Zdravko
DOVEDAN HAN

FORMALNI JEZICI I PREVODIOCI

• regularni izrazi, gramatike, automati

Recenzenti
Prof. dr. sc. Damir BORAS, FF, Zagreb
Prof. dr. sc. Mirko MALEKOVIĆ, FOI, Varaždin
Prof. dr. sc. Božidar TEPEŠ, UF, Zagreb

Lektorica
Dr. sc. Vjera LOPINA

Uređenje teksta
Autor

Predgovor

Teorija formalnih jezika (ili automata) jest kamen temeljac teorije računalnih znanosti. Međutim, njezin nastanak i motivacija dolaze iz različitih izvora:

1. *Uključivanja i isključivanja strujnih krugova* kao modela u elektrotehnici.
2. *Gramatikâ* kao modela strukture prirodnih jezika (Chomsky, 1956.)
3. Modela fenomena u biologiji:

   - *Neuronskih mreža* koje vode do konačnih automata (McCulloh, Pitts, 1943.).
   - *Lindenmayerovi sustavi* kao modeli za razvoj organizama (Lindenmayer, 1968).

4. Modela u različitim dijelovima teorije jezika za programiranje: sintaksnoj analizi, prevođenju („kompilaciji"), obradi teksta, itd.
5. Modela matematičkih (i filozofskih) pitanja izračunivosti (Turing, 1936.).

Ova lista također predstavlja primjere primjene formalnih jezika. Novija polja primjene su u kriptografiji i računalna grafika. Teorija formalnih jezika jest dio diskretne matematike i povezana je s mnogim drugim poljima: kombinatorikom, algebrom, logikom i izračunivošću.

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 više desetljeća bavljenja teorijom formalnih jezika i njezinom praktičnom primjenom u definiciji i izradbi prevodilaca jezika za programiranje, istraživanja u mom magistarskom i doktorskom radu, poslije preko trideset i pet godina predavanja na dodiplomskim, diplomskim i doktorskim studijima, te poslije velikog broja realiziranih algoritama teorije formalnih jezika i prevodilaca, posebno predprocesora, i nekoliko desetaka realiziranih informacijskih sustava u kojima je praktična primjena teorije formalnih jezika bila prilično zastupljena, odlučio sam sva ta znanja prikazati u tri knjige.

Sadržaji knjiga, odabir pojedinih tema, primjeri i programi rezultat su mojih istraživanja u promicanju i modernizaciji nastavnih sadržaja predmeta *Formalni jezici i prevodioci* na Odsjeku za informacijske i komunikacijske znanosti Filozofskog fakulteta u Zagrebu. Predmet se sastoji od tri kolegija: *Uvod u formalne jezike i automate*, *Teorija sintaksne analize i primjene* i *Teorija prevođenja i primjene*.

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.

Ovo je prva knjiga koja u potpunosti pokriva sadržaj kolegija *Uvod u formalne jezike i automate*. Ukratko, u knjizi se definira pojam formalnog jezika i detaljno opisuje njegovo definiranje ili generiranje uz pomoć tri formalizma: regularnim izrazima, gramatikama i automatima. Sve teme su obrađene u devet poglavlja koja predstavljaju određene logičke cjeline.

0. *Osnove* je uvodno poglavlje u kojem je dan kratki repetitorij onih dijelova matematike koji su neophodni za potpuno razumijevanje ostalih poglavlja, a to su: skupovi, relacije, funkcije, matematička logika i grafovi, te uveden pojam podataka i algoritama.

1. *Uvod u teoriju formalnih jezika* sadrži osnove teorije formalnih jezika, pojam znaka i niza znakova, simbola i niza simbola, klasifikaciju jezika i formalizme za definiranje jezika.

2. *Regularni skupovi i izrazi* poglavlje je u kojem se definira klasa linearnih jezika ili jezika tipa 3. Težište je u opisu regularnih izraza koji su odigrali posebno značajnu ulogu u razvoju teorije formalnih jezika.

3. *Gramatike* su jedna od najznačajnijih klasa generatora jezika. Dana je definicija gramatike i opisano kako generira jezik. Potom je prikazana klasifikacija gramatika, načini njezina prikaza te postupci transformiranja gramatika.

4. *Automati* je poglavlje u kojem se uvode temeljni pojmovi o automatima kao generatorima jezika. Dana je klasifikacija automata i njihova ekvivalentnost s posebnim klasama jezika. Posebno je opisan konačni automat kao generator regularnih (linearnih) jezika.

Poglavlja koja slijede opisuju pojedine klase jezika, od jezika tipa 3 do tipa 0 i, na kraju, posebne klase jezika koju smo nazvali "jezici sa svojstvima".

5. *Linearni jezici* prvo je poglavlje u kojem se bavimo detaljnijim opisom jezika tipa 3 ili linearnim jezicima. Prikazana su tri načina definiranja (generiranja) linearnih jezika: linearnim gramatikama, regularnim izrazima i konačnim automatima. Posebno je prikazana ekvivalentnost konačnih automata i regularnih gramatika i dani primjeri izvođenja gramatika nekih regularnih jezika.

6. *Beskontekstni jezici* najveće je poglavlje, a posvećeno je najširoj klasi formalnih jezika – beskonteksnim jezicima, koji se mogu definirati beskontekstnim gramatikama i stogovnim automatika. Posebno se bavimo transformiranjem beskonteksnih gramatika i definiranjem formi, Chomskyjevom (CNF) i Greibachinom (GNF). U dijelu *PROGRAMI* dali smo ustroj svih algoritama transformiranja beskontekstnih gramatika u Pythonu, a u dijelu *PRIMJENE* nekoliko mogućih primjena beskontekstnih jezika.

7. *Kontekstni jezici* poglavlje je u kojem je opisana klasa kontekstnih jezika. Prije definiranja svojstva kontekstnih jezika, konteksnih gramatika i linearno ograničenih ili dvostruko-stogovnih automata koji ih generiraju, dano je proširenje beskonteksnih gramatika – indeksirane gramatike. Posebno je opisan problem izvođenja kontekstnih gramatika prikazan kroz nekoliko primjera izvođenja kontekstnih gramatika poznatih kontekstnih jezika.

8. *Jezici bez ograničenja* poglavlje je u kojem su opisani generatori jezika najviše razine – jezika tipa 0 ili jezika bez ograničenja. Osim gramatika bez ograničenja opisali smo i Turingov stroj. Posebno se bavimo izvođenjem gramatika bez ograničenja gdje smo dali primjer kako jednostavno izvesti gramatike nekoliko "teških" jezika bez ograničenja.

9. *Jezici sa svojstvima* završno je poglavlje. Posvećeno je "jezicima sa svojstvima" ili "atributnim jezicima" koji sadrže određena semantička svojstva i mogu se generirati proširenjem značenja beskontekstnih gramatika, tzv. atributnim gramatikama ili uz pomoć automata koji su također prošireni skupom svojstava i procedura. Definicija jezika sa svojstvima obuhvaća sve tipove jezika (od tipa 3 do tipa 0) i jezike za programiranje.

Danas postoji nekoliko jezika za programiranje prikladnih za ustroj algoritama teorije formalnih jezika. Na primjer, Pascal, C, Lisp, Python itd. Odlučio sam se za Python čija su sintaksa i semantika najbliža pseudokodu opisa pojedinih algoritama. Svi su programi priloženi u glavnom dijelu knjige ili prilogu. Osim toga što programi prikazuju kako se pojedini algoritmi mogu ustrojiti u Pythonu, 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 studija društveno-humanističke informatike i opće informatologije na Odsjeku za informacijske i komunikacijske znanosti, kao temeljna literatura za kolegij *Uvod u formalne jezike i automate*, ali i za studij lingvistike i sve druge studije koji se bave obradom umjetnih i prirodnih jezika.

Siguran sam da će knjiga s ovakvim sadržajem zainteresirati studente sličnih usmjerenja, inženjere informatike i računarskih znanosti, napredne srednjoškolce i mnoge druge samouke informatičare.

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.

I, bit ću vam zahvalan ako pažljivo pročitate ovu knjigu, prihvatite barem jedan njezin djelić i primijenite u svojoj praksi. Sve primjedbe i pitanja možete mi poslati na e-mail adresu:

zdovedan@hotmail.com.

U Zagrebu, svibnja 2012. godine

Autor

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 han)

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 diskretne matematike: 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. Elemente skupa pišemo između vitičastih zagrada, odvojene zarezom. Na primjer, skup brojki možemo napisati kao $Brojke=\{0,1,2,3,4,5,6,7,8,9\}$. 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, $5 \in Brojke$, $pet \notin Brojke$. 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.

Ako skup sadrži konačan broj elemenata, naziva se konačnim, a ako ima beskonačno mnogo elemenata, onda je beskonačan. Na primjer, skup $Brojke$ je konačan skup, a skup prirodnih brojeva $\mathbb{N}$, $\mathbb{N}=\{1,2,3,...\}$ je beskonačan. U našim ćemo primjerima koristiti i prošireni skup prirodnih brojeva $\mathbb{N}$, $\mathbb{N}=\{0,1,2,3,...\}$. U matematici su poznati još i skupovi cijelih brojeva, $\mathbb{Z}$, realnih brojeva, $\mathbb{R}$ i racionalnih brojeva, $\mathbb{Q}$. 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 $P=\{2, 4, 6, 8\}$ podskup je skupa $Brojke$, tj. $P \subseteq Brojke$. 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 2^n elemenata (skupova). Na primjer, partitivni skup skupa $A=\{0, 1, 2\}$ čine osam skupova:

$$\varnothing,\ \{0\},\ \{1\},\ \{2\},\ \{0,1\},\ \{0,2\},\ \{1,2\},\ \{0,1,2\}$$

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 $\mathbb{N}$ skup prirodnih brojeva, može se napisati

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

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:

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

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

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

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

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

0.2 RELACIJE

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

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

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×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, ρ⊆N×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∈N, y∈M, da im je zbroj manji ili jednak 4. Isto svojstvo može se napisati kao xρy što se čita "x je u relaciji ρ s y" ili "između x i y postoji relacija ρ".

Svojstva relacija

Kaže se da je relacija ρ na skupu S:

1) refleksivna ako je aρa za sve a iz S
2) tranzitivna ako aρb i bρc implicira aρc
3) simetrična ako aρb implicira bρa
4) asimetrična ako nikad nije aρb ni bρa

Na primjer, relacija "<" na skupu cijelih brojeva je tranzitivna, budući a<b i b<c implicira a<c.

Za relaciju ρ na skupu S kažemo da je relacija ekvivalencije (klasifikacije) ako je istovremeno refleksivna, tranzitivna i simetrična. Takva je, na primjer relacija "=" na skupu cijelih (ili realnih) brojeva.

Ako je S neprazan skup i "<" (binarna) relacija na S, kažemo da je S uređen skup u odnosu na relaciju "<" ako su za svaki par x, y∈S zadovoljeni sljedeći uvjeti:

1) za x=y vrijedi x<y ili y<x, ali ne oboje
2) ako je x<y i y<z, tada je x<z
3) x<x nije istinito ni za jedan x∈S

Vidimo da je relacije "<" tranzitivna, asimetrična i nije refleksivna. Kaže se da je "<" relacija uređenja.

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. Piše se

$$f: A \to B$$

i kaže da je A domena, a B kodomena funkcije f. 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. Ako za svaki b∈B postoji najmanje jedan a∈A tako da je b=f(a), kaže se da je f surjekcija ili preslikavanje na B.

Za funkciju $f:A\to B$ kaže se da je injekcija, odnosno injektivno prelikavanje, ako iz $f(a)=f(b)$ slijedi da je $a=b$, za sve a i b iz A. Funkcija $f:A\to B$ naziva se bijekcija (ili 1-1 preslikavanje) ako je istovremeno injekcija i surjekcija.

Neka je A neprazan skup. Svaka funkcija $\circ$ sa (iz) $A\times A$ u A naziva se unutrašnja operacija (ili algebarska operacija) na (u) A. Uređeni par $(A,\circ)$ nepraznog skupa A i funkcije $\circ: A\times A \to A$ naziva se grupoid. Zbrajanje i množenje prirodnih brojeva primjeri su algebarskih operacija na $\mathbb{N}$. Dijeljenje realnih brojeva algebarska je operacija u skupu realnih brojeva itd.

Za funkciju $f:A\to A'$ kažemo da je morfizam (ili homomorfizam) grupoida $(A,\circ)$ u grupoid $(A',\circ')$, ako f algebarsku operaciju $\circ$ sa A prenosi u algebarsku operaciju na A', tj. ako vrijedi

$$f(a \circ b) = f(a) \circ' f(b)$$

za sve $a,b\in A$.

0.4 MATEMATIČKA LOGIKA

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

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"	"Darko je stariji od brata Igora"

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

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

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
- ekvivalencija

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".

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 s 3", "10 je veće od 5" istiniti su, pa je istinita i njihova konjunkcija "6 je djeljivo s 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".

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 $a \vee b$. Ako je a istinito, a b lažno, ili a lažno, a b istinito, istinit je i složeni sud $a \vee b$. Što je, međutim, s vrijednošću istinitosti suda $a \vee b$ ako su i a i b istiniti? U prvom slučaju, ako se složena izjava smatra istinitom, govori se u neisključnoj disjunkciji, 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 $a \vee b$, složen je sud koji je lažan onda i samo onda ako su i a i b lažni. Iz te definicije slijedi tablica:

a	b	$a \vee b$
F	F	F
F	T	T
T	F	T
T	T	T

Implikacija se obično označuje sa "$\Rightarrow$" i definira na sljedeći način: ako su a i b dva suda, $a \Rightarrow b$ (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	$a \Rightarrow b$
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 $\neg a \vee b$. U većini slučajeva pogodnije je sud $a \Rightarrow b$ zamijeniti sudom $\neg(a \wedge \neg b)$. Takvo značenje implikacije ističe činjenicu da pri istinitosti $a \Rightarrow b$ 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 $a \Rightarrow b$ ne podudara uvijek po smislu sa sudom "ako je a, onda je b". Otuda, da bi se izbjegli eventualni nesporazumi, složeni je sud $a \Rightarrow b$ bolje čitati "a implicira b".

Ekvivalencija se obično označuje sa "$\equiv$" i definira na sljedeći način: ako su a i b dva suda, $a \equiv b$ (čita se "a je ekvivalentno b") složeni je sud koji je istinit ako a i b imaju istu vrijednost istinitosti (oba su istinita ili su oba neistinita). Tablica istinitosti ekvivalencije dana je sa:

a	b	$a \equiv b$
F	F	T
F	T	F
T	F	F
T	T	T

Logički izrazi

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

$(a \vee b) \wedge c$ $(a \wedge b) \Rightarrow (c \vee d)$ $\neg a \vee (b \Rightarrow c)$

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.

Dokazivanje indukcijom

Indukcija je logička metoda. To je vrsta posrednog zaključka kod kojeg polazimo od pojedinačnog ka općem, to znači da ono što vrijedi za svaki pojedinačni slučaj jedne vrste vrijedi za cijelu vrstu.

Indukcija se u matematici često koristi pri dokazu nekih teorema, posebno u teoriji brojeva. Na primjer, pretpostavimo da je $S(n)$ istinito za sve brojeve n, $n \in \mathbb{N}$. Ako je $\mathbb{N}$ (beskonačni) skup prirodnih brojeva, koristimo jednostavnu matematičku indukciju. Tada, uz pretpostavku da je n_0 najmanja vrijednost iz $\mathbb{N}$, da bi se pokazalo da je $S(n)$ istinito za sve n iz $\mathbb{N}$, postupak je sljedeći:

(1) $S(n_0)$ je istinito. To je osnovni korak indukcije.
(2) Pretpostavimo da je $S(m)$ istinito za sve $m<n$ iz $\mathbb{N}$ i pokažimo da je $S(n)$ također istinito. To je induktivni korak.

Na primjer, pretpostavimo da je

$$S(n) = 1 + 3 + 5 + \ldots + (2n-1) = n^2$$

tj. da je zbroj n neparnih prirodnih brojeva jednaka kvadratu broja n („savršeni kvadrat“). Dakle, želimo dokazati da to vrijedi za skup svih prirodnih brojeva, $\mathbb{N}$.

(1) Osnovni korak:

$$S(1) = 1 = 1^2$$

(2) Induktivni korak:

$$S(n+1) = 1 +3 +5 +...+[2n-1] +[2(n+1) -1] = n^2 +2n +1$$
$$= (n+1)^2$$

iz čega slijedi da je $S(n+1)$ također istinito.

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 računalskim znanostima. Ovdje je izložen koncept teorije grafova koji ima posebne primjene u teoriji formalnih jezika.

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.

♦ Graf

Neka je $X=\{x_1,x_2,...,x_n\}$ 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 $\Gamma_1=(X_1,\rho_1)$.

♦ Graf prema C. Bergeu

Neka je $X=\{x_1,x_2,\ldots,x_n\}$ 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 $\Delta=(X,\delta)$.

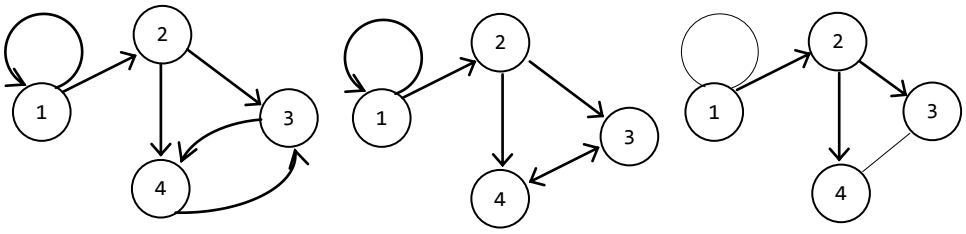
Na primjer, graf Δ iz prethodnog primjera može se definirati kao $\Delta=(X,\delta)$, gdje je $X=X_1$, a funkcija δ definirana je sa:

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

Graf može biti prikazan crtežom ili matricom susjedstva. Prikaz grafa crtežom je na sljedeći način: Najprije se nacrtaju čvorovi $x_1, \ldots, x_n$ 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

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

♦ Put u grafu

Niz čvorova $(x_0,\ldots, x_n)$, $n\geq1$, 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, $k\geq1$, 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.

♦ Kružni put

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

♦ Povezani graf

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

♦ Ulazni i izlazni stupanj

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.

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

♦ Stablo

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

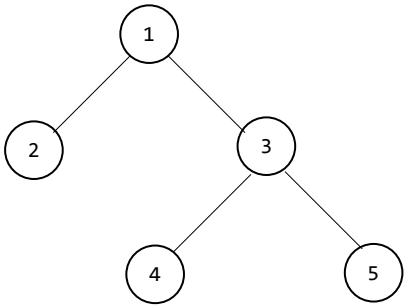
♦ Korijen i list stabla

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 je graf 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.

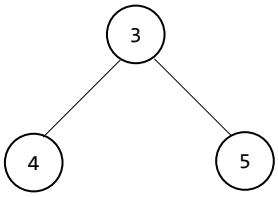
Ako je τ stablo, iz njegove definicije slijedi da ne sadrži petlje i da postoji jedinstveni put od korijena do svakog njegova čvora.

♦ Podstablo

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

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

Na kraju, evo primjera podstabla stabla iz prethodnog primjera:



0.6 PODACI

U računarstvu su podaci na razini strojnog jezika predočeni znakova binarnog alfabeta (`'0'` i `'1'`). U jezicima za programiranje visoke razine podaci ne pripadaju samo jednom skupu vrijednosti, niti su nad različitim skupovima dopuštene sve operacije. Zato se u tim jezicima uvodi pojam tipa podataka.

Tip podataka je skup vrijednosti koje imaju izvjesne zajedničke osobine. Najznačajnija od njih je skup operacija koje su definirane nad vrijednostima tog tipa. Najčešće su to sljedeći standardni tipovi podataka: brojčani (cjelobrojni i realni), logički i znakovni. Ova četiri tipa podataka nazivaju se još *primitivni tipovi*, zato što u jezicima za programiranje visoke razine predstavljaju nedjeljive cjeline i imaju izravan prikaz u memoriji kompjutera. Za naše primjene u ovoj knjizi najvažniji je znakovni tip podataka.

Polazeći od primitivnih tipova moguće je definirati strukturirane tipove - skupove vrijednosti čija struktura ima određeni smisao. One se mogu koristiti tako da se odnose na strukturiranu vrijednost u cjelini ili na pojedine komponente. U većini jezika za programiranje susreću se sljedeći strukturirani tipovi podataka: niz, skup, polje i slog. To su standardne strukture podataka, a postoji i nekoliko nestandarnih od kojih je za naše primjene najvažnija postisna lista ili stog. To je lista u kojoj se operacija dodavanja novog elementa („push“) izvodi na na njezinom vrhu (ili početku). Definirana je i operacija ukidanja ili izuzimanja elementa („pop“), također s vrha liste.

0.7 ALGORITMI

Algoritam je postupak ili pravilo za sustavno rješavanje određene vrste problema. Sastoji se od opisa konačnog skupa koraka. Svaki od njih sadrži jednu ili više izjava, a svaka izjava jednu ili više operacija.

Za prikazivanje algoritma potrebno je usvojiti određenu notaciju - tekstualnu, grafičku, skupom formula, kombiniranu ili neku drugu. Oblikovanje algoritma zahtijeva poznavanje nekoliko algoritamskih konstrukata. Tri su osnovna konstrukta, čijom se kompozicijom može oblikovati algoritam kao rješenje zadanog problema: slijed (sekvenca), grananje (selekcija) i ponavljanje (iteracija).

Slijed, odnosno sekvenca, jest skup jednog ili više koraka algoritma koji se odvijaju sekvencijalno - redno, u nizu - jedan za drugim. Broj je koraka proizvoljan. Svaki korak može biti skup od jedne ili više izjava, a može predstavljati i cijeli algoritamski konstrukt - sekvencu, selekciju ili iteraciju. Shematski, sekvenca se može prikazati kao

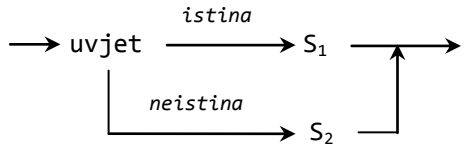
⟶ `korak1` ⟶ `...` ⟶ `korakN`

Na primjer, algoritam

1) Unesi `N` vrijednosti i zbroji ih
2) Izračunaj prosječnu vrijednost
3) Ispiši zbroj i prosječnu vrijednost

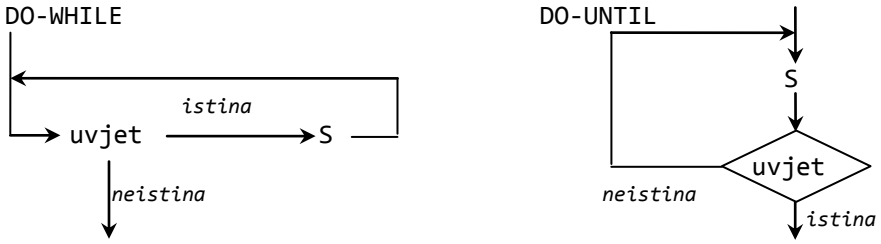
jest sekvenca triju koraka koji se odvijaju jedan za drugim, a svaki može biti algoritamski konstrukt.

