q,a,a) & = \{(q,\varepsilon)\} \\ (5)\ \delta(q,+,+) & = \{(q,\varepsilon)\} \\ (6)\ \delta(q,*,*) & = \{(q,\varepsilon)\} \\ (7)\ \delta(q,(,() & = \{(q,\varepsilon)\} \\ (8)\ \delta(q,),)) & = \{(q,\varepsilon)\} \end{array}$$

Na primjer, za ulazni niz a+a*a $\mathcal{R}$ će, između ostalog, načiniti sljedeća premještanja:

$$\begin{array}{ll} (q,a+a*a,E) & \vdash (q,\ a+a*a,\ E+T) \\ & \vdash (q,\ a+a*a,\ T+T) \\ & \vdash (q,\ a+a*a,\ F+T) \\ & \vdash (q,\ a+a*a,\ a+T) \\ & \vdash (q,\ +a*a,\ \ +T) \\ & \vdash (q,\ a*a,\ \ \ T) \\ & \vdash (q,\ a*a,\ \ T*F) \\ & \vdash (q,\ a*a,\ \ F*F) \\ & \vdash (q,\ a*a,\ \ a*F) \\ & \vdash (q,\ *a,\ \ \ *F) \\ & \vdash (q,\ a,\ \ \ \ F) \\ & \vdash (q,\ a,\ \ \ \ a) \\ & \vdash (q,\ \varepsilon,\ \ \ \ \varepsilon) \end{array}$$

U prethodnom je primjeru niz premještanja koje je načinio stogovni automat prihvaćajući ulazni niz ekvivalentan krajnjem izvođenju slijeva rečenice a+a*a počevši s početnim simbolom E. Takva vrsta analize naziva se "silazna (top-down) sintaksna analiza" ili "predikatna analiza". Detaljnije o sintaksnoj analizi napisano je u sljedeća tri poglavlja.

Pitanja i zadaci

1) Definirajte konačni automat ekvivalentan sljedećim regularnim izrazima:

   a) (0.)+((1+2+3+4+5+6+7+8+9)(0+1+2+3+4+5+6+7+8+9)*.)(0+1+2+3+4+5+6+7+8+9)$^+$
   b) (11+a)*(ε+0+00)
   c) (1+2+3+4+5+6+7+8+9)(0+1+2+3+4+5+6+7+8+9)*
   d) (a+b)*aa(a+b)*
   e) {0,1}* bez podniza 101

2) Dana je regularna $\mathcal{G}$ gramatika s produkcijama:

   S → 0A | 1S | ε
   A → 0B | 1A
   B → 0S | 1B

   Definirajte:

   a) Regularni izraz koji označuje skup nizova ekvivalentan jeziku $\mathcal{L}(\mathcal{G})$.
   b) Konačni automat $\mathcal{M}$ koji će prepoznavati taj jezik

3) Imena konstanti, varijabli, funkcija i procedura u Turbo Pascalu pišu se tako da moraju početi slovom (velikim ili malim) engleskog alfabeta ili podcrtom (znak _), potom, ako je duljina imena veća od 1, može slijediti slovo, brojka ili podcrta. Definirajte prvo gramatiku imena Turbo Pascala i iz nje izvedite konačni automat koji će ih prepoznavati.

4) Koristeći se primjerom 4.6 i 4.7 napišite regularne izraze koji će označivati nizove znakova koji predstavljaju prirodne brojeve djeljive s 3, 4 i 6.

5) Definirajte generator i regularnu gramatiku jezika:

   a) Parnih prirodnih brojeva
   b) Neparnih prirodnih brojeva
   c) Prirodnih brojeva djeljivih s 5, 7, 8, 9 i 13

6) U primjeru 4.5 izvedena je beskontekstna gramatika koja generira prirodne brojeve djeljive s 3. Iz primjera 4.5 slijedi da je jezik prirodnih brojeva djeljivih s 3 regularan. Izvedite regularnu gramatiku jezika prirodnih brojeva djeljivih s 3 iz tablice prijelaza primjera 4.5.

7) Definirajte stogovni automat gramatike:

   $E \rightarrow E+E \mid E*E \mid (E) \mid a$

5.

OPĆENITI POSTUPCI SINTAKSNE ANALIZE

sa svojom njuškom mješanca
židovskog lutalice i grčkog pastira
doći ću moja nježna zarobljenice
sestro moje duše
izvoru mog života
doći ću da upijem
tvojih dvadeset godina
bit ću princ krvi
sanjalica ili maloljetnik
već kako ti izabereš
i stvorit ćemo od svakog dana
cijelu jednu vječnost ljubavi
da bismo živjeli od nje umirući

sa svojom njuškom mješanca
židovskog lutalice i grčkog pastira
i sa svojim kosama
sa svih strana svijeta
s očima potpuno orošenim
koje mi daju izgled sanjalice
meni koji više ne sanjam često
s rukama kradljivca muzičara i skitnice
koje su opljačkale tolike vrtove
s ustima koja su pila
ljubila i grizla
a da nikad nisu zasitila svoju glad

sa svojom njuškom mješanca
židovskog lutalice i grčkog pastira
razbojnika i vagabunda
s grudima koje su grlile
pod suncem ljeta
sve ono što je suknju nosilo
sa srcem koje je znalo patiti
onoliko koliko je patilo
a da od toga ne pravi drame
s dušom koja nema
ni najmanje šanse da se spasi
i izbjegne čistilište

mješanac
métèque
(georges moustaki/
nepoznati student)

U prethodnom smo poglavlju pokazali kako se automati mogu primijeniti u specificiranju jezika i prihvaćanju ulaznih nizova. To je bila sintaksna analiza rečenica, premda je nismo posebno spomenuli.

Vidjeli smo kako gramatika $G=(N, T, P, S)$ *generira jezik* $L(G)$. *To je skup rečenica* w *dobivenih svim mogućim izvođenjima iz* S, *tj.*

$$L(G)=\{w\in T^* : S^*\Rightarrow w\}$$

U praksi se često susrećemo s problemom da je poznata gramatika nekog jezika i zadan niz znakova, a postavlja se pitanje je li to rečenica jezika generiranog danom gramatikom. U tom slučaju problem se svodi na nalaženje niza izvođenja, počevši od S, *koji bi rezultirao tim nizom (rečenicom). Takav postupak naziva se sintaksna analiza (ili sintaktička analiza). Gramatika u tom slučaju ima ulogu prepoznavatelja jezika. Ustrojbu postupka sintaksne analize na računalu (program u izabranom jeziku za programiranje) nazivat ćemo parser.*

Pri učenju nekog jezika gramatika je najčešće u ulozi prepoznavatelja. To je njezin pravi smisao. Bilo bi besmisleno znajući gramatiku nekog jezika prijeći odmah na izvođenje rečenica. Takav proces bi možda trajao nekoliko godina! I u teoriji formalnih jezika gramatika je češće u ulozi prepoznavatelja nego generatora jezika.

Algoritmi općenite sintaksne analize primjenjivi su nad širokom klasom beskontekstnih jezika. Postoji nekoliko takvih algoritama. Ovdje su opisana dva koja pripadaju klasi povratnih ("backtrack") algoritama: silazna (top-down) sintaksna analiza i uzlazna (bottom-up) sintaksna analiza, te dva općenita postupka koja pripadaju klasi tabličnih metoda, Cocke-Younger-Kasamijeva i Earlyjeva metoda.

5.1 SILAZNA SINTAKSNA ANALIZA

Naziv "silazna sintaksna analiza" ili "sintaksna analiza s vrha" dolazi od ideje da se pokuša izvesti stablo sintaksne analize ulaznog niza započeto od korijena (vrha) gradeći ga prema dolje, prema listovima.

Neformalno je postupak silazne sintaksne analize jezika generiranog nekom gramatikom G sljedeći:

1) Najprije se alternativama produkcija gramatike G pridruže, proizvoljno, indeksi. Na primjer, ako su α_1, α_2, ..., α_n sve alternative produkcije za A, onda je α_1 prva, α_2 druga alternativa, itd.

2) Neka je $w=a_1a_2...a_n$ ulazni niz za koji treba utvrditi je li u jeziku generiranom sa G. Bit će upotrijebljena kazaljka koja će pokazivati na tekući znak ulaznog niza (na početku je to znak a_1).

3) Početni simbol S korijen je stabla sintaksne analize i istovremeno aktivni čvor.

Sljedeća dva koraka izvode se rekurzivno:

4) Ako je aktivni čvor neterminalni simbol X, bira se njegova prva alternativa i generira k njegovih izravnih slijednika. Neka su označeni s X_1, ..., X_k. X_1 je novi aktivni čvor. Ako je k=0, aktivni čvor postaje simbol koji slijedi X.

5) Ako je aktivni čvor terminalni simbol, treba ga usporediti s tekućim simbolom ulaznog niza (simbolom na koji pokazuje kazaljka). Ako su jednaki, kazaljka se pomiče za jedan simbol udesno. Aktivni čvor postaje simbol desno od terminalnog simbola.

Ako terminalni simbol nije jednak tekućem simbolu ulaznog niza, vraća se na čvor gdje je bila upotrijebljena prethodna produkcija i izvodi sljedeća alternativa. Ako nijedna alternativa nije moguća, vraća se na prethodni čvor itd. Ako su upotrijebljene sve alternative i nije se uspjelo s izjednačivanjem ulaznog niza w, ulazni niz nije u jeziku generiranom gramatikom G. Na primjer, neka je dana gramatika G s produkcijama:

$S \rightarrow aSbS \mid aS \mid c$

Treba provjeriti je li ulazni niz w=aacbc u jeziku kojeg G generira, tj. postoji li, počevši sa S, niz izvođenja

$S \overset{*}{\Rightarrow} aacbc$

Najprije se može usvojiti da je

aSbS	prva,
aS	druga i
c	treća alternativa za S.

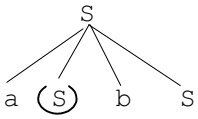
Postupak silazne sintaksne analize bio bi:

a)


```
aacbc
↑
```

S je korijen i inicijalni aktivni čvor. Kazaljka pokazuje na simbol a.

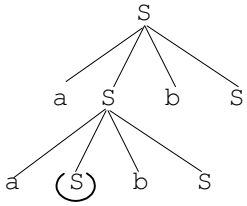
b)




```
aacbc
 ↑
```

Generirana su četiri izravna slijednika od S (prva alternativa). Terminal a je aktivni čvor i jednak je tekućem simbolu. Kazaljka se pomiče za jedno mjesto.

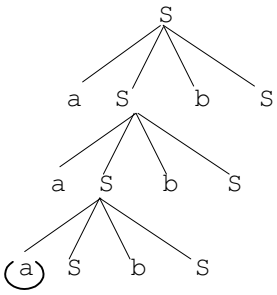
c)




```
aacbc
 ↑
```

Aktivni čvor je neterminal S, pa se ponovo generiraju četiri izravna slijednika (prva alternativa). Aktivni čvor je terminal a, jednak je simbolu ulaznog niza, pa se kazaljka pomiče za jedno mjesto.

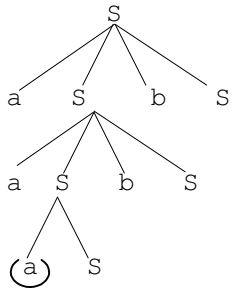
d)




```
aacbc
 ↑
```

Neterminal S ponovno je aktivni čvor. Generiraju se slijednici od S prema prvoj alternativi.

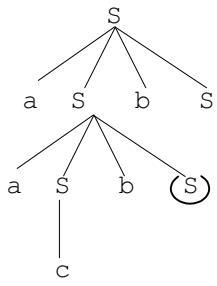
e)




```
aacbc
 ↑
```

Terminal a je aktivni čvor i nije jednak tekućem simbolu. Pokušava se s drugom alternativom.

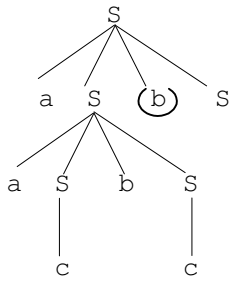
f)




```
aacbc
  ↑
```

Terminal a je aktivni čvor i nije jednak tekućem simbolu. Pokušava se s trećom alternativom. Prvo je terminal c aktivni čvor i jednak je tekućem simbolu. Kazaljka se pomiče za jedno mjesto. Sada je terminal b aktivni čvor i jednak je tekućem simbolu. Kazaljka se pomiče za jedno mjesto.

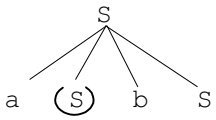
g)



aacbc
↑

Neterminal S je aktivni čvor. Generiraju se slijednici prema prvoj alternativi. Poslije toga je terminal a aktivni čvor i nije jednak tekućem znaku. Pokušava se s drugom alternativom. Opet je terminal a aktivni čvor i nije jednak tekućem znaku. Pokušava se s trećom alternativom. Tada je terminal c aktivni čvor i jednak je tekućem znaku. Kazaljka se pomiče za jedno mjesto.

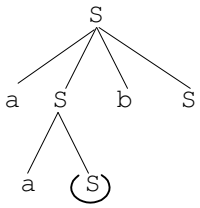
h)



aacbc
↑

Kazaljka je iza posljednjeg znaka ulaznog niza, a u stablu su ostala još dva znaka, b i S. Vraćamo se do mjesta gdje je posljednji put ekspandirano stablo. Ne postoji više alternativa, pa se vraćamo na prethodno mjesto ekspanzije stabla. Ni tu nije moguća nova ekspanzija, pa se dalje vraćamo, do koraka (b).

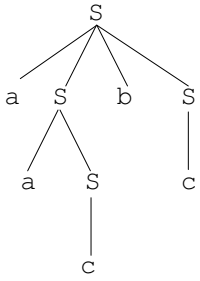
i)



aacbc
↑

Pokušava se s drugom alternativom. Aktivni čvor je terminal a i jednak je tekućem znaku. Kazaljka se pomiče za jedno mjesto.

k)



aacbc
↑

Aktivni čvor je S. Generira se podstablo iz prve alternative. Novi aktivni čvor je terminal a i nije jednak tekućem znaku. Pokušava se s drugom alternativom. Opet je aktivni čvor terminal a. Pokušava se s trećom alternativom. Aktivni čvor je terminal c i jednak je tekućem znaku. Kazaljka se pomiče za jedno mjesto i pokazuje iza posljednjeg znaka ulaznog niza. Više nema aktivnih čvorova. Ulazni niz je u jeziku.

Primjenljivost silazne sintaksne analize

Iz prethodnog primjera mogu se uočiti osnovni nedostaci postupka silazne (povratne) sintaksne analize.

Prvo, postoji veoma opasna klopka u toj proceduri. Ako je gramatika rekurzivna slijeva proces se ne bi nikada završio! Naime, u tom slučaju bi se smjenom nekog neterminala (aktivnog čvora) dobio isti aktivni čvor, pa njegovom smjenom opet isti itd.

Drugi je nedostatak povratne silazne sintaksne analize što uvodi određene semantičke aspekte. Na primjer, treba znati koji će dio tablice simbola biti izbrisan u slučaju neuspješnog izjednačivanja; koja je alternativa na redu da se ekspandira stablo sintaksne analize i na koji simbol ulaznog niza treba vratiti kazaljku.

Treći je nedostatak indeksiranje (uređenje) alternativa produkcija. Stavljanje neke alternative na prvo mjesto u nekoj produkciji može ili povećati ili smanjiti ukupno vrijeme potrebno za izvršavanje sintaksne analize. To znači da bi trebalo znati vjerojatnosti pojavljivanja svake od alternativa u nizovima izvođenja rečenica jezika i potom ih urediti stavljajući na prva mjesta one alternative koje imaju veću vjerojatnost pojavljivanja.

Intuitivno se može zaključiti da bi se sintaksna analiza s vrha mogla ubrzati ako sve alternative počinju terminalnim simbolima. Ako bi, pored toga, terminalni simboli bili različiti, moglo bi se jednoznačno, na osnovu tekućeg simbola ulaznog niza, odrediti koja će alternativa biti upotrijebljena. Ovakva razmišljanja, kao što će se vidjeti u sljedećem poglavlju, vode k pojmu $LL(k)$ gramatika i jednoprolaznih sintaksnih analiza.

Da zaključimo: silazna sintaksna analiza može se primijeniti samo za gramatike koje nemaju rekurzija slijeva. Međutim, znamo da prema teoremu 3.1 svaka beskontekstna gramatika koja je rekurzivna slijeva ima ekvivalentnu gramatiku koja nije rekurzivna slijeva, pa je taj postupak primjenljiv za sve beskontekstne jezike.

Gramatika iz uvodnog primjera silazne sintaksne analize nije rekurzivna slijeva (no jest zdesna), pa je moguće u konačno mnogo prolazaka iscrpsti sva izvođenja.

Algoritam silazne sintaksne analize

Poslije neformalnog opisa postupka silazne sintaksne analize i analize primjenljivosti takvog postupka, evo i njegova algoritma.

Algoritam koristi dva stoga (dvije "potisne" liste), L_1 i L_2, i kazaljku koja pokazuje na tekući simbol ulaznog niza. U opisu algoritma bit će korištena notacija slična onoj koja je bila upotrijebljena u opisu konfiguracije potisnog prepoznavača.

Algoritam 5.1
Silazna sintaksna analiza.

Ulaz
Gramatika $G=(N,T,P,S)$ koja nije rekurzivna slijeva i ulazni niz $w=a_1a_2...a_n$, n≥1. Pretpostavka je da su produkcije u P numerirane od 1 do p.

Izlaz
Stablo sintaksne analize za w, ako je w u jeziku $L(G)$; inače "pogreška".

Postupak
(1) Urediti alternative za svaki neterminal A iz N. Na primjer, ako su $A\rightarrow\alpha_1|\alpha_2|...\ |\alpha_k$ sve A-produkcije u P i alternative su uređene kao što je pokazano, tada je A_1 indeks za α_1, A_2 indeks za α_2, itd.

(2) Četvorka (s,i,L_1,L_2) označivat će konfiguraciju algoritma, gdje su

- s stanje algoritma: q - napredovanje, b – povrat i t – kraj
- i položaj ulazne kazaljke. Ako je i=n+1 znači da je dosegnut kraj ulaznog niza (bit će označen s $)
- L_1 potisna lista (stog) koja predstavlja povijest izbora alternativa i simbole ulaznog niza koji su bili jednaki aktivnim čvorovima stabla sintaksne analize. Vrh liste je zdesna
- L_2 potisna lista koja predstavlja tekuću lijevu rečeničnu formu. Simbol na vrhu (vrh je slijeva) označuje aktivni čvor stabla koje će biti generirano.

(3) Početna konfiguracija algoritma je $(q,1,\varepsilon,S\$)$.

(4) Postoji šest vrsta koraka. Bit će opisani konfiguracijom algoritma. Notacija $(s,i,\alpha,\beta)\vdash(s',i',\alpha',\beta')$, jer se ovdje radi o vrsti automata, ima značenje prelaska iz konfiguracije (s,i,α,β) u konfiguraciju (s',i',α',β'). Moguća su sljedeća premještanja:

(a) *Ekspanzija stabla*

$$(q,i,\alpha,A\beta)\vdash(q,i,\alpha A_1,\alpha_1\beta)$$

gdje je $A\rightarrow\alpha_1$ produkcija u P; α_1 je prva alternativa za A. Ovaj korak odgovara ekspanziji parcijalno izvedenog stabla koristeći prvu alternativu za prvi neterminal s lijeva u stablu.

(b) *Uspješno izjednačenje ulaznog i izvedenog simb*ola

$$(q,i,\alpha,a\beta)\vdash(q,i+1,\alpha a,\beta)$$

pod uvjetom da je $a_i=a$, $i \leq n$. Terminalni se simbol premješta s vrha liste L_2 na vrh liste L_1. Kazaljka se pomiče za jedno mjesto.

(c) *Neuspješno izjednačenje ulaznog i izvedenog simbola*

$(q,i,\alpha,a\beta) \vdash (b,i,\alpha,a\beta)$

ako je $a_i \neq a$. Lijeva rečenična forma koja je bila upravo izvedena nije konzistentna s ulazom. Prelazi se u "povratni" mod.

(d) *Uspješno okončanje*

$(q,n+1,\alpha,\$) \vdash (t,n+1,\alpha,\varepsilon)$

Dosegnut je kraj ulaznog niza i nađena lijeva rečenična forma koja odgovara ulazu.

(e) *Povrat kazaljke*

$(b,i,\alpha a,\beta) \vdash (b,i-1,\alpha,a\beta)$

za sve a iz $\mathcal{T}$. Svaki se put ulazni simbol premješta iz L_1 u L_2.

(f) *Pokušaj sljedeće alternative*

$(b,i,\alpha A_j,\alpha_j\beta) \vdash$

(1) $(q,i,\alpha A_{j+1},\alpha_{j+1}\beta)$, ako je α_{j+1} $(j+1)$-alternativa za A.

(2) Ne postoji sljedeća konfiguracija, ako je $i=1$, $A=S$ i postoji samo j alternativa za S. Taj uvjet pokazuje da su iscrpljene sve moguće lijeve rečenične forme konzistentne s ulazom w, a da se za njega nije uspjelo izvesti stablo sintaksne analize. Ulazni niz w nije u jeziku pa se prekida daljnje izvršavanje postupka.

(3) $(b,i,\alpha,A\beta)$, inače. Sve su alternative za A iscrpljene. Vraća se i izbacuju se sve alternative A_j iz L_1 i zamjenjuje α_j s A u L_2.

Izvršavanje algoritma je kao što slijedi:

(1) Krenuvši s početnom konfiguracijom, izračunaju se slijedeće uzastopne konfiguracije:

$C_0 \vdash C_1 \vdash \ldots \vdash C_i \vdash \ldots$

sve dok je definirana naredna konfiguracija.

(2) Ako je posljednje izračunata konfiguracija

$(t, n+1, \alpha, \varepsilon)$

prekida se daljnje izvršavanje. Ulazni je niz u jeziku generiranom gramatikom $\mathcal{G}$. Inače, dojavljuje se greška.

Algoritam silazne sintaksne analize - parser realiziran je u Pascalu i dan u prilogu. Program može raditi korak po korak, prikazujući svaku promjenu konfiguracije i pomicanje kazaljke, pa istovremeno predstavlja lekciju za učenje postupka silazne sintaksne analize.

Primjer 5.1
Primijenimo algoritam u sintaksnoj analizi jezika jednostavnih aritmetičkih izraza generiranog gramatikom:

```
E → T+E (1)     E → T  (2)
T → F*T (3)     T → F  (4)
F → a   (5)
```

gdje brojevi u zagradi označuju uređenje produkcija. Neka je `E1`, jednako `T+E`, prva alternativa za `E`, a `E2`, jednako `T`, druga alternativa za `E`. `T1` jednako `F*T` i `T2` jednako `F` su prva i druga alternativa za `T`, a `F1` jednako `a` je prva alternativa za `F`. Ako je ulazni niz `w` jednak `a+a`, algoritam će izračunati slijedeće konfiguracije:

```
(q, 1, ε, E$)
   ├ (q, 1, E1,                T+E$)
   ├ (q, 1, E1T1,             F*T+E$)
   ├ (q, 1, E1T1F1,           a*T+E$)
   ├ (q, 2, E1T1F1a,           *T+E$)
   ├ (b, 2, E1T1F1a,           *T+E$)
   ├ (b, 1, E1T1F1,           a*T+E$)
   ├ (b, 1, E1T1,             F*T+E$)
   ├ (q, 1, E1T2,               F+E$)
   ├ (q, 1, E1T2F1,             a+E$)
   ├ (q, 2, E1T2F1a,             +E$)
   ├ (q, 3, E1T2F1a+,            E$)
   ├ (q, 3, E1T2F1a+E1,        T+E$)
   ├ (q, 3, E1T2F1a+E1T1,     F*T+E$)
   ├ (q, 3, E1T2F1a+E1T1F1,   a*T+E$)
   ├ (q, 4, E1T2F1a+E1T1F1a,   *T+E$)
   ├ (b, 4, E1T2F1a+E1T1F1a,   *T+E$)
   ├ (b, 3, E1T2F1a+E1T1F1,   a*T+E$)
   ├ (b, 3, E1T2F1a+E1T1,     F*T+E$)
   ├ (q, 3, E1T2F1a+E1T2,       F+E$)
   ├ (q, 3, E1T2F1a+E1T2F1,     a+E$)
```

```
├ (q, 4, E1T2F1a+E1T2F1a,  +E$)
├ (b, 4, E1T2F2a+E1T2F1a,  +E$)
├ (b, 3, E1T2F1a+E1T2F1,  a+E$)
├ (b, 3, E1T2F1a+E1T2,    F+E$)
├ (b, 3, E1T2F1a+E1,      T+E$)
├ (q, 3, E1T2F1a+E2,        T$)
├ (q, 3, E1T2F1a+E2T1,    F*T$)
├ (q, 3, E1T2F1a+E2T1F1,  a*T$)
├ (q, 4, E1T2F1a+E2T1F1a,  *T$)
├ (b, 4, E1T2F1a+E2T1F1a,  *T$)
├ (b, 3, E1T2F1a+E2T1F1,  a*T$)
├ (b, 3, E1T2F1a+E2T1,    F*T$)
├ (q, 3, E1T2F1a+E2T2,      F$)
├ (q, 3, E1T2F1a+E2T2F1,    a$)
├ (q, 4, E1T2F1a+E2T2F1a,    $)
├ (t, 4, E1T2F1a+E2T2F1a,    ε)
```

Lista L_1 sadrži stablo sintaksne analize (lijeve rečenične forme):

$$E \Rightarrow T+E \Rightarrow F+E \Rightarrow a+E \Rightarrow a+T \Rightarrow a+F \Rightarrow a+a$$

5.2 UZLAZNA SINTAKSNA ANALIZA

Postoji općenita metoda sintaksne analize koja je po smislu suprotna od silazne sintaksne analize. To je uzlazna (bottom-up) sintaksna analiza.

U silaznoj se sintaksnoj analizi stablo sintaksne analize gradi od korijena prema dolje, do listova. Nasuprot tome, u uzlaznoj sintaksnoj analizi, počinje se od listova i pokušava izgraditi stablo prema gore, do korijena. Ovdje će biti izložen algoritam uzlazne sintaksne analize koji se naziva sintaksna analiza "s reduciranjem premještanja". Analiza se odvija koristeći najvažnije svojstvo sintaksne analize zdesna, prolazeći kroz sva moguća krajnja izvođenja zdesna koja su konzistentna s ulazom.

Premještanje se sastoji od ispitivanja niza na vrhu potisne liste i utvrđivanja postoji li desna strana produkcije koja može biti izjednačena s vrhom liste. Ako postoji, vrh potisne liste treba zamijeniti njome. Ako postoji više desnih strana produkcija koje se mogu upotrebiti za reduciranje vrha postisne liste, uz pretpostavku prethodnog proizvoljnog uređenja produkcija, bit će upotrebljena prva. Ako nije moguća nijedna redukcija, sljedeći se ulazni simbol premješta u listu i ponovi postupak. Uvijek će se pokušati s reduciranjem prije premještanja. Ako se dosegne kraj ulaznog niza i nije moguća nijedna redukcija, vraća se na posljednje premještanje gdje je bila učinjena redukcija i pokuša se s nekom drugom redukcijom.

Ulazni niz je u jeziku generiranom zadanom gramatikom ako lista sadrži početnii simbol i dosegnut je kraj ulaznog niza. Ako se pokušalo sa svim redukcijama i nije se postiglo takvo stanje, postupak se prekida uz

zaključak da ulazni niz nije rečenica jezika. Razmotrimo to na sljedećem primjeru. Neka je dana gramatika s produkcijama:

S→AB A→ab B→aba

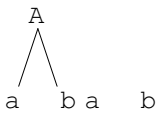
i neka je ulazni niz w=ababa. Pogledajmo kako će se odvijati uzlazna sintaksna analiza:

a)

ababa
↑

Najprije se prvi znak ulaznog niza, a, premješta u listu. Nije moguća nijedna redukcija, pa se i b prebacuje u listu.

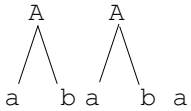
b)



ababa
↑

ab se može reducirati u A. Premješta se sljedeći znak u listu. Sada je u listi Aa i nije moguća redukcija. Premješta se sljedeći znak. Sadržaj je liste Aab.

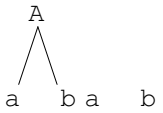
c)



ababa
↑

ab se može reducitari u A. Posljednji se znak premješta u listu. Sadržaj liste je AAa.

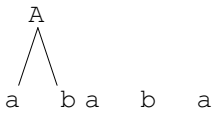
d)



ababa
↑

Nije moguća nijedna redukcija. Vraća se na prethodni korak kad je u listi bilo Aab (b je na vrhu).

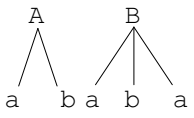
e)



ababa
↑

Nije moguća nijedna redukcija. Premješta se posljednji znak u listu. Sadržaj je liste: Aaba.