Često je u algoritmima potrebno odlučiti koju od dviju ili više sekvenci treba riješiti s obzirom na postavljeni uvjet. Jednostavniji oblik grananja jest selekcija, izbor jedne od dviju sekvenci, ovisno o tome je li postavljeni uvjet istinit, kad bi se izvela prva sekvenca (S_1), ili nije istinit, kad bi se izvela druga (S_2). To je poznati `IF-THEN-ELSE` konstrukt ili naredba za odabir u većini jezika za programiranje visoke razine, a shematski se može prikazati kao



Selekcija može imati jednu granu „praznu“, tj. može biti bez `ELSE` grane.

Ponavljanje, odnosno iteracija, jest sekvenca koraka algoritma koja se odvija izvjestan broj puta, sve dok je postavljeni uvjet ispunjen (okončava se kad uvjet više nije ispunjen – kad postane neistinit), ili sve dok određeni uvjet nije ispunjen (okončava se kad je uvjet ispunjen – postao je istinit). Prvi od iterativnih konstrukata poznat je pod imenom `DO-WHILE`, drugi kao `DO-UNTIL` (ili `REPEAT-UNTIL`). Prikažimo ta dva načina iteracije shematski:



U `DO-WHILE` konstruktu (ili, kako programeri kažu „`DO-WHILE` petlja“) slijed koraka algoritma `S` zaklonjen je i bit će izvršen samo ako je uvjet ispunjen (istinit). Drugim riječima, sekvenca `S` neće biti izvršena, odnosno, bit će izvršena jedanput ili više puta, ovisno o ispunjenju uvjeta. Tada je pretpostavka da će postavljeni uvjet poslije konačnog broja koraka biti ispunjen. U suprotnom ćemo imati beskonačnu petlju!

U konstruktu `DO-UNTIL` (ili „`REPEAT-UNTIL` petlji“) prvo se izvrši slijed `S`, a nakon toga se ispituje ispunjenost uvjeta. Ponavljanje izvršenja slijeda okončava se kad uvjet prvi put bude ispunjen. To znači da se u ovom konstruktu slijed `S` izvršava najmanje jednom. U tome i jest razlika između konstrukata `DO-WHILE` i `REPEAT-UNTIL`. I ovdje je pretpostavka da postavljeni uvjet mora postati ispunjen poslije konačnog broja izvršavanja slijeda `S`. U protivnom bismo imali beskonačnu petlju!

Rekurzija je, općenito, svojstvo objekta da se definira i pomoću samoga sebe (ili, da sudjeluje u definiciji samog sebe).

U matematici, rekurzija je takav algoritam, odnosno notacija ili pravilo opisa objekta, u čijim se definicijama kao parametar ili argument pojavljuju (i) sami objekti. Tako, na primjer, funkcija faktorijela nenegativnog cijelog broja definirana je rekurzivno sljedećim pravilom:

$$0! = 1$$
$$n! = n \times (n-1)!,\ n>0$$

U definiciji funkcije $n!$ javlja se rekurzivno, kao parametar, ista funkcija, ali s drugom vrijednošću argumenta.

Rekurzija u matematici omogućuje kompaktno definiranje proizvoljno velikog skupa vrijednosti objekata minimalnim skupom izjava. Na primjer, skupom izjava:

1) 1 je prirodni broj,
2) ako je n, $n\geq 1$, prirodni broj, $n+1$ također je prirodni broj

rekurzivno je definirano jedno svojstvo prirodnih brojeva.

Pri izračunavanju vrijednosti elemenata rekurzivno definiranih objekata svaka pojava objekta zamjenjuje se njezinom definicijom. Na primjer, faktorijel broja 3 rekurzivnim algoritmom izračunava se kao

$$3! = 3\times(3-1)! = 3\times 2!$$

a $2!$ se, istim algoritmom, izračunava kao

$$2! = 2\times(2-1)! = 2\times 1!$$

Dalje je

$$1! = 1\times(1-1)! = 1\times 0!$$

Iz definicije faktorijela je $0!=1$, pa se konačno dobije:

$$3! = 3\times 2! = 3\times 2\times 1! = 3\times 2\times 1\times 0! = 3\times 2\times 1\times 1 = 6$$

Rekurzivni, kao i svaki drugi algoritam, mora biti karakteriziran sljedećim svojstvima:

1) da je opisan konačnim brojem koraka,
2) da izračunavanje vrijednosti okonča nakon konačnog broja izračunavanja.

Dok je prvo svojstvo relativno jednostavno ostvariti, za ostvarenje drugog svojstva potrebno je uvesti tzv. *uvjet okončanja*. Ako se rekurzivni algoritam promatra kao funkcija jednog ili više argumenata, kažemo da će izračunavanje vrijednosti po tom algoritmu okončati ako se pri svakom pozivu rekurzivne funkcije bar jedna od izračunatih vrijednosti „približi“ uvjetu okončanja. Tako u primjeru izračunavanja faktorijela broja n, što smo mogli napisati kao

```
f(0) = 1
f(n) = n × f(n-1), n≥1
```

uvjet okončanja je `f(0)`. U svakom se koraku vrijednost argumenta umanjuje za `1` i tako se približava uvjetu okončanja.

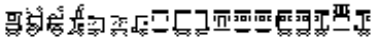
U jezicima za programiranje rekurzivni algoritmi implementirani su kao potprogrami (procedure i funkcijski potprogrami) radi mogućnosti rekurzivnog pozivanja.

Danas postoji nekoliko jezika za programiranje prikladnih za ustroj algoritama teorije formalnih jezika. Na primjer, Pascal, C, Lisp, Python itd. Odlučili smo se za Python čija su sintaksa i semantika najbliža pseudokodu opisa pojedinih algoritama.

1.
UVOD U TEORIJU FORMALNIH JEZIKA

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:

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 operacije nad jezicima.

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 nadovezivanje ili konkatenacija nizova. Na primjer, ako su x=1100 i y=0011 dva niza, tada je xy=11000011 i yx=00111100.

Duljina niza znakova jest broj znakova sadržanih u njemu. 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 znak. Naziva se prazan niz. Označivat ćemo ga s ε ili λ. Za bilo koji niz w vrijedi $\varepsilon w=w\varepsilon=w$. 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, $x^R=a_n...a_1$. Umjesto x^R može se pisati x^{-1}. Vrijedi $x=(x^{-1})^{-1}$.

Ako je $\mathcal{A}=\{a_1,...,a_n\}$ alfabet i $x\in\mathcal{A}^+$ s $N_{a_i}(x)$ označit ćemo broj pojavljivanja znaka a_i u nizu x. Ako su $\mathcal{A}=\{a_1,a_2, ..., a_m\}$ i $\mathcal{B}=\{b_1,b_2, ..., b_n\}$ dva alfabeta, definira se njihov produkt:

$$\mathcal{AB}=\{a_ib_j:\ a_i\in\mathcal{A}, b_j\in\mathcal{B}\}$$

a to je skup svih nizova znakova duljine 2 u kojima je prvi znak iz alfabeta $\mathcal{A}$, a drugi iz skupa $\mathcal{B}$. Ako je $\mathcal{A}\neq\mathcal{B}$ tada je i $\mathcal{AB}\neq\mathcal{BA}$ (produkt dvaju alfabeta nije komutativan).

Operacija potenciranja alfabeta, $\mathcal{A}^n$, $n \geq 0$, definirana je rekurzivno:

1) $\mathcal{A}^0 = \{\varepsilon\}$
2) $\mathcal{A}^n = \mathcal{A}\mathcal{A}^{n-1}$ za $n>0$

Dakle, zaključujemo da će $\mathcal{A}^n$ biti skup zvih nizova znakova nad alfabetom duljine n.

Skup svih nizova znakova koji se mogu izgraditi nad alfabetom $\mathcal{A}$, uključujući i prazan niz ε i sam alfabet, označivat ćemo sa $\mathcal{A}^*$. Vrijedi:

$$\mathcal{A}^* = \bigcup_{n=0}^{\infty} \mathcal{A}^n$$

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, a sa $\mathcal{A}^{+k}$ označivat ćemo konačan skup (podskup od $\mathcal{A}^+$) svih nizova znakova nad $\mathcal{A}$ duljine od 1 do k. Unarna operacija * poznata je i pod nazivom Kleeneova zvijezdica, jer ju je prvi put definirao Stephen Kleene. Operacija $^+$ je Kleeneov plus.

♣ **Primjer 1.1**
Nad binarnim alfabetom definirani su skupovi nizova znakova:

$\mathcal{A}^* = \{\varepsilon,0,1,00,01,10,11,000,001,010,011,100,\ldots\}$
$\mathcal{A}^{+2} = \{0,1,00,01,10,11\}$
$\mathcal{A}^3 = \{000,001,010,011,100,101,110,111\}$

1.2 DEFINICIJA FORMALNOG JEZIKA

Slijedi definicija formalnog jezika:

♦ Formalni jezik

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

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

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.

Izbor termina "jezik" može nam se učiniti neprikladnim. Međutim, mnogi se jezici, pa i hrvatski, mogu promatrati kao skupovi nizova znakova – rečenica, nad određenim alfabetom (ili abecedom). U računarskim znanostima nalazimo mnoge primjere. Poznati su jezici za programiranje, kao što su, na primjer, `C` i `Pascal`, u kojima su legalne

"rečenice" programi koji su sačinjeni od nizova znakova grupiranih u riječi nad alfabetom, najčešće podskupom ASCII znakova. Alfabeti se razliku od jednog do drugog jezika za programiranje, ali najčešće sadrže velika i mala slova engleskog alfabeta, brojke, interpunkciju i matematičke simbole.

S obzirom na to da je formalni jezik skup, u posebnom se slučaju mogu definirati dva trivijalna (ali različita!) jezika, $\mathcal{L}_1=\varnothing$ i $\mathcal{L}_2=\{\varepsilon\}$.

♣ Primjer 1.2

Evo nekoliko primjera jezika definiranih nad binarnim alfabetom $\mathcal{A}=\{0,1\}$:

$\mathcal{L}_1 = \{0(10)^i \mid i=0,1,...\} = \{0,010,01010,0101010,010101010,...\}$
$\mathcal{L}_2 = \{0^i1^i \mid i=1,2,...\} = \{01,0011,000111,...\}$
$\mathcal{L}_3 = \{xx^{-1} \mid x\in\mathcal{A}^*\} = \{\varepsilon,00,11,0000,0110,1001,1111,...\}$
$\mathcal{L}_4 = \{0^{n^2} \mid n=1,2,...\} = \{0,0000,000000000,...\}$
$\mathcal{L}_5 = \mathcal{A}^* = \{\varepsilon,0,1,00,01,10,11,000,001,010,011,100,...\}$
$\mathcal{L}_6 = \{x \mid x\in\mathcal{A}^+ \wedge N_0(x)=N_1(x)\} = \{01,10,0011,0101,0110,1010,1100,...\}$

♦ Svojstvo prefiksa

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

♣ Primjer 1.3

Jezik $\{0^n \mid n\geq 0\}$ nema svojstvo prefiksa jer je, na primjer, niz 0 svojstveni prefiks za sve rečenice (nizove) 0^n, $n>1$, dok jezik $\{0^n1 : n\geq 0\}$ 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 nadovezivanja.

♦ Produkt jezika

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 $\mathcal{L}_1\mathcal{L}_2$ nadovezivanje (ulančavanje ili konkatenacija) ili produkt jezika $\mathcal{L}_1$ i $\mathcal{L}_2$:

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

♣ Primjer 1.4

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

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

♦ Zatvarač jezika

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) $\mathcal{L}^* = \bigcup_{n\geq 0}^{\infty}\mathcal{L}^n$

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

1.3 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. To je uvijek konačan skup. 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

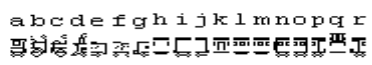
$\mathcal{V}$ = {I, V, IV, X, IX, L, XL, C, XC, D, CD, M, CM}

Ponekad ćemo definirati jezik nad rječnikom, tj. $\mathcal{L} \subseteq \mathcal{V}^*$. Rečenice takvog jezika i dalje su nizovi znakova iz $\mathcal{A}^*$, ali se mogu promatrati i kao nizovi simbola rječnika $\mathcal{V}$.

♣ Primjer 1.7

Alfabet:

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



Rječnik:

dcba

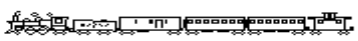
gfe

ihkj

nmlo

rqp

Rečenica:



♣ **Primjer 1.8**

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)

1.4 KLASIFIKACIJA JEZIKA

Prema hijerarhiji Chomskog jezike klasificiramo u četiri skupine (ili tipa), kao što je prikazano u sljedećoj tablici:

tip	naziv
0	bez ograničenja
1	kontekstni
2	beskontekstan
3	linearan

Da bismo odredili kojoj klasi jezika pripada neka rečenica korisno je znati svojstvo napuhavanja ("pumping lemma") kaže da svaki jezik dane klase može biti "napuhan" i pritom još uvijek pripadati danoj klasi. Jezik može biti napuhan ukoliko dovoljno dugi niz znakova jezika se može rastaviti na podnizove, od kojih neki mogu biti ponavljani proizvoljan broj puta u svrhu stvaranja novog, duljeg niza znakova koji je još uvijek u istom jeziku. Stoga, ukoliko vrijedi svojstvo napuhavanja za danu klasu jezika, bilo koji neprazni jezik u klasi će sadržavati beskonačan skup konačnih nizova znakova izgrađenih jednostavnim pravilom koje daje svojstvo napuhavanja.

1.5 DEFINIRANJE JEZIKA

Jezik je definiran kao podskup skupa svih nizova znakova (rečenica) nad alfabetom. Ako sadrži mali broj rečenica, neće ih biti problem specificirati. Međutim, najčešće je jezik ili konačan skup velikog broja rečenica ili beskonačan skup. Zbog toga bi teško bilo i nepraktično, a u slučaju beskonačnog broja rečenica i nemoguće, 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*. Primjenljiva je samo na opis jezika tipa 3.

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*.

U sljedeća tri poglavlja opisali smo regularne skupove i izraze, gramatike i automati. U preostalim poglavljima detaljno su opisane pojedine klase jezika, od tipa 3 do tipa 0.

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* $\mathcal{L}_1=\emptyset$ *i* $\mathcal{L}_2=\{\varepsilon\}$ *dva jezika, je li* $\mathcal{L}_1=\mathcal{L}_2$?

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) Znakovi alfabeta $\mathcal{A}$={0,1,2,3,4,5,6,7,8,9} *nazivaju se "brojke" ("znamenke" ili "cifre"). Nizovi znakova sačinjeni od brojki su "brojevi". Često ćemo na televiziji i radiju čuti, kad se želi istaknuti da je za određeni posao utrošeno puno novca, kako su to "velike cifre". Jesu li?*

2.
REGULARNI SKUPOVI I IZRAZI

$$\alpha+\beta = \beta+\alpha$$

$$\alpha+(\beta+\gamma) = (\alpha+\beta)+\gamma$$

$$\alpha(\beta\gamma) = (\alpha\beta)\gamma$$

$$\alpha(\beta+\gamma) = \alpha\beta+\alpha\gamma$$

```
d = (0|3|6|9)*   c = (3|6|9)d
a = (1|4|7)d     b = (2|5|8)d

R = (c| (a| bb){ab}b| (aa| b){ba}a)⁺
  = {3,6,9,12,15,18,21,24,27,30,…}
  = {3*n: n∈ℕ}
```

$$(\alpha+\beta)\gamma = \alpha\gamma+\beta\gamma$$

$$\alpha\varepsilon = \varepsilon\alpha = \alpha$$

$$\alpha^* = \alpha+\alpha^*$$

$$(\alpha^*)^* = \alpha^*$$

$$\alpha+\alpha = \alpha$$

Regularni skupovi jesu klasa jezika, tzv. regularnih jezika koji su dio linerarnih jezika ili jezika tipa 3. Imali su posebno značenje u početnom razvoju teorije formalnih jezika, krajem pedesetih godina dvadesetog stoljeća. Poslije dugog "mirovanja" ponovo su postali aktualni u primjenama na globalnoj mreži – internetu. Ovdje ćemo pokazati kako se teorija regularnih skupova i izraza može primijeniti u specifikaciji jezika.

2.1 REGULARNI SKUPOVI

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^*
   (d) (P)

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, produkta i (Kleeneove) operacije *. Izraz s regularnim skupovima može imati i zagrade da bi se naznačio prioritet izvršavanja ove tri operacije, pa najviši prioritet ima podizraz unutar zagrada, potom Kleenova operacija (potenciranje), produkt i, na kraju, unija.

♣ **Primjer 2.1**

Skupovi $A=\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$, $B=\{0\}$ i $C=\{1, 3, 5, 7, 9\}$ su regularni, (3). Regularni su i skupovi:

A^* B^* $A \cup B$ AB $A(A \cup B)^*$ $C \cup A(A \cup B)^*C$

Regularni skup $A(A \cup B)^*$ jest skup svih prirodnih brojeva, a $C \cup A(A \cup B)^*C$ skup svih neparnih prirodnih brojeva.

Svojstva regularnih skupova

Sada ćemo dati jedno svojstvo regularnih skupova čime će biti određeno je li dani skup regularni. To je "svojstvo napuhavanja", a definirano je u slijedećoj lemi:

♦ **Lema napuhavanja regularnih skupova**

Neka je $\mathcal{R}$ regularni skup. Tada postoji konstanta k takva da ako je $w \in \mathcal{R}$ i $|w| \geq k$, tada se w može napisati kao xyz, gdje je $0 < |y| \leq k$ i $xy^i z \in \mathcal{R}$ za sve $i \geq 0$.

Dakle, lema napuhavanja definira osnovno svojstvo nizova regularnog skupa da svi proizvoljno dugi nizovi (rečenice) regularnog jezika mogu biti "napuhane", tj. postoji središnji dio niza koji se ponavlja proizvoljan broj puta da bi proizveo novi niz koji je u istom jeziku. U praksi se lema napuhavanja često koristi da bi se dokazalo da dani jezik nije regularan.

♣ **Primjer 2.2**

Može se pokazati da jezik $L=\{0^n1^n:\ n>0\}$ nije regularan. Ako pretpostavimo da jest, tada za proizvoljno veliki n, 0^n1^n može biti napisano kao xyz, tako da je $y\neq\varepsilon$ i $xy^iz\in L$ za sve $i\geq 0$. Ako je $y\in 0^+$ ili $y\in 1^+$, tada $xz=xy^0z\notin L$. Ako je $y\in 0^+1^+$, tada $xyyz\notin L$. Dakle, imamo kontradikciju pa L nije regularni skup (jezik).

2.2 REGULARNI IZRAZI

Regularni izrazi (još i "pravilni izrazi") na alfabetu $\mathcal{A}$ označuju (generiraju) određene regularne skupove.

♦ Regularni izraz

Regularni izraz definira se 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:

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

Operaciju "+" ili "|" čitamo "ili". Za regularni izraz p^* koristit ćemo i notaciju $\{p\}$ (što ne treba poistovjećivati sa skupom). Ako je pp^* regularni izraz, može se napisati kao p^+ ili $p\{p\}$. Također ćemo koristiti i notaciju $\{p\}_m^n$ koja ima značenje dopisivanje izraza p najmanje m, najviše n puta, $m\leq n$. Izostavljeno m ima značenje 0, a izostavljeno n ima značenje *.

♣ **Primjer 2.3**

U sljedećoj je tablici dano nekoliko primjera regularnih izraza i jezika koje označuju (generiraju):

Regularni izraz	Jezik
a	{a}
ab	{ab}
a*	{ε,a,aa,...}
(a|b)(c|d)	{ac,ad,bc,bd}
$a^+=\{a\}_1$	{a,aa,aaa,...}
$\{a\|b\}^2$	{ε,a,b,aa,ab,ba,bb }

Ako ne postoji dvoznačnost u nekom regularnom izrazu, suvišne se zagrade mogu izbaciti. Može se zamisliti da operacija * (i +) ima najviši proritet, potom operacija nadovezivanja i, na kraju, operacija + (ili |).

♣ Primjer 2.4

Regularni izraz `101(101)*` opisuje skup (jezik) koji sadrži niz `101` ili nizove dobivene konkatenacijom niza `101`:

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

♣ Primjer 2.5

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

1) `01` označuje regularni skup {`01`}
2) `0*` označuje regularni skup {`0`}*
3) `(0|1)*` označuje regularni skup {`0,1`}*
4) `(0|1)*11` označuje skup svih nizova koji sadrže znakove `0` i `1`, ali uvijek sa sufiksom `11`.
5) `(a|b)(a|b|0|1)*` označuje skup nizova iz {`a,b,0,1`}* s prefiksom `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`.

♣ Primjer 2.6

Regularni izraz `(1|2|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9)*` označuje skup svih prirodnih brojeva, {1,2,3,4,5,6,7,8,9,10,11,...}.

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.

♦ Algebarska svojstva regularnih izraza

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) $\alpha+(\beta+\gamma) = (\alpha+\beta)+\gamma$ 3) $\alpha(\beta\gamma) = (\alpha\beta)\gamma$
4) $\alpha(\beta+\gamma) = \alpha\beta+\alpha\gamma$ 5) $(\alpha+\beta)\gamma = \alpha\gamma+\beta\gamma$ 6) $\alpha\varepsilon = \varepsilon\alpha = \alpha$
7) $\alpha^* = \alpha+\alpha^*$ 8) $(\alpha^*)^* = \alpha^*$ 9) $\alpha+\alpha = \alpha$

♣ Primjer 2.7

Jezik prirodnih brojeva djeljivih s 3 jest regularni skup {3,6,9,12,..}. Znamo da je uvjet djeljivosti prirodnog broja s 3 da zbroj njegovih znamenki mora biti djeljiv s 3. Pokušajmo sada izvesti regularni izraz koji će označivati regularni skup prirodnih brojeva djeljivih s 3. Ako zadovoljimo dani uvjet, riješili smo problem! Ali, znamo da u regularnim izrazima ne postoji operacija zbrajanja niti operacija dijeljenja. 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. Znamenke 3, 6 i 9 zadovoljavaju to svojstvo i jesu prvi članovi skupa prirodnih brojeva djeljivih s 3. I brojevi koji počinju tim znamenkama koje slijedi niz sačinjen od znamenki 0, 3, 6 i 9 također su djeljivi s 3. Ako uvedemo oznake:

```
d = (0|3|6|9)*
c = (3|6|9)d
```

tada je regularni izraz R koji označuje sve brojeve koji su djeljivi s 3, jednak c ili su to brojevi koji započinju s c, a potom slijedi niz d:

```
R = (c)
```

Ako, na primjer, 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 3 jednak 2 (a to su 2, 5 ili 8) ili dvije znamenke čiji je ostatak dijeljenja jednak 1. Na primjer, brojevi 18, 111 ili 741 djeljivi su s 3. Ili, ako broj započinje s 2, 5 ili 8 (ostatak dijeljenja s 3 jednak je 2), drugi dio broja mora sadržavati ili jednu znamenku, čiji je ostatak dijeljenja s 3 jednak 1, ili dvije znamenke čiji je ostatak dijeljenja s 3 jednak 2. Na primjer, takvi su brojevi 57, 81, 858 ili 555. Ako broj sadrži više znamenki, svako pojavljivanje niza {d} ne mijenja djeljivost s 3. Na primjer, brojevi **1**93060**104**039 i **2**322**54**3 djeljivi su s 3, jer su brojevi 114, 222 i 54 djeljivi s 3. Dakle, ako uvedemo oznake:

```
a = (1|4|7)d
b = (2|5|8)d
```

rezultirajući regularni izraz koji označuje regularni skup prirodnih brojeva djeljivih s 3 može se napisati kao:

$$R = (c|\ a\{ab\}b|\ aa\{ba\}a|\ b\{ba\}a|\ bb\{ab\}b)^+$$
$$= (c|\ (a|\ bb)\{ab\}b|\ (aa|\ b)\{ba\}a)^+$$

Na primjer, niz 2322543 djeljiv je s 3 jer je dobiven iz (b(d)bb)(ab(d)). Ako, na primjer, niz 90442693233 označimo s (cd)(a(abddd)b)(cd) zaključujemo da je djeljiv s 3, dok niz 12345678977 nije djeljiv s 3, jer je prefiks 123456789, dobiven izvođenjem (ab)(c)(ab)(c)(ab)(c), djeljiv s 3, a sufiks 77, dobiven izvođenjem aa, nije potpun (nedostaje a ili bb).

P R I M J E N E

Pokazati smo kako se teorija regularnih skupova i izraza može primijeniti u specifikaciji jezika. To je bilo njihovo osnovno značenje u početnom razvoju teorije formalnih jezika, krajem pedesetih i početkom šezdesetih godina dvadesetog stoljeća. Razvojem teorije gramatika i automata, šezdesetih godina prošloga stoljeća, regularni izrazi bili su malo "zaboravljeni". Tako je bilo tridesetak godina, do pojave interneta.

Pretraživanjem tekstova primjenom regularnih izraza bavit ćemo se u drugoj knjizi. Ovdje ćemo, s obzirom na to da se bavimo definiranjem jezika, dati proširena (novija) pravila pisanja regularnih izraza. Dakako, s obzirom na to da smo Python izabrali za naš "službeni jezik" u ovoj knjizi, dat ćemo prikaz definiranja regularnih izraza u njemu.

Regularni izrazi i Python

Ako je ASCII skup znakova, u sljedećoj je tablici dano značenje meta-simbola u pisanju regularnih izraza Pythona:

Metaznak	Značenje	Primjeri	Jezik
.	Bilo koji znak		
()	Podizraz (ili "blok")	(a)	{a}
|	Alternativa ("ili")	0|1|2|99 gr(a|e)y	{0,1,2,99} {gray,grey}
*	Ponavljanje prethodnog izraza 0, 1 ili bilo koji veći broj puta	(a)* ili a* (a|b)* go*gle	{ε,a,aa,aaa,...} {ε,a,b,aa,ab,ba,bb,...} {ggle,gogle,google,... }
?	Izostavljanje prethodnog izraza ili pojavljivanje jedanput.	colou?r M(i|a)r(k?)o ((great)?grand)? ((fa|mo)ther)	{color,colour} {Miro, Mirko, Maro, Marko} {father,mother,grand father,grand mother,great grand father,great grand mother}
+	Ponavljanje prethodnog izraza 1 ili bilo koji veći broj puta	go+gle	{gogle,google,goooogle,...}
$[a_1a_2...a_n]$	$=(a_1\|a_2\|...\|a_n)$	[abc]	{a,b,c}
$[a_i-a_j]$	$=(a_i\| a_{i+1}\| ...\| a_j)$ Skup znakova od a_i do a_j prema ASCII uređenju. Ako je redni broj a_i veći od rednog broja a_j, skup je prazan.	[A-F] (-|+)*[0-9]+.[0-9]+ ([0-9]| [1-9][0-9]| 1[0-9][0-9]| 2[0-4][0-9]| 25[0-5])	{A,B,C,D,E,F} Realni brojevi u Pascalu {0,1,2,...9, 10,...,99, 100,...,199, 200,...,249, 250,...,255}
$[a_1...a_nb_i-b_j]$	$=(a_1\|a_2\|...\|a_n\|b_i\|...\|b_j)$ Ako je znak '-' dio izraza piše se na početku ili na kraju. Ako su uglate zagrade dio izraza pišu kao [][...]	[abcq-z] [a-fu-z] [-abc]=[abc-] [][0123]	{a,b,c,q,r,s,t,u,v,w,x,y,z} {a,b,c,d,e,f,u,v,w,x,y,z} {-,a,b,c} {],[,0,1,2,3}
{n}	Izostavljanje prethodnog podizraza ili ponavljanje najviše n puta.	(0|1){2}	{ε,0,1,00,01,10,11}
{n,m}	Ponavljanje prethodnog podizraza najmanje m, najviše n puta.	A{1,8}(.|A{0,3})? gdje je A=([0-9]| [a-z]| [A-Z]|-|_|"|#|$|%| &|=|+|^|'|@)	Imena DOS datoteke
\d	=[0-9] (skup brojki)		{0,1,2,3,4,5,6,7,8,9}
\s	Razmak (blank)		
\w	=[a-zA-Z0-9_] (alfanumerički znakovi)		{0,...,9,_,A,...,Z,a,...,z}
\W	Skup svih znakova bez alfanumeričkih znakova ([a-zA-Z0-9_])		
\xFF	Znak čiji je ASCII kod jednak FF, gdje je FF heksadecimalni broj	\xA9	Znak © ako je kodna stranica Latin-1

Prioritet izvršavanja operacija jest: (1) izraz u zagradi, (2) `*`, `+`, `?` i (3)nastavljanje. Tamo gdje je prioritet očigledan bez zagrada mogu se izbrisati suvišne zagrade. Na primjer, `(a)` je jednako `a`, `(b)c` je jednako `bc` i `a(b)*` je jednako `ab*`.

U danoj tablici nije opisan potpuni skup pravila pisanja regularnih izraza. Izostavljen je dio koji se odnosi na prepoznavanje nizova znakova. Bit će dan u drugoj knjizi.

Pitanja i zadaci

1) Jesu li regularni izrazi a^+b^+ *i* $(ab)^+$ *jednaki?*

2) Napišite nekoliko rečenica (elemenata skupa) koje označuju sljedeći regularni izrazi:

a) a^+b*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)`

3) 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.*

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

5) Napišite regularne izraze koji označuju:

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

6) Napišite regularni izraz koji će generirati rimske brojeve `I,II,III,IV,...,MMMCMXCIX`.

7) Godina je prijestupna ako je djeljiva s 4 i nije djeljiva sa 100 ili ako je djeljiva s 400. Na primjer, 1900. godina nije bila prijestupna, a 2000. godina je bila prijestupna. Napišite regularni izraz koji označuje skup prijestupnih godina od 1600. do 9996. godine.

8) Napišite regularni izraz koji označuje skup datuma `2012.` *godine napisanih u obliku* `DD.MM.2012.` *Na primjer, datumi su* `01.01.2012.` *i* `29.02.2012.` *dok* `30.02.2012.` *i* `31.11.2012.` *nisu.*

9) Napišite regularni izraz koji označuje registarske oznake vozila u Bosni i Hercegovini. Ako je `s` *veliko slovo* `A, E, J, K, M, O` *ili* `T`*, na Wikipediji se može naći da su tri vrste registarskih oznaka u BiH:*

a) Stare registarske oznake, od `001-s-001` *do* `999-s-999`*, na primjer* `234-J-333`
b) Nove registarske oznake, od `s01-s-001` *do* `s99-s-999`*, na primjer* `A77-K-007`
c) Taksi vozila od `TA-000001` *do* `TA-999999`*, na primjer* `TA-222543`

10) Napišite regularni izraz koji označuje nizove `{b,aab,aba,baa,aaaab,...}`*, odnosno skup* `{a,b}*` *koji sadrži nula ili paran broj znakova* `a` *i neparan broj znakova* `b`.

3.
GRAMATIKE

3.1 DEFINICIJA GRAMATIKE

3.2 GRAMATIKA KAO GENERATOR JEZIKA

- Gramatika
- Rečenična forma
- Niz izvođenja

3.3 KLASIFIKACIJA GRAMATIKA

- Klasifikacija gramatika

3.4 PRIKAZ GRAMATIKA

Backus-Naurova forma (BNF)
Sintaksni dijagrami

3.5 TRANSFORMIRANJE GRAMATIKA

- Ekvivalentne gramatike

Supstitucija
Faktorizacija

Pitanja i zadaci

Gramatike su jedna od najznačajnijih klasa generatora jezika. Ovdje će biti razmatrane klase gramatika koje se ponekad nazivaju gramatike Chomskog ili "gramatike fraznih struktura". Poslije definicije gramatike, dana su klasifikacija, načini prikaza i postupci transformiranja gramatika.

3.1 DEFINICIJA GRAMATIKE

Gramatika jezika $\mathcal{L}$ koristi dva disjunktna skupa znakova. To su skup varijabli (neterminala ili sintaksnih kategorija), označit ćemo ga s $\mathcal{N}$, i skup terminalnih znakova (alfabet), označit ćemo ga 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.

♦ Gramatika

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

  $$\mathcal{T}\cap\mathcal{N}=\varnothing$$

- $\mathcal{P}$ 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 3.1

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, ε)}

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

S → 0A1
0A → 00A1
A → ε

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

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

piše se

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

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

$\alpha \rightarrow \varepsilon | \ \beta | \ \beta\beta | \ \beta\beta\beta | \ \ldots$

piše se $\alpha \rightarrow \{\beta\}$. Vitičaste zagrade omeđuju niz koji može biti izostavljen ili napisan jedanput, dvaput, triput, itd. Produkcija oblika:

$\alpha \rightarrow \varepsilon | \ \beta$

piše se $\alpha \rightarrow [\beta]$. Dakle, uglate zagrade omeđuju niz koji može biti izostavljen ili napisan jedanput. U daljnjem ćemo tekstu neterminale označivati velikim slovima engl. abecede. Terminali će biti mala slova engl. abecede i ostali znakovi (brojke, +, -, *, /, (,), itd.). Neterminal na početku prve produkcije bit će početni simbol.

3.2 GRAMATIKA KAO GENERATOR JEZIKA

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

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

i izabere se, proizvoljno, jedna alternativa za $\mathcal{S}$. Na primjer, $\mathcal{S} \rightarrow \alpha_1$. 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 mogu-ćnosti 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 3.2

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

S → 0S | 1

generira jezik $\mathcal{L}(\mathcal{G})$ ={1,01,001,0001,...}={0^n1: n≥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.

♦ Rečenična forma

Rečenična forma gramatike $\mathcal{G}=(\mathcal{N},\mathcal{T},\mathcal{P},\mathcal{S})$ 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.

♦ Niz izvođenja

Nad skupom $(\mathcal{N}\cup\mathcal{T})^*$ gramatike $\mathcal{G}=(\mathcal{N},\mathcal{T},\mathcal{P},\mathcal{S})$ 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 $\alpha_0, \alpha_1, \ldots, \alpha_n, \alpha_i\in(\mathcal{N}\cup\mathcal{T})^*$, n≥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 3.3

Gramatika s produkcijama:

E → T| T+E T → F| T*F F → a| (E)

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 jedna je rečenica jezika koji gramatika s danim produkcijama generira.

3.3 KLASIFIKACIJA GRAMATIKA

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

♦ Klasifikacija gramatika

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

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

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

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

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

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

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

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

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

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

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

   $\alpha \to \beta$ uz uvjet da je $|\alpha|\le|\beta|$

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

Sada možemo reći da je jezik bez ograničenja ako je generiran gramatikom tipa 0, kontekstan ako je generiran gramatikom tipa 1, beskontekstan ako je generiran gramatikom tipa 2 i linearan (ili regularan) ako je generiran gramatikom tipa 3 (ili regularnom gramatikom). 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 $\mathcal{L}_{i+1}\subseteq\mathcal{L}_i$, $0\le i<3$. Regularne gramatike generiraju najjednostavnije jezike koji mogu biti generirani regularnim izrazima. Idući prema gramatikama tipa 0 jezici su sve složeniji.

♣ Primjer 3.4

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

1) Regularna:

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

   generira jezik $\mathcal{L}=\{w: w\in\{a,b\}^*\}$ tj. regularni skup $\{a,b\}^*$ 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:

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

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

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

   $S \rightarrow CD \quad C \rightarrow aCA \mid bCB \mid \varepsilon \quad AD \rightarrow aD \quad BD \rightarrow bD \quad Aa \rightarrow aA$
   $Ab \rightarrow bA \quad Ba \rightarrow aB \quad Bb \rightarrow bB \quad D \rightarrow \varepsilon$

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

♣ Primjer 3.5

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

3 2 1 0

3.4 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 modifikacija je formalizma poznatog kao Backus-Naurova forma ili BNF. Prvi je put bio primijenjen u definiciji jezika ALGOL 60, 1963. godine i još uvijek je prisutan u praksi. Piše se 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 3.6**

Sljedeća gramatika generira "jezik kaubojskih vlakova":

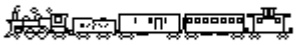
```
<kaubojski vlak>  ::= <lokomotiva><poštanska kola><dodatak>

<dodatak>         ::= {<putnička kola>}[<kola za konje>]

<lokomotiva>      ::=
```

```
<poštanska kola>  ::=
```

```
<putnička kola>   ::=
```

```
<kola za konje>   ::=
```

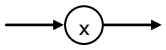
Evo nekoliko primjera rečenica generiranih ovom gramatikom:



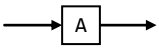
Sintaksni dijagrami

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



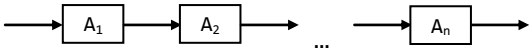
2) Neterminalni simbol A prikazan je kao



3) Produkcija oblika

$A \rightarrow A_1A_2 \ldots A_n \qquad A_i \in (\mathcal{N} \cup \mathcal{T})$

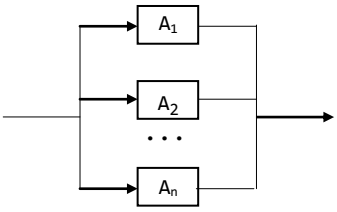
predstavlja se dijagramom



4) Produkcija oblika

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

prikazuje se dijagramom

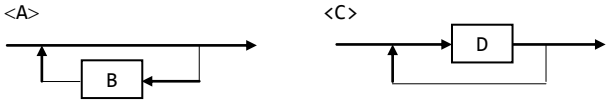


gdje je svaki A_i prikazan prema pravilima od (1) do (4). Ako je $A_i=\varepsilon$, ta se alternativa prikazuje punom crtom.

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:



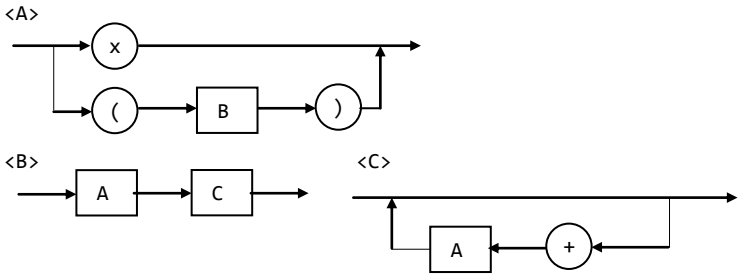
gdje su B i D prikazani dijagramima prema pravilima (1) do (4).

♣ Primjer 3.7

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

$A \rightarrow x | (B) \qquad B \rightarrow AC \qquad C \rightarrow \{+A\}$

generira jezik $\mathcal{L}(\mathcal{G})=\{x,(x),((x)),\ldots,(x+x),\ldots\}$. 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.

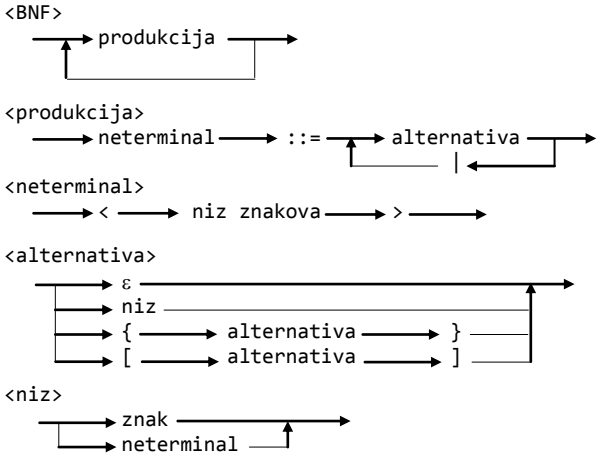
♣ Primjer 3.8

Formalizam pisanja produkcija, tj notacija kaju ovdje koristimo, regularni je jezik. Ako je $\mathcal{N}$ bilo koje veliko slovo (skup neterminala), a $\mathcal{T}$ bilo koji znak koji nije veliko slovo niti znak | (skup terminala), produkcija je definirana sa:

$$(\mathcal{T}+\mathcal{N})^*\mathcal{N}(\mathcal{T}+\mathcal{N})^* \rightarrow (\mathcal{T}+\mathcal{N})^* \; (\mid \; (\mathcal{T}+\mathcal{N})^*)^*$$

♣ Primjer 3.9

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).

3.5 TRANSFORMIRANJE GRAMATIKA

U primjeru `3.1` dana je gramatika tipa `0`, s produkcijama

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

koja generira jezik $\{0^n1^n: n>0\}$. I gramatika tipa 2, s produkcijama

$$S \rightarrow 01 \mid 0S1$$

generira isti jezik. Uspoređujući te dvije gramatike sigurno ćemo zaključiti da nisu nimalo slične, ali generiraju isti jezik! Što to znači? Pomoći će nam sljedeća definicija:

♦ Ekvivalentne gramatike

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.

U praksi će često trebati danu gramatiku, dobivenu u početnom pokušaju izvođenja gramatike danog jezika, transformirati u neku drugu, ekvivalentnu gramatiku. 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 i u drugoj knjizi, 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

Supstitucijom ćemo zamijeniti niz terminala i neterminala, ili više alternativa koje se pojavljuju kao dio dvije ili više produkcija. Pokažimo to na primjeru. Neka je dana gramatika $\mathcal{G}_1=(\mathcal{N}_1, \mathcal{T}, \mathcal{P}_1, \mathcal{S})$, gdje je $\mathcal{P}$

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

Prije svega treba uočiti da $\mathcal{G}_1$ generira jezik

$$\mathcal{L}(\mathcal{G}_1) = \{01, 0011, 000111, \ldots\} = \{0^n1^n: n \geq 1\}$$

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

$$B \rightarrow 0A$$

pa se $\mathcal{G}_1$ transformira u $\mathcal{G}_2$ u kojoj je skup produkcija:

```
S → B1
B → 0B1
B → 0A
A → ε
```

Dakle, svako pojavljivanje niza `0A` zamijenjeno je s `B` i dodana je produkcija `B→0A`. Primijetiti da je gramatika $\mathcal{G}_2$ beskontekstna (tipa `2`), dok je $\mathcal{G}_1$ gramatika bez ograničenja (tipa `0`).

Dalje se, s obzirom da je `A→ε` jedina alternativa za `A`, $\mathcal{G}_2$ može transformirati u gramatiku $\mathcal{G}_3$ s produkcijama:

```
S → B1
B → 0B1| 0
```

To nije sve. Promatrajući jezik koji gramatike $\mathcal{G}_1$, $\mathcal{G}_2$ i $\mathcal{G}_3$ generiraju, može se izvesti nova ekvivalentna gramatika, $\mathcal{G}_4$. Prvo, `01` je u jeziku, pa se može napisati:

```
S → 01
```

Sljedeća je rečenica `0011`, gdje smo s `01` označili prvu rečenicu. Ta se rečenica dobila dopisujući na početak prve rečenice znak `0`, na kraj znak `1`. Treća je rečenica `000111`, dobila se istom operacijom, ali primijenjenoj na drugu rečenicu, itd. Dakle, ako je α rečenica, tada je i $0\alpha1$ također rečenica, a to je "klasični" primjer rekurzije, pa je gramatika danog jezika:

```
S → 01| 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 → (0| 0S) 1
```

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

```
S → X1
X → 0| 0S
```

Ovo je gramatika $\mathcal{G}_5$ ekvivalentna svim prethodnim gramatikama. Dalje se produkcija za `X` može faktorizirati slijeva:

```
X → 0 (ε | S)
```

pa se uvođenjem smjene, na primjer `Y→ε|S`, dobiva gramatika $\mathcal{G}_6$ ekvivalentna svim prethodnim gramatikama:

```
S → X1
X → 0Y
Y → ε| S
```

Dakle, imamo šest ekvivalentnih gramatika od kojih je prva tipa 0, a preostalih pet tipa 2.