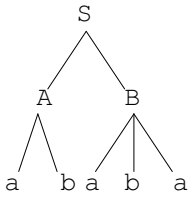
f)



ababa
↑

Sada se aba može reducirati u B. Sadržaj je liste: AB.

g)



ababa
↑

Konačno se AB može zamijeniti sa S. Time je stablo sintaksne analize izgrađeno. Niz je u jeziku koji dana gramatika generira i može se dobiti izvođenjem:

$$S \Rightarrow AB \Rightarrow Aaba \Rightarrow ababa$$

Primjenljivost uzlazne sintaksne analize

Izložena metoda može se razmatrati kao niz svih mogućih premještanja u nedeterminističkoj sintaksnoj analizi dane gramatike zdesna. Međutim, kao što je bilo u slučaju analize s vrha, i ovdje postoje izvjesna ograničenja u primjeni.

Prvo, postoje situacije u kojima bi broj mogućih premještanja bio beskonačan. Takva klopka može se pojaviti ako gramatika ima petlje, tj. postoji izvođenje $A^+ \Rightarrow A$ za neki neterminal A. Tada bi broj podstabala bio beskonačan.

Drugu poteškoću stvaraju ε-produkcije u gramatici. Tada se može načiniti proizvoljan broj redukcija u kojima bi prazan niz bio "reduciran" u neki neterminal. Postupak sintaksne analize može se proširiti da prihvaća i gramatike sa ε-produkcijama, ali je to izostavljeno u ovoj knjizi. Osim toga, u trećem je poglavlju dan algoritam za izbacivanje ε-produkcija, iz čega slijedi da je ovaj postupak primjenljiv za sve beskontekstne jezike.

Algoritam uzlazne sintaksne analize

Poslije neformalnog opisa postupka uzlazne sintaksne analize evo i algoritma koji dajemo u notaciji sličnoj onoj koja je upotrijebljena u algoritmu silazne sintaksne analize.

Algoritam 5.2

Uzlazna sintaksna analiza.

Ulaz

Gramatika $G=(N, T, P, S)$ bez petlji i ε-produkcija (svojstvena gramatika) i ulazni niz $w=a_1a_2...a_n$, $n \geq 1$. Uređene produkcije od 1 do p.

Izlaz

Stablo sintaksne analize za w, ako je w u $L(G)$. Inače, "pogreška".

Postupak

(1) Urediti alternative neterminala iz N počevši od startnog simbola i njegovih alternativa koje će imati indeks 1, 2, itd. sve do posljednjeg neterminala i njegovih alternativa koje će imati indekse ... $p-1$, p.

(2) Četvorka (s, i, L_1, L_2) označivat će konfiguraciju algoritma, gdje su:

- s stanje algoritma: q - napredovanje, b – povrat i t – kraj
- i lokacija ulazne kazaljke. Ako je $i=n+1$ znači da je dosegnut kraj ulaznog niza (bit će označen s $)
- L_1 potisna lista (vrh zdesna) koja će sadržavati niz terminala i neterminala koji izvodi dio ulaznog niza lijevo od pozicije kazaljke.
- L_2 potisna lista (vrh slijeva) koja će sadržavati povijest premještanja i reduciranja neophodnih za dobivanje sadržaja liste L_1.

(3) Početna je konfiguracija algoritma $(q,1,\$,\varepsilon)$.

(4) Izvršavanje počinje prvim korakom:

1. korak: *premještanje*

$$(q,i,\alpha,\gamma) \vdash (q,i+1,\alpha a_i,s\gamma)$$

pod uvjetom da je $i \neq n+1$. Potom se ide na drugi korak. Ako je $i=n+1$, prelazi se na treći korak. Ako je prvi korak bio moguć, upisuje se i-ti simbol na vrh liste L_1, pomiče se ulazna kazaljka i upisuje s na vrh liste L_2, što je znak da je bilo učinjeno premještanje.

2. korak: *pokušaj reduciranja*

$$(q,i,\alpha\beta,\gamma) \vdash (q,i,\alpha A,j\gamma)$$

pod uvjetom da je $A \to \beta$ j-ta produkcija u $\mathcal{P}$. β je prva desna strana u linearnom uređenju (1), tj. sufiks niza $\alpha\beta$. Broj produkcije upisan je u listu L_2. Ako je ovaj korak bio moguć, ponovo se vraća na njega.

3. korak: *prihvaćanje*

$$(q,n+1,\$\mathcal{S},\gamma) \vdash (t,n+1,\$\mathcal{S},\gamma)$$

Ako ovaj korak nije prihvatljiv, ide se na četvrti korak.

4. korak: *početak povratka*

$$(q,n+1,\alpha,\gamma) \vdash (b,n+1,\alpha,\gamma)$$

pod uvjetom da je $\alpha \neq \$\mathcal{S}$. Ide se na peti korak.

5. korak: *povratak*

a) $(b,i,\alpha A,j\gamma) \vdash (q,i,\alpha' B,k\gamma)$
ako je $A \to \beta$ j-ta produkcija u $\mathcal{P}$, a $B \to \beta'$ je sljedeća produkcija, označena s k prema uređenju (1), u kojoj je desna strana sufiks od $\alpha\beta$ (vrijedi $\alpha\beta = \alpha'\beta'$). Ide se na drugi korak.

b) $(b,n+1,\alpha A,j\gamma) \vdash (b,n+1,\alpha\beta,\gamma)$
ako je $A \to \beta$ j-ta produkcija u $\mathcal{P}$ i nije preostala nijedna alternativa za reduciranje αß. Ide se ponovo na peti korak.

c) $(b,i,\alpha A,j\gamma) \vdash (q,i+1,\alpha\beta a,s\gamma)$
ako je $i \neq n+1$, j-ta produkcija u $\mathcal{P}$ je $A \to \beta$, i nije preostala nijedna alternativa za reduciranje $\alpha\beta$. Ovdje je $a=a_i$ premješteno u L_1, a s u L_2. Ide se na drugi korak.

d) $(b,i,\alpha a,s\gamma) \vdash (b,i-1,\alpha,\gamma)$
ako je na vrhu liste L_2 simbol koji označuje premještanje (simbol s)

Primjer 5.2
Primijenimo algoritam u sintaksnoj analizi jezika jednostavnih aritmetičkih izraza generiranog gramatikom G:

E → E + T	(1)	E → T	(2)
T → T * F	(3)	T → F	(4)
F → a	(5)	F →(E)	(6)

Primijetimo da je G rekurzivna slijeva, što u slučaju primjene algoritma uzlazne sintaksne analize ne predstavlja nikakve probleme.

Prema danom uređenju produkcija pojavom E+T na vrhu liste L_1 najprije će se pokušati reduciranje koristeći E→E+T, potom E→T. Također će se uvijek prije pokušati reducirati s T→T*F nego s T→F. Razmotrimo promjenu konfiguracije algoritma u sintaksnoj analizi niza a*a:

```
(q, 1, $, ε)

    ├ (q, 2, $a,   s)
    ├ (q, 2, $F,   5s)
    ├ (q, 2, $T,   45s)
    ├ (q, 2, $E,   245s)
    ├ (q, 3, $E*,  s245s)
    ├ (q, 4, $E*a, ss245s)
    ├ (q, 4, $E*F, 5ss245s)
    ├ (q, 4, $E*T, 45ss245s)
    ├ (q, 4, $E*E, 245ss245s)
    ├ (b, 4, $E*E, 245ss245s)
    ├ (b, 4, $E*T, 45ss245s)
    ├ (b, 4, $E*F, 5ss245s)
    ├ (b, 4, $E*a, ss245s)
    ├ (b, 3, $E*,  s245s)
    ├ (b, 2, $E,   245s)
    ├ (q, 3, $T*,  s45s)
    ├ (q, 4, $T*a, ss45s)
    ├ (q, 4, $T*F, 5ss45s)
    ├ (q, 4, $T,   35ss45s)
    ├ (q, 4, $E,   235ss45s)
    ├ (t, 4, $E,   235ss45s)
```

Niz a*a je u jeziku $L(G)$. Može se dobiti nizom izvođenja (desnih rečeničnih formi):

E ⇒ T ⇒ T*F ⇒ T*a ⇒ F*a ⇒ a*a

5.3 TABLIČNI POSTUPCI SINTAKSNE ANALIZE

Opisat ćemo još dva postupka sintaksne analize koji pripadaju klasi tzv. tabličnih postupaka sintaksne analize. To su Cocke-Younger-Kasamijev i Earlyjev postupak. Oba postupka zahtijevaju oko n^3 vremena za analizu ulaznog niza duljine n.

Cocke-Younger-Kasamijev algoritam (CYK)

Vidjeli smo da silazni i uzlazni povratni postupak sintaksne analize imaju vrijeme trajanja analize koje raste eksponencijalno s povećanjem duljine ulaznog niza. Cocke-Younger-Kasamijev (CYK) algoritam, međutim, treba n^3 vremena i n^2 memorijskog prostora. Neformalno, postupak je sljedeći: Neka je $\mathcal{G}=(\mathcal{N},\mathcal{T},\mathcal{P},\mathcal{S})$ gramatika u Chomskyjevoj normalnoj formi, bez ε–produkcija. Neka je $w=a_1...a_n$ ulazni niz kojeg treba analizirati. Pretpostavimo da je svaki a_i u T, $1\le i\le n$. Bit CYK postupka jest da se izgradi tablica sintaksne analize **T** čiji će elementi biti označeni s t_{ij}, $1\le i\le n$ i $1\le j\le n-i+1$. Svaki t_{ij} ima vrijednost koja je podskup od N. Neterminal A će biti u t_{ij} ako i samo ako je $A^+\Rightarrow a_i...a_{i+j-1}$, tj. ako A izvodi j ulaznih simbola koji počinju od pozicije i. U posebnom slučaju, w je u $\mathcal{L}(\mathcal{G})$ ako i samo ako je $\mathcal{S}$ u t_{1n}.

Dakle, da bi se utvrdilo da je niz w u jeziku mora se izgraditi tablica **T** za w i provjeriti je li S na mjestu t_{1n}. Potom, ako želimo niz izvođenja koji završava s w, moramo upotrijebiti tu tablicu. Ovdje će prvo biti dan algoritam 5.3 koji će izgraditi tablicu, a algoritam 5.4 "čita" niz izvođenja.

Algoritam 5.3

Cocke-Younger-Kasamiev postupak sintaksne analize (CYK).

Ulaz

Gramatika $\mathcal{G}=(\mathcal{N},\mathcal{T},\mathcal{P},\mathcal{S})$ u Chomskyjevoj normalnoj formi, bez ε–produkcija i ulazni niz $w=a_1...a_n$, $w\in\mathcal{T}^+$.

Izlaz

Tablica sintaksne analize **T** za w tako da t_{ij} sadrži A ako i samo ako je $A^+\Rightarrow a_i...a_{i+j-1}$.

Postupak

(1) Staviti $t_{i1}=\{A:\ A\to a_i$ u $\mathcal{P}\}$ za svaki i. Poslije ovog koraka, ako t_{i1} sadrži A, sigurno postoji niz izvođenja $A^+\Rightarrow a_i$.

(2) Pretpostavimo da je $t_{ij'}$ bilo izračunato za sve i, $1\le i\le n$, i sve j', $1\le j'\le j$. Staviti:

$t_{ij}=\{A:$ za neki k, $1\le k<j$, $A\to BC$ je u P, B je u t_{ik}, C je u $t_{i+k,j-k}\}$

Budući da je $1 \le k < j$, i k i $j-k$ su manji od j. Dakle, i t_{ik} i $t_{i+k,j-k}$ bili su izračunati prije nego je t_{ij} bilo izračunato. Poslije ovoga koraka ako t_{ij} sadrži A, tada je:

$$A \Rightarrow BC \overset{+}{\Rightarrow} a_i \ldots a_{i+k-1} C \overset{+}{\Rightarrow} a_i \ldots a_{i+k-1} a_{i+k} \ldots a_{i+j-1}$$

(3) Ponavljati korak (2) sve dok t_{ij} ne postane poznato za sve $1 \le i \le n$ i $1 \le j \le n-i+1$.

Primjer 5.3
Razmotrimo gramatiku $\mathcal{G}$ s produkcijama:

$$S \rightarrow AA \mid AS \mid b$$
$$A \rightarrow SA \mid AS \mid a$$

Ulazni je niz $abaab$. Primjenjujući algoritam 5.3, tablica sintaksne analize je:

5	A,S				
4	A,S	A,S			
3	A,S	S	A,S		
2	A,S	A	S	A,S	
j↑ 1	A	S	A	A	S
i→	1	2	3	4	5

Iz koraka (1) je $t_{11}=\{A\}$ budući je $A \rightarrow a$ u $\mathcal{P}$ i $a_1=a$. U koraku (2) se dodaje S u t_{32}, budući je $S \rightarrow AA$ u $\mathcal{P}$ i A je i u t_{31} i t_{41}. Primijetiti, općenito, da ako su t_{ij} prikazani kao što je pokazano, možemo izračunati t_{ij}, $i>1$, ispitujući neterminale u sljedećim parovima ulaza:

$$(t_{i1}, t_{i+1,j-1}), (t_{i2}, t_{i+2,j-2}), \ldots, (t_{i,j-1}, t_{i+j-1,1})$$

Tada, ako je B u t_{ik} i C u $t_{i+k,j-k}$ za neki k tako da je $1 \le k < j$ i $A \rightarrow BC$ je u $\mathcal{P}$, treba dodati A u t_{ij}. Budući je S u t_{15}, $abaab$ je u jeziku $\mathcal{L}(\mathcal{G})$.

U sljedećem je algoritmu opisano kako se iz tablice sintaksne analize može dobiti sintaksna analiza slijeva.

Algoritam 5.4
Sintaksna analiza slijeva iz CYK tablice sintaksne analize.

Ulaz
Gramatika $\mathcal{G}=(\mathcal{N}, \mathcal{T}, \mathcal{P}, \mathcal{S})$ u Chomskyjevoj normalnoj formi, u kojoj su produkcije numerirane od 1 do p, ulazni niz $w=a_1 \ldots a_n$ i tablica sintaksne analize **T** izgrađena algoritmom 5.3.

Izlaz
Sintaksna analiza slijeva (niz lijevih rečeničnih formi) ili "pogreška".

Postupak
Najprije opišimo rekurzivnu proceduru **gen**(i,j,A) koja će generirati sintaksnu analizu slijeva sukladnu izvođenju $A^{+}\Rightarrow a_i a_{i+1}...a_{i+j-1}$, također slijeva. Procedura **gen**(i,j,A) definirana je kako slijedi:

(1) Ako je i=1 i m-ta produkcija u $\mathcal{P}$ jest $A\rightarrow a_i$, ispisati broj produkcije m.
(2) Ako je j>1, k je najmanji cijeli broj, $1\le k<j$, tako da je za neki B u t_{ik} i C u $t_{i+k,j-k}$, $A\rightarrow BC$ produkcija u $\mathcal{P}$, recimo numerirana s m (može biti više izbora za $A\rightarrow BC$, ali ćemo uzeti prvu). Ispisati broj m i izvršiti **gen**(i,k,B), potom **gen**(i+k,j-k,C).

Postupak počinje s **gen**(1,n,$\mathcal{S}$), ispitujući je li $\mathcal{S}$ u t_{1n}. Ako nije, dojavljuje se pogreška, ako jest, nastavlja se na opisani način.

Primjer 5.4
Razmotrimo gramatiku $\mathcal{G}$ iz primjera 5.3. Numerirajmo produkcije:

(1) S → AA
(2) S → AS
(3) S → b
(4) A → SA
(5) A → AS
(6) A → a

Ulazni je niz abaab, kao u primjeru 5.3, pa je tablica sintaksne analize također kao u primjeru 5.3. Budući da je S u t_{15}, ulazni je niz u jeziku L(G). Da bismo našli sintaksnu analizu slijeva, poziva se procedura **gen**(1,5,S). Pronađeno je A u t_{11} i u t_{24}, te produkcija $S\rightarrow AA$ u skupu produkcija. Dakle, ispisuje se 1 (broj produkcije $S\rightarrow AA$).

Dalje se poziva **gen**(1,1,A) i **gen**(2,4,A). **gen**(1,1,A) daje produkciju broj 6. Budući da je S u t_{21} i A je u t_{33} i $A\rightarrow SA$ je četvrta produkcija, **gen**(2,4,A) ispisuje 4 i poziva **gen**(2,1,S), pa **gen**(3,3,A). Nastavljajući, na kraju bismo dobili 164356263. Dakle, rečenica abaab dobiva se u nizu izvođenja:

S ⇒ AA ⇒ aA ⇒ aSA ⇒ abA ⇒ abAS ⇒ abaS ⇒ abaAS
⇒ abaaS ⇒ abaab

Budući da je gramatika $\mathcal{G}$ dvoznačna, moguće je dobiti više sintaksnih analiza slijeva, što je ovdje izostavljeno.

Earleyjev postupak sintaksne analize

Završavamo ovo poglavlje s još jednom tabličnom metodom sintaksne analize poznatom kao "Earlyjev postupak sintaksne analize".

Neformalno, postupak je sljedeći. Neka je $G=(N, T, P, S)$ beskontekstna gramatika i $w=a_1...a_n$ ulazni niz, $w \in T^*$. Objekt oblika

$$[A \rightarrow X_1X_2...X_k \bullet X_{k+1}...X_m, i]$$

naziva se stavka za w ako je $A \rightarrow X_1X_2...X_m$ produkcija u P i $0 \le i \le n$. Točka između X_k i X_{k+1} je metasimbol i nije u $N \cup T$. Cijeli broj k može biti bilo koji broj od 0 (tada je • prvi simbol) do m (tada je • posljednji simbol). Ako je produkcija $A \rightarrow \varepsilon$, tada je stavka jednaka $[A \rightarrow \bullet, i]$.

Za svaki j, $0 \le j \le n$, treba izgraditi listu stavaka $\mathbf{I}_j$ tako da je $[A \rightarrow \alpha \bullet \beta, i]$ u $\mathbf{I}_j$, za $0 \le i \le j$, ako i samo ako za neke γ i δ vrijedi $S^* \Rightarrow \gamma A \delta$, $\gamma^* \Rightarrow a_1...a_i$ i $\alpha^* \Rightarrow a_{i+1}...a_j$. Dakle, druga komponenta stavke i broj liste u kojoj se pojavljuje stavljaju u zagradu dio ulaznog niza izvedenog iz α. Drugi uvjeti stavke samo nas osiguravaju od mogućnosti da se produkcija $A \rightarrow \alpha\beta$ može upotrijebiti na način kako se pojavljuje u nekom ulaznom nizu konzistentnom s w do pozicije j.

Niz lista $\mathbf{I}_0$, $\mathbf{I}_1$, ..., $\mathbf{I}_n$ može se nazvati lista sintaksne analize ulaznog niza w. Primijetiti da je w u jeziku $L(G)$ ako i samo ako postoji stavka oblika $[S \rightarrow \alpha \bullet, 0]$ u $\mathbf{I}_n$.

Algoritam 5.5
Earleyjev postupak sintaksne analize.

Ulaz
Beskontekstna gramatika $G=(N, T, P, S)$ i ulazni niz $w=a_1...a_n$ iz T^*.

Izlaz
Sintaksna analiza slijeva (niz lijevih rečeničnih formi) ili "pogreška".

Postupak
Prvo se konstruira $\mathbf{I}_0$ na sljedeći način:

1) Ako je $S \rightarrow \alpha$ produkcija u P, dodati $[S \rightarrow \bullet \alpha, 0]$ u $\mathbf{I}_0$. Potom izvršiti korak (2) i (3) sve dok se nova stavka može dodati u $\mathbf{I}_0$.
2) Ako je $[B \rightarrow \gamma \bullet, 0]$ u $\mathbf{I}_0$, gdje γ može biti i ε, što znači da se ovaj korak bio izvršio inicijalno, dodati $[A \rightarrow \alpha B \bullet \beta, 0]$ u $\mathbf{I}_0$ za sve $[A \rightarrow \alpha \bullet B \beta, 0]$.
3) Pretpostavimo da je $[A \rightarrow \alpha \bullet B \beta, 0]$ stavka u $\mathbf{I}_0$. Dodati u $\mathbf{I}_0$, za sve produkcije iz P oblika $B \rightarrow \gamma$, stavku $[B \rightarrow \bullet \gamma, 0]$ (uz pretpostavku da ta stavka nije već bila u $\mathbf{I}_0$). Sada se može konstruirati $\mathbf{I}_j$ iz $\mathbf{I}_0$, $\mathbf{I}_1$, ..., $\mathbf{I}_{j-1}$.

4) Za svaki [B→α•aβ,i] u $\mathbf{I}_{j-1}$ tako da je a=a_j dodati [B→αa•β,i] u $\mathbf{I}_j$. Potom izvršavati korake (5) i (6) sve dok se može dodati nova stavka.

5) Neka je [A→α•,i] stavka u $\mathbf{I}_j$. Provjeriti nalaze li se stavke oblika [B→α•Aβ,k] u $\mathbf{I}_i$. Za sve nađene stavke dodati [B→αA•β,k] u $\mathbf{I}_j$.

6) Neka je [A→α•Bβ,i] stavka u $\mathbf{I}_j$. Za sve B→γ u $\mathcal{P}$ dodati [B→•γ,j] u $\mathbf{I}_j$.

Primijetiti da se pojavom stavke s terminalom desno od točke ne tvori nova stavka u koracima (2), (3), (5) i (6). Algoritam, dakle, tvori $\mathbf{I}_j$ za 0≤j≤n.

Primjer 5.5
Primijenimo algoritam 5.5 u sintaksnoj analizi jezika jednostavnih aritmetičkih izraza generiranog gramatikom $\mathcal{G}$:

```
E → T + E      (1)
E → T          (2)
T → F * T      (3)
T → F          (4)
F → (E)        (5)
F → a          (6)
```

i neka je (a+a)*a ulazni niz. Iz koraka (1) dodajemo dvije nove stavke, [E→•T+E,0] i [E→•T,0] u $\mathbf{I}_0$. Te će stavke biti uzete u obzir pri dodavanju stavki [T→•F*T,0] i [T→•F,0] u $\mathbf{I}_0$ prema pravilu (3). Nastavljajući, dodaju se [F→•(E),0] i [F→•a,0]. Više se nijedna stavka ne može dodati u $\mathbf{I}_0$.

Sada gradimo $\mathbf{I}_1$. Prema (4) dodajemo [F→(•E),0], budući da je a_1=(•. Potom se, prema pravilu (6), dodaju [E→•T+E,1], [E→•T,1], [T→•F*T,1], [T→•F,1], [F→•(E),1] i [F→•a,1]. Sada se više nijedna stavka ne može dodati u $\mathbf{I}_1$.

Da bismo izgradili $\mathbf{I}_2$, primijetimo da je a_2=a i da se prema pravilu (4) stavka [F→a•,1] može dodati u $\mathbf{I}_2$. Dalje, prema pravilu (5), promatramo tu stavku odlazeći u $\mathbf{I}_1$ i tražeći stavke s F koje slijedi točka. Pronalazimo dvije stavke, [T→F•*T,1] i [T→F•,1], i dodajemo ih u $\mathbf{I}_2$. Promatrajući prvu od njih, ništa, ali druga nas navodi da ponovno pregledamo $\mathbf{I}_1$, ovog puta stavke sa •T u sebi. Najviše se dvije stavke mogu dodati u $\mathbf{I}_2$, [E→T•+E,1] i [E→T•,1]. Opet druga stavka uzrokuje da se [F→(E•),0] može dodati u $\mathbf{I}_2$. Sada se više nijedna stavka ne može dodati u $\mathbf{I}_2$. Nastavljajući, na kraju bismo dobili liste:

$\mathbf{I}_0$	$\mathbf{I}_1$	$\mathbf{I}_2$	$\mathbf{I}_3$
[E→•T+E,0]	[F→(•E),0]	[F→a•,1]	[E→T+•E,1]
[E→•T,0]	[E→•T+E,1]	[T→F•*T,1]	[E→•T+E,3]
[T→•F*T,0]	[E→•T,1]	[T→F•,1]	[E→•T,3]
[T→•F,0]	[E→•F*T,1]	[E→T•+E,1]	[T→•F*T,3]
[F→•(E),0]	[T→•F,1]	[E→T•+E,1]	[E→•F,3]
[F→•a,0]	[F→•(E),1]	[F→(E•),0]	[F→•(E),3]
	[F→•a,1]		[F→•a,3]

$\mathbf{I}_4$	$\mathbf{I}_5$	$\mathbf{I}_6$	$\mathbf{I}_7$
[F→a•,3]	[F→(E)•,0]	[T→F*•T,0]	[F→a•,6]
[T→F•*T,3]	[T→F•*T,0]	[T→•F*T,6]	[T→F•*T,6]
[T→F•,3]	[T→F•,0]	[T→•F,6]	[T→F•,6]
[E→T•+E,3]	[E→T•+E,0]	[F→•(E),6]	[T→F*T•,0]
[E→T•,3]	[E→T•,0]	[F→•a,6]	[E→T•+E,0]
[E→T+E•,1]			[E→T•,0]
[E→(E•),0]			

Budući da je [E→T•,0] na kraju liste, niz (a+a)*a je u jeziku $\mathcal{L}(\mathcal{G})$.

Pitanja i zadaci

1) Definirajte gramatiku iz uvodnog primjera silazne sintaksne analize uz pomoć programa META-BNF danog u Prilogu knjige, potom uz pomoć programa silazne sintaksne analize, također danog u Prilogu knjige, analizirajte ulazni niz aacbc i usporedite ispisane konfiguracije s izvođenjem danim na stranicama 83 i 84.

2) Definirajte gramatiku cijelih brojeva i ispišite, bez upotrebe programa silazne sintaksne analize, prvih deset konfiguracija ako je ulazni niz -54321.

3) Usporedite vremena izvršavanja postupka silazne analize ako je ulazna gramatika kao u primjeru 5.1 mijenjajući ulazni niz $w=(^n a)^n$ za n=1, 2, ..., 20. Što zaključujete?

4) Usporedite silaznu i ulaznu sintaksnu analizu za isti ulazni niz koristeći dvije ekvivalentne gramatike, gramatiku iz primjera 5.1 za silaznu, a gramatiku s produkcijama:

   E → E+E| E*E | (E) | a

   za uzlaznu sintaksnu analizu. Što zaključujete? Zašto?

5) Dan je stogovni automat $R=(\{q,r\},\Sigma,\Gamma,\delta,q,\$,\{r\})$, gdje je funkcija prijelaza δ definirana sa:

$$
\begin{array}{lll}
(1) & \delta(q,\alpha,\varepsilon) & = \{(q,\alpha)\} \quad \text{za } \alpha\in\{a,+,*,(,)\} \\
(2) & \delta(q,\varepsilon,E+T) & = \{(q,E)\} \\
 & \delta(q,\varepsilon,T) & = \{(q,E)\} \\
 & \delta(q,\varepsilon,T*F) & = \{(q,T)\} \\
 & \delta(q,\varepsilon,F) & = \{(q,T)\} \\
 & \delta(q,\varepsilon,(E)) & = \{(q,F)\} \\
 & \delta(q,\varepsilon,a) & = \{(q,F)\} \\
(3) & \delta(q,\varepsilon,\$E) & = \{(r,\varepsilon)\}
\end{array}
$$

Pokazati na primjeru ulaznog niza $a+a*a$ da R radi kao uzlazni postupak sintaksne analize.

6.

JEDNOPROLAZNA SINTAKSNA ANALIZA

slobodo moja
dugo sam te čuvao
kao rijetki biser
slobodo moja
ti si mi pomogla
da odvežem konopce
i krenem bilo kamo
i krenem do kraja
putova sudbine
da uberem sanjajući
ružu vjetrova
na zraci mjesečevoj

slobodo moja
pred tvojim htijenjima
moja duša je pokorena
slobodo moja
sve sam ti dao
svoju posljednju košulju
i koliko sam patio
da mogu zadovoljiti
sve tvoje zahtjeve
mijenjao sam zemlje
gubio prijatelje
da zadobijem tvoje povjerenje

slobodo moja
znala si me odvići
i od najmanjih navika
slobodo moja
ti zbog koje sam zavolio
čak i samoću
ti zbog koje sam se smiješio
dok sam gledao kako završava
jedna lijepa avantura
ti koja si me štitila
dok sam se skrivao
da njegujem svoje rane

slobodo moja
ipak sam te napustio
jedne decembarske noći
napustio sam
puteve lutanja
koje smo zajedno slijedili
i nisam se čuvao
šaka i nogu sputanih
ostavio sam sve
i izdao sam te
zbog jednog zatvora ljubavi
i njegove lijepe tamničarke

slobodo moja
ma liberté
(georges moustaki/ nepoznati student)

U prethodnom su poglavlju opisane dvije povratne ("backtrack") metode nedeterminističke sintaksne analize slijeva i zdesna koje se, uz određene transformacije gramatika, mogu primijeniti na cijeloj klasi beskontekstnih gramatika, odnosno jezika koje one generiraju. Bilo je riječi o ograničenjima primjenljivosti takvih postupaka i o njihovim nedostacima, a glavni im je nedostatak bio vrijeme trajanja (ili broj koraka), posebno ako ulazni niz nije u jeziku.

U ovom će poglavlju biti riječi o klasi beskontekstnih jezika za koje je moguće konstruirati efikasne programe sintaksne analize koji čine $c_1 n$ operacija i koriste $c_2 n$ memorijskog prostora u obradi niza duljine n, gdje su c_1 i c_2 konstante. Takvi postupci rade deterministički, pretražujući ulazni niz samo jedanput. Primjenljivi su nad određenom klasom beskontekstnih jezika. Ovdje će biti obuhvaćene tri klase takvih jezika, odnosno gramatika koje ih generiraju.

To su jezici (gramatike) tipa LL(k) i LR(k), za koje je moguće konstruirati jednoprolazni postupak (program) sintaksne analize slijeva (za LL) ili zdesna (za LR), koji će raditi deterministički ako im se dopusti da "pogledaju" najviše k ulaznih znakova slijeva nadesno (prvo slovo L u LL i LR to naznačuje) od neke tekuće pozicije, te gramatike s prioritetom operatora za koje je moguće napisati deterministički postupak sintaksne analize upravljan tablicom prioriteta relacije.

6.1 JEZICI TIPA $\mathcal{LL}$(k)

Počinjemo s najvećom "prirodnom" klasom jezika za koje je moguća (silazna) analiza slijeva. To su $\mathcal{LL}(k)$ jezici. Najprije dajemo definiciju generatora tih jezika - gramatika tipa $\mathcal{LL}(k)$.

Definicija gramatike tipa $\mathcal{LL}$(k)

Neka je $\mathcal{G}=(\mathcal{N},\mathcal{T},\mathcal{P},\mathcal{S})$ jednoznačna gramatika i $w=a_1a_2\ldots a_n$ rečenica jezika $\mathcal{L}(\mathcal{G})$. Tada postoji jedinstven niz lijevih rečeničnih formi α_0, α_1, ..., α_m tako da vrijedi:

$$\mathcal{S} = \alpha_0, \qquad \alpha_i \underset{lm}{\overset{p_i}{\Rightarrow}} \alpha_{i+1}$$

za $0 \le i < m$ i $\alpha_m = w$. Sintaksna analiza slijeva za w je $p_0 p_1 \ldots p_{m-1}$. Pretpostavimo da je potrebno naći taj niz lijevih rečeničnih formi pretražujući ulazni niz w slijeva nadesno samo jednom. To se može pokušati uraditi konstruirajući niz lijevih rečeničnih formi.

Ako je

$$\alpha_i = a_1 \ldots a_j A \beta$$

tada na tom mjestu treba učitati prvih j znakova ulaznog niza i usporediti ih s prvih j znakova od α_i. Bilo bi dobro kad bi se α_{i+1} moglo odrediti znajući samo $a_1 \ldots a_j$ (dio ulaznog niza koji treba pretražiti do tog mjesta), nekoliko slijedećih ulaznih znakova ($a_{j+1} a_{j+2} \ldots a_{j+k}$ za neki fiksni k) i neterminal A. Ako te tri stvari jedinstveno određuju produkciju koja će biti upotrijebljena za ekspanziju neterminala A, može se precizno odrediti α_{i+1} iz α_i i k ulaznih simbola $a_{j+1} a_{j+2} \ldots a_{j+k}$.

Gramatika u kojoj svako krajnje izvođenje slijeva ima to svojstvo je tipa $\mathcal{LL}(k)$. Ovdje prvi "$\mathcal{L}$" označuje da se ulazni niz pretražuje slijeva, a drugi "$\mathcal{L}$" da se pri tome izvode lijeve rečenične forme, koristeći k tekućih znakova za donošenje odluke koja će produkcija biti upotrijebljena. Vidjet ćemo da je za gramatike tipa $\mathcal{LL}(k)$ moguće konstruirati deterministički postupak sintaksne analize. Prije potpune definicije gramatike tipa $\mathcal{LL}(k)$ dajemo definiciju skupa FIRST.

Definicija 6.1
Neka je $\alpha = x\beta$ lijeva rečenična forma gramatike $\mathcal{G}=(\mathcal{N},\mathcal{T},\mathcal{P},\mathcal{S})$ tako da je $x \in \mathcal{T}^*$, a β počinje ili neterminalom ili je jednako ε. Kažemo da je x zatvoreni dio od α. Granica između x i β jest međa.

Primjer 6.1
Neka je $\alpha = abacAaB$. Zatvoreni dio od α jest $abac$, otvoreni AaB. Ako je $\alpha = abc$, abc je zatvoreni dio, ε otvoreni. Međa je na desnom kraju.

Definicija 6.2
Neka je $\mathcal{G}=(\mathcal{N},\mathcal{T},\mathcal{P},\mathcal{S})$ beskontekstna gramatika i neka su $\alpha, \beta \in (\mathcal{N} \cup \mathcal{T})^*$. Definira se:

$$FIRST_k(\alpha) = \{x \mid \alpha^* \Rightarrow x\beta \text{ i } |x|=k, \text{ ili } \alpha^* \Rightarrow x \text{ i } |x|<k\}$$

tj. $FIRST_k(\alpha)$ se sastoji od svih prefiksnih terminala duljine k (ili manje, ako α izvodi niz terminala duljine manje od k), niza terminala koji može biti izveden iz α. Ako je $FIRST_k(\alpha)=\{w\}$, $w \in \mathcal{T}^*$, pisat ćemo $FIRST_k(\alpha)=w$.

Definicija 6.3
Neka je $\mathcal{G}=(\mathcal{N},\mathcal{T},\mathcal{P},\mathcal{S})$ beskontekstna gramatika. Kaže se da je $\mathcal{G}$ tipa $\mathcal{LL}(k)$, za neki fiksni cijeli broj k, ako u slučaju da postoje dva krajnja izvođenja slijeva:

1) $S \underset{lm}{*\Rightarrow} wA\alpha \underset{lm}{*\Rightarrow} w\beta\alpha \underset{lm}{*\Rightarrow} wx$ i

2) $S \underset{lm}{*\Rightarrow} wA\alpha \underset{lm}{*\Rightarrow} w\gamma\alpha \underset{lm}{*\Rightarrow} wy$

tako da je $FIRST_k(x)=FIRST_k(y)$, vrijedi $\beta=\gamma$. Kaže se da je gramatika tipa LL ako je tipa $LL(k)$ za neki k.

Neformalno, G je tipa $LL(k)$ ako za danu lijevu rečeničnu formu $wA\alpha$ iz $(N \cup T)^+$ i prvih k terminalnih znakova (ako egzistiraju) koji će biti izvedeni iz $A\alpha$ postoji najviše jedna produkcija koja će se upotrijebiti za A da bi se izveo niz znakova koji počinje s w iza kojeg slijedi k takvih znakova.

Primjer 6.2
Neka je G_1 gramatika s produkcijama:

$S \rightarrow aAS \mid b$
$A \rightarrow a \mid bSA$

Intuitivno, G_1 je gramatika tipa $LL(1)$ zato što za dani C, prvi neterminal u bilo kojoj lijevoj rečeničnoj formi, i c, slijedeći ulazni znak, postoji najviše jedna produkcija za C sposobna izvesti niz terminala započet s c. Prema definiciji gramatike tipa $LL(1)$, ako je

$S \underset{lm}{*\Rightarrow} wA\alpha \underset{lm}{*\Rightarrow} w\beta\alpha \underset{lm}{*\Rightarrow} wx$ i

$S \underset{lm}{*\Rightarrow} wA\alpha \underset{lm}{*\Rightarrow} w\gamma\alpha \underset{lm}{*\Rightarrow} wy$

i ako x i y počinju istim znakom, mora biti $\beta=\gamma$. Posebno, ako x i y počinju s a, tada se mora uporabiti produkcija $S \rightarrow aAS$ i $\beta=\gamma=aAS$. Alternativu $S \rightarrow b$ nije moguće upotrijebiti. Ako x i y počinju s b, mora se uporabiti alternativa $S \rightarrow b$ i vrijedi $\beta=\gamma=b$ ($x=y=\varepsilon$ je nemoguće jer se ne izvodi ε u gramatici G_1). Slično, ako se promatraju dva izvođenja:

$S \underset{lm}{*\Rightarrow} wA\alpha \underset{lm}{*\Rightarrow} w\beta\alpha \underset{lm}{*\Rightarrow} wx$ i

$S \underset{lm}{*\Rightarrow} wA\alpha \underset{lm}{*\Rightarrow} w\gamma\alpha \underset{lm}{*\Rightarrow} wy$

i $x=y=a$, onda je za A morala biti upotrijebljena produkcija $A \rightarrow a$, a ako je $x=z=b$, produkcija $A \rightarrow bSA$.

Gramatika G_1 iz prethodnog primjera primjer je primitivne gramatike tipa $LL(1)$. Slijedi definicija:

Definicija 6.4
Beskontekstna gramatika $\mathcal{G}=(\mathcal{N},\mathcal{T},\mathcal{P},\mathcal{S})$ bez ε-produkcija, tako da za sve $A\in\mathcal{N}$ svaka alternativa od A počinje različitim terminalom, naziva se primitivna $\mathcal{LL}(1)$ gramatika.

Dakle, u primitivnoj $\mathcal{LL}(1)$ gramatici za dani par (A,a), gdje je $A\in\mathcal{N}$, $a\in\mathcal{T}$, postoji najviše jedna produkcija oblika $A\rightarrow a\alpha$.

Primjer 6.3
Razmotrimo kompliciraniji slučaj gramatike $\mathcal{G}_2$ definirane sa:

$$S \rightarrow \varepsilon \mid abA$$
$$A \rightarrow Saa \mid b$$

Može se pokazati da je $\mathcal{G}_2$ tipa $\mathcal{LL}(2)$. Da bi se to uradilo, treba pokazati da ako je $wB\alpha$ bilo koja lijeva rečenična forma od $\mathcal{G}_2$ i wx je rečenica u $\mathcal{L}(\mathcal{G}_2)$, tada postoji najviše jedna produkcija $B\rightarrow\beta$ u $\mathcal{G}_2$ tako da $FIRST_2(\beta\alpha)$ sadrži $FIRST_2(x)$. Pretpostavimo da je:

$$S \underset{lm}{\overset{*}{\Rightarrow}} wS\alpha \underset{lm}{\overset{*}{\Rightarrow}} w\beta\alpha \underset{lm}{\overset{*}{\Rightarrow}} wx \quad \text{i}$$

$$S \underset{lm}{\overset{*}{\Rightarrow}} wS\alpha \underset{lm}{\overset{*}{\Rightarrow}} w\gamma\alpha \underset{lm}{\overset{*}{\Rightarrow}} wy$$

gdje su prva dva znaka x i y jednaka, ako postoje. Ili je $w=\alpha=\varepsilon$, ili je produkcija $A\rightarrow Saa$ bila upotrijebljena najmanje jedanput u izvođenju $S\underset{lm}{\overset{*}{\Rightarrow}}wS\alpha$. Dakle, ili je $\alpha=\varepsilon$, ili počinje s aa.

Pretpostavimo da je $S\rightarrow\varepsilon$ bilo upotrijebljeno u izravnom izvođenju $wS\alpha$ u $w\beta\alpha$. Tada je $\beta=\varepsilon$ i x je ili ε ili počinje s aa. Slično, ako je $S\rightarrow\varepsilon$ bilo upotrijebljeno u izravnom izvođenju $wS\alpha$ u $w\gamma\alpha$, tada je $\alpha=\varepsilon$ i y=ε, ili y počinje s aa. Ako je u izravnom izvođenju $wS\alpha$ u $w\beta\alpha$ bilo upotrijebljeno $S\rightarrow abA$, tada je $\beta=abA$ i x počinje s ab. Slično, ako je $S\rightarrow abA$ bilo upotrijebljeno u izravnom izvođenju $wS\alpha$ u $w\gamma\alpha$, tada je $\gamma=abA$ i y počinje s ab. Prema tome, ne postoji neka druga mogućnost osim da je $x=y=\varepsilon$, x i y počinju s aa, ili oba počinju s ab.

Posljedice definicije $\mathcal{LL}$(k)

Iz definicije $\mathcal{LL}(k)$ gramatike slijedi da, ako je dana lijeva rečenična forma $wA\alpha$, tada w i k znakova koji slijede w jedinstveno određuju koja će produkcija biti upotrijebljena za ekspanziju od A. Na prvi pogled, moraju se pamtiti svi w da bi se zaključilo koja će produkcija biti upotrijebljena kao sljedeća. Međutim, nije tako. Idući je teorem fundamentalan za razumijevanje $\mathcal{LL}(k)$ gramatika.

Teorem 6.1
Neka je $\mathcal{G}=(\mathcal{N},\mathcal{T},\mathcal{P},\mathcal{S})$ beskontekstna gramatika. $\mathcal{G}$ je tipa $\mathcal{LL}(k)$ ako i samo ako vrijedi sljedeći uvjet: ako su $A\to\beta$ i $A\to\gamma$ dvije različite produkcije u $\mathcal{P}$, tada je:

$$FIRST_k(\beta\alpha) \cap FIRST_k(\gamma\alpha) = \varnothing$$

za sve $wA\alpha$ tako da je $\mathcal{S}^* \underset{lm}{\Rightarrow} wA\alpha$.

Dakle, prema teoremu 6.1, beskontekstna je gramatika $\mathcal{G}=(\mathcal{N},\mathcal{T},\mathcal{P},\mathcal{S})$ tipa $\mathcal{LL}(1)$ ako i samo ako za sve A iz $\mathcal{N}$ svaki je skup A-produkcija $A\to\alpha_1|\alpha_2|\ldots|\alpha_n$ iz $\mathcal{P}$ takav da su svi parovi $FIRST_1(\alpha_1)$, $FIRST_1(\alpha_2)$, ..., $FIRST_1(\alpha_n)$, disjunktni skupovi.

Primjer 6.4
Gramatika $\mathcal{G}$ s produkcijama $S\to aS|a$ ne može biti tipa $\mathcal{LL}(1)$ jer je:

$$FIRST_1(aS) = FIRST_1(a) = a$$

Intuitivno, u sintaksnoj analizi niza koji počinje s a, gledajući samo prvi ulazni znak, nije moguće odrediti hoće li za ekspanziju od S biti upotrijebljeno $S\to aS$ ili $S\to a$. S druge strane, $\mathcal{G}$ je tipa $\mathcal{LL}(2)$. Prema teoremu 6.1, ako je $S^*\Rightarrow wA\alpha$, tada je $A=S$ i $\alpha=\varepsilon$. Za S su dane dvije produkcije, tako da je $\beta=aS$ i $\gamma=\varepsilon$. Budući da je $FIRST_2(aS)=aa$ i $FIRST_2(a)=a$, $\mathcal{G}$ je na temelju teorema 6.1 tipa $\mathcal{LL}(2)$.

Razmotrimo sada $\mathcal{LL}(1)$ gramatike s ε-produkcijama. Najprije uvodimo definiciju funkcije $FOLLOW_k$.

Definicija 6.5
Neka je $\mathcal{G}=(\mathcal{N},\mathcal{T},\mathcal{P},\mathcal{S})$ beskontekstna gramatika. Za $\beta\in(\mathcal{N}\cup\mathcal{T})^*$ definira se funkcija:

$$FOLLOW_k(\beta) = \{w:\ \mathcal{S}^*\Rightarrow\alpha\beta\gamma \text{ i } w\in FIRST_k(\gamma)\}$$

Dakle, $FOLLOW_1(A)$ uključuje skup terminalnih znakova koji se mogu pojaviti neposredno iza A u bilo kojoj rečeničnoj formi. Ako je αA rečenična forma, tada je ε također u $FOLLOW_1(A)$.

Proširimo funkcije $FIRST$ i $FOLLOW$ do domene u kojoj će se umjesto nizova pojavljivati skupovi nizova, tj. ako je $\mathcal{G}=(\mathcal{N},\mathcal{T},\mathcal{P},\mathcal{S})$ beskontekstna gramatika i $X\subseteq(\mathcal{N}\cup\mathcal{T})^*$, tada je:

$$FIRST_k(X) = \{w: \text{za neki } \alpha \text{ u } X,\ w\in FIRST_k(\alpha)\}$$
$$FOLLOW_k(X) = \{w: \text{za neki } \alpha \text{ u } X,\ w\in FOLLOW_k(\alpha)\}$$

Teorem 6.2
Beskontekstna gramatika $\mathcal{G}=(\mathcal{N},\mathcal{T},\mathcal{P},\mathcal{S})$ je tipa $\mathcal{LL}(1)$ ako i samo ako za svaki A u $\mathcal{N}$, gdje su $A\rightarrow\beta$ i $A\rightarrow\gamma$ dvije različite produkcije, vrijedi:

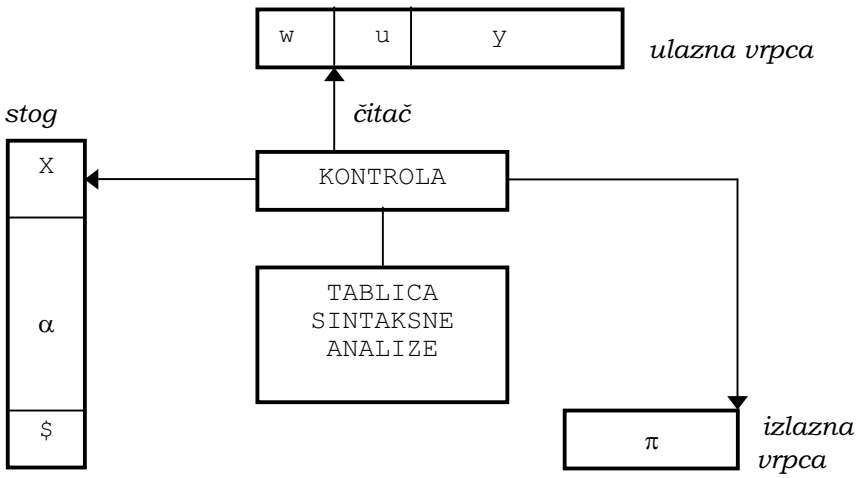
$$FIRST_1(\beta\ FOLLOW_1(A)) \cap FIRST_1(\gamma\ FOLLOW_1(A)) = \varnothing$$

Prema tome, gramatika $\mathcal{G}$ je tipa $\mathcal{LL}(1)$ ako i samo ako za svaki par A-produkcija $A\rightarrow\alpha_1|\alpha_2|\ldots|\alpha_n$ vrijede sljedeći uvjeti:

1) $FIRST_1(\alpha_1), FIRST_1(\alpha_2), \ldots, FIRST_1(\alpha_n)$ su disjunktni po parovima.
2) Ako je $\alpha_i \overset{*}{\Rightarrow} \varepsilon$, tada je $FIRST_1(\alpha_j) \cap FOLLOW_1(A) = \varnothing$, za $1 \le j \le n$, $i \ne j$.

Predikatna sintaksna analiza

$\mathcal{LL}(k)$ jezici mogu se analizirati veoma efikasno uporabom k-predikatnih postupaka. Postupak predikatne ("k-predikatne" ili "k-predvidljive") sintaksne analize $\mathcal{LL}(k)$ jezika generiranog gramatikom $\mathcal{G}=(\mathcal{N},\mathcal{T},\mathcal{P},\mathcal{S})$ shematski je prikazan na sljedećem crtežu:



Sl. 6.1 - *Model k-predikatnog postupka SA.*

Ulazna vrpca sadrži ulazni niz. Čitač može učitati do k slijedećih znakova, nazvat ćemo ih tekući niz. Na slici je to niz u.

Stog sadrži niz $X\alpha\$$, gdje je $ poseban znak za oznaku kraja stoga. Simbol X je na vrhu stoga. Sa Γ će biti označen alfabet znakova stoga.

Izlazna vrpca sadrži niz indeksa π produkcija gramatike $\mathcal{G}$ koje su bile upotrijebljene za izvođenje ulaznog niza.

Konfiguracija postupka predikatne sintaksne analize, jer se očigledno radi o prepoznavaču jezika $LL(k)$, definirana je uređenom trojkom:

$(x, X\alpha, \pi)$

gdje su:

- x neuporabljeni dio ulaznog niza
- $X\alpha$ niz stoga, X je na vrhu
- π izlazni niz

Akcija postupka predikatne SA, **A**, upravljana je tablicom sintaksne analize, **M**, i predstavlja preslikavanje iz skupa $(\Gamma \cup \{\$\}) \times T^{*k}$ u skup koji sadrži sljedeće elemente:

1) (β, i), gdje je β u Γ^*, a i je broj produkcije. Pretpostavlja se da je β ili desna strana produkcije s indeksom i, ili njezina reprezentacija
2) pop
3) accept
4) error

Postupak SA analizira ulazni niz čineći niz premještanja. U jednom premještanju utvrđuju se tekući niz, u, i simbol X na vrhu stoga. Tada se konzultira ulaz $\mathbf{M}(X,u)$ tablice sintaksne analize da bi se utvrdilo aktualno premještanje (akcija). I ovdje ćemo za premještanje koristiti relaciju $\vdash$ na skupu konfiguracija. Neka je u jednako $FIRST_k(x)$. Piše se:

1) $(x, X\alpha, \pi) \vdash (x, \beta\alpha, \pi i)$

   ako je $\mathbf{M}(X,u)=(\beta,i)$. Ovdje se znak X na vrhu stoga zamjenjuje nizom $\beta \in \Gamma^*$ i broj produkcije i dodaje izlazu. Čitač se ne pomiče.

2) $(x, a\alpha, \pi) \vdash (x', \alpha, \pi)$

   ako je $\mathbf{M}(a,u)=$pop i $x=ax'$, tj. kad je znak na vrhu stoga jednak tekućem znaku (prvom znaku tekućeg niza), stog "puca" i čitač se pomiče za jedno mjesto udesno.

3) Ako postupak dosegne konfiguraciju $(\varepsilon, \$, \pi)$, analiza je završena. Izlazni niz π sadrži indekse produkcija koje su bile upotrijebljene u izvođenju lijevih rečeničnih formi, počevši sa S i završavajući s ulaznim nizom. Može se pretpostaviti da je $\mathbf{M}(\$,\varepsilon)$ uvijek jednako accept. Konfiguracija $(\varepsilon, \$, \pi)$ je prihvatljiva.

4) Ako postupak dosegne konfiguraciju $(x, X\alpha, \pi)$ i $\mathbf{M}(X,u)=$error, prekida se daljnja analiza i dojavljuje pogreška. Konfiguracija $(x, X\alpha, \pi)$ je neprihvatljiva.

Ako je $w \in \mathcal{T}^*$ niz koji treba analizirati, tada je početna konfiguracija postupka sintaksne analize $(w, X_0\$, \varepsilon)$, gdje je X_0 početni znak. Ako je:

$$(w, X_0\$, \varepsilon) \ \ ^*\vdash \ (\varepsilon, \$, \pi)$$

piše se $\mathbf{A}(w)=\pi$ i π se naziva izlaz od **A** za ulaz w. Ako $(w, X_0\$, \varepsilon)$ ne dosegne prihvatljivu konfiguraciju, kaže se da je $\mathbf{A}(w)$ nedefinirano.

Kaže se da je **A** valjani postupak (algoritam) k-predikatne sintaksne analize beskontekstne gramatike $\mathcal{G}$ ako:

1) $\mathcal{L}(\mathcal{G}) = \{w: \mathbf{A}(w)$ je definirano$\}$, i
2) Ako je $\mathbf{A}(w)=\pi$, tada je π sintaksna analiza slijeva za w.

U ovom se slučaju za **M** kaže da je valjana tablica sintaksne analize za $\mathcal{G}$.

Primjer 6.5
Konstruirajmo 1-predikatni postupak sintaksne analize **A** gramatike:

(1) S→aAS (2) S→b (3) A→a (4) A→bSA

	a	b	ε
S	aAS, 1	b, 2	error
A	a, 3	bSA, 4	error
a	pop	error	error
b	error	pop	error
$	error	error	accept

Stupci su označeni znakovima iz alfabeta uz dodatak praznog niza, ε, i predstavljaju tekući znak, a redovi su označeni znakovima iz $\mathcal{N}$ i $\mathcal{T}$, uz dodatak znaka $, i predstavljaju znak na vrhu stoga. Koristeći tu tablicu, **A** će analizirati ulazni niz abbab kao što slijedi:

```
(abbab, S$, ε) ├ (abbab, aAS$, 1)
               ├ (bbab,   AS$, 1)
               ├ (bbab, bSAS$, 14)
               ├ (bab,   SAS$, 14)
               ├ (bab,   bAS$, 142)
               ├ (ab,     AS$, 142)
               ├ (ab,     aS$, 1423)
               ├ (b,       S$, 1423)
               ├ (b,       b$, 14232)
               ├ (ε,         $, 14232)
```

Dakle, abbab je u jeziku generiranom danom gramatikom. Taj se niz može dobiti nizom izvođenja:

$$S \Rightarrow aAS \Rightarrow abSAS \Rightarrow abbAS \Rightarrow abbaS \Rightarrow abbab$$

Niz π=14232 sadrži indekse (brojeve) produkcija koje su bile upotrijebljene u izravnim izvođenjima lijevih rečeničnih formi.

Sintaksna analiza $LL(1)$ jezika

Srce postupka k-predikatne sintaksne analize jest tablica sintaksne analize, **M**. U prethodnom primjeru smo je napisali intuitivno. Pogledajmo kako se ta tablica tvori u posebnom slučaju, kada je gramatika tipa $LL(1)$.

Algoritam 6.1 *Tvorba tablice sintaksne analize $LL(1)$ gramatike.*

Ulaz: Beskontekstna gramatika $G=(N,T,P,S)$ tipa $LL(1)$.

Izlaz: M, valjana tablica sintaksne analize za G.

Postupak: Pretpostavimo da je \$ na dnu stoga. **M** je definirano na $(\Gamma \cup \{\$\}) \times (T \cup \{\varepsilon\})$, gdje je $\Gamma = N \cup T$, kao što slijedi:

1) Ako je $A \to \alpha$ i-ta produkcija u P, tada je **M**$(A,a)=(\alpha,i)$ za sve a u $FIRST_1(\alpha)$, $a \neq \varepsilon$. Ako je ε također u $FIRST_1(\alpha)$, tada je **M**$(A,b)=(\alpha,i)$ za sve b u $FOLLOW_1(A)$.

2) **M**$(a,a)=$pop za sve $a \in T$.

3) **M**$(\$,\varepsilon)=$accept.

4) Inače, **M**$(X,a)=$error, $X \in (\Gamma \cup \{\$\})$, $a \in (T \cup \{\varepsilon\})$.

Primjer 6.6
Razmotrimo tvorbu tablice sintaksne analize za gramatiku G s produkcijama:

(1) E → TA (2) A → +TA (3) A → ε (4) T → FB
(5) B → *FB (6) B → ε (7) F → (E) (8) F → a

Gramatika G dobivena je transformacijom gramatike G_0 definirane sa:

E → E+T | T T → T*F | F F → (E) | a

G_0 nije tipa $LL(1)$ (rekurzivna je slijeva). Primjenom algoritma za eliminiranje rekurzija slijeva dobila se gramatika G za koju se, primjenjujući teorem 6.2, može pokazati da je tipa $LL(1)$.

Izračunajmo najprije E-red koristeći korak (1) algoritma 6.1. Ovdje je $FIRST_1(TA)=\{(,a\}$, pa je:

$$\mathbf{M}(E,()=\mathbf{M}(E,a)=TA,1$$

Svi ostali ulazi u E-redu su error. Sada izračunajmo ulaze A-reda. Uočavamo da je $FIRST_1(+TA)=+$, pa je:

$$\mathbf{M}(A,+)=+TA,2$$

Budući je $A\rightarrow\varepsilon$ produkcija, mora se izračunati $FOLLOW_1(A)$, a jednak je $\{\varepsilon,)\}$, pa je :

$$\mathbf{M}(A,\varepsilon)=\mathbf{M}(A,))=\varepsilon,3$$

Ostali ulazi za A su error. Nastavljajući s izračunavanjem redova ostalih produkcija, na koncu bismo dobili tablicu sintaksne analize (radi preglednosti su na mjestima gdje je trebalo stajati error ostale praznine):

	a	(	)	+	*	ε
E	TA, 1	TA, 1				
A			ε, 3	+TA, 2		ε, 3
T	FB, 4	FB, 4				
B			ε, 6	ε, 6	*FB, 5	ε, 6
F	a, 8	(E), 7				
a	pop					
(		pop				
)			pop			
+				pop		
*					pop	
$						accept

Ako, na primjer, treba analizirati niz (a*a), algoritam 1-predikatne sintaksne analize učinit će sljedeći niz premještanja (radi izbjegavanja dvoznačnosti, konfiguracije su napisane u uglatim zagradama):

```
[(a*a),E$,ε] ├─ [(a*a),      TA$,  1]
             ├─ [(a*a),     FBA$,  14]
             ├─ [(a*a),   (E)BA$,  147]
             ├─ [ a*a),    E)BA$,  147]
             ├─ [ a*a),   TA)BA$,  1471]
             ├─ [ a*a),  FBA)BA$,  14714]
             ├─ [ a*a),  aBA)BA$,  147148]
             ├─ [  *a),   BA)BA$,  147148]
             ├─ [  *a), *FBA)BA$,  1471485]
```

```
├─ [   a),  FBA)BA$,  1471485]
├─ [   a),  aBA)BA$,  14714858]
├─ [    ),   BA)BA$,  14714858]
├─ [    ),    A)BA$,  147148586]
├─ [    ),     )BA$,  1471485863]
├─ [    ε,      BA$,  1471485863]
├─ [    ε,       A$,  14714858636]
├─ [    ε,        $,  147148586363]
```

Dakle, ulazni niz (a*a) je u jeziku L(G) i može se dobiti nizom izvođenja:

$$E \Rightarrow TA \Rightarrow FBA \Rightarrow (E)BA \Rightarrow (TA)BA \Rightarrow (FB)BA \Rightarrow (aB)BA \Rightarrow (a*FB)BA \Rightarrow (a*aB)BA \Rightarrow (a*a)BA \Rightarrow (a*a)A \Rightarrow (a*a)$$

Rekurzivni spust

Često se koristi sintaksna analiza "s rekurzivnim spustom", koju je publicirao Wirth u sintaksnoj analizi jezika PL/0. Moglo bi se reći da je to pravi "školski primjer" realizacije problema SA $LL(1)$ jezika. Ako pretpostavimo da su produkcije neterminala A, $A \rightarrow \alpha_1 | \ldots | \alpha_n$, i ako je Sym tekući simbol, dio postupka sintaksne analize s rekurzivnim spustom koji se odnosi na A možemo općenito prikazati ovako:

```
PROCEDURE A; BEGIN
  IF Sym∈FIRST(α1FOLLOW(A)) THEN
     { Kod za α1 }
  ELSE IF Sym∈FIRST (α2 FOLLOW (A)) THEN
     { Kod za α1 }
          ...
  ELSE IF Sym∈FIRST (αn FOLLOW (A)) THEN
     { Kod za αn }
  ELSE
     Error
  END;
```

To znači da će se sparivanjem tekućeg simbola s jednim od očekujućih pozvati odgovarajuća procedura. Ako to nije moguće, bit će dojavljena pogreška.