P R O G R A M I

Kao što je pokazano u primjeru 3.9, 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 9. 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 početku se unosi ime gramatike, prema pravilima pisanja imena datoteke u Windowsima. Na kraju se upiše @ kao oznaka kraja unosa. Evo jednoga primjera unosa gramatike:

```
E -> T + E | T
T -> T * F | F
F -> (E) | a
```

Slijedi program **Meta_BNF.py** koji koristi modul **gramatika.py** dan u prilogu knjige.

🖫 Meta-BNF.py

```
# PROGRAM Meta-BNF
# -*- coding: cp1250 -*-

from gramatika import *

     #  N   T   ->  |   #   @
Tp = (( 1, -1, -1, -1, -1,  0),    # Tablica prijelaza
      (-1, -1,  2, -1, -1, -1),
      ( 3,  3,  0, -1,  4, -1),
      ( 3,  3,  0,  2, -1,  0),
      (-1, -1,  0,  2, -1,  0) )

Ta = (( 1,  0,  0,  0,  0,  4),    # Tablica akcija
      ( 0,  0,  0,  0,  0,  0),
      ( 3,  2,  0,  0,  0,  0),
      ( 3,  2,  0,  0,  0,  5),
      ( 0,  0,  0,  0,  0,  5) )

def Ucitaj_P (): # Unos izraza
    R = raw_input ()
    R = R +"@"
    return R
```

```
def Leks_An (R, i): # Leksička analiza
    MetaSym = ['|', '#', '@'];
    while R[i] == ' ' : i = i+1
    Z = R[i]; K = -1
    if Z in MetaSym : K = MetaSym.index (Z)
    i = i +1
    if K != -1                                  : return K+3, i
    if ord(Z) in range (ord('A'), ord('Z')+1) : return 0, i
    if Z == '-' and R[i] == '>' : i = i+1;      return 2, i
    return 1, i

def Sint_An (q, NT, C): # Sintaksna analiza
    Ss = Tp [q][C]; Er = Ss == -1
    if Er :
      print ('*** sint. pogreška ***')
      return True, Ss
    else :
      NT = w[i-1]; A = Ta[q][C]
      #if A == 1 and NT not in N0 : N0.append (NT)
      #if A == 2 and NT not in T  : T.append  (NT)
      #if A == 3 and NT not in N  : N.append  (NT)

    return False, Ss

Grm, Ok, G = Ucitaj_G ('Meta-BNF', '*.grm')

if Ok : Ispisi_G (Grm, G)

N, T, P, S = G
print NL, NL, 'Upišite produkcije: '

q = 0
x = Ucitaj_P ()
w = komp(x)
while w[0] != '@':
  i = 0;
  while i < len(w):
    C, i  = Leks_An (w, i)
    Er, q = Sint_An (q, w[i-1], C)
    if Er: break

  if not Er :
    w = w[0:-1]
    L = [w[0]]; y = ''
    for i in range (3, len(w)):
      if w[i] != '|' : y += w[i]
      else           : L.append (y); y = ''
    L.append (y)
    i = 0; k = 0;
    while i < len (P) and P[i][0][0] != L[0] : i += 1
    if i < len (P) : P[i] = L
    else           : P.append (L)
  w = Ucitaj_P ()

G = (N, T, P, S)
Ispisi_G (Grm, G)
Upamti_G (Grm, P)
```

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* $\mathcal{L}_1=\emptyset$ *i* $\mathcal{L}_2=\{\varepsilon\}$ *dva jezika, je li* $\mathcal{L}_1=\mathcal{L}_2$?

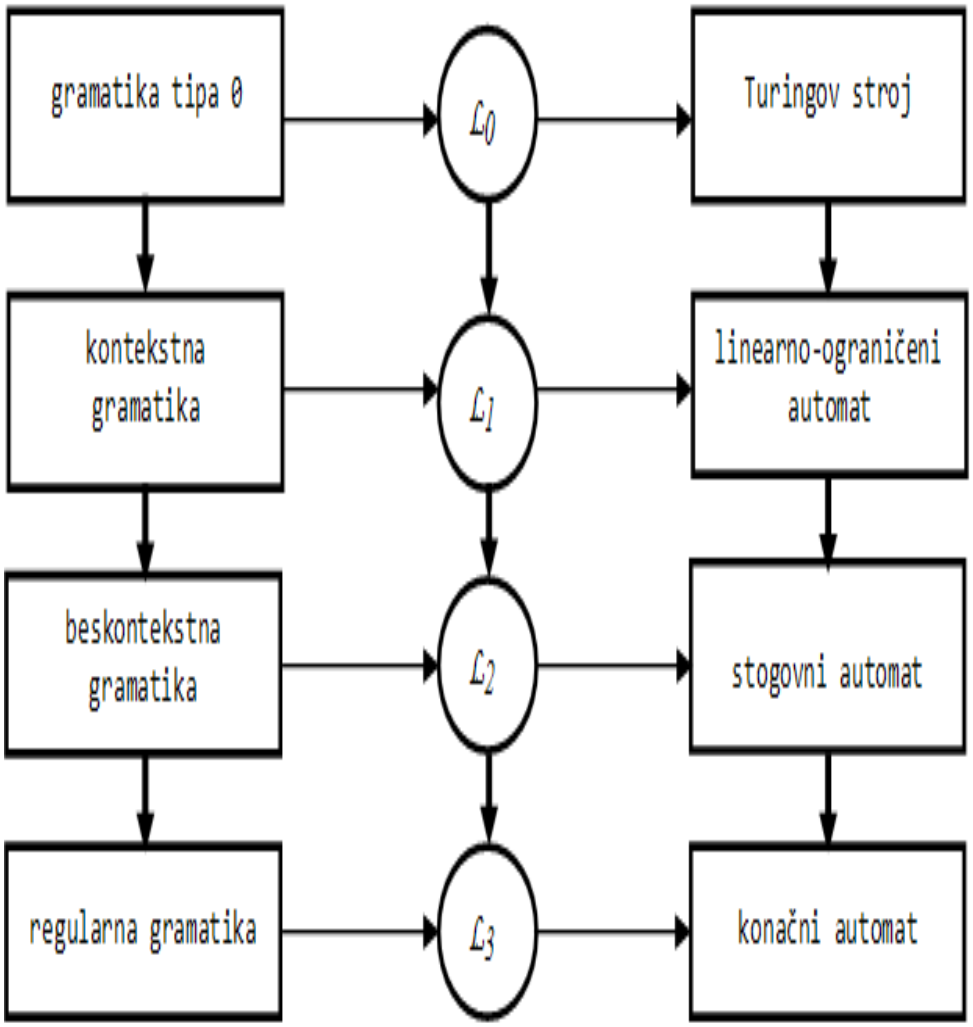
4) *Dan je alfabet* $\mathcal{A}=\{x,y\}$. *Napišite sve elemente jezika* $\mathcal{L}$ *definiranog nad* $\mathcal{A}$ *koji će imati svojstvo da sadrži nizove duljine do* 4.

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) *Znakovi alfabeta* $\mathcal{A}=\{0,1,2,3,4,5,6,7,8,9\}$ *nazivaju se "brojke" ("znamenke" ili "cifre"). Nizovi znakova sačinjeni od brojki su "brojevi". Često ćemo na televiziji i radiju čuti, kad se želi istaknuti da je za određeni posao utrošeno puno novca, kako su to "velike cifre". Jesu li?*

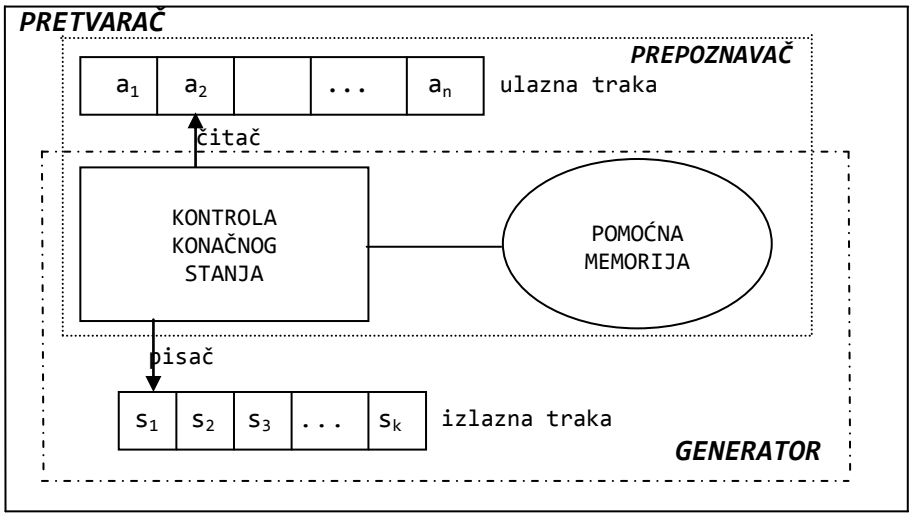
4. AUTOMATI


gramatika tipa 0
$\mathcal{L}_0$
Turingov stroj
kontekstna gramatika
$\mathcal{L}_1$
linearno-ograničeni automat
beskontekstna gramatika
$\mathcal{L}_2$
stogovni automat
regularna gramatika
$\mathcal{L}_3$
konačni automat

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

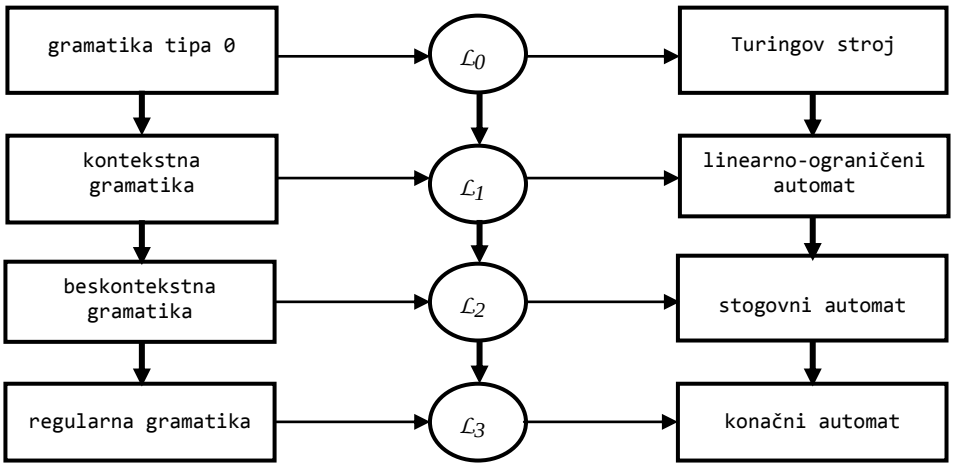
4.1 AUTOMATI I JEZICI

Opći model automata dan je na sl. 4.1. Automat koji sadrži sve navedene "dijelove" naziva se pretvarač, automat bez ulazne vrpce je generator, a automat bez izlazne vrpce je prepoznavač. U ovom ćemo pogavlju proučiti konačne generatore, koji generiraju regularne jezike, i pokazati njihovu vezu s regularnim izrazima.



Sl. 4.1 – *Opći model automata*.

Automati najčešće imaju ulogu prepoznavača. Ovisno o jeziku koji se prepoznaje, odnosno o tipu gramatike koja generira takav jezik, postoje i vrste prepoznavača dane na sljedećem crtežu.



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

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.

♦ Konačni automat

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

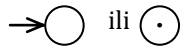
Evo primjera konačnog automata $\mathcal{M}=(Q, \Sigma, \delta, q_0, F)$

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

Dijagram prijelaza

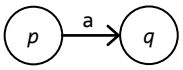
Iz definicije funkcije prijelaza zaključujemo da je to označeni, usmjereni graf čiji čvorovi odgovaraju stanjima automata, a grane su označene znakovima iz alfabeta. Ako,

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 ćemo stanje označivati s:

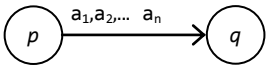


a završno s:

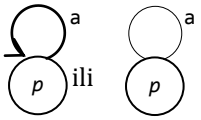
Ako je $p=\delta(q,a)$ tada postoji prijelaz iz stanja q u stanje p pa će grana u dijagramu prijelaza koja spaja q i p, s početkom u q i završetkom u p, biti označena s a:



Ako postoji više grana s početkom u q i završetkom u p, tj. ako je $p=\delta(q,a_1)=\ldots=\delta(q,a_n)$, to će biti prikazano s:

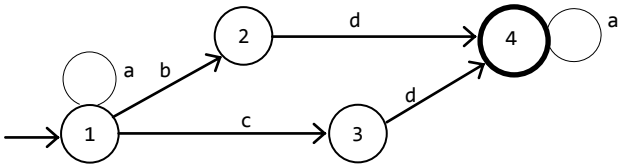


Povratak nekim prijelazom a u isto stanje, tj. ako je $p=\delta(p,a)$, dijagram prijelaza je:



♣ Primjer 4.2

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



Tablica prijelaza

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. Početno ćemo stanje označiti s $\rightarrow$ ili $>$, a konačno s $\otimes$ ili $*$.

♣ Primjer 4.3

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			

Deterministički i nedeterministički automat

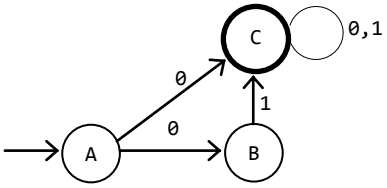
Iz definicije funkcije prijelaza vidimo da je moguć prijelaz iz tekućeg stanja u više narednih stanja s istim prijelazom a, $a \in \Sigma$, tj. da je ponašanje automata općenito nedeterminističko. Neka je $\mathcal{M}=(Q, \Sigma, \delta, q_0, F)$ konačni automat. Kažemo da je automat deterministički ako za sve $a \in \Sigma$ i sva stanja q, q' i q'' iz F i

$$\delta(q, a) = \{q', q''\}$$

slijedi da je $q'=q''$. Ako postoji barem jedan $a \in \Sigma$ tako da je $q' \neq q''$, automat je nedeterministički.

♣ Primjer 4.4

Evo jednoga primjera nedeterminističkog konačnog automata:



♣ Primjer 4.5

Dan je dio šahovske ploče, s 3×3 polja:

1	2	3
4	5	6
7	8	9

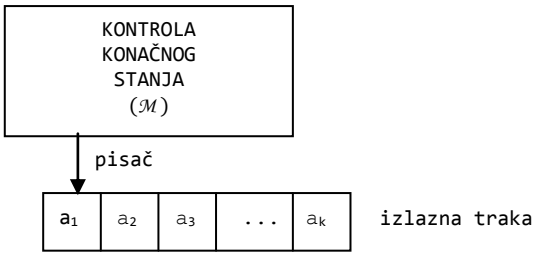
Svako polje predstavlja stanje, 1 je početno, a 9 konačno. Nedeterministički automat s tablicom prijelaza danom u nastavku predstavlja "kretanje skakača" u šahu.

	a	b	c	d	e	f	g	h
>1	6,8							
2					7,9			
3						4,8		
4							3,9	
6		1,7						
7			2,6					
8				1,3				
*9								2,4

Vidimo da je stanje 5 suvišno (nedokučivo) pa smo ga izostavili.

4.3 KONAČNI GENERATORI

Konačni automati u svojstvu generarora treći je formalizam za generiranje linearnih jezika. Na sl. 4.1 prikazan je opći model konačnog generatora, a sastoji se od kontrole konačnog stanja ($\mathcal{M}$), pisača i izlazne vrpce.



Sl. 4.3 – *Opći model konačnog generatora.*

Da bismo shvatili kako konačni generator generira rečenice linearnog jezika uvodimo definiciju njegove konfiguracije i pomak.

♦ Konfiguracija generatora

Ako je $\mathcal{M}=(Q, \Sigma, \delta, q_0, F)$ konačni generator, par $(q, w) \in Q\times\Sigma^*$ nazivat ćemo konfiguracija generatora $\mathcal{M}$. (q_0, ε) je početna konfiguracija generatora, a (q, w), $q\in F$, $w\in\Sigma^*$, jest završna konfiguracija generatora.

♦ Pomak

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

$(q, w) \vdash (q', wa)$ za sve $a\in\Sigma$

Ako postoje konfiguracije $C_0, C_1, C_2, \ldots, C_k$, tako da je $C_i \vdash C_{i+1}$, $0\le i<k$, tada je

$C_0 \vdash^k C_k$

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

$C \vdash^* C'$

što ima značenje

$C \vdash^k C'$ $k\ge 0$

a ako je postojao najmanje jedan pomak

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

Kaže se da je ulazni niz w generiran s $\mathcal{M}$ ako

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

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

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

Konačni automati generiraju regularne jezike.

♣ Primjer 4.6

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

$Q=\{A,B,C,D\}$ $\Sigma=\{0,1\}$ $q_0=A$ $\mathcal{F}=\{D\}$

$\delta(A,1)=\{B\}$ $\delta(B,0)=\{C\}$ $\delta(C,1)=\{D\}$ $\delta(D,1)=\{B\}$

generira jezik $\{(101)^n:\ n>0\}$. Na primjer:

$(A,\varepsilon)\vdash(B,1)\vdash(C,10)\vdash(D,\underline{101})$, ili $(A,\varepsilon)\vdash^3(D,\underline{101})$
$(A,\varepsilon)\vdash^{3x2}(D,(101)^2)$
...

Ako je

$$(A,\varepsilon)\vdash^{3xi}(D,(101)^i)$$

tada je

$(A,\varepsilon)\vdash^{3xi}(D,(101)^i)\vdash(B,(101)^i1)\vdash(C,(101)^i10)\vdash(D,(101)^i101)$
$(A,\varepsilon)\vdash^{3x(i+1)}(D,(101)^{i+1})$

čime smo pokazali da dani konačni automat generira jezik $\{(101)^n:\ n>0\}$.

4.4 KONAČNI GENERATORI I REGULARNI IZRAZI

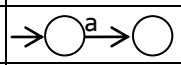
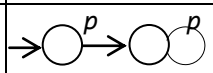
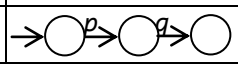
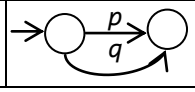
S obzirom na to da konačni generator generira rečenice linearnoga (regularnoga) jezika, postoji ekvivalentnost između ta dva formalizma. Drugim riječima, za svaki regularni izraz $\mathcal{R}$, koji označuje regularni jezik $\mathcal{L}(\mathcal{R})$, može se konstruirati nedeterministički konačni automat $\mathcal{M}(\mathcal{R})$koji generira isti jezik, jezik kojeg označuje regularni izraz $\mathcal{R}$:

$$\mathcal{L}(\mathcal{R}) = \mathcal{L}(\mathcal{M}(\mathcal{R}))$$

Često ćemo u praksi prvo definirati generator zadanog regularnog jezika, potom izvesti ekvivalentni regularni izraz, ili ćemo iz danog regularnog izraza izvesti konačni generator.

♦ Izvođenje konačnog generatora iz regularnih izraza

Ako je a iz $\mathcal{A}$ regularni izraz koji označuje regularni skup $\{a\}$, a p i q regularni izrazi koji označuju regularne skupove P i Q, redom, evo pravila izvođenja konačnog generatora (dijagrama prijelaza) iz regularnih izraza:

Regularni izraz	Regularni skup	Dijagram prijelaza
∅	∅	
ε	{ε}	
a	{a}	
(*p*)*	*P**	
(*p*)+	*P*+	
(*pq*)	*PQ*	
(*p*|*q*)	*P*∪*Q*	

♣ Primjer 4.7

Izvedimo dijagram (tablicu) prijelaza regularnog izraza `(a)*(b)*cd(e)*(f)*(g)*`

1) $q_0 a q_0 b q_1 b q_1 c q_2 d q_3 e q_3 f q_4 f q_4 g q_5 g q_5$ 2) $q_0 c q_2$ 3) $q_3 g q_5$

	a	b	c	d	e	f	g
→ q_0	q_0	q_1	q_2				
q_1		q_1	q_2				
q_2				q_3			
⊗ q_3					q_3	q_4	q_5
⊗ q_4						q_4	q_5
⊗ q_5							q_5

♣ Primjer 4.8

Deterministički konačni generator iz primjera 4.7 može se prikazati i kao nedeterministički u kojem je tablica prijelaza

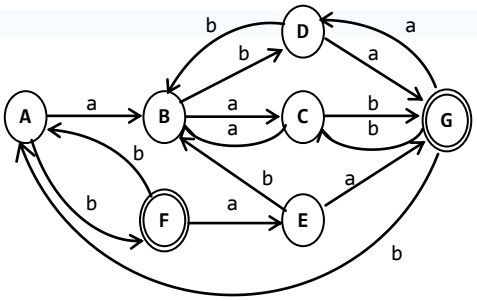
	a	b	c	d	e	f	g
→ q_0	q_0, q_1	q_1	q_2				
q_1		q_1, q_2	q_2				
q_2				q_3			
⊗ q_3					q_3, q_4	q_4	q_5
⊗ q_4						q_4, q_5	q_5
⊗ q_5							q_5

♣ Primjer 4.9

Regularni izraz koji označuje skup svih nizova koji sadrže paran broj znakova a i neparan znakova b:

```
(a(bb)*a)* (b|ab(bb)*a) (a(bb)*a|(b|ab(bb)*a)(a(bb)*a)*(b|ab(bb)*a))*
```

može se prevesti u dijagram, potom u tablicu prijelaza:



	a	b
→A	B	F
B	C	D
C	B	G
D	G	B
E	G	B
⊗ F	E	A
⊗ G	D	A,C

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 G gramatika s produkcijama:

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

Definirajte:

a) Regularni izraz koji označuje skup nizova ekvivalentan jeziku L(G).
b) Konačni automat 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) Dan je dio šahovske ploče, s 3×3 polja:

1	2	3
4	5	6
7	8	9

Ako je 1 *početno,* 9 *konačno stanje,* b *prijelaz iz bilo kojeg stanja u "bijelo" stanje i* s *prijelaz iz bio koje stanja u "sivo" stanje, definirajte funkciju prijelaza (nedeterministički automat) koji to opisuje. Na primjer,* $\delta(1,b)=\{2,4\}$, $\delta(1,s)=\{5\}$.