Postupak SA s rekurzivnim spustom, kao što slijedi iz njegova imena, sadrži rekurzije. Otuda i zaključak da će se najbolji efekti doseći njegovom ustrojbom u nekom jeziku za programiranje koji dopušta rekurzije (npr. Pascal). Tada je moguće iz sintaksnih dijagrama izravno izvesti program sintaksne analize danog jezika. Ako se postupak ustroji u jeziku koji ne dopušta rekurzije, treba uložiti dodatni napor za elimininiranje rekurzija, za što je potrebno uvesti pomoćne varijable i strukture podataka.

6.2 JEZICI TIPA $\mathcal{LR}$(k)

Za jezike tipa $\mathcal{LL}(k)$ moguće je, dakle, ustrojiti jednoprolazne silazne postupke sintaksne analize, dok se ulazni niz pretražuje slijeva nadesno. Postoji analogna klasa jezika za koju se mogu ustrojiti jednoprolazni postupci SA generirajući pritom stablo izvođenja odozdo.

Gramatike koje generiraju jezike s takvim svojstvom jesu tipa $\mathcal{LR}(k)$, $k \geq 0$. Ovdje "$\mathcal{L}$" označuje da se ulazni niz pretražuje slijeva ("left"), a "$\mathcal{R}$" da se pritom izvode desne ("right") rečenične forme, koristeći k tekućih simbola ulaznog niza za odluku koja će produkcija biti upotrijebljena.

Gramatike tipa $\mathcal{LR}$(0)

Najprije ćemo definirati gramatike tipa $\mathcal{LR}(0)$, podklasu $\mathcal{LR}$ gramatika, u kojoj $\mathcal{LR}(0)$ ima značenje "pretraživanje ulaznog niza slijeva izvodeći desnu rečeničnu formu, koristeći pritom 0 znakova ulaznog niza od tekuće pozicije u ulaznom nizu". Pokazuje se da takve gramatike generiraju jezike koji imaju svojstvo prefiksa (jezik $\mathcal{L}$ ima svojstvo prefiksa ako kad god je w u $\mathcal{L}$, nijedan svojstveni prefiks od w nije u $\mathcal{L}$; v. def. 1.2). Primijetiti da svojstvo prefiksa nije strog uvjet, jer svaki beskontekstni jezik može imati svojstvo prefiksa ako se uvede jedan znak kao oznaka (marker) kraja svih rečenica (tj. proširi se gramatika uvođenjem produkcije $S' \rightarrow S\#$, gje je "#" oznaka kraja).

Primjer 6.7
Gramatika s produkcijama:

$$S \rightarrow SA|A$$
$$A \rightarrow (S)|()$$

generira jezik "uparenih zagrada", $\{(),(()),()(),((())),(()()),\ ...\}$ i nije tipa $\mathcal{LR}(0)$, jer, na primjer, za rečenicu $w=x^n$, $x=()$, $n>1$, vrijedi da su rečenice x^k, $k=1, 2, ..., n-1$, svojstveni prefiksi od w. Ako gramatiku proširimo uvođenjem novog početnog simbola S' i produkcije $S' \rightarrow S\#$, dobili bismo gramatiku tipa $\mathcal{LR}(0)$.

Definicija 6.6
Stavka beskontekstne gramatike $\mathcal{G}=(\mathcal{N},\mathcal{T},\mathcal{P},S)$ jest produkcija koja sadrži znak "$\cdot$", "$\cdot" \notin (\mathcal{N} \cup \mathcal{T})$, bilo gdje u svojim alternativama, uključujući početak i kraj. Ako gramatika sadrži ε-produkciju, $A \rightarrow \varepsilon$, stavka je $A \rightarrow \cdot$.

Primjer 6.8
Neka je dana gramatika:

$$S' \rightarrow Sc \qquad S \rightarrow SA|A \qquad A \rightarrow aSb|ab$$

Ovo je također gramatika "sparenih zagrada", kao iz primjera 6.7, gdje je "c" oznaka kraja, "a" stoji umjesto "(", a "b" umjesto ")". Sve stavke ove gramatike su:

S' → ·Sc	S → ·SA	A → ·aSb
S' → S·c	S → S·A	A → a·Sb
S' → Sc·	S → SA·	A → aS·b
	S → ·A	A → aSb·
	S → A·	A → ·ab
		A → a·b
		A → ab·

Definicija 6.7
Držač (*handle*) desne rečenične forme γ, $\gamma=\delta\beta w$, beskontekstne gramatike $G=(\mathcal{N},\mathcal{T},\mathcal{P},\mathcal{S})$ jest podniz β ako je:

$$\mathcal{S} \underset{rm}{\overset{*}{\Rightarrow}} \delta A w \underset{rm}{\Rightarrow} \delta\beta w$$

Dakle, držač desne rečenične forme γ jest podniz β (alternativa za A) koji je bio unesen u posljednjem izravnom izvođenju zdesna. U tom je kontekstu važna položaj β unutar γ. Održivi prefiks (*viable*) desne rečenične forme γ jest bilo koji njezin prefiks koji se završava ne dalje desno od desnog kraja držača od γ.

Primjer 6.9
Analizirajmo desnu rečeničnu formu gramatike iz primjera 6.8:

$$S' \Rightarrow Sc \Rightarrow SAc \Rightarrow SaSbc$$

Dakle, SaSbc je desna rečenična forma. aSb je njezin držač. Održivi prefiksi su ε, S, Sa, SaS i SaSb.

Definicija 6.8
Reći ćemo da je $A\rightarrow\alpha\cdot\beta$ valjana stavka za održivi prefiks γ ako postoji krajnje desno izvođenje:

$$\mathcal{S} \underset{rm}{\overset{*}{\Rightarrow}} \delta A w \underset{rm}{\Rightarrow} \delta\alpha\beta w$$

i $\delta\alpha=\gamma$. Poznavanje stavki koje su valjane za dani održivi prefiks može nam pomoći pri nalaženju krajnjeg izvođenja zdesna. Reći ćemo da je neka stavka potpuna ako je točka njezin posljednji znak. Ako je $A\rightarrow\alpha\cdot$ potpuna stavka valjana za γ, tada se pojavljuje tako da je $A\rightarrow\alpha$ moglo biti upotrijebljeno u posljednjem koraku i niz je δAw bio prethodna desna rečenična forma u izvođenju γw.

Primjer 6.10
Razmotrimo gramatiku iz primjera 6.8 i desnu rečeničnu formu abc. Budući da je:

$$S' \underset{rm}{\overset{*}{\Rightarrow}} Ac \underset{rm}{\Rightarrow} abc$$

zaključujemo da je stavka A→ab· valjana za održivi prefiks ab. Također zaključujemo da je stavka A→a·b valjana za održivi prefiks a i da je A→·ab valjana za održivi prefiks ε. S obzirom da je A→ab· potpuna stavka, konačno zaključujemo da je Ac bila prethodna desna rečenična forma za abc.

Izračunavanje skupa valjanih stavki

Definicija $\mathcal{LR}(0)$ gramatika i postupak prihvaćanja jezika $\mathcal{L}(\mathcal{G})$ za $\mathcal{LR}(0)$ gramatiku $\mathcal{G}$ deterministčkim potisnim automatom (DPDA) u potpunosti je ovisno o postojanju skupa valjanih stavaka za svaki održivi prefiks γ. Pokazuje se da je za bilo koju beskontekstnu gramatiku $\mathcal{G}$ skup održivih prefiksa regularan skup i da se može prihvatiti nedeterminističkim konačnim automatom (NFA) u kojem su stanja stavke za $\mathcal{G}$. Koristeći podskup konstrukcije tog automata dolazi se do potisnog automata u kojem stanja koja odgovaraju održivom prefiksu γ jesu skup valjanih stavki za γ.

Definicija 6.9
Nedeterministički konačni automat (NKA) koji prepoznaje održive prefikse gramatike $\mathcal{G}=(\mathcal{N},\mathcal{T},\mathcal{P},\mathcal{S})$ definiran je kao $\mathcal{M}=(Q,\mathcal{N}\cup\mathcal{T},\delta,q_0,Q)$, gdje je Q skup stavki za $\mathcal{G}$ plus stanje q_0 koje nije stavka, a funkcija prijelaza δ definirana je kao:

1) $\delta(q_0, \varepsilon) = \{\mathcal{S} \rightarrow \cdot\alpha : \mathcal{S}\rightarrow\alpha$ je produkcija$\}$
2) $\delta(A\rightarrow\alpha\cdot B\beta, \varepsilon) = \{B \rightarrow \cdot\gamma : B\rightarrow\gamma$ je produkcija$\}$
3) $\delta(A\rightarrow\alpha\cdot X\beta, X) = \{A \rightarrow \alpha X\cdot\beta\}$

Pravilo (2) dopušta ekspanziju simbola B koji se nalazi u sredini, desno od točke, a pravilo (3) dopušta pomicanje točke preko bilo kojeg simbola gramatike X, ako je X idući ulazni simbol.

Primjer 6.11
Neka je dana gramatika:

S' → Sc S → SA|A A → aSb|ab

(v. primjer 6.8). Funkcija prijelaza nedeterminističkog konačnog automata koji prepoznaje održive prefikse dana je sljedećom tablicom (prijelaza):

	ε	S	A	a	b	c	
→ q_0	q_1						
q_1	q_4, q_7	q_2					S' → ·Sc
q_2						q_3	S' → S·c
⊗ q_3							S' → Sc·
q_4	q_4, q_7	q_5					S → ·SA
q_5	q_9, q_{13}		q_6				S → S·A
⊗ q_6							S → SA·
q_7	q_9, q_{13}		q_8				S → ·A
⊗ q_8							S → A·
q_9				q_{10}			A → ·aSb
q_{10}		q_{11}					A → a·Sb
q_{11}					q_{12}		A → aS·b
⊗ q_{12}							A → aSb·
q_{13}				q_{14}			A → ·ab
q_{14}					q_{15}		A → a·b
⊗ q_{15}							A → ab·

Stanja q_1 do q_{15} oznake su stavki danih u posljednjem stupcu.

Teorem 6.3
NKA $\mathcal{M}$ definiran u 6.9 ima svojstvo da $\delta(q_0,\gamma)$ sadrži $A\to\alpha\cdot\beta$ ako i samo ako je $A\to\alpha\cdot\beta$ valjana stavka za γ.

Definicija gramatike tipa $\mathcal{LR}(0)$

Poslije svih ovih uvodnih razmatranja vrijeme je da definiramo gramatiku tipa $\mathcal{LR}(0)$.

Definicija 6.10
Reći ćemo da je $\mathcal{G}$ gramatika tipa $\mathcal{LR}(0)$ ako su ispunjeni sljedeći uvjeti:

1) početni se simbol ne pojavljuje niti u jednoj alternativi (tj. na desnoj strani produkcija), i
2) za svaki održivi prefiks γ iz $\mathcal{G}$, kad god je $A\to\alpha\cdot$ potpuna valjana stavka za γ, tada ne postoji nijedna druga potpuna stavka niti bilo koja stavka s terminalom desno od točke koja bi bila valjana za γ.

Teorem 6.3 daje postupak izračunavanja skupa valjanih stavki za bilo koji održivi prefiks, odnosno kako treba pretvoriti NKA čija će stanja biti stavke deterministčkog konačnog automata (DKA). U DKA put od početnog stanja označenog s γ vodi do stanja koje je skup valjanih stavki za γ. Dakle, treba izgraditi DKA i provjeriti svako stanje da bi se vidjelo jesu li prekršeni uvjeti $\mathcal{LR}(0)$.

Primjer 6.12
Tablica prijelaza DKA izgrađenog iz NKA primjera 6.11, iz kojeg su izbačeni "mrtvo" stanje (prazan skup stavaka) i prijelazi u mrtvo stanje, dana je s:

	S	A	a	b	c
→ I_0	I_1	I_2	I_3		
I_1		I_5	I_3		I_4
⊗ I_2					
I_3	I_6	I_2	I_3	I_7	
⊗ I_4					
⊗ I_5					
I_6		I_5	I_3	I_8	
⊗ I_7					
⊗ I_8					

gdje su:

I_0
S'→ ·Sc
S → ·SA
S → ·A
A → ·aSb
A → ·ab

I_1
S'→ S·c
S → S·A
A → ·aSb
A → ·ab

I_2
S → A·

I_3
A → a·Sb
A → a·b
S → ·SA
S → ·A
A → ·aSb
A → ·ab

I_4
S'→ Sc·

I_5
S'→ Sc·

I_6
A → aS·b
S → S·A
A → ·aSb
A → ·ab

I_7
A → ab·

I_8
A → aSb·

Stanja koja imaju više od jedne stavke nemaju nijednu potpunu stavku i, dakako, početni simbol ne pojavljuje se niti u jednoj alternativi. Zbog toga je gramatika iz primjera 6.11 tipa $\mathcal{LR}(0)$.

$\mathcal{LR}(0)$ gramatike i deterministički stogovni automati

Gramatika tipa $\mathcal{LR}(0)$ jednoznačna je pa generira deterministički beskontekstni jezik i, obrnuto, svaki deterministički beskontekstni jezik sa svojstvom prefiksa ima gramatiku tipa $\mathcal{LR}(0)$. Osim toga, gramatike tipa $\mathcal{LR}(0)$ su beskontekstne, pa se za njih može konstruirati deterministički stogovni automat (DSA).

Prije nego opišemo postupak izgradnje DSA, promijenimo prvobitnu definiciju konfiguracije stogovnog automata (q, w, α^R) u $[q, \alpha, w]$ (umjesto okruglih sada su uglate zagrade, a sadržaj stoga, α, sada ima vrh na svom desnom kraju).

Da bismo simulirali krajnje izvođenje zdesna u $\mathcal{LR}(0)$ gramatici ne samo da imamo održivi prefiks u stogu, već se iznad svakog simbola nalazi stanje DSA koje prepoznaje održive prefikse. Ako je održivi prefiks

$$X_1 \ldots X_k$$

u stogu, tada je potpuni sadržaj stoga

$$s_0X_1s_1 \ldots X_ks_k$$

gdje je

$$s_i = \delta(X_1 \ldots X_i)$$

i δ je funkcija prijelaza DSA. Stanje na vrhu stoga, s_k, osigurava valjane stavke za $X_1 \ldots X_k$.

Ako s_k sadrži $A \to \alpha\cdot$, tada je $A \to \alpha\cdot$ valjana stavka za $X_1 \ldots X_k$. Dakle, α je sufiks od $X_1 \ldots X_k$, na primjer $\alpha = X_{i+1} \ldots X_k$ (primijetiti da α može biti ε, tada je $i=k$). Međutim, postoji neki w tako da je $X_1 \ldots X_k w$ krajnja desna rečenična forma i postoji izvođenje:

$$S \overset{*}{\underset{rm}{\Rightarrow}} X_1 \ldots X_i A w \underset{rm}{\Rightarrow} X_1 \ldots X_k w$$

Prema tome, da bi se dobila desna rečenična forma koja je prethodila krajnjoj desnoj rečeničnoj formi $X_1 \ldots X_k w$, α je u izvođenju zdesna reducirano u A, zamjenjujući $X_{i+1} \ldots X_k$ na vrhu stoga s A. Ili, ako to prikažemo preko niza pomaka:

$$[q, s_0X_1 \ldots s_{k-1}X_ks_k, w] \vdash^* [q, s_0X_1 \ldots s_{i-1}X_is_iAs, w]$$

gdje je $s = \delta(s_i, A)$. Ako je gramatika tipa $\mathcal{LR}(0)$, s_k sadrži samo $A \to \alpha\cdot$, osim ako je $\alpha = \varepsilon$, kada s_k može sadržavati nekoliko nepotpunih stavki. Međutim, iz definicije $\mathcal{LR}(0)$, nijedna od tih stavki nema terminal desno od točke, ili je potpuna. Dakle, za bilo koji y tako da je $X_1 \ldots X_k y$ desna rečenična forma, $X_1 \ldots X_i A y$ mora biti prethodna desna rečenična forma, tako da je reduciranje α u A ispravno, bez obzira na tekući ulaz.

Razmotrimo sada slučaj kad s_k sadrži samo nepotpune stavke. Tada desna rečenična forma koja prethodi rečeničnoj formi $X_1 \ldots X_k w$ nije mogla biti dobivena reduciranjem sufiksa od $X_1 \ldots X_k$ u neku varijablu, inače bi postojala potpuna valjana stavka za $X_1 \ldots X_k$. Tada mora postojati držač koji se završava desno od X_k u $X_1 \ldots X_k w$, i to tako da je $X_1 \ldots X_k$ održivi prefiks. Dakle, jedina moguća akcija DSA jest premjestiti sljedeći ulazni znak u stog, to jest

$[q, s_0X_1 \ldots s_{k-1}X_k s_k, ay] \vdash [q, s_0X_1 \ldots s_{k-1}X_k s_k a t, y]$

gdje je $t=\delta(s_k,a)$. Ako je t neprazan skup stavki, $X_1 \ldots X_k a$ jest održivi prefiks. Ako je t prazan, može se dokazati da ne postoji prethodna desna rečenična forma od $X_1 \ldots X_k a y$, tako da ulazni niz znakova nije rečenica jezika kojeg gramatika generira i DSA "umire" umjesto da načini pomak.

Primjer 6.13
Pogledajmo DKA kao u primjeru 6.12 u kojem su stanja označena s 0, 1, ..., 8 umjesto I_0, I_1, ..., I_8, redom, i neka je ulazni niz aababbc. Tada će DSA s funkcijom prijelaza jednakom funkciji prijelaza DKA, načiniti niz pomaka:

	stog	preostali ulaz	opis
1)	0	aababbc	inicijalizacija
2)	0a3	ababbc	premještanje
3)	0a3a3	babbc	premještanje
4)	0a3a3b7	abbc	premještanje
5)	0a3A2	abbc	reduciranje s A→ab
6)	0a3S6	abbc	reduciranje s S→A
7)	0a3S6a3	bbc	premještanje
8)	0a3S63b7	bc	premještanje
9)	0a3S6A5	bc	reduciranje s A→ab
10)	0a3S6	bc	reduciranje s S→SA
11)	0a3S6b8	c	premještanje
12)	0A2	c	reduciranje s A→aSb
13)	0S1	c	reduciranje s S→A
14)	0S1c4	-	premještanje
15)	-	-	prihvaćanje

Na primjer, u redu (1) stanje 0 je na vrhu stoga. Ne postoji potpuna stavka u skupu I_0, pa se premješta tekući znak na vrh stoga. Prvi ulazni simbol je a i postoji prijelaz iz I_0 u I_3 označen s a. Sadržaj stoga u redu (2) je 0a3, itd. U redu (9), na primjer, stanje 5 je na vrhu stoga. I_5 sadrži potpunu stavku S→SA. Izbacuje se SA iz stoga, pa ostaje 0a3. Potom se dodaje S u stog. Postoji prijelaz iz I_3 u I_6 označen sa S, pa dopisujemo stanje 6, što rezultira sadržajem stoga 0a3S6 u redu (10)itd.

Gramatike tipa $\mathcal{LR}$(k)

Neformalno, kaže se da je gramatika tipa $\mathcal{LR}$(k) ako se u danom krajnjem izvođenju zdesna:

$$S = \alpha_0 \underset{rm}{\Rightarrow} \alpha_1 \underset{rm}{\Rightarrow} \ldots \underset{rm}{\Rightarrow} \alpha_m = z$$

u svakoj rečeničnoj formi α_i može izolirati neki $\beta \in (\mathcal{N} \cup \mathcal{T})^*$ i jednoznačno utvrditi, pretražujući α_i slijeva nadesno, ali najviše k znakova za β, kojem je neterminalu A niz β alternativa.

Pretpostavimo da je $\alpha_{i-1}=\alpha A w$ i $\alpha_i=\alpha\beta w$, gdje je $\alpha \in (\mathcal{N} \cup \mathcal{T})^*$, $w \in \mathcal{T}^*$ i $A \rightarrow \beta$ produkcija u $\mathcal{G}$. Dalje, pretpostavimo da je $\alpha\beta = x_1 x_2 \ldots x_r$, $X_i \in \mathcal{N} \cup \mathcal{T}$. Ako je gramatika $\mathcal{G}$ tipa $\mathcal{LR}(k)$, tada možemo biti sigurni da vrijedi sljedeće:

1) Znajući $X_1 X_2 \ldots X_j$ i prvih k znakova od $X_{j+1} \ldots X_r \omega$ sigurni smo da kraj od β nije dosegnut sve dok nije $j=r$.
2) Znajući $\alpha\beta$ i najviše k prvih znakova od ω možemo uvijek odrediti β i neterminal A iz kojeg je β izvedeno.
3) Kada je $\alpha_{i-1}=S$ može se sa sigurnošću dojaviti da je ulazni niz prihvaćen (u jeziku je $\mathcal{L}(\mathcal{G})$).

Primijetiti da prolazeći kroz niz α_m, α_{m-1}, ..., α_0 počinjemo promatranjem samo $FIRST_k(\alpha_m)=FIRST_k(\omega)$. U svakom koraku promatramo samo k ili manje početnih znakova.

Definicija 6.11
Neka je dana beskontekstna gramatika $\mathcal{G}=(\mathcal{N},\mathcal{T},\mathcal{P},\mathcal{S})$. Definira se proširena gramatika izvedena iz $\mathcal{G}$, gramatika $\mathcal{G}'$ dobivena proširenjem gramatike $\mathcal{G}$, $\mathcal{G}'=(\mathcal{N} \cup \{\mathcal{S}'\}, \mathcal{T}, \mathcal{P} \cup \{\mathcal{S}' \rightarrow \mathcal{S}\}, \mathcal{S}')$. Dakle, gramatika $\mathcal{G}'$ dobivena je iz $\mathcal{G}$ dodavanjem novog početnog simbola, $\mathcal{S}'$, i produkcije $\mathcal{S}' \rightarrow \mathcal{S}$.

Definicija 6.12
Neka je dana beskontekstna gramatika $\mathcal{G}=(\mathcal{N},\mathcal{T},\mathcal{P},\mathcal{S})$ i njezina proširena gramatika, $\mathcal{G}'=(\mathcal{N}', \mathcal{T}, \mathcal{P}', \mathcal{S}')$. Kaže se da je gramatika $\mathcal{G}$ tipa $\mathcal{LR}(k)$, $k \geq 0$, ako tri uvjeta:

1) $\mathcal{S}' \underset{\mathcal{G}'rm}{\overset{*}{\Rightarrow}} \alpha A w \underset{\mathcal{G}'rm}{\Rightarrow} \alpha\beta w$
2) $\mathcal{S}' \underset{\mathcal{G}'rm}{\overset{*}{\Rightarrow}} \gamma B x \underset{\mathcal{G}'rm}{\Rightarrow} \alpha\beta y$
3) $FIRST_k(w)=FIRST_k(y)$

impliciraju $\alpha A y=\gamma B x$ (tj. $\alpha=\gamma$, $A=B$ i $y=x$).

Intuitivno, prethodna definicija kaže da su $\alpha\beta w$ i $\alpha\beta y$ desne rečenične forme proširene gramatike s $FIRST_k(w)=FIRST_k(y)$, i ako je $A \rightarrow \beta$ posljednje upotrijebljena produkcija u izvođenju $\alpha\beta w$ u krajnjem izvođenju zdesna, tada $A \rightarrow \beta$ mora također biti upotrijebljeno u reduciranju $\alpha\beta y$ na

αAy u sintaksnoj analizi zdesna. Budući da A može izvesti β neovisno o w, $\mathcal{LR}(k)$ uvjet kaže da postoji dostatna informacija u $FIRST_k(w)$ na temelju koje se može zaključiti da je $\alpha\beta$ bilo izvedeno iz αA. Dakle, nikada ne može postojati dilema kako treba reducirati bilo koju desnu rečeničnu formu proširene gramatike. Osim toga, s $\mathcal{LR}(k)$ gramatikom uvijek se zna može li se prihvatiti ulazni niz ili se nastavlja analiza. Ako se početni simbol ne pojavljuje na desnoj strani niti u jednoj produkciji, može se alternativno definirati gramatiku $\mathcal{LR}(k)$ kao gramatiku $\mathcal{G}=(\mathcal{N},\mathcal{T},\mathcal{P},\mathcal{S})$ u kojoj tri uvjeta:

1) $\mathcal{S} \overset{*}{\underset{\mathcal{G}'rm}{\Rightarrow}} \alpha Aw \underset{\mathcal{G}'rm}{\Rightarrow} \alpha\beta w$

2) $\mathcal{S} \overset{*}{\underset{\mathcal{G}'rm}{\Rightarrow}} \gamma Bx \underset{\mathcal{G}'rm}{\Rightarrow} \alpha\beta y$

3) $FIRST_k(w)=FIRST_k(y)$

impliciraju $\alpha Ay=\gamma Bx$.

Primjer 6.14
Pogledajmo gramatiku $\mathcal{G}$ s produkcijama

$S \rightarrow SaSb \mid \varepsilon$

Proširena gramatika je

$S' \rightarrow S$
$S \rightarrow SaSb \mid \varepsilon$

Iz

1) $S' \overset{*}{\underset{\mathcal{G}'rm}{\Rightarrow}} SaSaSbb \underset{\mathcal{G}'rm}{\Rightarrow} SaSabb$

2) $S' \overset{*}{\underset{\mathcal{G}'rm}{\Rightarrow}} SaSbab \underset{\mathcal{G}'rm}{\Rightarrow} SaSaSbbab$

3) $FIRST_1(b)=FIRST_1(b)=b$

slijedi da je $SaSb=Sab$, što ne implicira da je $S'a=S$.

Primjer 6.15
Pogledajmo gramatiku $\mathcal{G}$ s produkcijama

$S \rightarrow Sa \mid a$

Ako zanemarimo činjenicu da se početni simbol pojavljuje na desnoj strani produkcije, tj. ako gledamo drugu alternativu, $\mathcal{G}$ bi mogla biti $\mathcal{LR}(0)$ gramatika. Međutim, primjenjujući definiciju, $\mathcal{G}$ nije $\mathcal{LR}(0)$, budući da uvjeti

1) $S' \overset{0}{\underset{\mathcal{G}'rm}{\Rightarrow}} S' \underset{\mathcal{G}'rm}{\Rightarrow} S$

2) $S' \underset{\mathcal{G}'rm}{\Rightarrow} S \underset{\mathcal{G}'rm}{\Rightarrow} Sa$

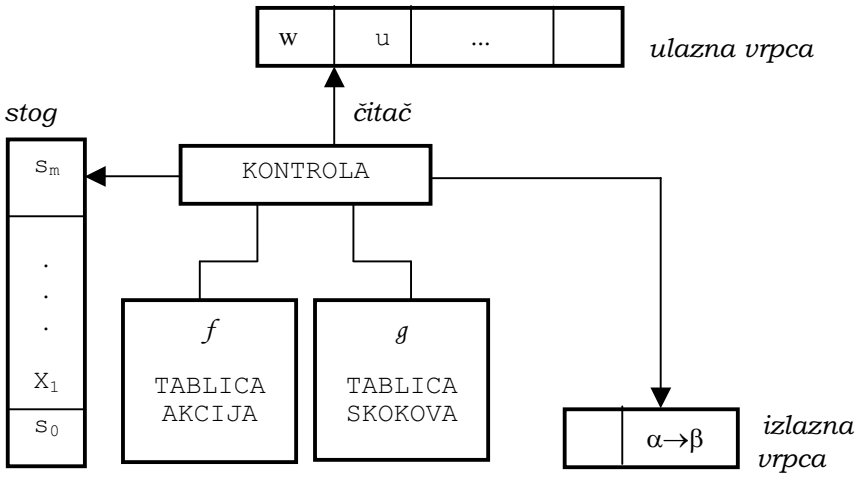
3) $FIRST_0(\varepsilon)=FIRST_0(a)=\varepsilon$

ne impliciraju da je $S'a=S$.

Sintaksna analiza $\mathcal{LR}(1)$ jezika

U praksi se često promatra klasa $\mathcal{LR}(1)$ jezika. U teoriji je formalnih jezika poznato da svi deterministički beskontekstni jezici imaju gramatiku tipa $\mathcal{LR}(1)$. Ta je klasa gramatika posebno važna u oblikovanju prevodilaca većine jezika za programiranje.

Općenito se model SA jezika tipa $\mathcal{LR}(1)$ može predočiti kao što je prikazano na sljedećem crtežu:



Sl. 6.2 – *Model sintaksne analize $\mathcal{LR}(1)$ jezika.*

Vidimo da je to pretvornik sastavljen od ulazne vrpce, čitača, kontrole, stoga, tablice akcija, tablice skokova i izlazne vrpce.

Tablica akcija f i tablica skokova g, diktiraju ponašanje postupka sintaksne analize. Sadržaj stoga u bilo kojem trenutku jest niz:

$$s_0X_1s_1X_2s_2 \ \ldots \ X_ms_m$$

gdje su s_i simboli *stanja*, a X_i simboli iz gramatike.

Ukratko, postupak SA $\mathcal{LR}(1)$ jezika ponaša se na sljedeći način. Najprije pretpostavimo da stog sadrži niz $s_0X_1s_1X_2s_2...X_ms_m$. Kontrola određuje s_m, stanje na vrhu potisne liste, i a, tekući ulazni znak. Potom ispituje $f(s_m,a)$, ulaz za s_m i a u tablici akcija. Taj ulaz može prouzročiti jednu od četiriju različitih vrsta akcija, ovisno o vrijednosti funkcije f:

1) $f\ (s_m,a) = \{$ shift $s\}$

   Premješta se ulazni simbol a na vrh potisne liste, potom stanje s. Nakon toga potisna lista sadrži:

   $$s_0X_1s_1X_2s_2\ \ldots\ X_ms_mas$$

2) $f\ (s_m,a) = \{$ reduciranje produkcijom $A\rightarrow Y_1Y_2\ldots Y_r\ \}$

   Izuzima se $2r$ simbola iz potisne liste, pa je s_{m-r} sada na vrhu. Zatim se konzultira tablica skokova, tj. $g(s_{m-r},A)$. Ako je vrijednost te funkcije jednaka s, upisuje se As na vrh potisne liste. Produkcija $A\rightarrow Y_1Y_2...\ Y_r$ ispisuje se na izlaznu vrpcu. Čitač se pomiče za jedno mjesto udesno. Poslije toga sadržaj potisne liste će biti:

   $$s_0X_1s_1\ \ldots\ X_{m-r}s_{m-r}As.$$

3) $f(s_m,a) = \{$ accept $\}$

   Prekida se postupak i prihvaća ulazni niz kao sintaksno korektan.

4) Ako je $f(s_m,a) = \{$ error $\}$ ili ako se ne može dosegnuti nijedno od danih premještanja, dojavljuje se pogrešaka i prekida postupak.

Postupak sintaksne analize $\mathcal{LR}(1)$ jezika počinje nekom početnom konfiguracijom u kojoj ulazna vrpca sadrži niz koji treba analizirati, sa znakom $\$$ na svom kraju. Čitač je ispod prvog pravokutnika slijeva na ulaznoj vrpci. Potisna lista sadrži samo početno stanje s_0. Izlazna je vrpca prazna. Postupak SA potom obavljai niz premještanja, sve dok ne prihvati ulazni niz, ako je u jeziku, ili dok ne dojavi pogrešku.

Primjer 6.16
Razmotrimo gramatiku G s produkcijama:

(1) $S \rightarrow S(S)$ (2) $S \rightarrow \varepsilon$

Tablice sintaksne analize i skokova su:

	(	)	$
0	red 2	err	red 2
1	sh 2	err	acc
2	red 2	red 2	err
3	sh 4	sh 5	err
4	red 2	red 2	err
5	red 1	err	red 1
6	sh 4	sh 7	err
7	red 1	red 1	err

	S
0	1
1	err
2	3
3	err
4	6
5	err
6	err
7	err

Stanje 0 je početno. Ovdje sh i stoji umjesto "premjesti stanje i u potisnu listu", red i označuje da treba reducirati produkciju numeriranu s i, acc stoji umjesto accept i err umjesto error.

Pogledajmo kako će postupak sintaksne analize analizirati ulazni niz w=(()). Vrh stoga je na desnoj strani. Pozicija čitača označena je sa "↑". Postupak će obaviti premještanja:

	Stog	Ulazni niz	Izlaz
početak	0	(())	
		↑	
1.	0S1	(())$	S→ε
		↑	
2.	0S1(2	())$	
		↑	
3.	0S1(2S3	())$	S→ε
		↑	
4.	0S1(2S3(4	))$	
		↑	
5.	0S1(2S3(4S6	))$	S→ε
		↑	
6.	0S1(2S3(4S6)7	)$	
		↑	
7.	0S1(2S3	)$	S→S(S)
		↑	
8.	0S1(2S3)5	$	
		↑	
9.	0S1	$	S→S(S)
		↑	accept

Niz izvođenja je:

S ⇒ S(S) ⇒ S(S(S)) ⇒ S(S()) ⇒ S(()) ⇒ (())

Objasnimo detaljnije premještanja u postupku sintaksne analize:

1. Početno je stanje 0. Ulazni je znak (. Akcija je f[0,(] = red 2 što znači da treba upotrijebiti produkciju S→ε za reduciranje. Najprije se izbacuje 2r znakova s vrha stoga (r=0, pa se ne izbacuje nijedan znak). Na vrhu stoga je stanje 0. Funkcija g(0,S) je 1. Na vrh stoga premješta se najprije S pa stanje 1. Na izlaznu se vrpcu upisuje produkcija kojom se reduciralo.
2. Najprije se na vrh stoga premješta tekući znak (, potom stanje 2 (jer je f[1,(] = sh 2). Čitač se pomiče za jedno mjesto.
3. Reducira se niz na vrhu stoga (kao u prvom koraku).
4. Najprije se na vrh stoga premješta znak (, potom stanje 4. Čitač se pomiče za jedno mjesto udesno.
5. Opet se reducira prazan niz u S i unosi stanje 6 na vrh stoga.
6. Najprije se na vrh stoga premješta tekući znak (, potom stanje 7. Čitač se pomiče za jedno mjesto udesno.
7. Sada se niz na vrhu stoga može reducirati u S, koristeći prvu produkciju S→S(S). Najprije se izbacuje 2r znakova s vrha stoga (r=4, pa će biti izbačeno 8 znakova). Poslije toga na vrhu je stanje 2, pa je g(2,S)=3. Dakle, najprije se na vrh stoga upisuje S pa stanje 3. Na izlaznu se vrpcu ispisuje produkcija S→S(S).
8. Na vrh se stoga premješta tekući znak), potom stanje 5. Čitač se pomiče za jedno mjesto udesno i sada je na oznaci kraja ulaznog niza, znaku $.
9. Opet se, kao u koraku 7, reducira niz na vrhu stoga koristeći prvu produkciju. Poslije toga na vrhu je stoga stanje 0, pa je vrijednost funkcije g za ulaz (0,S) jednaka 1. Dakle, na vrh se stoga upisuje S1. Na izlaznu se vrpcu ispisuje S→S(S). Tekuće je stanje 1, tekući znak $, pa je f(1,$)=acc, tj. ulazni niz je prihvaćen.

6.3 GRAMATIKE S RELACIJOM PRIORITETA

Postoji još nekoliko deterministčkih uzlaznih postupaka sintaksne analize koji rade nad podskupovima $\mathcal{LR}$ jezika. Posebno je zapažena sintaksna analiza jezika sa slabim prioritetom i s prioritetom operatora koje ćemo ovdje opisati.

Definicija 6.13
Neka je $\mathcal{G}=(\mathcal{N},\mathcal{T},\mathcal{P},\mathcal{S})$ svojstvena gramatika (v. def. 3.5) i $ novi znak koji nije u $\mathcal{N}\cup\mathcal{T}$. Definiraju se tri relacije prioriteta $\lessdot$, $\doteq$ i $\gtrdot$ na $\mathcal{N}\cup\mathcal{T}\cup\{\$\}$ na sljedeći način:

1) $X \lessdot Y$ ako postoji produkcija $A \rightarrow \alpha XB\beta$ u P tako da $B^+ \Rightarrow Y\gamma$ za neki γ.
2) $X \doteq Y$ ako postoji produkcija $A \rightarrow \alpha XY\beta$.
3) $X \gtrdot Y$, Y je terminal, ako postoji produkcija $A \rightarrow \alpha XZ\beta$ tako da $B^* \Rightarrow \gamma X$ i $Z^* \Rightarrow Y\delta$ za neke γ i δ.
4) $\$ \lessdot Y$ ako $S^+ \Rightarrow Ya$ za neki a.
5) $X \gtrdot \$$ ako $S^+ \Rightarrow \alpha X$ za neki a.

Definicija 6.14
Gramatika $\mathcal{G}=(\mathcal{N}, \mathcal{T}, \mathcal{P}, \mathcal{S})$ jest gramatika sa slabim prioritetom ako:

1) $\mathcal{G}$ je svojstvena.
2) Definirana je najmanje jedna relacija prioriteta između svakog para simbola iz $\mathcal{N} \cup \mathcal{T} \cup \{\$\}$.
3) Ne postoje dvije produkcije koje imaju jednaku desnu stranu.

Za gramatike $\mathcal{G}$ sa slabim prioritetom može se definirati parser (prepoznavatelj) kao automat koji se sastoji od ulazne vrpce, stoga i matrice relacija prioriteta za $\mathcal{G}$. Ulazni niz $x=a_1a_2\ldots a_n$ koji se analizira nalazi se na ulaznoj vrpci i ima $\$$ na svom kraju. Inicijalno se čitač nalazi na prvom znaku i stog sadrži $\$$. Postupak će SA tada načiniti niz pomaka sve dok ne prihvati ulazni niz ili ne dojavi pogrešku.

Pomaci će biti određeni relacijom prioriteta između znaka na vrhu stoga i tekućeg znaka. Pretpostavimo da stog sadrži niz $X_1X_2\ldots X_m$, gdje je X_m na vrhu, te da je a_i tekući znak ($a_{n+1}=\$$). Idući pomak određen je vrijednošću matrice relacije prioriteta između X_m i a_i:

1) Ako je $X_m \lessdot a_i$ ili $X_m \doteq a_i$, premješta se a_i na vrh stoga.
2) Ako je $X_1X_2\ldots X_m=\$S$ i $a_i=\$$, prekida se postupak SA i prihvaća ulazni niz.
3) Ako je $X_m \gtrdot a_i$, pretražuje se stog dok se ne pronađe simbol X_k tako da je $X_{k-1} \lessdot X_k \doteq X_{k+1} \doteq \ldots \doteq X_m$. Tada se pronalazi produkcija $A \rightarrow X_kX_{k+1}\ldots X_m$ i zamjenjuje $X_kX_{k+1}X_m$ na stogu s A. Iz definicije gramatike s jednostavnim prioritetom može postojati najviše jedna produkcija s takvim svojstvom. Ako skup produkcija P ne sadrži takvu produkciju ili ako za neki $j \leq m$, $X_j \doteq X_{j+1} \doteq \ldots \doteq X_m$ ne postoji relacija prioriteta između X_{j-1} i X_j, dojavljuje se pogreška i prekida analiza.
4) Ako ne postoji relacija prioriteta između X_m i a_i, dojavljuje se pogreška i prekida analiza.

Svaka gramatika sa slabim prioritetom istodobno je tipa $\mathcal{LR}(1)$. No, klasa jezika generiranih gramatikama sa slabim prioritetom mali je podskup deterministčkih beskontekstnih jezika.

Definicija 6.15
Svojstvena gramatika $\mathcal{G}=(\mathcal{N}, \mathcal{T}, \mathcal{P}, \mathcal{S})$ jest gramatika s jakim prioritetom ako:

1) Operacija prioriteta $\dot{>}$ prema uniji $\dot{<}$ i $\cong$.
2) Ako su $A \to \alpha X \beta$ i $B \to \beta$ produkcije, ne postoji nijedna operacija prioriteta $X \dot{<} B$ niti $X \cong B$ (tj. X se ne smije pojaviti ispred B ni u jednoj rečeničnoj formi).

Parser gramatike s jakim prioritetom operira slično parseru gramatika sa slabim prioritetom. Pretpostavimo da stog sadrži $X_1 \ldots X_m$ i da je a_i tekući ulazni simbol. Da bi načinio sljedeći pomak, parser izvodi relaciju prioriteta između X_m i a_i:

1) Ako je $X_m \dot{<} a_i$ ili $X_m \cong a_i$, premješta se a_i na vrh stoga.
2) Ako je $X_1 \ldots X_m = \$S$ i $a_i = \$$, prekida se postupak SA i prihvaća ulazni niz.
3) Ako je $X_m \dot{>} a_i$, pretražuje se lista produkcija gramatike dok se ne pronađe najdulja desna strana jednaka sufiksu od $X_1 \ldots X_m$. Ako je produkcija $A \to X_k \ldots X_m$ s takvim svojstvom, parser zamjenjuje $X_1 \ldots X_m$ na vrhu s A.
4) Ako nijedan od prethodnih uvjeta ne stoji, dojavljuje se pogreška i prekida analiza.

Definicija 6.16
Operatorska gramatika jest gramatika bez praznih produkcija, s alternativama koje sadrže bar jedan terminal, a ne sadrže dva uzastopna neterminala.

Primjer 6.17
Gramatika s produkcijama

$$S \to SOS \mid (S) \mid -S \mid x \qquad O \to + \mid - \mid * \mid /$$

nije operatorska gramatika, jer u alternativi SOS nije zadovoljen uvjet da ne smiju biti dva uzastopna neterminala (tu su čak tri). No, zamijenimo li O sa svim njegovim alternativama, dobit ćemo operatorsku gramatiku:

$$S \to S+S \mid S-S \mid S*S \mid S/S \mid (S) \mid -S \mid x$$

Za operatorsku gramatiku $\mathcal{G}=(\mathcal{N}, \mathcal{T}, \mathcal{P}, \mathcal{S})$ definirane su relacije prioriteta operatora $\dot{<}$, $\dot{>}$ i $\cong$ nad $\mathcal{T} \cup \{\$\}$ na sljedeći način:

1) $a \dot{<} b$ ako je $A \to \alpha a B \beta$ produkcija tako da $B \overset{+}{\Rightarrow} \gamma b \delta$ za neki γ iz $\mathcal{N} \cup \{\varepsilon\}$.
2) $a \cong b$ ako je $A \to \alpha a \beta b \gamma$ produkcija tako da je β iz $\mathcal{N} \cup \{\varepsilon\}$.
3) $a \dot{>} b$ ako je $A \to \alpha B b \beta$ produkcija tako da $B \overset{+}{\Rightarrow} \gamma a \delta$ za neki δ iz $\mathcal{N} \cup \{\varepsilon\}$.
4) $\$ \dot{<} b$ ako $\mathcal{S} \overset{+}{\Rightarrow} \alpha b \beta$ za neki α iz $\mathcal{N} \cup \{\varepsilon\}$.
5) $a \dot{>} \$$ ako $\mathcal{S} \overset{+}{\Rightarrow} \alpha a \beta$ za neki β iz $\mathcal{N} \cup \{\varepsilon\}$.

Definicija 6.17
Gramatika $G=(N, T, P, S)$ jest gramatika s prioritetom operatora ako je operatorska i ako postoji najviše jedna relacija prioriteta operatora između svakog para simbola iz $T \cup \{\$\}$.

Primjer 6.18
Gramatika s produkcijama:

$$E \rightarrow E+T \mid T \qquad T \rightarrow T*F \mid F \qquad F \rightarrow (E) \mid a$$

jest gramatika s prioritetom operatora. U sljedećoj su tablici prikazane relacije prioriteta operatora između pojedinih parova simbola:

	$	(	+	*	a	)
$		$<\cdot$	$<\cdot$	$<\cdot$	$<\cdot$	
(		$<\cdot$	$<\cdot$	$<\cdot$	$<\cdot$	$\cong$
+	$\cdot>$	$<\cdot$	$\cdot>$	$<\cdot$	$<\cdot$	$\cdot>$
*	$\cdot>$	$<\cdot$	$\cdot>$	$\cdot>$	$<\cdot$	$\cdot>$
a	$\cdot>$		$\cdot>$	$\cdot>$		$\cdot>$
)	$\cdot>$		$\cdot>$	$\cdot>$		$\cdot>$

Na primjer, iz produkcije $F \rightarrow (E)$ slijedi da je $(\cong)$, ili iz $E \rightarrow E+T$ i $T^+ \Rightarrow T*F$ i $T^+ \Rightarrow (E)$, slijedi da je $+ <\cdot *$ i $+ <\cdot ($, itd.

Definicija 6.18
Neka je $G=(N, T, P, S)$ operatorska gramatika. Definira se skeletna gramatika $G_S=(\{S\}, T, P', S)$ za G koja se sastoji od produkcija

$$S \rightarrow X_1 \ldots X_m$$

tako da postoji produkcija $A \rightarrow Y_1 \ldots Y_m$ u P i, za $1 \le i \le m$, vrijedi:

(1) $X_i = Y_i$ ako je $Y_i \in T$
(2) $X_i = S$ ako je $Y_i \in N$

i nije dopuštena produkcija $S \rightarrow S$ u P'.

Dakle, skeletna gramatika G_S gramatike G s prioritetom operatora jest gramatika dobivena zamjenom svih neterminala u produkcijama početnim simbolom i, potom, izbacivanjem produkcija $S \rightarrow S$, ako postoje. Općenito će vrijediti uvjet da je $L(G) \subseteq L(G_S)$, tj. $L(G_S)$ može sadržavati rečenice koje nisu u $L(G)$.

Primjer 6.19
Skeletna gramatika G_S gramatike G iz primjera 6.18 jest:

$$E \rightarrow E+E \mid E \qquad E \rightarrow E*E \mid E \qquad E \rightarrow (E) \mid a$$

odnosno, poslije izbacivanja produkcija E→E:

E → E+E | E*E | (E) | a

Primijetiti da je $\mathcal{L}(\mathcal{G}) = \mathcal{L}(\mathcal{G}_S)$.

Algoritam 6.2

Parser gramatika s prioritetom operatora.

Ulaz: Gramatika s prioritetom operatora, $\mathcal{G} = (\mathcal{N}, \mathcal{T}, \mathcal{P}, S)$.

Izlaz: funkcije akcije i skokova, f i g, za $\mathcal{G}_S$.

Postupak:

Neka je β jednako S ili ε.
(1) $f(a\beta, b) =$ shift ako je $a \lessdot b$ ili $a \doteq b$.
(2) $f(a\beta, b) =$ reduce ako je $a \gtrdot b$.
(3) $f(\$S, \$) =$ accept.
(4) $f(\alpha, w) =$ error, inače.
(5) $g(a\beta b\gamma, w) = i$ ako je
  (a) β jednako S ili ε
  (b) $a \lessdot b$
  (c) Vrijedi relacija $\doteq$ između prva dva terminala od γ, ako postoje, i
  (d) $S \rightarrow \beta b\gamma$ je produkcija označena s i u $\mathcal{G}_S$.
(6) $g(\alpha, w) =$ error, inače.

Parser gramatike s prioritetom operatora radi slično parseru gramatika sa slabim prioritetom, ali najčešće se realizira za skeletnu gramatiku. Skeletna gramatika može biti i višeznačna (a može biti i ekvivalentna osnovnoj gramatici, kao što je to bio slučaj u prethodnom primjeru), ali relacije prioriteta operatora uvijek će jednoznačno odrediti moguću analizu ulaznog niza. Štoviše, ustrojbom parsera skeletne gramatike, uvijek se mogu koristiti originalne produkcije osnovne gramatike pri dojavi pogreške, ako to želimo. Ono što treba posebno istaknuti jest da je jezik $\mathcal{L}(\mathcal{S}_S)$ najčešće nadskup jezika $\mathcal{L}(\mathcal{S})$, pa je postupak sintaksne analize uporabom skeletne gramatike općenitiji.

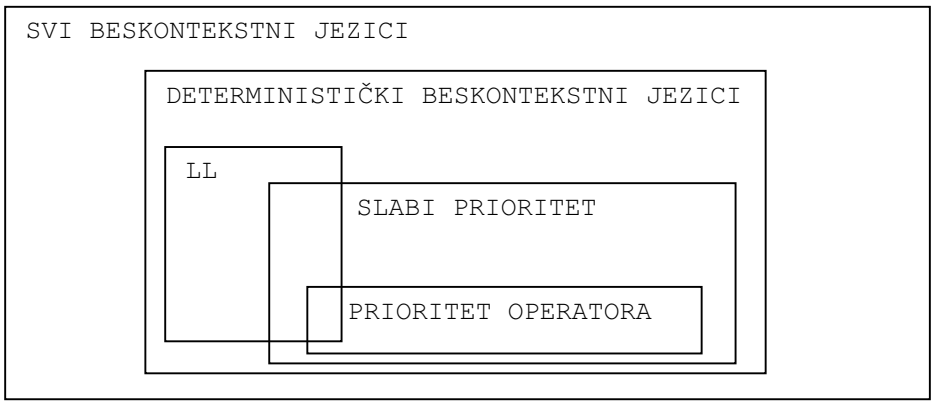
Primjer 6.20
Parser gramatike jednostavnih aritmetičkih izraza, koristeći tablicu relacija prioriteta operatora iz primjera 6.19, analizirat će ulazni niz a+a*a kao što slijedi:

	stog	preostali ulaz	opis
1)	$	a+a*a$	premještanje a
2)	$a	+a*a$	reduciranje s E→a
3)	$E	+a*a$	premještanje +
4)	$E+	a*a$	premještanje a
5)	$E+a	*a$	reduciranje s E→a
6)	$E+E	*a$	premještanje *
7)	$E+E*	a$	premještanje a
8)	$E+E*a	$	reduciranje s E→a
9)	$E+E*E	$	reduciranje s E→E*E
10)	$E+E	$	reduciranje s E→E+E
11)	$E	$	prihvaćanje

6.4 EFIKASNOST JEDNOPROLAZNIH POSTUPAKA SINTAKSNE ANALIZE

Za sve jednoprolazne postupke sintaksne analize vrijedi općeniti zaključak da su efikasniji od povratnih (*backtrack*) postupaka sintaksne analize. Potrebno im je $k=c_1 n$ operacija i koriste $c_2 n$ memorijskog prostora u analizi niza duljine n, gdje su c_1 i c_2 konstante. Vrijednost koeficijenta k ovisi o konkretnom ustrojenju postupka, odnosno o jeziku i gramatici koja ga generira. Samo u posebnim slučajevima moguće je statističkim metodama izračunati najvjerojatnije uporabe određenih procedura i poslije toga preurediti program tako da se tekući simbol pokuša najprije izjednačiti sa simbolom čija je pojava najvjerojatnija. Time bi se moglo utjecati na smanjenje koeficijenta k, pa bi postupak bio brži.

Osnovni nedostatak jednoprolaznih postupaka sintaksne analize jest ograničena primjenjivost, nad relativno malom klasom beskontekstnih jezika, sl. 6.3.



Sl. 6.3 – *Hijerarhija beskontekstnih jezika.*

Pitanja i zadaci

1) Pokažite da gramatika s produkcijama

$S \rightarrow aAaB \mid bAbB$
$A \rightarrow a \mid ab$
$B \rightarrow aB \mid a$

jest $LL(3)$ ali nije $LL(2)$.

2) Napišite algoritam za izračunavanje $FOLLOW_k(A)$ za neterminal A.

3) Pokažite da je gramatika G s produkcijama

$S \rightarrow aaSbb \mid a \mid \varepsilon$

tipa $LL(2)$. Nađite ekvivalentnu gramatiku gramatici G koja će biti tipa $LL(1)$.

4) Pragram 1-predikatne analize dan je u prilogu 5. Proučite ga unosom ulaznih gramatika iz primjera 6.2 i 6.6.

5) Definirajte tablicu akcija i skokova za gramatiku G s produkcijama

$S \rightarrow SaSb \mid \varepsilon$

Potom provjerite je li niz $aabb$ u jeziku $L(G)$.

7.

JEZICI SA SVOJSTVIMA

treba putovati, treba otići
tamo gdje ćeš biti netko
tamo gdje nitko ništa ne zna

govoriti neki drugi jezik
slušati neke druge glasove
kušati neko drugo voće
živjeti u drugim legendama

treba putovati, treba otići
izgubiti se na kraju svijeta
i sam se vratiti

tražiti druge prijatelje
lutati nekim drugim ulicama
spavati u drugim krevetima
u rukama nepoznatima

treba putovati, treba otići
na putove zemaljske
i staze morske

promijeniti sat i dan
udisati druge mirise
opijati se drugim vinima
otkriti neke druge ljubavi

treba putovati, treba otići
reći zbogom samome sebi
onome što se bilo

i vratiti se možda
kao iscrpljeni novi bogataš
da se umre ili uskrsne
tu otkud se otišlo,
otišlo, otišlo

treba putovati
il faut voyager
(georges moustaki/
zdravko dovedan)

U ovom je završnom poglavlju opisan jedan jednoprolazni postupak sintaksne analize primjenljiv na svim beskontekstnim i kontekstnim jezicima, te posebno pogodan za ustrojbu na računalima. Postupak sintaksne analize počiva na definiciji jezika za koji se treba ustrojiti. Temelj definicije jezika sa svojstvima bit će generator, a ne gramatika.

U prvom dijelu poglavlja definiran je generator jezika sa svojstvima. Potom slijedi opis postupka sintaksne analize jezika sa svojstvima i, na kraju, dani su rezultati usporedbe s ranije opisanim postupcima sintaksne analize beskontekstnih jezika.

7.1 DEFINICIJA JEZIKA SA SVOJSTVIMA

Generator u kojem je kontrola konačnog stanja u potpunosti zadana funkcijom prijelaza generira regularne jezike. Drugim riječima, jezik je regularan ako sadrži rečenice:

$\omega=a_1 \ldots a_n \qquad \omega \in \Sigma^+$

za koje vrijedi:

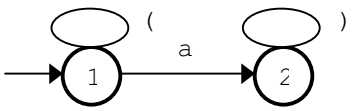
$q_1=\delta(q_0,a_1) \qquad q_2=\delta(q_1,a_2) \quad \ldots \quad q_k=\delta(q_j,a_n)$

odnosno

$q_k = \delta(\delta(\ldots,\delta(q_0,a_1),a_2),\ldots,a_n)$

gdje su q_1, $i=1$, ..., k, stanja iz Q, q_0 jest početno i q_k jedno od završnih stanja, $q_k \in F$.

Primjer 7.1
Generator zadan dijagramom prijelaza:



gdje je $q_0=1$, $F=\{2\}$, a skup stanja, rječnik i funkcija prijelaza vide se iz dijagrama, generira regularni jezik:

$$\mathcal{L} = \{(^m a)^n : m \geq 0,\ n \geq 0\}$$

Proširenjem značenja kontrole konačnog stanja generatora regularnih jezika dolazimo do definicije jezika sa svojstvima:

Definicija 7.1
Jezik sa svojstvima jest jezik čiji je generator zadan kao uređena osmorka:

$$\mathcal{J} = (Q, \Sigma, V, \delta, q_0, F, \alpha, M)$$

gdje su:

Q	konačan skup stanja (kontrole završnog stanja)
Σ	alfabet
V	rječnik, $V \subseteq \Sigma^+$
δ	funkcija prijelaza, definirana kao $\delta: Q \times V \rightarrow P(Q)$ gdje je $P(Q)$ particija od Q
q_0	početno stanje, $q_0 \in Q$
F	skup završnih stanja, $F \subseteq Q$
α	skup akcija pridruženih svakom paru (q_i, s_j), $q_i \in Q$, $s_j \in V$, za koji je definirana funkcija prijelaza
M	pomoćna memorija

Dakle, kontrola završnog stanja generatora jezika sa svojstvima podijeljena je na dva dijela: funkciju prijelaza i skup akcija. Funkcijom prijelaza δ (primijetiti da je definirana nad rječnikom) zadaju se sve moguće promjene stanja. Akcija je, općenito, postupak koji, ovisno o tekućem stanju q_i i informacijama dobivenih od pomoćne memorije, reducira broj mogućih prijelaza iz stanja q_i, zadanih funkcijom δ, u trenutačno ostvarive prijelaze. I ne samo to, akcija može promijeniti vrijednosti funkcije prijelaza. Ako je prijelaz iz stanja q_i u stanje q_j prijelazom s_k uvijek ostvariv, smatrat ćemo da je takvom prijelazu pridružena prazna akcija. Generator regularnih jezika primjer je u kojem su svi prijelazi uvijek ostvarivi. Otuda generator regularnih jezika nema potrebe za pomoćnom memorijom.