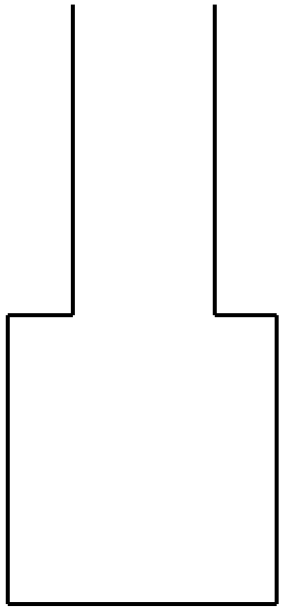
8) Dan je deterministički konačni generator jezika (a|b)*ab *u kojem je tablica prijelaza:*

```
   a b
>1 2 1
 2 2 3
*3 2 1
```

Definirajte ekvivalentni nedeterministički generator.

5.
LINEARNI JEZICI

3



Linearni jezici, ili jezici tipa 3, najjednostavniji su jezici prema klasifikaciji jezika. Promatrajući rečenice nekog jezika postavit ćemo si pitanje kojeg je tipa. Dakako, svi su jezici tipa 0, ali se postavlja pitanje koje je najniže razine. Ovdje je riječ o jezicima tipa 3, pa nas interesira kako ih "prepoznati". Zasigurno će nam u tome pomoći "svojstvo (ili lema) napuhavanja regularnih skupova" dano u poglavlju posvećenom regularnim skupovima i izrazima.

Postoje tri formalizma generiranja rečenica linearnih jezika: regularni izrazi, linearne gramatike i konačni automati (generatori). Prvi smo način opisali u drugom poglavlju. Ovdje opisujemo preostala dva.

5.1 LINEARNE GRAMATIKE

Prema hijerarhijskoj strukturi gramatika definiranoj u trećem poglavlju, linearni su jezici (ili jezici tipa 3) oni koje generira gramatika $G=(N, T, P, S)$ linearna zdesna, u kojoj su sve produkcije iz P oblika:

$A \to xB$ ili $A \to x$ $\quad A, B \in N,\ x \in T^*$

ili gramatika linearna slijeva, u kojoj su sve produkcije iz P oblika:

$A \to Bx$ ili $A \to x$ $\quad A, B \in N,\ x \in T^*$

Posebno je važna podklasa jezika linearnih zdesna, regularni jezici, a to su jezici generirani gramatikama linearnim zdesna u kojoj je svaka produkcija oblika

$A \to aB$ ili $A \to a$ $\quad A, B \in N,\ a \in T$

i jedino je dopuštena produkcija $S \to \varepsilon$, ali se tada S ne smije pojavljivati niti u jednoj alternativi ostalih produkcija, generira regularni jezik.

♣ Primjer 5.1

Evo nekoliko primjera linearnih gramatika:

1) Gramatika linearna zdesna koja generira jezik prirodnih brojeva:

$$N \to 1B|\ 2B|\ 3B|\ 4B|\ 5B|\ 6B|\ 7B|\ 8B|\ 9B$$
$$B \to\ \varepsilon|\ 0B|\ 1B|\ 2B\ |\ 3B\ |\ 4B\ |\ 5B\ |\ 6B|\ 7B|\ 8B|\ 9B$$

2) Linearni jezik $\{(01)^*1\}$ može biti generiran trima ekvivalentnim gramatikama:

a) gramatikom linearnom zdesna:

$$S \to 1|\ 10S$$

b) gramatikom linearnom slijeva:

$S \rightarrow 1 | S01$

c) regularnom gramatikom:

$S \rightarrow 1B | 1$
$B \rightarrow 0A$
$A \rightarrow 1 | 1B$

5.2 IZVOĐENJE GRAMATIKA LINEARNIH JEZIKA

Često će nam pri izvođenju neke gramatike biti dovoljno opisati rečenice jezika kojeg generira i potom dobivena pravila prevesti u produkcije gramatike.

♣ Primjer 5.2

Na primjer, ako želimo izvesti gramatiku jezika prirodnih brojeva, {1,2,3,...}, to se može definirati rekurzivnim pravilom:

1) Brojke 1, 2, 3, 4, 5, 6, 7, 8 i 9, označene s A, su prirodni brojevi.
2) Ako je N prirodni broj, prirodni brojevi su i N0, N1, N2, ..., N9.

pa je gramatika prirodnih brojeva, $\mathcal{G}_1$, definirana produkcijama:

$N \rightarrow A | N0 | N1 | N2 | N3 | N4 | N5 | N6 | N7 | N8 | N9$
$A \rightarrow 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9$

S obzirom na to da za svaki linerni jezik možemo definirati gramatiku, regularni izraz ili konačni generator, očekujemo da je moguće iz prvo definiranog formalizma izvesti preostala dva. Na primjer, ako imamo definiran regularan izraz koji generira linearni jezik iz njega ćemo moći izvesti linearnu gramatiku i konačni generator. Ili, ako imamo definiran konačni generator iz njega je moguće izvesti linearnu gramatiku i regularni izraz.

Izvođenje linearnih gramatika iz regularnih izraza

Pokažimo kako se iz regularnog izraza može izvesti regularna (linearna) gramatika. Ako su α i β regularni izrazi,

$\alpha=(a_1|a_2|...|a_m)$
$\beta=(b_1|b_2|...|b_n)$

ekvivalentne produkcije definirane su sa:

α: $A \rightarrow a_1 | a_2 | ... | a_m$
β: $B \rightarrow b_1 | b_2 | ... | b_n$

Dalje je

$\alpha\beta$: $X \rightarrow a_1B \mid a_2B \mid \ldots \mid a_mB$
$\beta\alpha$: $Y \rightarrow b_1A \mid b_2A \mid \ldots \mid b_nA$
α^*: $A \rightarrow \varepsilon \mid a_1A \mid a_2A \mid \ldots \mid a_mA$
α^+: $A \rightarrow a_1 \mid a_2 \mid \ldots \mid a_m \mid a_1A \mid a_2A \mid \ldots \mid a_mA$

♣ Primjer 5.3

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

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

S obzirom da se regularni izraz $101(101)^*$ može napisati kao $(101)^+$ možemo izvesti gramatiku linearnu zdesna:

$$S \rightarrow 101 \mid 101S$$

Regularni izraz (1|2|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9)* označuje skup svih prirodnih brojeva, {1,2,3,4,5,6,7,8,9,10,11,...}. Ako ga napišemo kao

(1|2|3|4|5|6|7|8|9)B

gdje je

B=(0|1|2|3|4|5|6|7|8|9)*

možemo izvesti gramatiku:

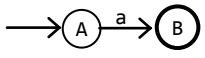
$S \rightarrow 1B \mid 2B \mid 3B \mid 4B \mid 5B \mid 6B \mid 7B \mid 8B \mid 9B$
$B \rightarrow \varepsilon \mid 0B \mid 1B \mid 2B \mid 3B \mid 4B \mid 5B \mid 6B \mid 7B \mid 8B \mid 9B$

Konačni automati i linearne gramatike

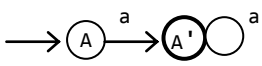
Ako konačni automati generiraju regularne jezike, onda postoji ekvivalentnost konačnih automata i gramatika tipa 3, tj. za konačni automat $\mathcal{M}$ postoji bar jedna gramatika tipa 3 tako da je $\mathcal{L}(\mathcal{M})=\mathcal{L}(\mathcal{G})$, i obrnuto, za danu gramatiku $\mathcal{G}$ tipa 3 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:

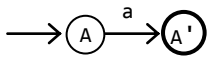
(1) $A \rightarrow a \mid aB$



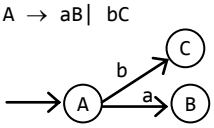
(2) $A \rightarrow a \mid aA$ (rekurzija)



(3) A → a



(4) A → aB| bC

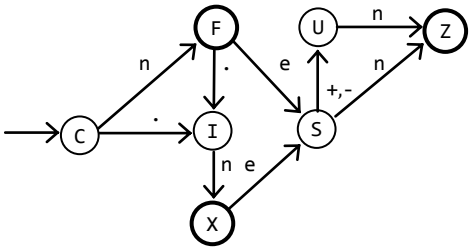


♣ Primjer 5.4

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

	a_1	a_2	...	a_j	...	a_n
→S_0	$\delta(S_0,a_1)$	$\delta(S_0,a_2)$		$\delta(S_0,a_j)$		$\delta(S_0,a_n)$
S_1	$\delta(S_1,a_1)$	$\delta(S_1,a_2)$		$\delta(S_1,a_j)$		$\delta(S_1,a_n)$
.	...	...		...		...
⊗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, produkcijama za S_i dodati praznu alternativu.

♣ Primjer 5.5

Automat u kojem je tablica prijelaza dana sa

	a	b	c	d	e	f	g
→ A	A	B	C				
B		B	C				
C				D			
⊗ D					D	E	F
⊗ E						E	F
⊗ F							F

može se prevesti u

$A \rightarrow aA|\ bB|\ cC$
$B \rightarrow bB|\ cC$
$C \rightarrow dD$
$D \rightarrow \varepsilon|\ eD|\ fE|\ gF$
$E \rightarrow \varepsilon|\ fE|\ gF$
$F \rightarrow \varepsilon|\ gF$

♣ Primjer 5.6

Generator prirodnih brojeva definiran je tablicom prijelaza:

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

Koristeći pravila izvođenja linearne gramatike dobivamo ekvivalentnu gramatiku linearnu zdesna:

$N \rightarrow 1B|\ 2B|\ 3B|\ 4B|\ 5B|\ 6B|\ 7B|\ 8B|\ 9B$
$B \rightarrow \varepsilon|\ 0B|\ 1B|\ 2B|\ 3B|\ 4B|\ 5B|\ 6B|\ 7B|\ 8B|\ 9B$

odnosno, poslije supstitucije, gramatiku:

$N \rightarrow 1B|\ 2B|\ 3B|\ 4B|\ 5B|\ 6B|\ 7B|\ 8B|\ 9B$
$B \rightarrow \varepsilon|\ 0B|\ N$

♣ Primjer 5.7

Jezik rimskih brojeva može biti generiran automatom u kojem je tablica prijelaza:

	m	d	c	l	x	v	i
→**A**	B	I	E	L	M	S	Q
m ⊗**B**	C	I	E	L	M	S	Q
mm ⊗**C**	D	I	E	L	M	S	Q
mmm ⊗**D**		I	E	L	M	S	Q
c ⊗**E**	H	H	F	L	M	S	Q
cc ⊗**F**			G	L	M	S	Q
ccc ⊗**G**				L	M	S	Q
cd ⊗**H**				L	M	S	Q
d ⊗**I**			J	L	M	S	Q
dc ⊗**J**			K	L	M	S	Q
dcc ⊗**K**			G	L	M	S	Q

		m	d	c	l	x	v	i
l	⊗**L**					N	S	Q
x	⊗**M**			P	P	O	S	Q
lx	⊗**N**					O	S	Q
xx	⊗**O**					P	S	Q
xxx	⊗**P**						S	Q
i	⊗**Q**					V	V	R
ii	⊗**R**							V
v	⊗**S**							T
vi	⊗**T**							U
vii	⊗**U**							V
	⊗**V**							

Ekvivalentna gramatika linearna zdesna definirana je skupom produkcija:

$A \to mB|\ dI|\ cE|\ lL|\ xM|\ vS|\ iQ$ $B \to \varepsilon|\ mC|\ dI|\ cE|\ lL|\ xM|\ vS|\ iQ$
$C \to \varepsilon|\ mD|\ dI|\ cE|\ lL|\ xM|\ vS|\ iQ$ $D \to \varepsilon|\ dI|\ cE|\ lL|\ xM|\ vS|\ iQ$
$E \to \varepsilon|\ mH|\ dH|\ cF|\ lL|\ xM|\ vS|\ iQ$ $F \to \varepsilon|\ cG|\ lL|\ xM|\ vS|\ iQ$
$G \to \varepsilon|\ lL|\ xM|\ vS|\ iQ$ $H \to \varepsilon|\ lL|\ xM|\ vS|\ iQ$
$I \to \varepsilon|\ cJ|\ lL|\ xM|\ vS|\ iQ$ $J \to \varepsilon|\ cK|\ lL|\ xM|\ vS|\ iQ$
$K \to \varepsilon|\ cG|\ lL|\ xM|\ vS|\ iQ$ $L \to \varepsilon|\ xN|\ vS|\ iQ$
$M \to \varepsilon|\ cP|\ xP|\ xO|\ vS|\ iQ$ $N \to \varepsilon|\ xO|\ vS|\ iQ$
$O \to \varepsilon|\ xP|\ vS|\ iQ$ $P \to \varepsilon|\ vS|\ iQ$
$Q \to \varepsilon|\ xV|\ vV|\ iR$ $R \to \varepsilon|\ iV$
$S \to \varepsilon|\ iT$ $T \to \varepsilon|\ iU$
$U \to \varepsilon|\ iV$ $V \to \varepsilon$

Poslije uvođenja supstitucija:

$Y \to vS|\ iQ$ $W \to lL|\ xM|\ Y$ $X \to dI|\ cE|\ W$

imamo još jednu ekvivalentnu gramatiku:

$A \to mB|\ X$ $B \to \varepsilon|\ mC|\ X$
$C \to \varepsilon|\ mD|\ X$ $D \to \varepsilon|\ X$
$E \to \varepsilon|\ mH|\ dH|\ cF|\ W$ $F \to \varepsilon|\ cG|\ W$
$G \to \varepsilon|\ W$ $H \to G$
$I \to \varepsilon|\ cJ|\ W$ $J \to \varepsilon|\ cK|\ W$
$K \to \varepsilon|\ cG|\ W$ $L \to \varepsilon|\ xN|\ Y$
$M \to \varepsilon|\ cP|\ xP|\ xO|\ Y$ $N \to \varepsilon|\ xO|\ Y$
$O \to \varepsilon|\ xP|\ Y$ $P \to \varepsilon|\ Y$
$Q \to \varepsilon|\ xV|\ vV|\ iR$ $R \to \varepsilon|\ iV$
$S \to \varepsilon|\ iT$ $T \to \varepsilon|\ iU$
$U \to \varepsilon|\ iV$ $V \to \varepsilon$
$W \to lL|\ xM|\ Y$ $X \to dI|\ cE|\ W$
$Y \to vS|\ iQ$

Evo i nekoliko primjera izvođenja:

$A \Rightarrow X \Rightarrow W \Rightarrow Y \Rightarrow iQ \Rightarrow \underline{i}$
$A \Rightarrow mB \Rightarrow mX \Rightarrow mW \Rightarrow mY \Rightarrow miQ \Rightarrow \underline{mi}$
$A \Rightarrow mB \Rightarrow mmC \Rightarrow mmmD \Rightarrow mmmX \Rightarrow mmmcE \Rightarrow mmmcmH \Rightarrow mmmcmG \Rightarrow mmmcmW$
$\Rightarrow mmmcmxM \Rightarrow mmmcmxcP \Rightarrow mmmcmxcY \Rightarrow mmmcmxciQ \Rightarrow mmmcmxcixV \Rightarrow \underline{mmmcmxcix}$

Može se dokazati, da svaka gramatika linearna zdesna ima ekvivalentnu gramatiku linearnu slijeva, i obrnuto. Takve su, na primjer, gramatike 2(a) i 2(b) u primjeru 5.1.

♣ Primjer 5.8

Gramatika

$S \to 101|\ 101S$

dana u primjeru 5.3 generira jezik:

$\mathcal{L} = \{101,101101,101101101,101101101101,...\} = \{(101)^+\}$

S obzirom na to da se regularni izraz $(101)^+$ može napisati kao $(101)^*101$ možemo izvesti gramatiku linearnu slijeva koja će generirati isti jezik:

$S \rightarrow 101 \mid S101$

Ili, Regularni izraz (1|2|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9)* označuje skup svih prirodnih brojeva, {1,2,3,4,5,6,7,8,9,10,11,...}. Ako ga napišemo kao

(1|2|3|4|5|6|7|8|9)B

gdje je

B=(0|1|2|3|4|5|6|7|8|9)*

možemo izvesti gramatiku:

$S \rightarrow 1B \mid 2B \mid 3B \mid 4B \mid 5B \mid 6B \mid 7B \mid 8B \mid 9B$
$B \rightarrow \varepsilon \mid 0B \mid 1B \mid 2B \mid 3B \mid 4B \mid 5B \mid 6B \mid 7B \mid 8B \mid 9B$

P R O G R A M I

GRAMATIKA JEZIKA PRIRODNIH BROJEVA DJELJIVIH S k

Treba izvesti gramatiku jezika prirodnih brojeva djeljivih sa zadanim brojem *k*, *k*>0. Ako je *k*=1 ili *k*=2, gramatika je trivijalna. Ali, već za *k*=3 ili *k*=4, naizgled rješenje nije baš jednostavno. A tek za *k*=13!

Očigledno problem djeljivosti prirodnih brojeva sa zadanim brojem k uvodi neka kontekstna svojstva koja na prvi pogled nadilaze mogućnosti linearnih jezika, pa time i linearnih gramatika. Znamo da u gramatikama nisu definirane nikakve operacije, ni množenje, ni dijeljenje. One su potpuno "ravnodušne" u odnosu na značenje ili semantiku jezika.

Sada ćemo pokazati da to ipak nije tako i da se primjenom konačnog automata, a potom izvođenja ekvivalentne gramatike linearne zdesna, taj problem može riješiti vrlo jednostavno, za bilo koji *k*! Ako je *k*=1, a tada je to cijeli skup prirodnih brojeva, tablica prijelaza jest:

	0	1	2	3	4	5	6	7	8	9
$\rightarrow S_0$		S_1	S_1	S_1	S_1	S_1	S_1	S_1	S_1	S_1
$\otimes S_1$	S_1	S_1	S_1	S_1	S_1	S_1	S_1	S_1	S_1	S_1

Dakle, prirodni su brojevi brojke 1, 2, ..., 9 ili brojevi koji počinju s 1, 2, ..., 9, a potom se dopisuje brojka 0, 1, ..., 9. Konačni automat (generator prirodnih brojeva) ima dva stanja, početno S_0, i konačno, S_1. Iz dane tablice prijelaza jednostavno se izvede ekvivalentna gramatika linearna zdesna (primjer 5.6).

Za *k*=2 jezik prirodnih brojeva djeljivih s 2 jesu svi parni brojevi. A to su brojke 2, 4, 6 i 8, odnosno bilo koji prirodni broj veći od 9 koji završava na 0, 2, 4, 6 ili 8. Generator će prirodnih brojeva djeljivih s 2 imati tri stanja, $\{S_0,S_1,S_2\}$. S_0 je početno, a S_2 konačno stanje. Ako broj počinje s 2, 4, 6 ili 8, prelazi se u konačno stanje, a ako počinje s 1, 3, 5, 7 ili 9, prelazi se u stanje S_1, gdje se vraća ako slijedi neparna brojka. U konačno će se

stanje prijeći s 0 i parnom brojkom. Sada nije problem izvesti tablicu prijelaza generatora jezika prirodnih brojeva djeljivih s 2:

	0	1	2	3	4	5	6	7	8	9
→S_0		S_1	S_2	S_1	S_2	S_1	S_2	S_1	S_2	S_1
S_1	S_2	S_1	S_2	S_1	S_2	S_1	S_2	S_1	S_2	S_1
⊗S_2	S_2	S_1	S_2	S_1	S_2	S_1	S_2	S_1	S_2	S_1

Za k=3 jezik prirodnih brojeva djeljivih s 3 jesu brojevi 3, 6, 9, 12, odnosno svaki treći prirodni broj. Generator će prirodnih brojeva djeljivih s 3 imati četiri stanja, {S_0,S_1,S_2,S_3}. S_0 je početno, a S_3 konačno stanje. Ako broj počinje s 3, 6 ili 9, prelazi se u konačno stanje, a ako počinje s 1, 4 ili 7 prelazi se u stanje S_1, s 2, 5 ili 8 prelazi se u stanje S_2, itd. Evo kompletne tablice prijelaza:

	0	1	2	3	4	5	6	7	8	9
→S_0		S_1	S_2	S_3	S_1	S_2	S_3	S_1	S_2	S_3
S_1	S_1	S_2	S_3	S_1	S_2	S_3	S_1	S_2	S_3	S_1
S_2	S_2	S_3	S_1	S_2	S_3	S_1	S_2	S_3	S_1	S_2
⊗S_3	S_3	S_1	S_2	S_3	S_1	S_2	S_3	S_1	S_2	S_3

Na primjer, ako s A, B, C i D označimo stanja S_0 do S_3 tablice prijelaza konačnog automata koji generira jezik prirodnih brojeva djeljivih s 3

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

ekvivalentna gramatika linearna zdesna koja generira isti jezik jest:

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

ili

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

Iz danih primjera izvođenja linearnih gramatika jezika prirodnih brojeva djeljivih s 1, 2 i 3 dolazimo do definicije konačnog automata jezika prirodnih brojeva djeljivih sa zadanim brojem k:

$$\mathcal{M} = (\{S_0,S_1,...,S_k\}, \{0,1,2,3,4,5,6,7,8,9\}, \delta, S_0, \{S_k\})$$

gdje je funkcija prijelaza, δ, definirana tablicom prijelaza u kojoj prijelazna stanja napisana redom od S_1 do S_k, S_1... S_k, nastavljajući taj slijed prelaskom u novi red tablice i sve do njezina kraja. Tako je, na primjer, slijed prijelaznih stanja jezika prirodnih brojeva djeljivih s 3 bio S_1 S_2 S_3 S_1 S_2 S_3 S_1... Uvijek će prijelazno stanje biti jednako S_k za one brojeve koji su djeljivi s k.

REDUCIRANJE TABLICE PRIJELAZA

Ponekad će biti moguće reducirati tablicu prijelaza dobivenu opisanim postupkom. Ako postoje dva stanja, S_p i S_q, uz uvjet da je $S_p \neq S_k$ i $S_q \neq S_k$, tako da vrijedi $\delta(S_p,a) = \delta(S_q,a)$ za sve $a \in \Sigma$, smije se ukinuti ili red s prijelaznim stanjem S_p ili s prijelaznim stanjem S_q. Ako se, na primjer, ukine red sa stanjem S_q, prije njegova ukidanja treba još promijeniti sva prijelazna stanja u tablici prijelaza, S_q, u novo prijelazno stanje, S_p.