U ustrojbi generatora jezika sa svojstvima kontrolu konačnog stanja i dalje ćemo prikazivati dijagramom (tablicom) prijelaza, ali ćemo svakom prijelazu dodati kôd njemu pridružene akcije sa značenjem:

0 (ili izostavljeno)	- prazna akcija
1, 2, ...	- neprazna akcija

Kaže se da regularne gramatike generiraju regularne jezike, beskontekstne gramatike beskontekstne jezike, itd. Međutim, to je oprečno s tvrdnjom da je svaka regularna gramatika istodobno beskontekstna, svaka beskontekstna gramatika da je istodobno kontekstna i, konačno, da je svaka kontekstna gramatika istodobno i gramatika bez ograničenja, jer iz toga slijedi da je svaki regularni jezik istodobno beskontekstan, itd.

Iz definicije jezika sa svojstvima slijedi da će takvi jezici u pravilu imati neka kontekstna svojstva, pa se postavlja pitanje: Može li beskontekstna gramatika generirati jezik sa svojstvima? Isto pitanje vrijedi i za regularne gramatike.

Odgovor je potvrdan! To će biti u onim slučajevima kad beskontekstna gramatika u svojim produkcijama sadrži ekplicitno napisane "prave" rekurzije, odnosno, ako se za gramatiku bilo kojeg tipa može napisati niz izvođenja:

$X \overset{+}{\Rightarrow} \alpha X \beta \qquad \alpha, \beta \in (\mathcal{N} \cup \mathcal{T})^*$

Tada pomoćna memorija prepoznavača takvog jezika ima strukturu stoga.

Primjer 7.2
Beskontekstna gramatika $\mathcal{G}_1$ s produkcijama:

$E \rightarrow E+E \mid E*E \mid (E) \mid a$

generira jezik jednostavnih aritmetičkih izraza koji moraju zadovoljavati kontekstno svojstvo: broj otvorenih zagrada jednak je broju zatvorenih zagrada. Isto vrijedi i za ekvivalentnu linearnu (tipa 3) gramatiku $\mathcal{G}_2$:

$E \rightarrow aB \mid (C$
$B \rightarrow *E \mid +E \mid \varepsilon$
$C \rightarrow E)$

No, generator jezika $\mathcal{L}(\mathcal{G}_1)$, odnosno $\mathcal{L}(\mathcal{G}_2)$, imat će tablicu prijelaza:

	(	)	+	*	a	@
1	1				2	
2		2	1	1		1

gdje @ označuje kraj generiranja, i tablicu akcija:

	(	)	+	*	a	@
1	1	0	0	0	0	0
2	0	2	0	0	0	3

odnosno, ako tablicu prijelaza i akcija spojimo u jednu u kojoj će akcije biti prikazane kao eksponenti (prazna je akcija, 0, izostavljena):

	(	)	+	*	a	@
1	1^1				2	
2		2^2	1	1		1^3

Ako je Tp tablica prijelaza, Ta tablica akcija, R izlazni (generirani) niz, B brojač prijelaza sa "(", akcije su:

```
1: B := B+')'; Tp[2,2]=2
2: B := copy (B, 2, 255); IF B='' THEN Tp[2,2]=0
3: R := R +B
```

Generator napisan kao program u Turbo Pascalu :

```
PROGRAM Generator_artimetickih_izraza;

USES    Crt;

CONST   Mq = 2; Ms = 6;
        Tp : ARRAY [1..Mq, 1..Ms] OF byte = (
               { (  )  +  *  a  @ }
         { 1 } ( 1, 0, 0, 0, 2, 0 ),
         { 2 } ( 0, 0, 1, 1, 0, 1 ) );

        Ta : ARRAY [1..Mq, 1..Ms] OF byte = (
               { (  )  +  *  a  @ }
         { 1 } ( 1, 0, 0, 0, 0, 0 ),
         { 2 } ( 0, 2, 0, 0, 0, 3 ) );

        s  : STRING = '()+*a@';

VAR     B, R : STRING;        q, k : byte;
        d    : STRING;        Kraj : boolean;
        i    : byte;

PROCEDURE a_1; BEGIN B := B +')';  Tp [2, 2] := 2 END;

PROCEDURE a_2; BEGIN
  B := copy (B, 2, 255); q := Tp[q, k];
  IF B='' THEN Tp[2, 2] := 0
  END;

PROCEDURE a_3; BEGIN  R := R +B; Kraj := True END;


BEGIN
  ClrScr; Randomize;
  B := ''; q := 1; R := ''; Tp[2, 2] := 0; Kraj := False;
  REPEAT
    d := '';
    FOR i := 1 TO Ms DO
      IF Tp[q, i] <> 0 THEN
        d := d +chr (ord('0') +i);
    k := ord (d[random (length (d)) +1]) -ord('0');
    CASE Ta[q, k] OF
      1: BEGIN R := R +s[k]; a_1; q := Tp[q, k] END;
      2: BEGIN R := R +s[k]; a_2                END;
      3: a_3;
      ELSE BEGIN
        R := R +s[k]; q := Tp[q, k]
        END
      END
    UNTIL Kraj;
  WriteLN (Gen, R); Readln
END.
```

Evo nekoliko primjera generiranih izraza:

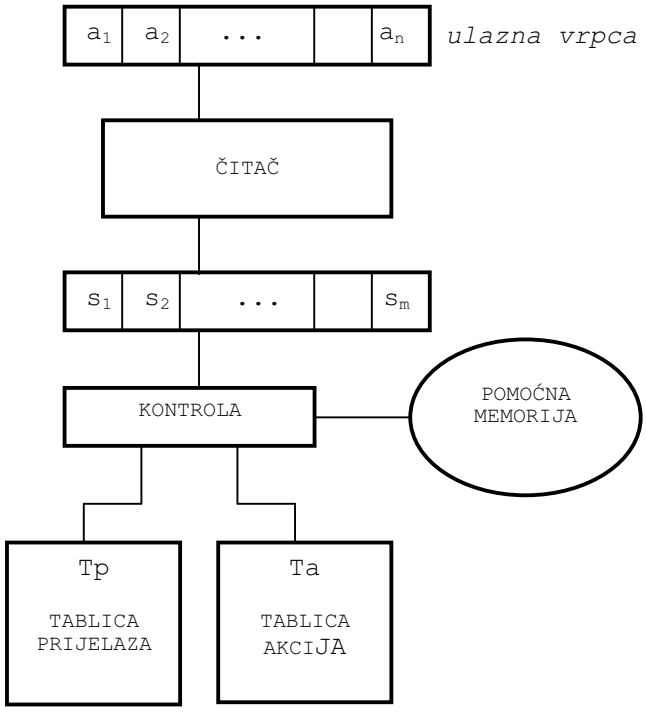
```
((a*a*a+a*(a)))    (a*a*(a*a+((((a))))))   (((a*(((a)))))))
(a)                a*a                     (a+a)
a                  (a+a*((a*a)))           (((((a)))))
(a*(a+a))          (a*(a*(a+a)))           (a+a)*(a+a)
```

7.2 PREPOZNAVAČ JEZIKA SA SVOJSTVIMA

Kod generatora jezika sa svojstvima akcijom je za svako stanje dijagrama (tablice) prijelaza bio određen skup ostvarivih prijelaza, što je u svakom trenutku bilo uvjetovano informacijama dobivenim iz pomoćne memorije.

U slučaju prepoznavača jezika sa svojstvima postavlja se pitanje: "Postoji li za simbol $s_j \in V$ ostvariv prijelaz iz tekućeg stanja q_i u neko stanje q_k?". Prvi dio odgovora na ovo pitanje dat će nam funkcija (tablica) prijelaza. Ako je $g_k=\delta(q_i,s_j)$ definirano, prijelaz je moguć, a akcija pridružena paru (q_i,s_j) odredit će je li ostvariv.

Općenito se struktura prepoznavača jezika sa svojstvima može prikazati kao na sljedećem crtežu:



Sl. 7.1 - *Model prepoznavača jezika sa svojstvima.*

Pomoćna memorija

Zamislit ćemo da pomoćnu memoriju čine, primitivne i strukturirane varijable, svih tipova, kao što je to, na primjer, u Turbo Pascalu. Inicijalno će te varijable sadržavati određene vrijednosti.

Čitač

Čitač je jednostavni pretvoritelj (program leksičke analize) koji prihvaća niz ulaznih znakova, kojima je na kraju dodan znak ¨@¨, i prevodi ih u niz simbola. Najčešće će rječnik biti jednak alfabetu, pa će skup simbola biti jednak skupu znakova alfabeta. Svakom simbolu pridužen je jedinstveni cjelobrojni kôd, od `1` do `k`, ako rječnik sadrži k simbola, te `k+1` za znak ¨@¨.

Sintaksna analiza

Ako je `C` kôd učitanog simbola, `S` tekuće stanje kontrole konačnog stanja i `Tp` tablica prijelaza, mogući prijelaz `Ss` zadan je s:

```
Ss := Tp [S, C]
```

a akcija kodom na mjestu `Ta[S,C]`, gdje je `Ta` tablica akcija.

Tada se kontrola završnog stanja prepoznavatelja jezika sa svojstvima (postupak sintaksne analize) može prikazati sljedećim dijelom programa:

```
REPEAT
  Ucitaj_Sim; Ss := Tp [S, C];
  IF (C=0) OR (Ss=0) THEN BEGIN
    WriteLN ('* Sintaksna pogreška'); Pogreska := TRUE
    END
  ELSE
   CASE Ta [S, C] OF
      0 : { Prazna akcija } S := Ss;
      1 : { Akcija broj 1 };
      2 : { Akcija broj 2 };
            { ... }
      {N : (* Akcija broj N *) }
      END
  UNTIL Kraj OR Pogreska;
```

Najprije će u posebnoj proceduri biti inicijalizirane varijable programa, tablica prijelaza i tablica akcija. Procedura `Ucitaj_Sim` učitat će tekući simbol i vratiti njegov kôd, `C`. Ako je kôd različit od `0` i ako je definirano naredno stanje, izvršit će se odgovarajuća akcija. Akcija može biti prazna, tada se bezuvjetno prelazi u naredno stanje. Neprazna se akcija izvodi u posebnoj proceduri.

Postupak se sintaksne analize nastavlja sve do dosezanja kraja ulaznog niza i njegova prihvaćanja ili se prekida ako je učitan simbol koji nije u rječniku ili je pronađena sintaksna pogreška.

Primjer 7.3
U dodatku je knjige dan primjer prepoznavača meta-jezika BNF čiju smo gramatiku definirali na kraju poglavlja posvećenog gramatikama. Ovdje dajemo primjer prepoznavača jezika jednostavnih artimetičkih izraza iz primjera 7.2.

```
PROGRAM Prepoznavac_aritmetickih_izraza;
USES    Crt;
CONST   Mq = 2; Ms = 6;
        Tp : ARRAY [1..Mq, 1..Ms] OF byte = (
               { (  )  +  *  a  @ }
         { 1 } ( 1, 0, 0, 0, 2, 0 ),
         { 2 } ( 0, 2, 1, 1, 0, 1 ) );

        Ta : ARRAY [1..Mq, 1..Ms] OF byte = (
               { (  )  +  *  a  @ }
         { 1 } ( 1, 0, 0, 0, 0, 0 ),
         { 2 } ( 0, 2, 0, 0, 0, 3 ) );

CONST   V  : STRING = '()+*a@';
VAR     B, Niz         : STRING;    S, C, i : byte;
        Kraj, Pogreska : boolean;

PROCEDURE Inicijalizacija; BEGIN
  ClrScr; Randomize; B := ''; S := 1; Tp[2, 2] := 0;
  Kraj := False; Pogreska := False; i := 0  END;

PROCEDURE Citac; BEGIN
  WriteLN ('Unesite ulazni niz:'); ReadLN(Niz);
  Niz := Niz+'@'  END;

PROCEDURE a_1; BEGIN B := B +')';  Tp [2, 2] := 2  END;

PROCEDURE a_2; BEGIN B := copy (B, 2, 255);  S := Tp[S, C];
  IF B='' THEN Tp[2, 2] := 0  END;

PROCEDURE a_3; BEGIN Pogreska := B <> '';  Kraj := True  END;

BEGIN
  Inicijalizacija; Citac;
  REPEAT
    i := i +1;C := pos (Niz[i], V);
    IF (C=0) OR (Tp[S,C]=0) THEN Pogreska := True
    ELSE CASE Ta[S, C] OF
      0: S := Tp[S, C];        1: BEGIN a_1; S := Tp[S, C] END;
      2: a_2;                  3: a_3
     END
    UNTIL Kraj OR Pogreska;
  IF Pogreska THEN BEGIN WriteLN (^G, ' ' :i-1, chr(24));
    Write   ('POGREŠKA, niz nije u jeziku!')
    END
  ELSE
    WriteLN (^M, 'NIZ JE U JEZIKU!');
  Readln
END.
```

U ovom je primjeru rječnik jednak alfabetu, pa se ulazni niz odmah učita do kraja i doda mu se simbol "@". Kodovi simbola rječnika određeni su inicijalizacijom varijable V, tj. kôd im je pozicija u tom nizu.

Izvođenje dijagrama prijelaza iz produkcija BNF-a

Vidimo da je program sintaksne analize jezika sa svojstvima prilično jednostavan. Najveći je problem ispravno zadati tablicu prijelaza i akcija, te napisati procedure akcija. Za neke jednostavnije, "školske" primjere beskontekstnih jezika, kao što je bio jezik jednostavnih aritmetičkih izraza, to možda i nije teško. Ali, nažalost, u praksi se ne susrećemo s trivijalnim jezicima, pa je izvođenje dijagrama prijelaza prilično kompleksan problem. Međutim, posebno je interesantno pitanje: Kako iz dane beskontekstne gramatike G izvesti dijagrame prijelaza - generator jezika $\mathcal{L}(\mathcal{G})$? Jamačno, ne možemo ponuditi konačno rješenje, ali smo mu se pokušali približiti.

Kao ni u slučaju izvođenja gramatike zadanog jezika, ne postoje pravila koja bi nam pomogla da izvedemo odgovarajuće dijagrame prijelaza jezika za koji želimo riješiti problem sintaksne analize upravljan dijagramom prijelaza i akcija. Međutim, izvođenje dijagrama prijelaza može biti znatno olakšano ako nam je za dani jezik poznata bar jedna od ekvivalentnih gramatika. Polazeći od nje poslije niza transformacija može se doći do konačnog dijagrama.

Znamo da se jezik $\mathcal{L}$ generiran gramatikom $\mathcal{G}$ može općenito napisati kao:

$$\mathcal{L}(\mathcal{G}) = \{\omega:\ \omega \in \mathcal{T}^*,\ \mathcal{S} \overset{+}{\Rightarrow} \omega\ \}$$

a to je skup rečenica ω iz $\mathcal{T}^*$ koje se mogu dobiti izvođenjem iz početnog simbola (rečenične forme) $\mathcal{S}$. Svaku rečeničnu formu izvedenu iz početnog simbola, tako da se uvijek zamjenjivao prvi neterminal slijeva, možemo napisati kao

$$\mathcal{S} \overset{+}{\Rightarrow} xA\alpha \qquad x \in \Sigma^*,\ A \in \mathcal{N},\ \alpha \in (\mathcal{T} \cup \mathcal{N})^*$$

Promatrajući sva moguća izvođenja rečeničnih formi, može se doći do početnog dijagrama prijelaza, a daljnjim postupkom njegove transformacije u ekvivalentni konačni dijagram prijelaza. Prije nego što definiramo pravila izvođenja i reduciranja dijagrama prijelaza, proširimo značenje dijagrama (funkcije) prijelaza generatora jezika sa svojstvima u:

$$\delta:\ Q \times (\mathcal{T} \cup \mathcal{N}) \rightarrow P(Q)$$

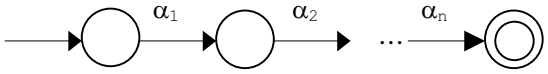
tj. proširili smo prijelaze simbolima iz skupa neterminala.

Neka je $\mathcal{G}=(\mathcal{N}, \mathcal{T}, \mathcal{P}, \mathcal{S})$ beskontekstna gramatika. Pravila izvođenja dijagrama prijelaza, ovisno o tipu produkcija gramatike $\mathcal{G}$, jesu sljedeća:

i1) Produkcija oblika:

$$A \rightarrow \alpha_1\alpha_2 \ldots \alpha_n \qquad \alpha_i \in (\mathcal{T}\cup\mathcal{N})^*$$

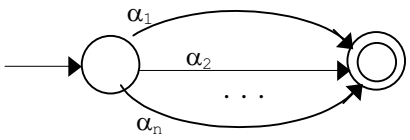
prevodi se u dijagram prijelaza:



i2) Produkcija oblika:

$$A \rightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n \qquad \alpha_i \in (\mathcal{T}\cup\mathcal{N})^*$$

prevodi se u dijagram prijelaza:

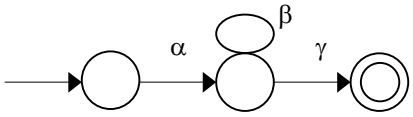


gdje svaki α_i ima dijagrame prijelaza prema pravilima i1 do i8.

i3) Produkcija oblika:

$$A \rightarrow \alpha\{\beta\}\gamma \qquad \alpha,\gamma \in (\mathcal{T}\cup\mathcal{N})^*, \quad \beta \in (\mathcal{T}\cup\mathcal{N})^+$$

prevodi se u dijagram prijelaza:



i4) Rekurzija zdesna:

$$A \rightarrow \alpha A \mid \beta \qquad \alpha \in (\mathcal{T}\cup\mathcal{N})^+, \ \beta \in (\mathcal{T}\cup\mathcal{N})^*$$

Iz konačnog niza izvođenja:

$$A \Rightarrow \alpha A \Rightarrow \alpha\alpha A \Rightarrow \ldots \Rightarrow \alpha^n A \Rightarrow \alpha^n\beta$$

i izravnog izvođenja:

$$A \Rightarrow \beta$$

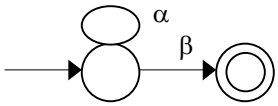
što se, prema definiciji praznog niza ε, može dalje pisati:

$$\beta \Rightarrow \varepsilon\beta = \alpha^0\beta$$

pa se rekurzija zdesna može napisati kao:

$$A \rightarrow \{\alpha\}\beta$$

a to je poseban slučaj primjene pravila (i3), pa je, konačno, dijagram prijelaza rekurzije zdesna:



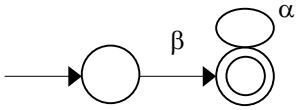
i5) Rekurzija slijeva:

$$A \rightarrow A\alpha \mid \beta \qquad \alpha \in (\mathcal{T} \cup \mathcal{N})^+, \quad \beta \in (\mathcal{T} \cup \mathcal{N})^*$$

Promatranjem konačnog niza izvođenja, kao i u slučaju rekurzije zdesna, na kraju bi se dobilo:

$$A \rightarrow \beta\{\alpha\}$$

što je, također, poseban slučaj primjene pravila (i3), pa je dijagram prijelaza rekurzije slijeva:



i6) Obostrana rekurzija:

$$A \rightarrow A\alpha A \mid \beta \qquad \alpha \in (\mathcal{T} \cup \mathcal{N})^+, \ \beta \in (\mathcal{T} \cup \mathcal{N})^*$$

Iz promatranja nekoliko nizova izvođenja:

$$A \Rightarrow \beta \qquad A \Rightarrow A\alpha A \Rightarrow \beta\alpha A \Rightarrow \beta\alpha\beta$$
$$A \overset{*}{\Rightarrow} \beta\alpha\beta\alpha\beta = \beta(\alpha\beta)^2 \qquad A \overset{*}{\Rightarrow} \beta(\alpha\beta)^3$$

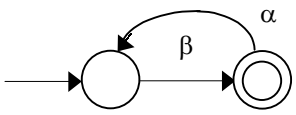
zaključujemo da se umjesto produkcije

$$A \rightarrow A\alpha A \mid \beta$$

može napisati:

$$A \rightarrow \beta\{\alpha\beta\}$$

pa je odgovarajući dijagram prijelaza:



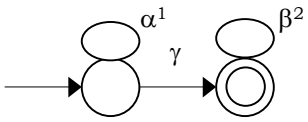
i7) Rekurzija:

$A \rightarrow \alpha A \beta \mid \gamma \qquad \alpha, \beta \in (\mathcal{T} \cup \mathcal{N})^+, \ \gamma \in (\mathcal{T} \cup \mathcal{N})^*$

Kao i u prethodna tri slučaja, promatranjem svih konačnih nizova izvođenja, na kraju bi se dobilo:

$A \rightarrow \alpha^n \gamma \beta^n$

pa je dijagram prijelaza prikazan s:



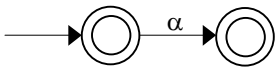
gdje su 1 i 2 kodovi akcija sa značenjem

1 - brojanje prijelaza sa α
2 - broj prijelaza sa β; mora biti jednak broju prijelaza sa α

i8) ε-produkcije:

$A \rightarrow \alpha \mid \varepsilon \qquad \alpha, \beta \in (\mathcal{T} \cup \mathcal{N})^+$

imaju dijagram prijelaza:



Eliminiranje neterminala

Krajnji je cilj dobiti dijagrame prijelaza koji neće sadržavati neterminale kao prijelaze. To znači da je potrebno eliminirati sve neterminale iz dijagrama supstituirajući ih njihovim dijagramima. Tako što je dopušteno, jer su dijagrami prijelaza izvedeni iz produkcija ekvivalentni produkcijama gramatike, pa je supstitucija neterminala u njima ekvivalentna supstituciji neterminala u odgovarajućoj rečeničnoj formi.

Neka je $\mathcal{G}=(\mathcal{N},\mathcal{T},\mathcal{P},\mathcal{S})$ beskontekstna gramatika. Pravila izvođenja konačnog dijagrama prijelaza iz produkcija gramatike $\mathcal{G}$, jesu sljedeća:

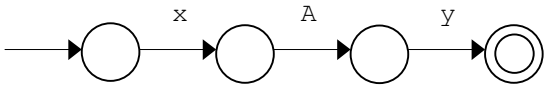
p1) Za svaku produkciju iz P izvede se ekvivalentni dijagram prijelaza prema pravilima (i1) do (i7). U njima se označe samo početno i završna stanja.

p2) Počevši s dijagramom prijelaza početnog simbola $\mathcal{S}$, supstituirati neterminale njihovim dijagramima prijelaza sve dok se ne dobije dijagram prijelaza koji će sadržavati samo simbole iz $\mathcal{T}$.

p3) Reducirati dobiveni dijagram prijelaza, ako je moguće, potom označiti stanja (od 1 nadalje).

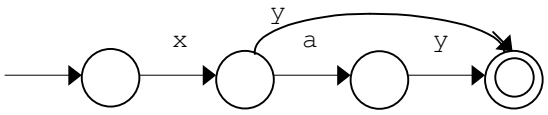
Pravila supstituiranja su sljedeća:

z1) Dijagram prijelaza:

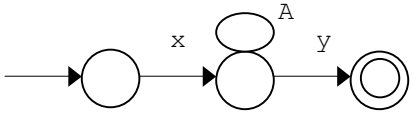


$x,y\in\mathcal{T}$, $A\in\mathcal{N}$

Ako je dijagram prijelaza za A dobiven primjenom pravila (i1) do (i6), poslije supstitucije na njegovom mjestu bit će odgovarajući dijagram prijelaza, a ako je A dobiven primjenom pravila (i7), poslije supstitucije A dobit će se dijagram:

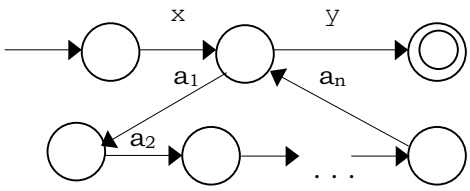


z2) Poslije supstitucije neterminala A, u dijagramu oblika:

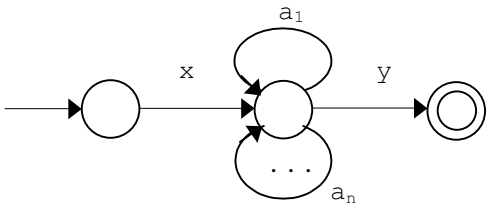


rezultirajući dijagram prijelaza, ovisno o dijagramu prijelaza za A, može biti:

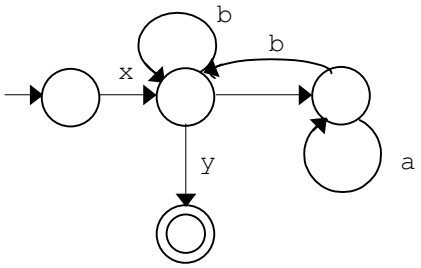
a) $A \rightarrow a_1a_2\ldots a_n$



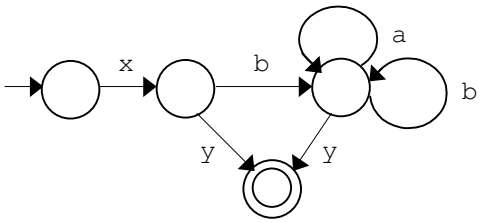
b) $A \rightarrow a_1 \mid a_2 \mid \ldots \mid a_n$



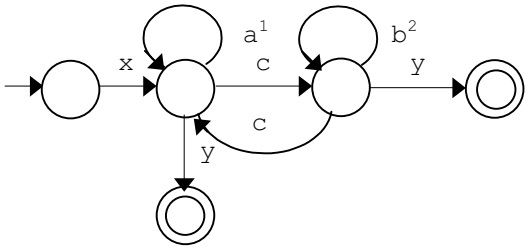
c) $A \rightarrow aA \mid b$



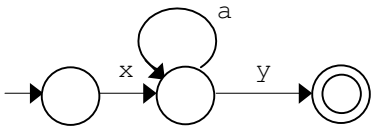
d) $A \rightarrow Aa \mid b$



e) $A \rightarrow aAb \mid c$



f) $A \rightarrow a \mid \varepsilon$



Do tih dijagrama dolazi se promatrajući ekvivalentne rečenične forme. Na primjer, u slučaju rekurzije zdesna, smjenjujući neterminal `A` u

```
a { A } y
```

s `{a}b`, dobije se:

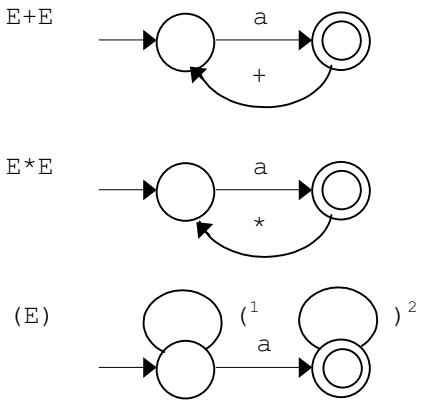
```
x {{ a } b } y
```

pa primjenjujući pravilo (i3) dvaput, dobije se dani dijagram prijelaza.

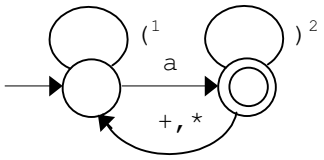
Primjer 7.4
Za gramatiku jednostavnih izraza iz primjera 7.2, s produkcijama:

$$E \rightarrow E+E \mid E*E \mid (E) \mid a$$

odgovarajući su dijagrami prijelaza:



pa je rezultirajući dijagram prijelaza:

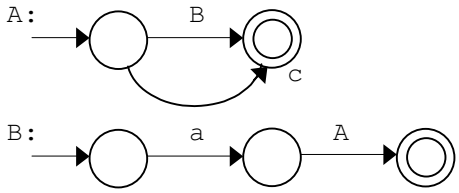


Ako se poslije supstitucije neterminala `B` u dijagramu prijelaza neterminala A novi dijagram sadrži prijelaz označen s `A` (što znači da je osnovna beskontekstna gramatika rekurzivna, ali je rekurzija implicitna), treba dijagram prevesti u ekvivalentnu rečeničnu formu, identificirati tip rečenične forme i, primjenjujući odgovarajuće pravilo (i4) do (i6), izvesti ekvivalentni dijagram prijelaza.

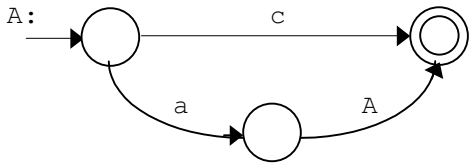
Primjer 7.5
Gramatika s produkcijama

```
A → B | c
B → aA
```

ima implicitnu rekurziju. Odgovarajući su dijagrami prijelaza:



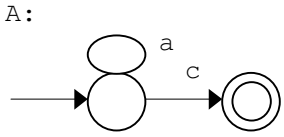
Poslije supstitucije neterminala B u dijagramu za A, imamo:



što je ekvivalentno produkciji

```
A → aA | c
```

pa je konačno (rekurzija zdesna):



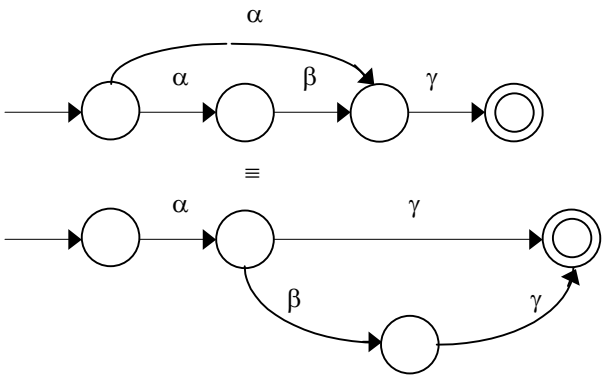
Reduciranje dijagrama prijelaza

Primjenom pravila za izvođenje dijagrama prijelaza, te eliminiranjem neterminala, često će se dobiveni dijagrami ili rezultirajući dijagram moći pojednostavniti i prevesti u ekvivalentan dijagram prijelaza. Na primjer, ako postoji prijelaz s dva jednaka simbola iz jednakog stanja (nedeterminizam) u isto naredno stanje. Pravila reduciranja su sljedeća:

r1)

r2)

r3)



7.3 USPOREDBA POSTUPAKA SINTAKSNE ANALIZE

Na kraju, usporedimo postupak sintaksne analize jezika sa svojsvima s višeprolaznim povratnim silaznim i ulaznim postupkom, te predikatnom sintaksnom analizom. Analizirat će se rečenice jezika jednostavnih aritmetičkih izraza definiranog gramatikom $\mathcal{G}$ s produkcijama:

$\mathcal{G}$: E → E+E | E*E | (E) | a

Ova je gramatika istodobno ulaz uzlazne (bottom-up) SA. Za primjenu silazne (top-down) i 1-predikatne SA potrebno je transformirati gramatiku $\mathcal{G}$ u gramatiku $\mathcal{G}_1$, koja neće sadržati rekurzije slijeva, i ekvivalentnu gramatiku $\mathcal{G}_2$ koja će biti tipa $\mathcal{LL}(1)$:

$\mathcal{G}_1$: E → T+E | T
T → F*T | F
F → (E) | a

$\mathcal{G}_2$: E → TA
A → +TA | ε
T → FB
B → *FB | ε
F → (E) | a

Konačno, za postupak SA upravljan tablicama prijelaza i akcija potrebno je definirati te tablice, što je bilo učinjeno u primjeru 7.3 i 7.4.

Kompletni programi – realizacija navedenih postupaka – napisani su u Turbo Pascalu i dani su u dodatku. U sljedećoj tablici dan je broj koraka koji je bio potreban za svaki od navedenih postupaka u sintaksnoj analizi nekoliko rečenica jezika $\mathcal{L}(\mathcal{G})$.

Ulazni niz	top-down	bottom-up	1-predikatna	tablice
a	32	3	7	2
a+a	47	7	13	4
a*a	56	7	11	4
a*a+a	53	11	17	6
a*(a+a)	330	14	24	8
(a+a)*(a+a)	528	21	37	12
(a+(a+(a+a)))	6,326	24	46	14

Očigledno je postupak SA upravljan tablicama prijelaza i akcija najefikasniji. Možda začuđuju rezultati postupka bottom-up. S obzirom na to da se radi o nedeterminističkom postupku, ne bi se očekivalo da će biti efikasniji od 1-predikatne SA. Međutim, poslije detaljnije analize može se zaključiti da je uzrok tome ulazna gramatika koja je omogućavala uvijek jednoznačnu redukciju pri izgradnji stabla sintaksne analize pa je u ovom slučaju općenito nedeterministički postupak radio deterministički. Pripadao bi klasi $\mathcal{LR}(1)$ SA, a poznato je da su takvi postupci efikasniji od $\mathcal{LL}(1)$ postupaka.

Neefikasnost nedeterminističkih postupaka SA posebno dolazi do izražaja u analizi rečenice koja sadrži sintaksnu pogrešku. Opet smo iste postupke usporedili u analizi nekoliko nizova koji ne mogu biti prihvaćeni kao rečenice jezika $\mathcal{L}(\mathcal{G})$. Evo rezultata:

Ulazni niz	top-down	bottom-up	1-predikatna	tablice
a+	67	10	8	3
a+b	67	14	8	3
((a+a)	1,397	36	24	7
a+a+a+	143	86	20	7
(())	629	10	9	3

Opet je prednost na strani postupka upravljanog tablicama prijelaza i akcija. Sada dolazi do izražaja nedeterminizam postupka "bottom-up".

Prednost determinističkih postupaka nije samo u bržem otkrivanju sintaksne pogreške, već i u mogućnosti njezinog lociranja. Nedeterministički postupci mogu otkriti sintaksnu pogrešku tek poslije iscrpljenja svih alternativa, bez preciznog lociranja pogreške.

Pitanja i zadaci

1) Definirajte tablicu prijelaza i akcija za sljedeće jezike:

   a) Parne priorodne brojeve
   b) Parne brojeve djeljive s 3, 4, 6 i 9 (svaki jezik posebno)

2) Definirajte tablicu prijelaza i akcija gramatike s produkcijama:

```
S → AB:BC| B
A → x| y
B → ND
C → ;S| ε
N → 1| 2| 3| 4| 5| 6| 7| 8| 9
D → N| DD| 0| ε
```

3) Napišite program sintaksne analize upravljan tablicom prijelaza i akcija za jezik definiran gramatikom s produkcijama:

```
I → IOI| (I)| B
O → +| -| *| /
B → 1| 2| 3| 4| 5| 6| 7| 8| 9
```

Literatura

AHO, V. Alfred; ULLMAN, D. Jeffrey:
The Theory of Parsing, Translation, and Compiling, vol. I: *Parsing*, Prentice-Hall, 1972.

AHO, V. Alfred; ULLMAN, D. Jeffrey:
Principles of Compiler Design, Addison-Wesley Publishing Company, 1979.

AHO; SETHI; ULLMAN:
Compilers: Principles, Techiques, and Tools, Addison-Wesley Publishing Company, 1986.

BACKHOUSE, C. R.:
Sintax of Programming Languages: Theory and Practice, Prentice-Hall, 1979.

BERRY, E. R.:
Programming Language Translation, Ellis Horwood Limited, 1981.

DENNING, J. P.; DENNIS, B. J.; QUALITZ, E. J.:
Machines, Languages, and Computation, Prentice-Hall, 1978.

DOVEDAN, Zdravko:
Jedan model sintaktičke analize jezika za programiranje, disertacija, Zagreb, Filozofski fakultet, 1992.

DOVEDAN, Zdravko:
Pascal i programiranje (1), Zagreb, don, 1995.

DOVEDAN, Zdravko:
SIntaktička analiza jezika sa svojstvima, Ljubljana, Informatica 82/3, 1983.

GOOS, G.; HARTMANIS, J., editors:
Compiler Construction, An Advanced Course, Springer-Verlag, 1976.

GRUNE, D.:
Parsing Techniques – A Practical Guide, Ellis-Horwood, 1990.

HOPCROFT, E. J.; ULLMAN, D. J.:
Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, 1979.

KALUŽNIN, A. L.:
Što je matematička logika, Zagreb, Školska knjiga 1975.

KUREPA, Svetozar:
Uvod u matematiku, Zagreb, Tehnička knjiga, 1970.

TOMITA, M., editor:
Current Issues in Parsing Technology, Kluwer Academic Publishers, 1991.

WAITE, M. W.; GOOS, G.:
Compiler Construction, Springer-Verlag, 1984.

WIRTH, N.:
Algorithms + Data Structures = Programs, Prentice-Hall, 1976.

YEH, T. R., editor:
Applied Computation Theory: Analysis, Design, Modeling, Prentice-Hall, 1976.

PRILOZI

Ovdje su dani programi – realizacija nekoliko postupaka sintaksne analize u Turbo Pascalu: sintaksne analize jezika BNF, silazne sintaksne analize, uzlazne sintaksne analize, CYK sintaksne analize i predikatne sintaksne analize. Sva dodatna objašnjenja dana su u komentarima programa. Programe možete dobiti ako se javite na e-mail adresu:

zdovedan@hotmail.com

1. Sintaksna analiza jezika BNF

Kao što je pokazano u primjeru 2.13, formalizam pisanja produkcija u BNF-u takodjer je jezik. U ovom je prilogu dan primjer sintaksne analize tog jezika primjenom tablice prijelaza i akcija, opisane u poglavlju 7. Pravila pisanja (unosa) produkcija su sljedeća:

- neterminali su velika slova engleskog alfabeta
- terminali su mala slova engleskog alfabeta, brojke i ostali znakovi
- umjesto simbola → pišu se dva znaka -> bez razmaka
- alternative su odvojene znakom | (Alt Gr W)
- prazna produkcija je znak ε (Alt pa 238 na brojčanoj tastaturi)

Na početku se unosi ime gramatike, prema pravilima pisanja imena datoteke u MS-DOSu. Ime može sadržavati točku i ekstenziju. Na primjer, pravilno su napisana imena: IZRAZ.GRM, TOP-01, BOOLEAN.TXT itd. Na kraju se upiše @ kao oznaka kraja unosa. Evo jednoga primjera unosa gramatike:

```
E -> T + E | T
T -> T * F | F
F -> (E) | a
```

Datoteka koja sadrži produkcije gramatike koristi se kao ulaz u programu silazne i uzlazne sintaksne analize i u predikatnoj sintaksnoj analizi.

```
PROGRAM Meta_BNF;  {Sintaksna analiza jezika BNF}
  USES Crt;
  TYPE  vs = 'A'..'Z'; mat = ARRAY [1..5, 1..6] OF integer;
  CONST       { N   T   ->  |   ε   @  }
    Tp: mat = ((2,  0,  0,  0,  0,  1),
               (0,  0,  3,  0,  0,  0),
               (4,  4,  0,  0,  5,  0),   {T. prijelaza}
               (4,  4,  0,  3,  0,  1),
               (0,  0,  0,  3,  0,  1) );
    Ta: mat = ((1,  0,  0,  0,  0,  3),
               (0,  0,  0,  0,  0,  0),
               (0,  4,  0,  0,  0,  0),   {T. akcija}
               (0,  4,  0,  0,  0,  2),
               (0,  0,  0,  0,  0,  2) );
    MetaSym = '|ε@';
```

```
  VAR  Net    : string;      { Skup neterminala          }
       Ter    : string;      { Skup terminala            }
       Pr     : ARRAY [vs]
                  OF string; { Skup produkcija           }
       P      : integer;     { Pozicija tekućeg znaka    }
       R      : string;      { Ulazni niz znakova        }
       Z      : char;        { Tekući znak               }
       C      : integer;     { Kod tekućeg znaka         }
       E      : boolean;     { Oznaka kraja upisa        }
       S, Ss  : integer;     { Tekuće i sljedeće stanje  }
       Gramat : text;        { Gramatika (skup neterminala,
                               terminala i produkcije)   }
       Ime_gr : string [12]; { Ime gramatike             }
       Y, R1  : string;    N     : char;
       K      : integer;   Line  : string;

PROCEDURE Inic; BEGIN { Inicijalizacija }
  Net := ''; Ter := ''; R := ''; E := false; P := 1; S := 1;
  FOR N := 'A' TO 'Z' DO Pr[N] := '';
  Write ('Upisite ime gramatike  '); Readln (Ime_gr);
  Assign (Gramat, Ime_gr); {$I-} Reset (Gramat); {$I+}
  IF IOresult = 0 THEN BEGIN
    Readln (Gramat, Net); Readln (Gramat, Ter);
    WHILE NOT eof (Gramat) DO BEGIN
      Readln (Gramat, Line); Z := Line[1];
      Pr[Z] := Line; Writeln (Line)
      END
    END
  END;

PROCEDURE Leks_An; BEGIN { Leksička analiza }
  IF P > length(R) THEN BEGIN
    readln (R); R := R+'@'; P := 1
    END;
  WHILE R[P] = ' ' DO P := P+1;
  Z := R[P]; P := P+1; K := pos (Z, MetaSym);
  IF K > 0           THEN BEGIN C := K+3; EXIT END;
  IF Z IN ['A'..'Z'] THEN BEGIN C := 1;   EXIT END;
  IF (Z = '-') AND
     (R[P] = '>')    THEN BEGIN C := 3; P := P+1; EXIT END;
  C := 2
  END;

PROCEDURE Sint_An; { Sintaksna analiza }
  PROCEDURE _0; BEGIN { Prazna akcija } END;
  PROCEDURE _1; BEGIN { Novi neterminal }
    N := Z; IF pos(N, Net) = 0 THEN Net := Net+Z
    END;
  PROCEDURE _2; BEGIN { Kraj produkcije }
    Pr [N] := copy (R, 1, length(R)-1)
    END;
```

```
  PROCEDURE _3;
      VAR  i, n : integer; { Pomocne varijable }
           Z    : char;
    BEGIN { Kraj upisa }
      ClrScr; Rewrite (Gramat);
      Writeln ('G = (N, V, P, S), gdje su:');
      Writeln;
      Write ('    N = {', Net[1]); n := length (Net);
      FOR i := 2 TO n DO Write (', ', Net[i]);
      Writeln ('}');  Writeln;  Writeln (Gramat, Net);
      Write ('    V = {', Ter[1]);
      FOR i := 2 TO length (Ter) DO Write (', ', Ter[i]);
      Writeln ('}'); Writeln; Writeln (Gramat, Ter);
      Writeln ('    P:'); Writeln;
      FOR i := 1 TO n DO BEGIN
        Z := Net[i];
        Writeln (Gramat, Pr[Z]);
        Writeln ('      ', Pr[Z])
        END;
      Close (Gramat);
      E := True
      END;

  PROCEDURE _4; BEGIN { Prosirenje alfabeta }
    IF pos (Z, Ter) = 0 THEN Ter := Ter+Z
    END;

  BEGIN { Sint_An }
    Ss := Tp[S, C];
    IF Ss = 0 THEN BEGIN
      Writeln ('^': P-1); writeln ('*** sint. pogreska ***');
      Y := copy (R, 1, P-2);
      Write (Y); Readln (R1);
      R := Y+R1+'@'; P := P-1;
      EXIT
      END;
    CASE Ta[S, C] OF  0: _0; 1: _1; 2: _2; 3: _3; 4: _4 END;
    S := Ss
    END;

BEGIN
  ClrScr;
  Inic;
  WHILE NOT E DO BEGIN
    Leks_An;
    Sint_An
    END;
  Readln;
  ClrScr
  END.
```

2. Silazna sintaksna analiza

Program silazne sintaksne analize realiziran je na temelju algoritma silazne sintaksne analize danog na str. 86. Ulazna gramatika mora biti prethodno definirana programom Meta_BNF, a unosi se upisom imena datoteke (s ekstenzijom, ako je ima) u posebnom modulu Upis_Gr. Gramatika ne smije biti rekurzivna slijeva (v. "Primjenljivost silazne sintaksne analize", str. 85). Dodatno je ograničenje da produkcije ne smiju imati više od 9 alternativa. Program ispisuje ulaznu gramatiku, konfiguracije algoritma za vrijeme izvođenja sintaksne analize i niz izvođenja na kraju analize u tekstualnu datoteku s imenom jednakim imenu datoteke ulazne gramatike i ekstenzijom TOP. To može korisno poslužiti u učenju samog postupka silazne sintaksne analize.

```
UNIT Upis_Gr;

INTERFACE
  USES Crt;
  TYPE vs = 'A' .. 'Z';
  VAR
    Net, Ter               : string;
    Alfabet                : SET OF char;
    Pr                     : ARRAY [vs, 1..9] OF string[20];
    Br_alt                 : ARRAY [vs] OF byte;
    i, k, n                : integer;
    Start, S, Sym, T, C, Z : char;
    Alfa, Beta, Sf, W, D   : string;
    j, L, M, Code          : integer;
    St                     : string [1];
    Err, Leks_gr, Sint_gr  : boolean;
    Isp_konf               : boolean;
    Izlaz                  : text;
    Ime_Izl, Out           : string;

  PROCEDURE Upis; PROCEDURE Upis_Niz; PROCEDURE Leks_An;

IMPLEMENTATION

PROCEDURE Upis;
    VAR F, j, L : integer;  Line    : string;
        N       : char;     Gramat  : text;
        Ime_gr  : string[12];

  PROCEDURE Kompr (VAR S: string);
      VAR  i: integer;
    BEGIN
      i := pos(' ', S);
      WHILE i <> 0 DO BEGIN
        Delete (S, i, 1); i := pos(' ', S)
        END
      END;
```

```
  BEGIN { Upis }
    Write ('Upisite ime gramatike  ');  Readln (Ime_Gr);
    Assign (Gramat, Ime_gr);      {$I-} Reset (Gramat) {$I+};
    IF IOresult <> 0 THEN BEGIN
      Write ('NE POSTOJI DATOTEKA "', Ime_Gr, '"'); Readln;
      HALT END;
    j := pos('.', Ime_Gr);
    IF j > 0 THEN Ime_Izl := copy(Ime_Gr, 1, j-1) +'.TOP'
             ELSE Ime_Izl := Ime_Gr +'.TOP';
    Assign (Izlaz, Ime_Izl); Rewrite (Izlaz);
    Readln (Gramat, Net);  Start   := Net[1];
    Readln (Gramat, Ter);  Alfabet := [];
    FOR j := 1 TO length(Ter) DO
      Alfabet := Alfabet +[Ter[j]];
    FOR j := 1 TO length(Net) DO BEGIN
      Readln (Gramat, Line);
      Kompr (Line);
      N := Line[1];  Br_alt[N] := 0;
      F := 4;  L := F-1 +pos('|', copy(Line, F, 255));
      WHILE L >= F DO BEGIN
        Br_alt[N]         := Br_alt[N] +1;
        Pr[N, Br_alt [N]] := copy(Line, F, L-F);
        F := L +1; L := F -1 + pos('|', copy(Line, F, 255))
        END;
      Br_alt[N]         := Br_alt[N] +1;
      Pr[N, Br_alt [N]] := copy(Line, F, 255);
      END
    END;

PROCEDURE Upis_Niz; BEGIN
  Writeln ('Upišite niz znakova'); Readln (W);
  Writeln (Izlaz, 'w = ', W); Writeln (Izlaz);
  N    := length(W);
  Sym  := W [1]
  END;

PROCEDURE Leks_An;
    VAR i: integer;
  BEGIN
    FOR i := 1 TO N DO
      IF NOT (W[i] IN Alfabet) THEN BEGIN
        Writeln (chr (24): i);
        Writeln ('*** ZNAK NIJE U ALFABETU!');
        Leks_gr := True; Write ('ENTER za nastavak'); Readln;
        ClrScr;
        EXIT
        END;
    ClrScr;  Writeln (W)
    END;
  END.
```

```
PROGRAM TopDown;  { SILAZNA SINTAKSNA ANALIZA }
  USES Crt, Upis_Gr;

PROCEDURE Init; BEGIN {Inicijalizacija varijabli}
  i    := 1;  Err  := False;  S       := 'q';
  Alfa := ''; Beta := Start +'$';
  END;

PROCEDURE Reader; BEGIN { Učitavanje ulaznog niza }
  ClrScr;  Writeln ('Zadaj recenicu');  Readln (W);  Clrscr;
  GotoXY  (1, 1);  Writeln (' ', W)
  END;

PROCEDURE Ulazna_Gr;
    VAR i, n: integer; Z: char; Ch: vs; Out: string;
  BEGIN
    Writeln ('Ulazna gramatika G = (N, T, P, S)');
    Writeln (Izlaz, 'Ulazna gramatika'); Writeln (Izlaz);
    Writeln; Write ('  N = {' +Net[1]);  N := length(Net);
    FOR i := 2 TO n DO Write (', '+Net[i]);
    Writeln ('}'); Writeln;
    Write ('  T = {' +Ter[1]);
    FOR i := 2 TO length(Ter) DO Write (', ', Ter[i]);
    Writeln ('}'); Writeln;
    FOR i := 1 TO length(Net) DO BEGIN
      Ch := Net[i];
      Write (Ch, ' -> ', Pr[Ch, 1]);
      Write (Izlaz, Ch, ' -> ', Pr[Ch, 1]);
      FOR j := 2 TO Br_Alt[Ch] DO BEGIN
        Write ('| ', Pr[Ch, j]);
        Write (Izlaz, '| ', Pr[Ch, j]);
        IF Pr[Ch, j] = 'ε' THEN
          Pr[Ch, j] := ''
        END;
      Writeln; Writeln (Izlaz)
      END;
    Writeln (Izlaz); Readln
    END;

PROCEDURE Ispis; BEGIN
  Writeln (Izlaz, '(', S,', ', i,', ', Alfa,', ', Beta,')');
  IF Isp_konf THEN BEGIN
    GotoXY (1, 2);
    Writeln (chr (24): i, '    ');
    GotoXY (1, 3);
    Writeln ('(', S, ', ', i, ', ', Alfa,
             ', ', Beta, ')                    ');
    Readln
    END
  END;
```

```
PROCEDURE Forw; BEGIN { Napredovanje }
  T := Beta[1];  k := pos(T, Net);
  IF k <> 0 THEN BEGIN
    { Ekspanzija stabla }
    Alfa := Alfa +T +'1';
    Beta := Pr[T, 1] +copy(Beta, 2, 255)
    END
  ELSE BEGIN
    { Izjednačivanje }
    IF T <> Sym THEN
      S := 'b'
    ELSE BEGIN
      i := i+1;  Sym := W[i];
      Alfa := Alfa +T;
      Delete (Beta, 1, 1);
      IF (Beta = '$') AND (i = n+1) THEN S := 't'
      END
    END
  END;

PROCEDURE Back; BEGIN { Povrat }
  T := Alfa [length (Alfa)];
  IF (T IN Alfabet) AND
    (pos (Alfa [length (Alfa) - 1], Net) = 0) THEN BEGIN
    { Vraćanje kazaljke }
    Delete (Alfa, length (Alfa), 1);
    Beta := T + Beta;
    i    := i -1;  Sym := W[i]
    END
  ELSE BEGIN
    { Pokušaj sljedeće alternative }
    T := Alfa [length(Alfa) -1];
    Val (Alfa [length(Alfa)], j, Code);
    K := pos(T, Net);
    IF j = Br_alt[T] THEN BEGIN
      {Iscrpljene su sve alternative}
      Delete (Alfa, length(Alfa) - 1, 2);
      Delete (Beta, 1, length(Pr[T, j]));
      Beta := T +Beta;
      IF (Alfa = '') AND (i = 1) THEN  Err := true
      END
    ELSE BEGIN
      { Sljedeća alternativa }
      j := j +1;  Str (j: 1, St);
      Alfa := copy(Alfa, 1, length (Alfa) - 2) +T +St;
      Beta := Pr[T,j] +copy(Beta, length(Pr[T,j-1]) +1, 255);
      S := 'q'
      END
    END
  END;
```

```
PROCEDURE Izvodjenje; BEGIN
  Writeln; Writeln (Izlaz);
  SF := Net[1]; Writeln (SF); Writeln (Izlaz, SF);
  D := '';
  FOR L := 1 TO length(Alfa) DO BEGIN
    C := Alfa[L];
    IF pos(C, Net) > 0 THEN
      D := D +copy(Alfa, L, 2);
    END;
  L := 1;
  REPEAT
    C  := D[L]; val (D[L+1], j, Code);
    k  := pos(C, SF); M := pos(C, Net);
    SF := copy(SF, 1, k-1) + Pr[C, j] +copy(SF, k+1, 255);
    Writeln ('  => ', SF); Writeln (Izlaz, '  => ', SF);
    L := L+2
    UNTIL L > length(D)
  END;


BEGIN { TopDown }
  ClrScr;
  Upis; Ulazna_Gr; Init;
  ClrScr;
  REPEAT
    Leks_gr := False;
    Upis_Niz;
    Leks_An;
    UNTIL NOT Leks_gr;
  W := W +'@';
  Write ('Želite li ispis konfiguracija (D/N)? ');
  Readln (Z);
  Isp_konf := upcase(Z) = 'D';
  GotoXY (1, WhereY -1);
  Writeln (' ': 80);
  REPEAT
    Ispis;
    IF S = 'q' THEN Forw ELSE Back;
    IF S = 't' THEN Ispis
    UNTIL  (S = 't') OR Err;
  IF Err THEN BEGIN
    GotoXY (1, 3); Out := '*** NIZ NIJE U JEZIKU!';
    Writeln (Out); Writeln (Izlaz, Out)
    END
  ELSE BEGIN
    Writeln; Izvodjenje
    END;
  Close (Izlaz);
  Readln
  END.
```