Na primjer, tablica prijelaza jezika brojeva djeljivih s 4:

		0	1	2	3	4	5	6	7	8	9
→	S_0		S_1	S_2	S_3	S_4	S_1	S_2	S_3	S_4	S_1
	S_1	S_2	S_3	S_4	S_1	S_2	S_3	S_4	S_1	S_2	S_3
	S_2	S_4	S_1	S_2	S_3	S_4	S_1	S_2	S_3	S_4	S_1
	S_3	S_2	S_3	S_4	S_1	S_2	S_3	S_4	S_1	S_2	S_3
⊗	S_4	S_4	S_1	S_2	S_3	S_4	S_1	S_2	S_3	S_4	S_1

ima dva jednaka reda, s prijelaznim stanjima S_1 i S_3. Ukinemo li red S_3, i na svim mjestima S_3 zamijenimo s S_1, dobit ćemo reduciranu tablicu:

		0	1	2	3	4	5	6	7	8	9
→	S_0		S_1	S_2	S_1	S_4	S_1	S_2	S_1	S_4	S_1
	S_1	S_2	S_1	S_4	S_1	S_2	S_1	S_4	S_1	S_2	S_1
	S_2	S_4	S_1	S_2	S_1	S_4	S_1	S_2	S_1	S_4	S_1
⊗	S_4	S_4	S_1	S_2	S_1	S_4	S_1	S_2	S_1	S_4	S_1

ALGORITAM ZA IZVOĐENJE GENERATORA I GRAMATIKE

Sada možemo napisati program za generiranje konačnog automata (generatora) i gramatike jezika prirodnih brojeva djeljivih s *k*, *k*=1, 2, ..., 25. Početno stanje označeno je s A, a preostala stanja s B, C, ..., Z, tako da se može načiniti izravni prijevod u gramatiku u kojoj će ta ista slova označivati neterminale; A će biti startni simbol, a posljednje stanje je završno. S obzirom na to da imamo 26 slova engleskog alfabeta, priloženi program može izvesti gramatike brojeva djeljivih s 1 (cijeli skup prirodnih brojeva), 2, 3, ..., 25.

Poslije izvođenja tablice prijelaza u drugom se koraku izbacuju suvišna stanja. Potom se iz novodobivene tablice prijelaza izvede odgovarajuća gramatika (linearna zdesna). Ako je to moguće, dobivena gramatika se transformira u ekvivalentnu beskontekstnu gramatiku.

Pogledajmo što će program ispisati za gramatiku prirodnih brojeva djeljivih s 15:

```
  0 1 2 3 4 5 6 7 8 9          0 1 2 3 4 5 6 7 8 9
--------------------        --------------------
A   B C D E F G H I J       I F G H I J K L M N O
B K L M N O P B C D E       J P B C D E F G H I J
C F G H I J K L M N O       K K L M N O P B C D E
D P B C D E F G H I J       L F G H I J K L M N O
E K L M N O P B C D E       M P B C D E F G H I J
F F G H I J K L M N O       N K L M N O P B C D E
G P B C D E F G H I J       O F G H I J K L M N O
H K L M N O P B C D E       P P B C D E F G H I J
```

reducirana tablica prijelaza je:

```
  0 1 2 3 4 5 6 7 8 9
--------------------
A   B C D B C D B C D
B B C D B C P B C D B
C C D B C D B C D B C
D P B C D B C D B C D
P P B C D B C D B C D
```

a izvedena gramatika, linearna zdesna:

```
A -> 1B|2C|3D|4B|5C|6D|7B|8C|9D
B -> 0B|1C|2D|3B|4C|5P|6B|7C|8D|9B
C -> 0C|1D|2B|3C|4D|5B|6C|7D|8B|9C
D -> 0P|A
P -> ε|D
```

Gen-PB.pas *Izvođenje gramatike jezika brojeva djeljivih s* k

```
PROGRAM Generator_PB;
 { Generator gramatika prirodnih brojeva djeljivih s k, k = 1, ..., 25 }
  USES Crt;
  TYPE
    prod = ARRAY ['A'..'Z'] OF STRING;
  VAR
    k    : 1..25;           { Djelitelj         }
    TP   : ARRAY ['A'..'Z']
             OF STRING[10]; { Tablica prijelaza }
    C,                      { Pomocne varijable }
    C0,
    C1,
    F    : char;            { Konacno stanje    }
    S    : STRING;
    i    : integer;
    Ok   : boolean;
    P    : prod;
    Free : STRING;
    Izl  : text;

  CONST
    Pr   : STRING  = '0123456789';
    DN   : boolean = False;

  PROCEDURE Ispis_TP (VAR Dat : text); BEGIN
    Writeln (Dat, '  0 1 2 3 4 5 6 7 8 9');
    WriteLN (Dat, '--------------------');
    FOR C := 'A' TO F DO
      IF TP[C] <> '' THEN BEGIN
        Write (Dat, C);
        FOR i := 1 TO 10 DO Write (Dat, TP[C][i] :2);
        WriteLN (Dat)
        END;
    WriteLN (Dat)
    END;
```

```
PROCEDURE Reduciraj; BEGIN { Izbacivanje suvisnih stanja }
  C := 'B'; Ok := False;
  WHILE C < F DO BEGIN
    FOR C0 := succ(C) TO pred(F) DO BEGIN
      IF (TP[C] <> '') AND (TP[C0] = TP[C]) THEN BEGIN
        Ok := True;
        TP[C0] := ''; Free := Free +C0;
        FOR C1 := 'A' TO F DO BEGIN
          S := TP[C1];
          FOR i := 1 TO 10 DO
            IF S[i] = C0 THEN S[i] := C;
          TP[C1] := S
          END
        END
      END;
    C := succ (C)
    END
  END;

PROCEDURE Gramatika;
  VAR St : char;
  BEGIN { Izvodjenje gramatike }
    FOR C := 'A' TO F DO IF TP[C] <> '' THEN BEGIN
      FOR i := 1 TO length (Pr) DO BEGIN
        St := TP[C][i];
        IF St <> ' ' THEN
          IF P[C] <> '' THEN P[C] := P[C] +'|' +Pr[i] +St
                        ELSE P[C] := Pr[i] +St
        END;
      IF C = F THEN P[F] := chr (136) +'|' +P[F]
      END;

    FOR C := 'A' TO pred(F) DO
      IF P[C] <> '' THEN
        FOR C0 := succ(C) TO F DO
          IF (P[C0] <> '') THEN BEGIN
            i := pos (P[C], P[C0]);
            IF i > 0 THEN
              P[C0] := copy (P[C0], 1, i-1) +C +copy (P[C0],
                             i +length(P[C]), 255)
            END
    END;

PROCEDURE Ispis_GR (VAR Dat : text); BEGIN { Ispis gramatike }
  FOR C := 'A' TO pred (F) DO IF P[C] <> '' THEN
    WriteLn (Dat, C, ' -> ', P[C]);
  FOR C := succ(F) TO 'Z' DO IF P[C] <> '' THEN
    WriteLn (Dat, C, ' -> ', P[C]);
  WriteLn (Dat, F, ' -> ', P[F]);
  WriteLn (Dat)
  END;

  PROCEDURE Trans_GR; { Transformiranje gramatike }
    VAR
      S, A, Alt  : STRING;   Ok, Kraj  : boolean;
      C0, C2     : char;     B, j, k   : integer;
```

```
PROCEDURE Supstituiraj (X, Y : char); BEGIN
       S := P[Y] +'|'; i := 1;
       REPEAT
         A := '';
         WHILE (Alt[i] <> '|') AND (i < length(Alt)) DO BEGIN
           A := A +Alt[i]; i := i +1
           END;
         A := A +'|';
         j := pos (A, S);
         IF j > 0 THEN
           S := copy (S, 1, j-1) +copy (S, j+length(A), 255);
         i := i +1
         UNTIL i >= length (Alt);
       P[Y] := S +X
       END;

     PROCEDURE Supstitucija; BEGIN
       FOR C := 'A' TO 'Z' DO BEGIN
         IF P[C] <> '' THEN BEGIN
           S := P[C] +'|'; Alt := ''; A := ''; i := 1;
           WHILE (S[i] <> '|') AND (i < length(S)) DO BEGIN
             A := A +S[i]; i := i +1
             END;
           A  := A +'|'; Ok := False; C2 := succ (C);
           WHILE NOT Ok AND (C2 <= 'Z') DO BEGIN
             Ok := pos (A, P[C2] +'|') > 0;
             IF NOT Ok THEN
               C2 := succ (C2)
             END;
           IF Ok THEN BEGIN
             Alt := A; B := 1;
             REPEAT
               A := ''; i := i +1;
               WHILE (S[i] <> '|') AND (i < length(S)) DO BEGIN
                 A := A +S[i]; i := i +1
                 END;
               A  := A +'|';
               Ok := pos (A, P[C2] +'|') > 0;
               IF Ok THEN BEGIN
                 Alt := Alt +A; B := B +1
                 END
               UNTIL i >= length(S);
             IF (B > 1) AND (length(Free) > 0) THEN BEGIN
               C0 := Free[1]; Free := copy (Free, 2, 255);
               P[C0] := copy (Alt, 1, length (Alt) -1);
               Supstituiraj (C0, C);
               Supstituiraj (C0, C2);
               END
             END
           END
         END
       END;

   BEGIN
     Supstitucija;
     IF DN THEN Ispis_GR (Izl) ELSE Ispis_GR (Output);

     { Faktorizacija }
```

```
FOR C := 'A' TO 'Z' DO BEGIN
  k := 0;
  IF P[C] <> '' THEN REPEAT
    Kraj := False;
    S := P[C] +'|'; Ok := False;
    IF k > 0 THEN BEGIN
      i := 1; j := 0;
      WHILE NOT Ok AND (i < length (S) - 1) DO BEGIN
        IF S[i] = '|' THEN j := j +1;
        Ok := k = j;
        i  := i +1
        END;
      IF Ok THEN S := copy (S, i, 255)
      END;
    IF Ok OR (k = 0) THEN BEGIN
      Alt := ''; A := ''; i := 1;
      WHILE (S[i] <> '|') AND (i < length(S)) DO BEGIN
        A := A +S[i]; C0 := S[i]; i := i +1
        END;
      Alt := Alt +A +'|'; C2 := ' '; B := 1;
      S   := copy (S, i+1, 255);     i := 0;
      REPEAT
        A := ''; i := i +1; j := i;
        WHILE (S[i] <> '|') AND (i < length(S)) DO BEGIN
          A := A +S[i]; C2 := S[i]; i := i +1
          END;
        IF C2 = C0 THEN BEGIN
          Alt := Alt +A +'|'; B := B +1;
          S   := copy (S, 1, j-1) +copy (S, i+1, 255);
          i   := j-1
          END
        UNTIL i >= length (S);
      IF B > 1 THEN BEGIN
        A := '';
        FOR i := 1 TO length(Alt) -1 DO
          IF Alt[i] <> C0 THEN
            A := A +Alt[i];
        Alt := A;
        OK := False; C2 := 'A';
        WHILE NOT Ok AND (C2 < 'Z') DO BEGIN
          Ok := Alt = P[C2];
          IF NOT Ok THEN C2 := succ (C2)
          END;
        IF NOT Ok THEN BEGIN
          IF (length(Free) > 0) THEN BEGIN
            C2    := Free[1]; Free := copy (Free, 2, 255);
            P[C2] := Alt;
            Ok    := True
            END
          END;
        IF Ok THEN BEGIN
          IF length(S) >= 1 THEN S := copy (S, 1, length(S) -1);
          IF k = 0 THEN
            P[C] := C2 +C0
          ELSE BEGIN
            j := 0; i := 1;
            Ok := False;
```

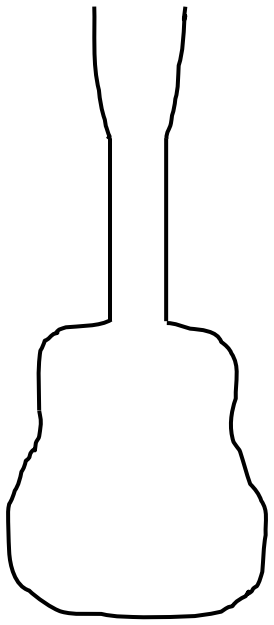
```
                  WHILE NOT Ok AND (i < length (P[C])) DO BEGIN
                    IF P[C][i] = '|' THEN j := j +1;
                    Ok := k = j;
                    i  := i +1
                    END;
                  P[C] := copy (P[C], 1, i-1) +C2 +C0
                  END;
                IF S <> '' THEN
                  P[C] := P[C] +'|' +S
                END
              END
            END
          ELSE
            Kraj := True;
          k := k +1
          UNTIL Kraj
        END;

      { Redukcija }
      FOR C := 'A' TO 'Z' DO BEGIN
        IF P[C] <> '' THEN
          FOR C2 := 'A' TO 'Z' DO IF C <> C2 THEN BEGIN
            i := pos (P[C], P[C2]);
            IF i > 0 THEN
              P[C2] := copy (P[C2],1,i-1) +C +copy (P[C2], i+length (P[C]), 255)
            END
        END
      END;

BEGIN REPEAT
  ClrScr;
  FOR C := 'A' TO 'Z' DO BEGIN
    TP[C] := ''; P[C] := '' END;
  REPEAT
    IF NOT DN THEN BEGIN
      Write ('Ispis na datoteku (D/N)? ');
      DN := upcase (ReadKey) = 'D'
      END;
    Write ('Zadaj djelitelja prirodnih brojeva (od 1 do 25) '); Readln (k);
    IF NOT (k IN [1..25]) THEN
      WriteLn ('Ponovi upis!')
    UNTIL k IN [1..25];
  Str (k :2, S); IF S[1] = ' ' THEN S[1] := '0';
  IF DN THEN BEGIN
    Assign (Izl, 'Djelj-' +S +'.GRM');
    Rewrite (Izl)
    END;

  F := chr (ord ('A') +k);
  FOR C := 'A' TO 'Z' DO TP[C] := '';
  S := '';
  FOR C := 'B' TO F DO S := S +C;

  Free := '';
  FOR C := succ (F) TO 'Z' DO Free := Free +C;

  TP['A'] := ' ' +S;
```

```
  ClrScr;
  IF NOT DN THEN BEGIN
    Writeln ('Brojevi djeljivi s ', k); WriteLN
    END;
  FOR C := 'A' TO F DO BEGIN
    IF TP[C] = '' THEN BEGIN
      i := pos (TP[pred(C)][10], S);
      IF i < k THEN
        TP[C] := copy (S, i+1, 25)
      ELSE
        TP[C] := S
      END;
    WHILE length (TP[C]) < 10 DO
      TP[C] := TP[C] +S;
    END;
  IF DN THEN
    Ispis_TP (Izl);

  Reduciraj; WHILE Ok DO BEGIN {Ispis_TP; Readln;} Reduciraj END;
  IF DN THEN Ispis_TP (Izl) ELSE Ispis_TP (Output);

  Gramatika;
  Trans_GR;
  IF DN THEN BEGIN
    Ispis_GR (Izl); Close (Izl)
    END
  ELSE Ispis_GR (Output);
  Writeln;
  Write ('<Enter> ponovi; . za kraj'); C := ReadKey
  UNTIL C = '.'
END.
```

Pitanja i zadaci

6.
BESKONTEKSTNI JEZICI

2



Beskontekstni jezici su važna klasa formalnih jezika i u razvoju teorije formalnih jezika zauzimaju posebno mjesto. Razvijene su mnoge klase beskontekstnih jezika. 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 i nekoliko formi koje imaju posebno značenje u sintaksnoj analizi beskontekstnih jezika koriste se kao osnova u raznim shemama za specificiranje prevođenja.

U ovom su poglavlju dane definicije i transformacije beskontekstnih gramatika na kojima će se temeljiti proučavanje sintaksne analize beskontekstnih jezika. Opisani su i stogovni generatori, klasa automata koji generiraju beskontekstne jezike.

6.1 SVOJSTVO BESKONTEKSNIH JEZIKA

Najprije definirajmo svojstvo beskontekstnih jezika ili jezika tipa 2. Tu nam ponovo "pomaže" teorem kojim je definirano svojstvo rečenica beskontekstnih jezika. To je Ogdensova lema iz koje slijedi svojstvo napuhavanja beskontekstnih jezika. No, prije toga dajemo definiciju pozicije u nizu znakova.

♦ Pozicija u nizu

Pozicija u nizu duljine k jest cijeli broj i tako da je $1 \le i \le k$. Kažemo da se simbol a pojavljuje na poziciji i niza w ako je $w = w_1 a w_2$ i $|w_1| = i-1$.

♣ Primjer 6.1

Ako promatramo niz w=caaab, simbol (znak) c pojavljuje se na prvoj, simbol a na drugoj, trećoj i četvrtoj, te simbol b na petoj poziciji.

♦ Ogdensova lema

Za svaku beskontekstnu gramatiku $G = (N, T, P, S)$, postoji cijeli broj $k \ge 1$ takav da ako je $z \in L(G)$, $|z| \ge k$, i postoji k ili više različitih pozicija u z, tada se z može napisati kao $z = uvwxy$, tako da vrijedi

(1) w se pojavljuje na najmanje jednoj poziciji
(2) ili su u i v ili x i y na različitim pozicijama
(3) vwx se pojavljuje na najviše k različitih pozicija
(4) postoji neterminal A tako da je

$$S \overset{+}{\Rightarrow} uAy \overset{+}{\Rightarrow} uvAxy \overset{+}{\Rightarrow} \ldots \overset{+}{\Rightarrow} uv^i Ax^i y \overset{+}{\Rightarrow} uv^i wx^i y$$

za sve $i \ge 0$ (ako je $i=0$ tada je $S \overset{+}{\Rightarrow} uAy \overset{+}{\Rightarrow} uwy$)

♦ Svojstvo napuhavanja beskontekstnih jezika

Neka je B beskontekstni jezik. Tada postoji konstanta k takva da ako je $z \in B$ i $|z| \ge k$, z može napisati kao $uvwxy$ tako da vrijedi $vx \ne \varepsilon$, $|vwx| \le k$, i za sve i je $uv^i wx^i y \in B$.

♣ Primjer 6.2

U prethodnom smo poglavlju, primjeru 5.1, dokazali da jezik $L=\{0^n1^n: n\geq 1\}$ nije regularni skup (linearni jezik). Provjerimo je li beskontekstan. Ako pretpostavimo da jest, tada je za proizvoljno dugi niz 0^i1^i $v=0$, $x=1$, $u=w=y=\varepsilon$, $k=2i$ i vrijedi $vx\neq\varepsilon$, $|vx|\leq k$, za sve $i=1,2,...,n$. Dakle, $0^i1^i\in L$.

♣ Primjer 6.3

Provjerimo je li jezik $L=\{0^n1^n2^n: n\geq 1\}$ beskontekstan. Ako jest, tada za proizvoljno dugi niz $z=0^n1^n2^n$ vrijedi $|z|=3n\geq n$. S obzirom na to da je $z\in\mathcal{L}$ možemo ga napisati kao $z=uvwxy$ pri čemu, prema svojstvu napuhavanja, vrijede uvjeti

(1) $vx\neq\varepsilon$,
(2) $|vwx|\leq n$, i
(3) $uv^iwx^iy\in\mathcal{L}$. za sve $i\geq 0$.

Budući je $0^n1^n2^n = z = uvwxy$, (1) nam kaže da vwx ne može istivremeno sadržavati 0 i 2. Dakle, ili vwx ne sadrži nijedan znak 0 ili ne sadrži nijedan znak 2, pa imamo dva slučaja:

(a) Ako vwx ne sadrži 0 tada ni vx ne sadrži 0 pa, prema (2), imamo da vx sadrži 1 ili 2. To znači da podniz uwy ima n znakova 0 ili manje od n znakova 1 ili manje od n znakova 2. Ali, (3) nam kaže da $uwy=uv^0wx^0y$ L, tako da uwy nema jednak broj znakova 0, 1 i 2, što je kontradikcija!
(b) Ako vwx ne sadrži 2 na sličnan način ćemo također doći do konradikcije.

Prema tome, dani jezik $L=\{0^n1^n2^n: n\geq 1\}$, nije beskontekstan.

6.2 BESKONTEKSTNE GRAMATIKE

U poglavlju posvećenom gramatika definirali smo gramatiku koja generira beskontekstni jezik. Bila je to gramatika u kojoj je svaka produkcija oblika

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

tj. lijeva strana svake produkcije sadrži samo neterminal (kao i kod linearnih gramatika), a alternative sadrže prazan niz ili niz neterminala i terminala. Ovdje ćemo detaljno opisati transformiranje beskonteksnih jezika.

♣ Primjer 6.4

Beskontekstna gramatika jezika prirodnih brojeva može se napisati kao

$N \rightarrow A|\ NB \qquad A \rightarrow 1|\ 2|\ 3|\ 4|\ 5|\ 6|\ 7|\ 8|\ 9 \qquad B \rightarrow \varepsilon|\ 0|\ A$

Rekurzije

Važno je svojstvo beskonteksnih gramatika da sadrže <u>rekurzije</u> u svojim produkcijama, što proizlazi iz svojstva napuhavanja beskontekstnih jezika.

♦ Rekurzije

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

1) <u>Rekurzivna slijeva</u> ako $\mathcal{P}$ sadrži produkciju oblika

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

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

$$A \to \alpha A \qquad \alpha \in (N \cup T)^+$$

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

$$A \to \alpha A \beta \qquad \alpha, \beta \in (N \cup T)^+$$

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

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

Posljedica rekurzija u produkcijama gramatike jest da gramatika rekurzivna slijeva, zdesna, odnosno, općenito rekurzivna, generira beskonačan jezik.

♣ Primjer 6.5

Regularna gramatika iz prethodnog primjera:

$$S \to aS \mid bS \mid \varepsilon$$

rekurzivna je zdesna, a beskontekstna gramatika iz istog primjera:

$$S \to aSb \mid ab$$

jest rekurzivna. Gramatika s produkcijama:

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

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

♣ Primjer 6.6

Beskontekstna gramatika jezika prirodnih brojeva može se napisati kao

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

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 (i linearnih) 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 $A \to X_1 X_2 \ldots X_{n-1} X_n$ produkcija u $\mathcal{P}$.

♦ Stablo izvođenja

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

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

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

   $$S \rightarrow X_1 \ldots X_k$$

   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.