3. Uzlazna sintaksna analiza

Program uzlazne sintaksne analize realiziran je na temelju algoritma uzlazne sintaksne analize danog na str. 91. Ulazna gramatika, kao i za program silazne sintaksne analize, mora biti prethodno definirana programom `Meta_BNF`. Gramatika ne smije imati prazne produkcije (v. "Primjenljivost uzlazne sintaksne analize", str. 91). Dodatno je ograničenje da ukupni broj alternativa ne smije biti veći od 99. Program ispisuje konfiguracije algoritma za vrijeme izvođenja sintaksne analize što može korisno poslužiti u učenju samog postupka uzlazne sintaksne analize.

```
PROGRAM Bottom_Up; { UZLAZNA SINTAKSNA ANALIZA }
  USES Crt;
  VAR
    Alt     : ARRAY [1..99]
              OF STRING;    { Produkcije }
    P       : STRING;       { Uređenje produkcija }
    Br_A  : integer;      { Broj alternativa }
    Ter     : STRING;       { Skup terminalnih znakova }
    Alfabet : SET OF char;  { Alfabet }
    Net     : STRING;       { Skup neterminala}
    Sint_gr : Boolean;      { Indikator sintaksne pogreške }
    Leks_gr : Boolean;      { Indikator leksičke pogreške }
    S       : char;         { Stanje algoritma }
    i       : integer;      { Pozicija kazaljke }
    A       : STRING;       { Stog L1 }
    B       : STRING;       { Stog L2 }
    W       : STRING;       { Ulazni niz }
    N       : integer;      { Duljina ulaznog niza }
    Znak    : char;         { Tekući znak }
    Gramat  : text;         { Ulazna gramatika }
    Ime_gr  : STRING [12];  { Ime gramatike (datoteka) }
    Start   : char;         { Početni simbol }
    Isp     : boolean;      { Ispis konfiguracija }
    ImaPP   : boolean;      { Prazna produkcija }

PROCEDURE Inic;  {Inicijalizacija varijabli programa }
    VAR  F, j, L : integer; { Pomoćne varijable }
         Pr      : STRING;  { Produkcija }
         N       : char;    { Neterminal }

  PROCEDURE Kompr (VAR S: STRING);
                  {"Komprimiranje" niza znakova}
      VAR  i: integer;
    BEGIN
      i := pos (' ', S);
      WHILE i <> 0 DO BEGIN
        Delete (S, i, 1);
        i := pos (' ', S)
        END
      END;
```

```
  BEGIN { Inic }
    Write  ('Upišite ime gramatike  '); Readln (Ime_gr);
    Assign (Gramat, Ime_gr);  Reset (Gramat);
    Readln (Gramat, Net);     Start   := Net [1];
    Readln (Gramat, Ter);     Alfabet := [];
    FOR j := 1 TO length(Ter) DO
      Alfabet := Alfabet +[Ter[j]];
    Br_A := 0; P := ''; ImaPP := False;
    FOR j := 1 TO length(Net) DO BEGIN
      Readln (Gramat, Pr);
      ImaPP := ImaPP OR (pos('#', Pr) > 0);
      Kompr (Pr);
      N := Pr [1];  F := 4;
      L := F-1 +pos ('|', copy (Pr, F, 255));
      WHILE L >= F DO BEGIN
        Br_A       := Br_A+1;
        Alt [Br_A] := copy (Pr, F, L-F);
        P := P +N;
        F := L +1;  L := F -1 + pos('|', copy(Pr, F, 255))
        END;
      Br_A      := Br_A +1;
      Alt[Br_A] := copy (Pr, F, 255);
      P           := P +N
      END;
    Sint_gr := False;
    S       := 'q';
    I       := 1;
    A       := '$';
    B       := '';
    END; { Inic }

PROCEDURE Upis; BEGIN { Upisivanje rečenice }
  Writeln ('Upišite rečenicu'); Readln (W);
  N    := length(W);
  Znak := W[1];
  Writeln ('Ispis konfiguracija (D/N)?');
  Isp  := upcase(ReadKey) = 'D'
  END;

PROCEDURE Leks_an; { Leksička analiza }
    VAR i: integer;
  BEGIN
    FOR i := 1 TO N DO IF NOT (W[i] IN Alfabet) THEN BEGIN
      Writeln (chr (24): i);
      Writeln ('Leks. gr. - znak nije u alfabetu');
      Leks_gr := True; Write ('ENTER za nastavak');
      Readln; ClrScr;
      EXIT
      END;
    ClrScr; Writeln (W)
  END;
```

```
PROCEDURE Sintaksna_Analiza;

  PROCEDURE Reduciranje;
    VAR
      L, K, M: integer;    St: STRING [2];
    BEGIN
      FOR M := 1 TO Br_A DO BEGIN
        L := length(Alt[M]); K := length(A);
        IF L <= K-1 THEN
          IF copy (A, K-L+1, L) = Alt[M] THEN BEGIN
            A := copy (A, 1, K-L) +P[M]; Str (M :2, St);
            IF St[1] = ' ' THEN St[1] := '0';
            B := St +B;
            EXIT
            END
        END;
      IF I < N+1 THEN BEGIN
        { Premještanje }
        B := 's' +B; A := A +Znak; i := i +1; Znak := W[i]
        END
      ELSE BEGIN
        IF A = '$' +Start THEN BEGIN
          S := 't'; EXIT
          END;
        S := 'b'
        END
      END;

  PROCEDURE Vracanje;
      VAR
        T:    STRING [2];
    PROCEDURE X; { Pomoćna procedura }
      VAR
        V, C, L, K, A_d, Alt_d : integer; St: STRING [2];
      BEGIN
        IF B[1] = 's' THEN Delete (B, 1, 1)
                      ELSE Delete (B, 1, 2);
        Delete (A, length (A), 1);  Val (T, V, C);
        A := A +Alt[V];  L := V +1;
        WHILE L <= Br_A DO BEGIN
          A_d := length (A); Alt_d := length (Alt[L]);
          IF A_d >= Alt_d THEN
            IF copy(A, A_d -Alt_d +1, 255)=Alt[L] THEN BEGIN
              Delete (A, A_d -Alt_d +1, 255); A := A +P[L];
              Str(L:2,St); IF St[1]=' ' THEN St[1] := '0';
              B := St + B; S := 'q'; EXIT
              END;
          L := L +1
          END
      END;
```

```
  PROCEDURE Y; BEGIN { Pomoćna procedura }
    IF i > 1 THEN BEGIN
      i := i -1;  Znak := W[i]; Delete (A, length (A), 1);
      IF B[1] = 's' THEN Delete (B, 1, 1)
                    ELSE Delete (B, 1, 2)
      END
    END;

  BEGIN { Vracanje }
    IF i < N +1 THEN BEGIN
      { Vraćanje kazaljke }
      T := '';
      IF B[1] = 's' THEN
        T := B[1]
      ELSE IF length (B) > 0 THEN
        T := copy (B, 1, 2);
      IF T = ''  THEN BEGIN Sint_gr := true; EXIT END;
      IF T = 's' THEN BEGIN Y; EXIT END;
      X;
      IF S = 'b' THEN S := 'q';
      A := A + Znak;  B := 's' + B;
      i := i+1; Znak := W[i]; EXIT
      END
    ELSE BEGIN
      IF B[1] = 's' THEN T := B[1]
                    ELSE T := copy (B, 1, 2);
      IF T = 's' THEN BEGIN Y; EXIT END;
      X
      END
    END;

BEGIN { Sintaksna_Analiza }
  WHILE NOT (S = 't') AND NOT Sint_gr DO BEGIN
    GotoXY (1, 2); Write (chr(24): i, '  ');
    IF Isp THEN BEGIN
      GotoXY (1, 3);
      Write ('(', S, ',',i, ', ',A, ', ',B, ')         ');
      ReadLN;
      END;
    IF S = 'q' THEN Reduciranje ELSE Vracanje
    END;
  IF Sint_gr THEN BEGIN
    GotoXY (1, 2); Writeln (chr(24): i);
    Writeln ('*** Sintaksna pogreška ***')
    END
  ELSE BEGIN
    GotoXY (1, 3);
    Writeln ('(', S, ',', I, ', ', A, ', ', B, ')')
    END
  END; { Sintaksna_Analiza }
```

```
PROCEDURE Ispis;  {Ispisivanje niza izvodjenja}
    VAR
      C, D, Sf         : STRING;
      K, L, M, Kod, P  : integer;
  BEGIN
    Writeln; Writeln (Start);
    D := ''; L := 1;
    WHILE L <= length (B) DO BEGIN
      IF B[L] <> 's' THEN BEGIN
        D := D +copy(B, L, 2);
        L := L +1
        END;
      L := L +1
      END;
    Sf := Start; L  := 1;
    WHILE L <= length (D) DO BEGIN
      val (copy (D, L, 2), K, Kod);
      M := length (Sf);
      REPEAT
        P := pos (Sf [M], Net);
        M := M-1
        UNTIL (P <> 0) OR (M=0);
      IF P <> 0 THEN BEGIN
        M := M+1;
        Sf := copy (Sf, 1, M-1) +Alt[K] +copy(Sf, M+1, 255);
        writeln ( '   => ', Sf)
        END;
      L := L +2
      END
    END;

  PROCEDURE Ulazna_GR;
      VAR  i, N : integer;
           Z    : char;
           Ch   : char;
    BEGIN
      Writeln ('Ulazna gramatika G=(N,T,P,S), gdje su:');
      Writeln;
      Write ('  N = {', Net [1]);  N := length(Net);
      FOR i := 2 TO N DO Write (', ', Net [i]);
      Writeln ('}');  Writeln;
      Write ('  T = {', Ter [1]);
      FOR i := 2 TO length (Ter) DO Write (', ', Ter [I]);
      Writeln ('}'); Writeln;
      Writeln ('  P:');
      WriteLN (' ' :6, P[1], ' -> ', Alt [1]);
      FOR i := 2 TO Br_A DO
          Writeln (' ':6, P[i], ' -> ', Alt[i]);
      Readln
      END;
```

```
BEGIN
  ClrScr;
  Inic;
  Ulazna_Gr;
  IF ImaPP THEN BEGIN
    WriteLN ('Ulazna gramatika sadrži praznu produkciju!');
    Readln; ClrScr; HALT
    END;
  REPEAT
    Leks_gr := False;
    Upis;
    Leks_an;
    UNTIL NOT Leks_gr;
  Sintaksna_Analiza;
  IF NOT Sint_gr THEN Ispis;
  Readln;
  END.
```

4. CYK sintaksna analiza

Slijedi program sintaksne analize realiziran na temelju Cocke-Younger-Kasamijeva algoritma opisanog u 5. poglavlju, str. 94 do 96. Ulazna gramatika mora biti u CNF-u i mora biti prethodno pripremljena u tekstualnoj datoteci. U prvom redu je početni simbol, potom se u idućim redovima unose produkcije, svaka alternativa posebno, bez razmaka. Maksimalan broj produkcija ograničen je na 30. Na primjer, datoteka s imenom `PR-5-3.CYK` sadrži produkcije gramatike iz primjera 5.3, str. 95 (`S` je početni simbol):

```
S
S->AA
S->AS
S->b
A->SA
A->AS
A->a
```

Poslije unosa imena datoteke koja sadrži ulaznu gramatiku zadaje se ulazni niz koji će se analizirati. Postupak se može ponoviti s drugim ulaznim nizom. Prekid programa je kad se za ulazni niz upiše `@`.

```
PROGRAM CYK;  { Cocke-Younger-Kasamijeva sintaksna analiza }
              { Autor programa: Dr. Milan STAMENKOVIĆ      }

  USES  Crt, Graph;
  TYPE  String2     = string[ 2];
        String10    = string[10];
  CONST MaxProd =   30;   { Maksimalan broj produkcija }
```

```
  VAR S           : string2; { početni simbol }
      Neterminali : SET OF char;   { skup neterminala }
      Terminali   : SET OF char;   { skup terminala }
      P           : ARRAY [1..MaxProd] OF RECORD
                      L : string2; { lijeva  str. prod. }
                      D : string2  { desna str. prod. }
                      END;
      T           : ARRAY [1..MaxProd,1..MaxProd] OF
                          SET OF char;
      Tstr        : ARRAY [1..MaxProd] OF string;
      BrProd      : integer;  { broj produkcija }
      w           : string;   { ulazna rečenica }
      nn          : integer;  { duljina ulazne rečenice }
      ii,jj,kk,ll : integer;  { kontrolne varijable }
      Tree        : ARRAY [0..40] OF string[80];
      Astr, Bstr,
      Cstr, ai    : string2; Achar, Bchar, Cchar   : char;
      TijExists   : boolean; Korak, z, dd, mm, ni  : integer;
      MaxAlt      : integer; Xstep, Ystep, jjj, iii: integer;
      HS, RedProd,
      TreeStr,
      TreeStr1    : string;
      LeftD       : ARRAY [1..100] OF integer;

PROCEDURE GetData;
  VAR F     : text; imeD  : string; Chh   : char;
      Unija : SET OF char;          i1    : integer;
  BEGIN
    Terminali := []; Neterminali := []; Unija := [];
    ClrScr; Write ('Upišite ime datoteke s gramatikom ');
    Write (' (NEXT za GRA.CYK) : '); Readln(imeD);
    IF imeD = '' THEN ImeD := 'EXP.CYK';
    Writeln ('Učitavanje produkcija s datoteke ',
             imeD, ' ...');
    Assign (F, ImeD); Reset (F); i1 := 0; Readln (F, S);
    WHILE NOT eof(F) DO BEGIN
      i1 := i1 +1;   Read (F, Chh); P[i1].D := '';
      P[i1].L := Chh; { lijeva str. prod. }
      Neterminali := Neterminali +[Chh]; Read (F, Chh);
      Read (F, Chh); { preskok za oznaku -> }
      WHILE NOT eoln(f) DO BEGIN
        Read (F, Chh);
        P[i1].D := P[i1].D +Chh;
        Unija   := Unija   +[Chh]
        END;
      Readln (F);
      END;
    BrProd    := i1; { Broj produkcija }
    Terminali := Unija -Neterminali
    END;
```

```
PROCEDURE PrikaziGramatiku;
  VAR i1 : integer; Tstring, Nstring : string;
  BEGIN
    Tstring := ''; NString := '';
    FOR i1 := 31 TO 200 DO IF chr(i1) IN Terminali THEN
      Tstring := Tstring +chr(i1) + ',';
    FOR i1 := 31 TO 200 DO IF chr(i1) IN Neterminali THEN
      Nstring := Nstring +chr(i1) +',';
    IF length(Tstring) > 1 THEN
      Tstring := copy(Tstring, 1, length(Tstring) -1);
    IF length(Nstring) > 1 THEN
      Nstring := copy(Nstring, 1, length(Nstring) -1);
    ClrScr;
    Writeln ('Cocke-Younger-Kasamijeva sintaksna analiza. ');
    Writeln ('Ulazna gramatika: G = ({',NString,'} ,
             {',Tstring,'}, P ,',S,')');
    Writeln ('P: '); Writeln;
    FOR i1 := 1 TO BrProd DO
      Writeln ('(',i1:(BrProd DIV 10 + 1),')   ',
               P[i1].L,' -> ', P[i1].D)
    END;

PROCEDURE ObrisiTablicu;
  VAR i1, j1: integer;
  BEGIN
    FOR i1 := 1 TO maxProd DO
       FOR j1 := 1 TO MaxProd DO T[i1,j1] := [];
    FOR i1 := 1 TO MaxProd DO Tstr[i1] := ''
    END;

PROCEDURE FormirajDonjiRed;
  VAR i1, j1, k1, m1 : integer;   Ch1 : string[1];
      char1          : char;      St  : SET OF char;
  BEGIN
    FOR i1 := 1 TO nn DO FOR j1:=1 to BrProd DO BEGIN
      Ch1   := copy(P[j1].L,1,1); Char1 := Ch1[1];
      IF P[j1].D = copy(w, i1, 1) THEN
        T[i1,1] := T[i1,1] +[Char1] END;
    FOR i1 := 1 TO nn DO BEGIN
      Tstr[1] := Tstr[1] + ' '; St := []; m1 := 0;
      FOR k1 := 1 TO BrProd DO BEGIN
        Ch1   := copy(P[k1].L, 1, 1); Char1 := Ch1[1];
        IF (NOT (Char1 IN St)) AND
          (Char1 IN T[i1,1]) THEN BEGIN
          Tstr[1]:=Tstr[1] +P[k1].L;
          St := St +[Char1]; m1 := m1 +1
          END
        END;
      IF m1 > MaxAlt THEN MaxAlt := 1
      END
    END;
```

```
PROCEDURE PrikaziRed (Red: integer);
  VAR i1, j1, k1, l1, m1 : integer;      Ch1   : string;
      St                 : SET OF char;  Char1 : char;
  BEGIN
    FOR i1 := 1 TO nn -red +1 DO BEGIN
      Tstr[jj] := Tstr[jj]+ ' '; St := []; m1 := 0;
      FOR k1 := 1 TO BrProd DO BEGIN
        Ch1 := copy(P[k1].L, 1, 1);  Char1 := Ch1[1];
        IF (NOT (Char1 IN St)) AND
          (Char1 IN T[i1, Red]) THEN BEGIN
          Tstr[jj] := Tstr[jj] +P[k1].L;
          St       := St +[Char1];  m1 := m1 +1
          END
        END;
      IF m1 > MaxAlt THEN MaxAlt := m1
      END
    END;

PROCEDURE Gen (VAR i8, j8: integer; VAR A8: string2);
  VAR m, k, i9, j9 : integer;  Bch,Cch : char;
      Bstr,Cstr    : string2;  Ok      : Boolean;
  BEGIN
    ai := copy(w,i8,1);
    {1} IF j8 = 1 THEN BEGIN
       m := 0;
       REPEAT m := m+1 UNTIL (m = BrProd) OR (P[m].D=ai);
       IF P[m].D=ai THEN BEGIN ni := ni+1; LeftD[ni] := m END
       END;
    {2} IF j8 > 1 THEN BEGIN
       k := 0;
       REPEAT
         k  := k +1;  Ok := False; m  := 0;
         REPEAT
           m := m +1;
           IF length(P[m].D) = 2 THEN BEGIN
             Bstr := copy (P[m].D,1,1);
             Cstr := copy (P[m].D,2,1);
             Bch  := Bstr[1]; Cch := Cstr[1];
             Ok   := (Bch IN T[i8, k]) AND
                     (Cch IN T[i8+k, j8-k]) AND(A8=P[m].L)
             END
           UNTIL Ok OR (m=BrProd)
         UNTIL Ok OR KeyPressed OR (k>=j8-1);
       IF Ok THEN BEGIN
         ni := ni +1; LeftD[ni] := m;    gen (i8, k, Bstr);
         i9 := i8+k;  j9        := j8-k; gen (i9, j9, Cstr)
         END
       END
    END;
```

```
BEGIN { CYK  }
  GetData; { Učitaj gramatiku }
  REPEAT
    MaxAlt :=0;  ObrisiTablicu; ClrScr;
    PrikaziGramatiku;
    GotoXY(1,14); Write ('Upisite recenicu : '); Readln(w);
    IF W[1] = '@' THEN BEGIN ClrScr; HALT END;
    nn    := length(W);  korak := 1;
    FormirajDonjiRed; {1. korak: popuni donji red}
    jj    := 1; korak := 2; ii := 2;
    REPEAT
      CASE korak OF
        2: BEGIN {KORAK 2 - popunjavanje idućeg reda tablice}
             korak :=3; jj := jj +1;
             FOR ii:=1 TO nn-jj +1 DO FOR ll:=1 TO brProd DO
               IF length(P[ll].D)=2 THEN
                 FOR kk := 1 TO jj-1 DO BEGIN
                   Astr  := copy(P[ll].L,1,1);
                   Bstr  := copy(P[ll].D,1,1);
                   Cstr  := copy(P[ll].D,2,1);
                   Achar := Astr[1];  Bchar := Bstr[1];
                   Cchar := Cstr[1];
                   IF (Bchar IN t[ii, kk]) AND
                     (Cchar IN t[ii+kk, jj-kk]) THEN
                     T[ii,jj] := T[ii,jj] +[Achar]
                   END;
             Prikazired(jj)
             END;
        3: korak := 2
        END;
      TijExists:=False;
      FOR z := 1 TO nn -jj +1 DO
        IF T[z,jj]<>[] THEN TijExists := True
      UNTIL (jj>nn) OR (NOT TijExists);
    dd    := Detect; InitGraph (dd, mm, '');
    Xstep := round(2*MaxAlt*TextWidth('W'));
    Ystep := round(2*TextHeight('W')); jjj := 1;
    REPEAT
      Rectangle (Xstep,Ystep*(jj-jjj+2), Xstep*(nn-jjj+2),
                 Ystep*(jj-jjj+1));
      Line      (Xstep*(nn-jjj+2), Ystep*(jj-jjj+2),
                 Xstep*(nn-jjj+2), Ystep*(jj+1));
      FOR ii := 1 TO nn -jjj +1 DO BEGIN Hs := '';
        FOR dd := 32 TO 255 DO IF chr(dd) IN T[ii,jjj] THEN
          Hs := Hs+ chr(dd);
        OutTextXY (round(0.25 *Xstep) +Xstep *ii,
          Ystep*(jj-jjj +1)+round(0.5 *TextHeight('H')),Hs)
        END;
      jjj := jjj +1;
      UNTIL (jjj = jj) OR KeyPressed;
```

```
  ii := 1; ni := 0;  Gen (ii, nn, S);
  FOR ii := 0 TO ni DO Tree[ii] := '';
  Tree[0] := copy(S,1,1);
  FOR ii := 1 TO ni DO BEGIN
    jj := 0;
    REPEAT
      jj   := jj +1; Bstr := copy(Tree[ii-1], jj, 1);
      UNTIL ( Bstr[1] IN Neterminali ) OR
            (jj > length(Tree[ii -1]));
    Tree[ii]:= copy(Tree[ii-1], 1, jj -1)
              +P[LeftD[ii]].D
              +copy(Tree[ii-1], jj+1,
              length(Tree[ii-1])-jj+1)
    END;
  TreeStr := '';
  FOR ii := 0 TO ni -1 DO
    Treestr := TreeStr +Tree[ii]+'=>';
  TreeStr := TreeStr +Tree[ni];
  RedProd := 'red. produkcija : ';
  FOR ii := 1 TO ni DO BEGIN
    Str(LeftD[ii]: 2, Bstr); RedProd := RedProd +Bstr
    END;
  OutTextXY (30, Ystep*(nn+3),
                 'Tabela CYK-algoritma');
  IF length(TreeStr)*TextWidth('W') > GetmaxX-30 THEN BEGIN
    TreeStr1 := copy(TReeStr, 1,
                     length(TreeStr) DIV 2);
    TreeStr  := copy(TreeStr, 1 +length(TreeStr) DIV 2,
                          -1 +length(TreeStr) DIV 2);
    OutTextXY (30, Ystep*(nn+5), TreeStr1);
    OutTextXY (30, Ystep*(nn+6), TreeStr);
    OutTextXY (30, Ystep*(nn+7), RedProd)
    END
  ELSE BEGIN
    OutTextXY (30, Ystep*(nn+5), TreeStr);
    OutTextXY (30, Ystep*(nn+7), RedProd)
    END;
  Readln; CloseGraph
  UNTIL False
END.
```

5. Predikatna sintaksna analiza

Na kraju dajemo program 1-predikatne sintaksne analize opisane u šestom poglavlju, str. 108 do str. 113. I ovdje ulazna gramatika mora biti prethodno definirana programom `Meta_BNF`. Gramatika mora biti tipa $LL(1)$. Dodatno je ograničenje da broj alternativa po produkciji ne smije biti veći od 9. Program sadrži proceduru `Tablica`, za tvorbu tablice sintaksne analize.

```
PROGRAM Predikatna_Analiza; {1-predikatna sintaksna analiza}
  USES Crt;
  TYPE
    vs      = 'A' .. 'Z';
    ind     = '!' .. '~';
  VAR
    Ter     : string [20];          {Skup terminalnih znakova}
    Alfabet : SET OF char;          {Alfabet}
    Net     : string [26];          {Skup neterminala}
    P       : string [ 9];          {Uredjenje produkcija}
    Sint_gr : Boolean;              {Indikator sint. pogreške}
    Leks_gr : Boolean;              {Indikator leks. pogreške}
    Alfa    : string;               {Stog L1}
    W       : string;               {Ulazni niz}
    N       : integer;              {Duljina ulaznog niza}
    Znak    : char;                 {Tekući znak}
    Gramat  : text;                 {Ulazna gramatika}
    Ime_gr  : STRING [12];          {Ime gramatike (datoteka)}
    Start   : char;                 {Početni simbol}
    M       : ARRAY [1..30, 1..30] OF
                    STRING [10];    {Tablica SA}
    Pr      : ARRAY [vs, '1'..'9'] OF
                    STRING [20];    {Produkcije}
    Br_A    : ARRAY[vs] OF integer;{Broj alter. svake prod.}
    X       : STRING;               {Oznake redova od M}
    Y       : STRING;               {Oznake stupaca od M}
    Pi      : string;               {Izlazna vrpca }
    U       : string;               {Preostali dio ul. niza}
    Line    : string; IspKonf: boolean; i, j: integer;

PROCEDURE Tablica; {Tvorba tablice SA}
    VAR  F, J, K, L, i2, j1, F1 : integer;
         N, N1, N2, Z, Ch       : char;
         Fi, Fo                 : ARRAY [vs] OF SET OF char;
         First                  : SET OF char;
         St                     : string [2];  RF: string;
  PROCEDURE Kompr (VAR S: string);
     VAR  i: integer;
    BEGIN
      i := pos (' ', S);
      WHILE i <> 0 DO BEGIN Delete(S,i,1); i:=pos(' ', S) END
      END;
  BEGIN
    Write ('Upisi ime gramatike  ');  readln (Ime_gr);
    Assign (Gramat, Ime_gr);  Reset (Gramat);
    Readln (Gramat, Net);  Start   := Net [1];
    Readln (Gramat, Ter);  Alfabet := [];
    FOR j := 1 TO length(Ter) DO
      Alfabet := Alfabet +[Ter[j]];
    X := Net +Ter +'$';  Y := Ter +'#';
```

```
FOR i := 1 TO length(X) DO
  FOR j := 1 TO length(Y) DO M[i, j] := '';
FOR j := 1 TO length (Net) DO BEGIN
  Readln (Gramat, Line); Kompr (Line);
  N := Line [1];  Br_A[N] := 0;  F := 4;
  L := F-1 + pos ('|', copy (Line, F, 255));
  WHILE  L >= F DO BEGIN
    Br_A[N] := Br_A[N] + 1;
    Ch        := chr (ord('0') +Br_A[N]);
    Pr [N, Ch] := copy (Line, F, L-F);
    F := L+1;  L := F-1 +pos ('|', copy (Line, F, 255))
    END;
  Br_A[N] := Br_A[N]+1; Ch := chr(ord('0') +Br_A[N]);
  Pr[N, Ch] := copy (Line, F, 255) END;
FOR Ch:='A' TO 'Z' DO BEGIN Fi[Ch]:=[]; Fo[Ch]:=[] END;
FOR i := 1 TO length(Net) DO BEGIN
  N := Net[i];
  FOR j := 1 TO Br_A[N] DO BEGIN
    Ch := chr (ord('0') +j);
    IF Pr [N, Ch] <> '' THEN BEGIN
      Z := Pr [N, Ch] [1];  First := [Z];
      IF First <= Alfabet THEN BEGIN
        Fi[N] := Fi[N] +[Z]; Str (j, St);
        M[i, pos(Z, Ter)] := N +St
        END
      ELSE BEGIN
        WHILE NOT (First <= Alfabet +['#']) DO BEGIN
          FOR k := 1 TO length(Net) DO BEGIN
            Z := Net[K];
            IF Z IN First THEN BEGIN
              First := First -[Z];
              FOR L := 1 TO Br_A[Z] DO BEGIN
                Ch := chr (ord('0') +L);
                First := First +[Pr[Z,Ch][1]]
                END
              END
            END
          END;
        FOR k := 1 TO length(Ter) DO BEGIN
          Z := Ter [k];
          IF Z IN First THEN BEGIN
            Fi[N] := Fi[N] +[Z]; Str (j, St);
            M [i, k] := N +St
            END
          END
        END
      END
    END
  END;
```

```
FOR i := 1 TO length(Net) DO BEGIN
  N := Net[i];
  FOR j := 1 TO Br_A[N] DO BEGIN
    Ch := chr (ord('0') +j);
    IF Pr [N, Ch] [1] = '#' THEN BEGIN
      Str (j, St); M[i, pos ('#', Y)] := N +St;
      FOR k := 1 TO length(Net) DO BEGIN
        N1 := Net[k];
        IF N1 <> N THEN BEGIN
          FOR L := 1 TO Br_A[N1] DO BEGIN
            Ch   := chr (ord('0') +L);
            Line := Pr[N1, Ch]; F := pos (N, Line);
            IF F = length(Line) THEN BEGIN
              FOR i2 := 1 TO length(Net) DO BEGIN
                N2 := Net[i2];
                IF N2 <> N THEN BEGIN
                  FOR j1 := 1 TO Br_A[N2] DO BEGIN
                    Ch := chr(ord('0')+j1);
                    Line:= Pr[N2,Ch]; F := pos(N1,Line);
                    IF (F>0) AND (F<length(Line))
                      THEN BEGIN
                      Z := Line[F+1];
                      IF Line[F+1] IN Alfabet THEN
                        Fi[N] := Fi[N] +[Z]
                      ELSE
                        Fi[N] := Fi[N] +Fo[Z] +Fi[Z]
                      END
                    END
                  END
                END
              END
            END;
          FOR L := 1 TO length(Ter) DO BEGIN
            Z := Ter[L];
            IF (Z IN Fi[N]) AND (M[i, L] = '') THEN
              M[i, L] := N +St
            END
          END
        END
      END
    END;
  FOR i := 1 TO length(Net) DO BEGIN
    N := Net[i];
    FOR j := 1 TO Br_A[N] DO BEGIN Ch:=chr(ord('0') +j);
      IF Pr [N, Ch] [1] = '#' THEN Pr [N, Ch] := '' END
  END;
L := length (Net);
FOR i := 1 TO length (Ter) DO M [L+i, i] := 'pop';
M[length(X), length(Y)] := 'accept' END;
```

```
PROCEDURE Upis; BEGIN {Upisivanje rečenice}
  Writeln ('Upisite recenicu'); Readln (W); N := length (W);
  Znak := W [1]; Write ('Ispis konfiguracija (D/N)? ');
  Readln (Line); IspKonf := upcase (Line[1]) = 'D'
  END;

PROCEDURE Leks_An;  {Leksička analiza}
    VAR i: integer;
  BEGIN
    FOR i := 1 TO N DO
      IF NOT (W [i] IN Alfabet) THEN BEGIN
        Writeln (chr (24): i);
        Writeln ('Leks. gr. - znak nije u alfabetu');
        Leks_gr := True; Write ('ENTER za nastavak');
        Readln; ClrScr; EXIT
        END;
    ClrScr; Writeln (W)
    END;

PROCEDURE Sint_An;  {Sintaksna analiza}
  VAR Ai     : char; {Tekući znak ulaznog niza}
      Akcija : string;  j, k   : integer;
      Kraj   : boolean; Ch     : char;
  BEGIN
    U    := W; Alfa := Start +'$'; Pi      := '';
    i    := 1; Ai   := U[1]; Sint_Gr := False; Kraj := False;
    WHILE NOT Kraj AND (U <> '') AND NOT Sint_Gr DO BEGIN
      j      := pos (Alfa[1], X); k := pos (Ai, Y);
      Akcija := M [j, k];
      IF Akcija = 'pop' THEN BEGIN
        Delete (U, 1, 1);
        IF U = '' THEN U := '#';
        Ai := U[1]; Delete (Alfa, 1, 1); i := i +1
        END
      ELSE IF (Akcija <> '') AND
        (Akcija <> 'accept') THEN BEGIN
        Ch   := Akcija[2];
        Alfa := Pr[Akcija[1], Ch] +copy(Alfa, 2, 255);
        Pi   := Pi +Akcija
        END;
      Sint_Gr := Akcija = '';  Kraj := Akcija = 'accept';
      IF IspKonf THEN BEGIN
        GotoXY (1, 1);
        Write ('(',U, ', ', Alfa, ', ', Pi, ')           ');
        Readln END;
      IF Sint_Gr THEN BEGIN
        GotoXY (1, 2); Write ('*** Sintaksna pogreska!');
        Readln END
      END
    END;
```

```
PROCEDURE Ispis;  {Ispisivanje niza izvodjenja}
  VAR i, j, k : integer;   N, Ch   : char;
      RF      : string;    Ok      : boolean;
  BEGIN
    Writeln ('Niz ',
             W,' je u jeziku. Može se dobiti izvodjenjem:');
    Writeln;
    Write (Start); RF := Start;
    WHILE Pi <> '' DO BEGIN
      Ok := False; k := 1;
      WHILE NOT Ok DO BEGIN
        Ok := pos (RF[k], Net) > 0;
        IF NOT Ok THEN k := k +1
        END;
      N := RF[k]; Ch := Pi[2];  i := pos (N, Net);
      RF := copy(RF, 1, k-1) +Pr[N, Ch] +copy(RF, k+1, 255);
      GotoXY (3, WhereY); Writeln ('=> ', RF); Delete(Pi,1,2)
      END;
    Readln
    END;

BEGIN
  ClrScr;
  Tablica;
  REPEAT
    Leks_gr := false;
    Upis;
    Leks_An;
    UNTIL NOT Leks_gr;
  Sint_An;
  ClrScr;
  {* ISPIS TABLICE SINTAKSNE ANALIZE
     Write (' ');
     FOR j := 1 TO length(Y) DO Write (Y [j]: 10);
     Writeln; Writeln;
     FOR i := 1 TO length(X) DO BEGIN
       write (X [I]);
       FOR  j := 1  TO length(Y) DO
          write (M [i, j]: 10);
       writeln
       END;
    *}
  IF NOT Sint_gr THEN Ispis
  END.
```

Kazalo