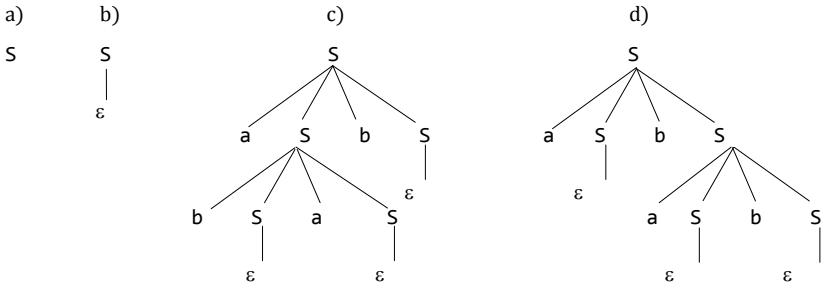
3) Inače, ako je $\mathcal{D}_1$ jedino podstablo korijena $\mathcal{D}$ i korijen od $\mathcal{D}_1$ označen je sa ε, tada je $S\rightarrow\varepsilon$ produkcija u $\mathcal{P}$.

♣ Primjer 6.7

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

$$S \rightarrow aSbS|\ bSaS|\ \varepsilon$$

dani su na sljedećoj slici:



♣ Primjer 6.8

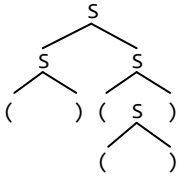
Gramatika koja generira jezik poznat kao "jezik zagrada" može se definirati kao

$$S \rightarrow ()|\ SS|\ (S)$$

Izvođenja niza w=()(())

$$S \Rightarrow SS \Rightarrow ()S \Rightarrow ()(S) \Rightarrow ()(())$$

može se prikazati stablom izvođenja:



♦ Granica stabla izvođenja

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

♣ Primjer 6.9

Granice stabala izvođenja iz primjera 6.7 jesu:

a) S b) ε c) abab d) abab

a iz primjera 6.8 ()(()).

♦ Krajnje izvođenje slijeva i zdesna

Neka je $\mathcal{G}=(\mathcal{N},\mathcal{T},\mathcal{P},S)$ 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 $A\to\delta$ 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

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

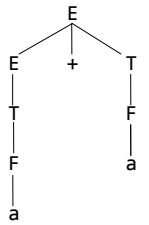
α se naziva lijeva rečenična forma. Definira se izvođenje zdesna i krajnje izvođenje zdesna, analogno izvođenju slijeva. Na svim mjestima umjesto "lijevo" treba stajati "desno", a oznaka "lm" prelazi u "rm".

♣ Primjer 6.10

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

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

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:

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

♦ Dvoznačnost gramatike

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

♣ Primjer 6.11

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

Transformiranje beskontekstnih gramatika

U trećem smo poglavlju uveli dvije općenite transformacije gramatika: supstituciju i faktorizaciju. Ovdje ćemo dati još nekoliko transformacija beskontekstnih gramatika:

- izbacivanje neupotrebljivih simbola
- izbacivanje praznih produkcija
- izbacivanje jediničnih produkcija

i dvije normalne forme:

- Chomskyjeva normalna forma (CNF)
- Greibachina normalna forma (GNF)

Također dajemo algoritme za svaku transformaciju i prevođenje u normalne forme.

IZBACIVANJE NEUPOTREBLJIVIH SIMBOLA

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

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

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.

♦ Neupotrebljivi simboli

Kaže se da je simbol $X \in \mathcal{N} \cup \mathcal{T}$ neupotrebljiv 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 6.1 *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.

♣ Primjer 6.12

Evo dva primjera provjere korektnosti definicija gramatika (procedura `ALG_6_1`):

```
GRAMATIKA:  Exp.GRM

N = { E , T , F }
T = { + , * , ( , ) , a }
S = E
P:
  E -> T+E | T
  T -> F*T | F
  F -> (E) | a

Ne = { F , T , E }

Jezik je neprazan

GRAMATIKA:  P-6-12.GRM

N = { S , A , B }
T = { a , b }
S = S
P:
  S -> a | A
  A -> AB
  B -> b

Ne = { S , B }

Jezik je neprazan
```

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

♦ Nedokučivi simboli

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 6.2 *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
$\mathcal{G}' = (\mathcal{N}', \mathcal{T}', \mathcal{P}', \mathcal{S})$

Ustroj algoritma 6.2 dan je u dijelu *PROGRAMI*. Sada se može može dati algoritam za izbacivanje neupotrebljivih simbola.

♥ Algoritam 6.3 *Izbacivanje neupotrebljivih simbola.*

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

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

Postupak
(1) Upotrijebiti algoritam 6.1 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 $\mathcal{P}_1$ sadrži samo one produkcije iz $\mathcal{P}$ koje uključuju simbole iz $N_e\cup\mathcal{T}$.
(2) Primijeniti algoritam 6.2 na $\mathcal{G}_1$ da bi se dobilo $\mathcal{G}'=(\mathcal{N}', \mathcal{T}', \mathcal{P}', \mathcal{S})$.

Korak (1) algoritma 6.3 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:

$\mathcal{S}$ `*⇒ wXy *⇒ wxy`

Da smo najprije upotrijebili algoritam 6.2, potom algoritam 6.1, ne bismo uvijek dobili konačnu gramatiku bez neupotrebljivih simbola.

♣ Primjer 6.13

Primijenimo algoritam 6.3 na gramatiku `P-6-12.GRM` iz primjera 6.12:

```
P-6-12.GRM

N = { S , A , B }
T = { a , b }
S = S
```

```
P:
  S -> a | A
  A -> AB
  B -> b
```

U koraku (1) je N_e={S,B}, tako da je $\mathcal{G}'$=({S,B}, {a,b}, {S→a, B→b}, S):

```
P-6-12.GRM'

N = { S , B }
T = { a , b }
S = S
P:
  S -> a
  B -> b
```

Dalje, primjenom algoritma 6.2, dobili bismo $V_2=V_1$={S,a}. Dakle, $\mathcal{G}$"=({S}, {a}, {S→a}, S):

```
P-6-12.GRM"

N = { S }
T = { a }
S = S
P:
  S -> a
```

Da smo prvo upotrijebili algoritam 6.2 na gramatiku $\mathcal{G}$, zaključili bismo da su svi simboli dokučivi, pa bi gramatika ostala nepromijenjena. Primjenom potom algoritma 6.1, 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 $\mathcal{S} \to \varepsilon$.

♦ Gramatika bez ε–produkcija

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 $\mathcal{S} \to \varepsilon$ i $\mathcal{S}$ se ne pojavljuje niti u jednoj alternativi (desnoj strani) ostalih produkcija iz $\mathcal{P}$.

♥ Algoritam 6.4 *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 6.1.

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

(a) Ako je $A\rightarrow\alpha_0B_1\alpha_1B_2\ldots B_k\alpha_k$ u $\mathcal{P}$, $k\geq0$, 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:

$$A\rightarrow\alpha_1X_1\alpha_1\ldots B_k\alpha_k$$

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:

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

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 6.14

Primijenimo algoritam 6.4 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
```

Isto ćemo dobiti izvršenjem algoritma ako je ulazna gramatika `#.GRM`:

```
#.GRM

N = { S }
T = { a , b }
S = S
P:
  S -> aSbS | bSaS | #

Ekvivalentna gramatika bez e-produkcija

#.###

N = { S }
T = { a , b }
S = A
P:
  A -> S | #
  S -> aSbS | aSb | abS | ab | bSaS | bSa | baS | ba
```

IZBACIVANJE JEDINIČNIH PRODUKCIJA

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

♥ Algoritam 6.5 *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 $N_A=\{B : A \stackrel{*}{\Rightarrow} B\}$ 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 6.15

Primijenimo algoritam 6.5 na gramatiku `Exp.GRM` i `Exp1.GRM`:

```
Exp.GRM

N = { E , T , F }
T = { + , * , ( , ) , a }
S = E
P:
  E -> T+E | T
  T -> F*T | F
  F -> (E) | a

Ekvivalentna gramatika bez jediničnih produkcija

N = { E , T , F }
T = { + , * , ( , ) , a }
S = E
P:
  E -> T+E | F*T | (E) | a
  T -> F*T | (E) | a
  F -> (E) | a

Exp1.GRM

N = { E , T , F }
T = { + , * , ( , ) , a }
S = E
P:
  E -> E+T | T
  T -> T*F | F
  F -> (E) | a
```

```
Ekvivalentna gramatika bez jediničnih produkcija

N = { E , T , F }
T = { + , * , ( , ) , a }
S = E
P:
  E -> E+T | T*F | (E) | a
  T -> T*F | (E) | a
  F -> (E) | a
```

♦ Svojstvena gramatika

Beskontekstna gramatika G je svojstvena (engl. *proper*) ako nema produkcija $A \to \varepsilon$ 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 dobivajući ekvivalentne gramatike bez praznih ili jediničnih produkcija, ili izbacujući suvišne produkcije. Sada ćemo definirati posebnu formu gramatika (preciznije, produkcija gramatika) poznatu kao "Chomskyjeva normalna forma" (CNF), koja je posebno važna za primjene nekih postupaka sintaksne analize danih u drugoj knjizi.

♦ Chomskyjeva normalna forma (CNF)

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

(1) $A \to BC$, $A,B,C \in N$, ili

(2) $A \to a$, $a \in T$, ili

(3) Ako je ε u jeziku $L(G)$, tada je $S \to \varepsilon$ i S se ne pojavljuje niti u jednoj alternativi (desnoj strani) ostalih produkcija iz P.

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

♥ Algoritam 6.6 *Konverzija u CNF.*

Ulaz
Svojstvena beskontekstna gramatika $G=(N, T, P, S)$.

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

Postupak
Skup produkcija P' izgraditi na sljedeći način:

(1) Dodati sve produkcije iz P oblika $A \to a$ u P'.

(2) Dodati sve produkcije iz P oblika $A \to BC$ u P'.

(3) Ako je $S \to \varepsilon$ u P dodati $S \to \varepsilon$ skupu produkcija P'.

(4) Za sve produkcije oblika $A \to X_1 \ldots X_k$ 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:

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

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

(5) Za sve produkcije oblika $A \to X_1 X_2$ iz $\mathcal{P}$, gdje je $X_1 \in \mathcal{T} \vee X_2 \in \mathcal{T}$, dodati u $\mathcal{P}'$ produkciju $A \to X'_1 X'_2$.

(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 6.16

Primijenimo algoritam 6.6 na gramatiku `Exp.GRM`

```
Exp.GRM

N = { E , T , F }
T = { + , * , ( , ) , a }
S = E
P:
  E -> T+E | T
  T -> F*T | F
  F -> (E) | a

Ekvivalentna gramatika bez jediničnih produkcija:

N = { E , T , F }
T = { + , * , ( , ) , a }
S = E
P:
  E -> T+E | F*T | (E) | a
  T -> F*T | (E) | a
  F -> (E) | a

Ekvivalentna gramatika u CNF-u

N = { E , T , F , A , B , C , D , G , H , I }
T = { + , * , ( , ) , a }
S = E
P:
  E -> TA | FB | CD | a
  T -> FB | CD | a
  F -> CD | a
  A -> GE
  B -> HT
  C -> (
  D -> EI
  G -> +
  H -> *
  I -> )
```

Ulazna je gramatika sadržavala jedinične produkcije pa je prvo bila transformirana u ekvivalentnu gramatiku bez jediničnih produkcija, potom je ta gramatika transformirana u ekvivalentnu gramatiku u CNF-u.

♣ Primjer 6.17

Evo još jednog primjera transformacije gramatike u CNF:

```
CNF.GRM

N = { S , A , B }
T = { a , b }
S = S
P:
  S -> aAB | BA
  A -> BBB | a
  B -> AS | b

Ekvivalentna gramatika u CNF-u

N = { S , A , B , C , D , E }
T = { a , b }
S = S
P:
  S -> CD | BA
  A -> BE | a
  B -> AS | b
  C -> a
  D -> AB
  E -> BB
```

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

$S \to BA \quad A \to a \quad B \to AS \mid b$

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

$$S \to a'\langle AB\rangle \mid BA \quad A \to B\langle BB\rangle \mid a \quad B \to AS \mid b$$
$$\langle AB\rangle \to AB \quad \langle BB\rangle \to BB \quad a' \to a$$

Ako uvedemo supstituciju C za a', D za $\langle AB\rangle$ i E za $\langle BB\rangle$, dobili bi rezultirajuću gramatiku kao što je gore prikazano.

GREIBACHINA NORMALNA FORMA (GNF)

Završavamo ovo potpoglavlje s još jednom formom beskontekstnih gramatika, tzv. "Greibachinom normalnom formom" (GNF), koju je definirala Sheila Adele Greibach. 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 6.7. Najprije definicija Greibachine normalne forme:

♦ Greibachina normalna forma (GNF)

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

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

Ako gramatika nije rekurzivna slijeva, moguće je naći linearno uređenje označeno s $<$ na skupu neterminala $\mathcal{N}$ tako da ako je $A \to B\alpha$ produkcija u $\mathcal{P}$, vrijedi $A<B$. Može se dokazati da svaka beskontekstna gramatika može biti transformirana u ekvivalentnu beskontekstnu gramatiku koja je u Greibachinoj normalnoj formi.

♥ Algoritam 6.7 *Konverzija gramatike u Greibachinu 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 Greibachinoj 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 $A_i \to A_j\alpha$, $j>i$, s $A_i \to \beta_1\alpha|\ldots|\beta_m\alpha$, gdje je $A_j \to \beta_1|\ldots|\beta_m$, 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 $\mathcal{S} \to \varepsilon$, ako postoji) počinju terminalom. U svakoj produkciji, $A \to aX_1\ldots X_k$, 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 $X'_j \to X_j$.

♣ Primjer 6.18

Transformirajmo gramatiku `Exp.GRM` u GNF:

```
Exp.GRM

N = { E , T , F }
T = { + , * , ( , ) , a }
S = E
P:
  E -> T+E | T
  T -> F*T | F
  F -> (E) | a

Ekvivalentna gramatika u GNF-u

N = { E , T , F }
T = { + , * , ( , ) , a }
S = E
P:
  E -> (EABTCE | aCE | (EACE | aBTCE | (EABT | a | (EA | aBT
  T -> aBT | (EABT | a | (EA
  F -> (EA | a
  A -> )
  B -> *
  C -> +
```

♣ Primjer 6.19

Ulazna gramatika ne smije biti rekurzivna slijeva, kao što je, na primjer, gramatika:

```
Exp-1.GRM

N = { E , T , F }
T = { + , * , ( , ) , a }
S = E
P:
  E -> T | E+T
  T -> F | T*F
  F -> (E) | a

Ekvivalentna gramatika u GNF-u

N = { E , T , F }
T = { + , * , ( , ) , a }
S = E
P:
  E -> (EA | a | EBT
  T -> a | (EA | TCF
  F -> (EA | a
  A -> )
  B -> +
  C -> *
```

Dobivena je ekvivalentna gramatika koja nije u GNF-u. Ako se dana gramatika transformira u ekvivalentnu gramatiku bez rekurzija slijeva (a takva je gramatika `Exp.GRM`), na primjer u gramatiku:

```
Exp2.GRM

N = { E , T , F , A , B }
T = { ( , ) , a , + , * }
S = E
P:
  E -> T | TA
  T -> F | FB
  F -> (E) | a
  A -> +T | +TA
  B -> *F | *FB
```

Prvo uvedimo linearno uređenje na skupu neterminala: A<E<B<T<F. Neterminal F je najveći u tom uređenju i sve njegove alternative počinju terminalom. T je sljedeći simbol s produkcijama T→F|FB, tako da se F supstituira sa svojim alternativama, pa se dobije:

```
T → (E) |a |(E)B |aB
```

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

```
E → (E)| a| (E)B| aB| (E)A| aA| (E)BA| aBA
```

A-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 C, pa je rezultirajuća gramatika u Greibachinoj normalnoj formi:

```
N = { E , T , F , A , B, C }
T = { ( , ) , a , + , * }
S = E
P:
  E -> (EC | aB | (ECB | a | (ECA | aBA | (ECBA | aA
  T -> a | (EC | aB | (ECB
  F -> (EC | a
  A -> +T | +TA
  B -> *F | *FB
  C -> )
```

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.

♦ Operatorska gramatika

Kaže se da je CFG operatorska gramatika ako nijedna njezina produkcija ne sadrži u alternativama dva susjedna neterminala. Može se dokazati (što nije predmet našeg bavljenja) da svaka CFG ima ekvivalentnu operatorsku gramatiku.

♣ Primjer 6.20

Već smo u našim primjerima imali operatorske gramatike. Na primjer, takva je gramatika:

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

6.3 STOGOVNI AUTOMATI

Pokazali smo da je konačni automat ekvivalentan gramatikama tipa 3. Konačni generator, na primjer, ne bi mogao generirati rečenice beskontekstnog jezika $\{a^n b^n : n \geq 1\}$ jer bi to zahtijevalo beskonačan broj stanja da se zapamte prijelazi s a da bi se potom načinilo jednako toliko prijelaza s b. Problem može biti riješen uvođenjem pomoćne memorije, a za generiranje rečenica beskontekstnog jezika lako se može pokazati da pomoćna memorija sa strukturom stoga rješava problem. Takve automate nazivamo "stogovnim". Dakle, stogovni automat, koji ima stog kao pomoćnu memoriju, drugi je formalizam za definiranje (i prepoznavanje) beskontekstnog jezika.

♦ Stogovni automat

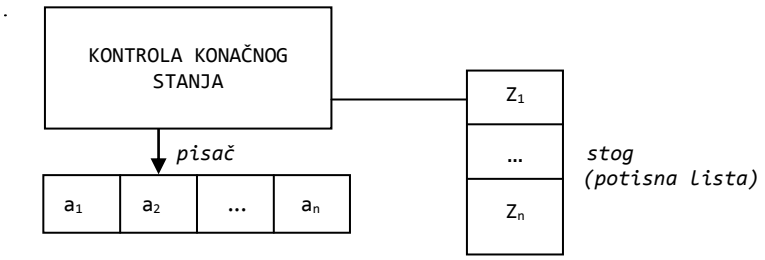
Stogovni automat PDA (Push Down Automata) jest uređena sedmorka, $\mathcal{P}=(Q, \Sigma, \Gamma, \delta, q_0, Z_0, \mathcal{F})$, gdje su:

- 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 stoga (potisne liste), $Z_0 \in \Gamma$
F skup završnih stanja, $F \subseteq Q$

Stogovni generator jest stogovni automat bez ulazne vrpce i čitača, sl. 6.1.



Sl. 6.1 – *Stogovni generator.*

♦ Funkcija prijelaza stogovnog automata

Funkcija prijelaza stogovnog automata definirana je kao

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

odnosno

$$\delta(q,a,X)=\{(p,Y),\ldots\}$$

gdje su:

q tekuće stanje, $q \in Q$
a znak (prijelaz), $a \in \Sigma$, ili ε
X simbol na vrhu stoga, $X \in \Gamma$
p naredno stanje, $p \in Q$
Y niz znakova stoga, $Y \in \Gamma^*$:
= ε za pop(X)
= X vrh stoga se ne mijenja
= $Z_1 \ldots Z_k$ *pop*(X) i *push*(Y) u reverznom redu (Z_1 će biti novi vrh stoga)

♦ Konfiguracija stogovnog generatora

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

q tekuće stanje
w generirani niz znakova
α niz znakova koji predstavlja sadržaj potisne liste; vrh je prvi znak niza

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

♦ Pomak stogovnog generatora

Pomak stogovnog generatora $\mathcal{P}$ jest relacija $\vdash_{\mathcal{P}}$ (ili samo $\vdash$ ako se $\mathcal{P}$ podrazumijeva):

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

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 niz w generiran s $\mathcal{P}$ ako

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

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

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

♣ Primjer 6.18

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

$\delta(q_0, 0, \$) = \{(q_1, 0\$)\}$
$\delta(q_1, 0, 0) = \{(q_1, 00)\}$ } tekući se znak postavlja na vrh stoga
$\delta(q_1, \varepsilon, \varepsilon) = \{(q_2, \varepsilon)\}$ ⟵ prelazak u pop mode
$\delta(q_2, 1, 0) = \{(q_2, \varepsilon)\}$ ⟵ smanjivanje stoga izbacujući znakove s vrha
$\delta(q_2, \varepsilon, \$) = \{(q_0, \varepsilon)\}$ ⟵ prekid

Na primjer, rečenica 000111 može biti generirana nizom premještanja:

$(q_0, \varepsilon, \$) \vdash (q_1, 0, 0\$) \vdash (q_1, 00, 00\$) \vdash (q_1, 000, 000\$) \vdash (q_2, 000, 000\$)$
$\vdash (q_2, 0001, 00\$) \vdash (q_2, 00011, 0\$) \vdash (q_2, 000111, \$) \vdash (q_0, 000111, \varepsilon)$

♣ Primjer 6.19

Evo još jednoga stogovnog automata koji generira beskontekstni jezik $\mathcal{L} = \{ww^R:\ w \in \{a,b\}^*\}$:

$Q = \{p,q,r\}$ $\Sigma = \{a,b\}$ $\Gamma = \{a,b,Z\}$ $q_0 = p$ $Z_0 = Z$ $F = \{r\}$

$\delta(p,a,Z) = \{(p,aZ)\}$
$\delta(p,b,Z) = \{(p,bZ)\}$ } prvi se znakovi postavljaju na vrh stoga

$\delta(p,a,a) = \{(p,aa)\}$
$\delta(p,a,b) = \{(p,ab)\}$
$\delta(p,b,a) = \{(p,ba)\}$
$\delta(p,b,b) = \{(p,bb)\}$ } tekući se znak postavlja na vrh stoga

$\delta(p,\varepsilon,a) = \{(q,a)\}$
$\delta(p,\varepsilon,b) = \{(q,b)\}$ } prelazak u pop mode

$\delta(q,a,a) = \{(q,\varepsilon)\}$
$\delta(q,b,b) = \{(q,\varepsilon)\}$ } smanjivanje stoga izbacujući znakove s vrha

$\delta(q,\varepsilon,Z) = \{(r,\varepsilon)\}$ ⟵ prekid

Na primjer, rečenica aabbaa može biti generirana nizom premještanja:

$(p,\varepsilon,Z) \vdash (p,a,aZ) \vdash (p,aa,aaZ) \vdash (p,aab,baaZ)$
$\vdash (q,aab,baaZ) \vdash (q,aabb,aaZ) \vdash (q,aabba,aZ)$
$\vdash (q,aabbaa,Z) \vdash (r,aabbaa,\varepsilon)$

6.4 EKVIVALENTNOST STOGOVNIH AUTOMATA I BESKONTEKSTNIH GRAMATIKA

Da rekapituliramo: rečenice beskontekstnih jezika mogu biti generirane beskontekstnim gramatikama ili stogovnim automatom (generatorom), pa očekujemo da će beskontekstna gramatika zadanog beskontekstnog jezika imati ekvivalentni stogovni automat. Da bismo to pokazali najprije definiramo stogovni automat s praznim stogom.

♦ Stogovni automat s praznim stogom

Neka je $\mathcal{P}=(Q,\Sigma,\Gamma,\delta,q_0,Z_0,F)$ stogovni automat. Kaže se da je niz $w\in\Sigma^*$ generiran automatom $\mathcal{P}$ s praznim stogom ako je

$$(q_0,\varepsilon,Z_0)\ ^{+}\!\vdash (q,w,\varepsilon)$$

za neki $q\in Q$. Dakle, rečenica je generirana uz uvjet da je stog prazan, bez obzira na stanje, pa takav automat nema konačnih stanja, tj. $F=\varnothing$ i definiran je kao uređena šestorka $\mathcal{R}=(Q,\Sigma,\Gamma,\delta,q_0,Z_0)$. Skup generiranih nizova (jezik) označujemo s $\mathcal{L}_\varepsilon(\mathcal{R})$.

• Teorem 6.1

Neka je $\mathcal{G}=(\mathcal{N},\mathcal{T},\mathcal{P},S)$ beskontekstna gramatika. Ako se iz $\mathcal{G}$ definira nedeterministički stogovni automat s praznim stogom, $\mathcal{R}=(\{q_0\},\mathcal{T},\mathcal{T}\cup\mathcal{N},\delta,q_0,S)$, gdje je funkcija prijelaza δ definarana kao

$$\delta(q_0,\varepsilon,A) = \{(q_0,\alpha):\ (A\to\alpha)\in\mathcal{P},\ A\in\mathcal{N},\ \alpha\in(\mathcal{T}\cup\mathcal{N})^*\}$$
$$\delta(q_0,a,a) = \{(q_0,\varepsilon):\ a\in\mathcal{T}\}$$

tada je stogovni automat $\mathcal{R}$ ekvivalentan gramatici $\mathcal{G}$, tj. $\mathcal{L}(\mathcal{R})=\mathcal{L}(\mathcal{G})$.

Dakako, važno je upamtiti što teorem 6.1 tvrdi, dok nas njegov dokaz ne zanima. Dalje, valja primijetiti da za automat s praznom stogovnom listom ne trebamo imati nijedno završno stanje, jer je prazna potisna lista uvjet okončanja postupka generiranja rečenice jezika (ili prihvaćanja, ako je automat u ulozi prepoznavača).

♣ Primjer 6.20

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

$$E \to T \mid T+E$$
$$T \to F \mid F*T$$
$$F \to (E) \mid a$$

Stogovni automat će biti $\mathcal{R}=(\{q\},\ \{a,+,*,(,)\},\ \{E,T,F,a,+,*,(,)\},\delta,q,E)$, gdje su:

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

Na primjer, rečenica a+a*a bit će generirana nizom:

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

Ovaj primjer zorno prikazuje općenitu ideju ekvivalentnosti gramatike i automata s praznim stogom izvedenim iz nje:

$$w\in\mathcal{L}(\mathcal{G}) \Leftrightarrow S \overset{*}{\Rightarrow} w \quad\approx\quad (q_0,\varepsilon,S) \overset{*}{\vdash} (q_0,w,\varepsilon) \Leftrightarrow w\in\mathcal{L}(\mathcal{R}(\mathcal{G}))$$

U drugoj ćemo knjizi pokazati da primjena stogovnog automata s praznim stogom u prepoznavanju beskontekstnih jezika ima značenje silazne (top-down) sintaksne analize ili "predikatne" analize.

6.5 IZVOĐENJE BESKONTEKSTNIH 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 6.21

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 $a\alpha b$ također rečenica, a to je "klasični" primjer rekurzije, pa je gramatika danog jezika:

$$S \rightarrow ab \mid aSb$$

Za mnoge linearne jezike, kao što su na primjer jezik svih prirodnih brojeva, parnih i neparnih prirodnih brojeva, prirodnih brojeva djeljivih s 3 ili 4, kompaktija (preglednija) će biti beskontekstna gramatika od ekvivalentne linearne gramatike.

♣ Primjer 6.22

Izvedimo gramatiku jezika rimskih brojeva {i,ii,iii,iv,v,...,xcix,c,ci,...,mmmcmxcix}. Znamo da je to linearni jezik i pokazali smo kako se može definirati automat koji ga generira, odnosno izvesti linearna gramatika, primjer 5.7. Izvedimo sada ekvivalentnu beskontekstnu gramatiku:

```
R → J| DA| SB| TE
J → i| ii| iii| iv| v| vi| vii| viii| ix
D → x| xx| xxx| xl| l| lx| lxx| lxxx| xc
S → c| cc| ccc| cd| d| dc| dcc| dccc| cm
T → m| mm| mmm
A → ε| J
B → A| DA
E → B| SB
```

ili

```
R → J| DA| SB| TE
J → I| iv| v| vI| ix   I → i| ii| iii
D → X| xl| l| lX| xc   X → x| xx| xxx
A → ε| J
S → C| cd| d| dC| cm   C → c| cc| ccc
B → A| DA
T → m| mm| mmm
E → B| SB
```

♣ Primjer 6.23

U prethodnom smo poglavlju definirali linearnu gramatiku prirodnih brojeva:

```
N → A| N0| N1| N2| N3| N4| N5| N6| N7| N8| N9
A → 1| 2| 3| 4| 5| 6| 7| 8| 9
```

što se, prvo faktorizacijom

```
N → A| N0| N(1| 2| 3| 4| 5| 6| 7| 8| 9)
```

i supstitucijom,

```
A → 1| 2| 3| 4| 5| 6| 7| 8| 9
```

može transformirati u ekvivalentnu beskontekstnu gramatiku:

```
N → A| N0| NA
A → 1| 2| 3| 4| 5| 6| 7| 8| 9
```

Još je moguća jedna faktorizacija:

```
N → A| N(0| A)
```

i supstitucija uvođenjem novog neterminala B, pa imamo još jednu ekvivalentnu beskontekstnu gramatiku koja generira prirodne brojeve:

```
N → A| NB
A → 1| 2| 3| 4| 5| 6| 7| 8| 9
B → 0| A
```

Najčešće ćemo do konačne beskontekstne gramatike doći izravno, "prepisujući" pravila do kojih smo došli promatrajući svojstva rečenica danog jezika i koristeći produkcije ranije izvedenih gramatika ako su jezici koje one generiraju sadržani kao podnizovi u jeziku čiju gramatiku želimo izvesti. Slijedeći primjer možda to najbolje prikazuje. Prvo smo izveli gramatiku jezika prirodnih brojeva, koju smo iskoristili za izvođenje gramatika jezika parnih i neparnih prirodnih brojeva i, potom te dvije gramatike za izvođenje gramatike jezika prirodnih brojeva djeljivih s 4.

♣ Primjer 6.24

Pođimo od treće gramatike iz prethodnog primjera koja generira prirodne brojeve:

$N \to A|\ NB$
$A \to 1|\ 2|\ 3|\ 4|\ 5|\ 6|\ 7|\ 8|\ 9$
$B \to 0|\ A$

Sada nije teško izvesti gramatiku jezika parnih i neparnih brojeva. Parni su brojevi 2, 4, 6, 8 ili prirodni brojevi koji završavaju na 0, 2, 4, 6 ili 8:

$P \to C|\ ND$
$C \to 2|\ 4|\ 6|\ 8$
$D \to 0|\ C$

a neparni brojevi su 1, 3, 5, 7, 9 ili prirodni brojevi koji završavaju tim brojkama:

$Q \to E|\ NE \quad E \to 1|\ 3|\ 5|\ 7|\ 9$

Da bismo izveli gramatiku jezika prirodnih brojeva djeljivih s 4 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, 13456900 ili 101010196. Ali, ako napišemo sve dvocifrene brojeve koji su djeljivi s 4: 12, 16, 20, 24, 28, 32, ..., 80, 84, 88, 92, 96, doći ćemo do zaključka: Višecifreni broj djeljiv je s 4 ako ima paran prefiks, P, i sufiks 0, 4 ili 8, odnosno, ako ima neparan prefiks, Q, i sufiks 2 ili 6:

$S \to 4|\ 8|\ PX|\ QY \quad X \to 0|\ 4|\ 8 \quad Y \to 2|\ 6$

Takvi su, na primjer, brojevi 170568 i 78248452342368412. Dakle, rezultirajuća je gramatika:

$S \to 4|\ 8|\ PX|\ QY$
$P \to C|\ ND$
$C \to 2|\ 4|\ 6|\ 8$
$D \to 0|\ C$
$Q \to E|\ NE$
$E \to 1|\ 3|\ 5|\ 7|\ 9$
$N \to A|\ NB$
$A \to 1|\ 2|\ 3|\ 4|\ 5|\ 6|\ 7|\ 8|\ 9$
$B \to 0|\ A$
$X \to 0|\ 4|\ 8$
$Y \to 2|\ 6$

Na primjer, rečenica 1064 može se izvesti nizom izvođenja:

$S \Rightarrow PX \Rightarrow NDX \Rightarrow NBDX \Rightarrow ABDX \Rightarrow 1BDX \Rightarrow 10DX \Rightarrow 10CX \Rightarrow 106X \Rightarrow \underline{1064}$

Rezultirajuća gramatika može se reducirati uvođenjem nekih supstitucija u ekvivalentnu gramatiku s produkcijama:

$S \to Z|\ PX|\ QY$
$Z \to 4|\ 8$
$X \to 0|\ Z$
$Y \to 2|\ 6$
$P \to C|\ ND$
$C \to Y|\ Z$
$D \to 0|\ C$
$Q \to E|\ NE$
$E \to 1|\ 3|\ 5|\ 7|\ 9$
$N \to A|\ NB$
$A \to C|\ E$
$B \to 0|\ A$

U prethodnom smo poglavlju definirali konačni automat koji je generirao linearne gramatike jezika prirodnih brojeva djeljivih s bilo kojim brojem, od 1 do 25 (program GEN-PB.pas). Taj bi program za jezik prirodnih brojeva djeljivih s 4 generirao linearnu gramatiku s produkcijama:

```
A -> 2C|4E|6C|8E|F
B -> 0C|2E|4C|6E|8C|F
C -> 0E|A
F -> 1B|3B|5B|7B|9B
E -> ε|C
```

odnosno, poslije faktorizacije i supstitucije, ekvivalentnu beskontekstnu gramatiku s produkcijama:

```
A -> GC|HE|F
B -> IC|GE|F
C -> 0E|A
F -> JB
G -> 2|6
H -> 4|8
I -> 0|H
J -> 1|3|5|7|9
E -> ε|C
```

♣ Primjer 6.25

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:

$$S \rightarrow AB|\ BA|\ C$$

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:

$$B \rightarrow 2|\ 5|\ 8|\ BX|\ AA$$
$$A \rightarrow 1|\ 4|\ 7|\ AX|\ BB$$

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

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

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

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

Gramatika jezika $w \in (a+b)^+$ uz uvjet da je $N_a(w)=N_b(w)$, gdje su $N_a(w)$ i $N_b(w)$ broj pojavljivanja znaka a i znaka b u nizu w.

Izvođenje beskontekstnih gramatika iz regularnih izraza

Iz regularnog izraza koji označuje regularni skup (jezik) prirodnih brojeva djeljivih s 3 dan u drugom poglavlju:

$R = (c| (a|bb)\{ab\}b| (aa|b)\{ba\}a)^+$
$d = (0|3|6|9)$
$c = (3|6|9)\{d\}$
$a = (1|4|7)\{d\}$
$b = (2|5|8)\{d\}$

možemo izvesti beskontekstnu gramatiku:

$R \rightarrow X| XR$
$X \rightarrow C| IJ| KL$
$I \rightarrow A| BB$
$J \rightarrow MB$
$M \rightarrow \varepsilon| AB| MM$
$K \rightarrow AA| B$
$L \rightarrow NA$
$N \rightarrow \varepsilon| BA| NN$
$T \rightarrow 3| 6| 9$
$D \rightarrow \varepsilon| 0| T| DD$
$C \rightarrow TD$
$A \rightarrow 1| 4| 7| AD$
$B \rightarrow 2| 5| 8| BD$

P R O G R A M I

USTROJ ALGORITAMA ZA TRANSFORMIRANJE GRAMATIKA

Prilažemo ustroj svih algoritama danih u ovom poglavlju. Njihovo je značenje dvojako: dopunjuju razumijevanje algoritma koji je ustrojen i prikazuju tehnike programiranja u Pythonu kojima se pseudo kod algoritma gotovo "doslovce" preslikava u Python što će vam omogućiti da i sami realizirate neke druge algoritme iz teorije formalnih jezika i šire.

ALGORITMI_FJ.py

```
# ALGORITMI TRANSFORMIRANJA GRAMATIKA
# -*- coding: cp1250 -*-

from gramatika import *
from fun       import *
import Tkinter

# 6.1 --- Izbacivanje jediničnih produkcija ------------------

def Skup_Ne (P, Ter): # Izracunavanje skupa Ne
  Ne = [];  End = False;  n = 0
  while not End:
    for i in range (len (P)):
      A = P[i][0]
      if A not in Ne:
        for j in range (1, len(P[i])):
          W = P[i][j]; C = W[0];  k = 0;  Ok = True
          while k < len (W) : Ok = Ok and C in Ter +['#'] +Ne;  k += 1
          if Ok and A not in Ne: Ne.append (A)
    End = n == len (Ne)
    n = len (Ne)
  return Ne

# 6.2 --- Izbacivanje nedokučivih simbola --------------------

def Izbaci_P (Ne, G) :
  N, T, P, S = G
  i = 0
  while i < len(P) :
    if P[i][0] not in Ne :
      P.remove (P[i])
    else:
      j = 1
      while j < len (P[i]) :
        k = 0; Beta = P[i][j]; Ok = True
        while k < len (Beta) and Ok:
          Ok = Beta[k] in (Ne +T + ['#'])
          if not Ok : P[i].remove (Beta)
          k += 1
        j += 1
    i += 1
  return (Ne, T, P, S)

# 6.3 --- Izbacivanje neupotrebljivih simbola -----------------

def trans (G) :

  def presjek (X, Y): return [x for x in X if x in Y]

  N, T, P, S = G
  P2 = []; V = [S];
  End = False; n = 0
  while not End:
    End = True
    for i in range (len (P)) :
      A = P[i][0]
      if A in V:
```

```
        for j in range (1, len(P[i])):
          W   = P[i][j]
          Vi  = [x for x in W if x != '#']
          for x in V:
            if x not in Vi: Vi.append(x)
          if len(Vi) != len(V): End = False
          V = Vi
      if not End: P2.append (P[i])
  N2 = presjek (N, V);  T2 = presjek (T, V)
  G  = (N2, T2, P2, S)
  G  = Izbaci_P (N2, G)
  return G

# 6.4 --- Izbacivanje e-produkcija ---------------------------

def Izbaci_e (G):
  N, T, P, S = G;  Ne = '';  X = list(A)
  for x in N : X.remove(x)

  for i in range (len(P)):
    e = False; j = 1
    while not e and j < len(P[i]):
      e = P[i][j] == '#'
      j += 1
    if e : Ne += P[i][0]

  for i in range (len(P)):
    j = 1
    while j < len (P[i]):
      Alfa = Beta = P[i][j]; k = 0; Y = []
      for x in Ne :
        l = pos (x, Beta)
        while l != -1:
          Y.append (k+l); k += l+1; Beta = Beta [l+1:]; l = pos (x, Beta)
      if len(Y) > 0:
        m = len(Y); n = pow(2,m)
        for k in range (1,n):
          Z = bin (k); Z = Z[2:]; Z = '0'*(m-len(Z)) +Z
          Beta = Alfa; d = 0
          for l in range (len(Z)):
            if Z[l] == '1':
              p = Y[l]-d; Beta = Beta[:p] +Beta[p+1:]; d += 1
          j += 1
          P[i].insert (j, Beta)
      else:
        j += 1
    if '#' in P[i]: P[i].remove ('#')

  if S in Ne : P.insert (0, [X[0], S, '#']); S = X[0]
  return (N, T, P, S)

# 6.5 --- Izbacivanje jediničnih produkcija ------------------

def Izbaci_JP (G):
  N, T, P, S = G;  P2 = []
  for i in range (len(P)):  P2.append ([])
    for x in P[i]: P2[i].append(x)
```

```
    "(1) Izgradnja skupa neterminala NA "
    X = [[A] for A in N]
    for i in range (len(P2)-1,-1,-1):
      for j in range (1, len(P2[i])):
        Beta = P2[i][j]
        if Beta in N : X[i].append(Beta)
      for j in range (i, len(X)):
        if X[j][0] in X[i]:
          for k in range (1,len(X[j])):
            x = X[j][k]
            if x not in X[i]: X[i].append(x)

    "(2) zamjena jediničnih produkcija "
    for i in range (len(X)):
      for j in range (1, len(X[i])):
        x = X[i][j]
        m = N.index (x)
        if x in P2[i]:
          k = P2[i].index(x);  P2[i].remove(x)
          for l in range (1,len(P2[m])):
            Beta = P2[m][l]
            P2[i].insert(k, Beta); k+= 1

    return (N, T, P2, S)

  # 6.6 --- CNF (Chomskyjeva normalna forma) ------------------

  def CNF (G):
    def zamijeni (x, y):
      k = n;
      while k < len(P):
        if P[k][1] == x: y = P[k][0]; return y
        k += 1
      y = Nf[0]; P.append([y, x]); Nf.remove(y); N.append(y)
      return y
    N, T, P, S = G;  Nf = list(A); X = '';  Y = ''
    for x in N : Nf.remove(x)
    n  = len(N);  i  = 0
    while i < len(P):
      for j in range (1, len(P[i])):
        Beta = P[i][j]
        if len(Beta) == 1:
          if not Beta in T:
            print 'jedinična produkcija, ', P[i][0], '->', Beta
            return G
        elif len(Beta) == 2:
          x1 = Beta[0]; x2 = Beta[1]; y1, y2 = x1, x2
          if x1 not in N: y1 = zamijeni (x1, y1)
          if x2 not in N: y2 = zamijeni (x2, y2)
          P[i][j] = y1+y2
        else:
          x1 = Beta[0]; x2 = Beta[1:]; y1, y2 = x1, x2
          if x1 not in N: y1 = zamijeni (x1, y1)
          y2 = zamijeni (x2, y2);
          P[i][j] = y1+y2
      i += 1
    return (N, T, P, S)
```

```
### 6.7 ### GNF (Greibachina normalna forma)----------------------------

def GNF (G) :
  N, T, P, S = G;
  Nf = list(A); X = ''; Y = ''
  for x in N : Nf.remove(x)
  n  = len(N);

  i  = n-1
  while i != -1:
    k = 1
    while k < len(P[i]):
      NT = P[i][k][0];  Y = []
      j = -1
      if NT in N: j = N.index(NT)
      if j > i:
        Alfa = P[i][k][1:]
        for m in range (1,len(P[j])):
          Beta = P[j][m]
          if Beta[0] in T:
            if m == 1 : P[i][k] = Beta+Alfa
            else      : P[i].insert(k, Beta+Alfa); k+= 1
      else:
        k += 1
    i = i -1

  X = ''; Y = ''; n = len(P)
  for i in range (n):
    for j in range (1, len(P[i])):
      Beta = P[i][j]
      for k in range (1,len(Beta)):
        NT = Beta[k]
        if NT in T :
          if pos(NT, Y) == -1:
            Y += NT; Z = Nf[0]; X += Z; Nf = Nf[1:]
            P.append ([Z, NT]);  N.append(Z)
          else:
            Z = X[pos(NT, Y)]
          Beta = Beta.replace(NT, Z)
      P[i][j] = Beta

  return (N, T, P, S)


Alg = ('1  Je li L(G) neprazan?',
       '2  Izbacivanje nedokučivih simbola',
       '3  Izbacivanje neupotrebljivih simbola',
       '4  Izbacivanje e-produkcija',
       '5  Izbacivanje jediničnih produkcija',
       '6  Konverzija u CNF (Chomskyjeva normalna forma)',
       '7  Konverzija u GNF (Greibachina normalna forma)' )

print ('IZABERITE ALGORITAM TRANSFORMIRANJA GRAMATIKE: ')
for i in range (len(Alg)):  print ' ', Alg[i]

i = input ('> ')
Alg2 = list (Alg)
```

```
if i < len(Alg):
  Ime = Alg[i-1][2:]
  Grm, Ok, G = Ucitaj_G (Ime, '*.grm')

  if Ok :
    Ispisi_G (Grm, G);  N, T, P, S = G
    if   i == 1:
      "Je li L(G) neprazan?"
      Ne = Skup_Ne (P, T)
      print
      Print_skup ('Ne', Ne);  print
      if S in Ne : print 'Jezik je neprazan'
      else       : print 'Jezik je prazan'

    elif i == 2:
      "Izbacivanje nedokučivih simbola"
      Ne = Skup_Ne (P, T)
      G  = Izbaci_P (Ne, G)
      Ispisi_G (Grm +"'", G)

    elif i == 3:
      "Izbacivanje neupotrebljivih simbola"
      Ne = Skup_Ne (P, T)
      G  = Izbaci_P (Ne, G)
      G2 = trans (G)
      print
      Ispisi_G (Grm +'"', G2)

    elif i == 4:
      "Izbacivanje e-produkcija"
      G2 = Izbaci_e(G)
      print NL, 'Ekvivalentna gramatika bez e-produkcija'
      Ispisi_G (Grm[:-3]+'###', G2)

    elif i == 5:
      "Izbacivanje jediničnih produkcija"
      G2 = Izbaci_JP (G)
      if G2 != G:
        print; Ispisi_G ('(bez jediničnih produkcija)', G2)
      else:
        print; print 'Gramatika ne sadrži jedinične produkcije!'

    elif i == 6:
      "Konverzija u CNF (Chomskyjeva normalna forma)"
      G2 = Izbaci_JP (G)
      if G2 != G:
        print
        print ('Gramatika sadrži jedinične produkcije!')
        Y = raw_input ('Eliminiram jedinične produkcije (D/N)? > ')
        if Y[0] in ['d','D']:
          G2 = CNF(G2)
          print
          Ispisi_G ('Ekvivalentna gramatika u CNF-u', G2)
      else:
        G2 = CNF(G)
        print
        Ispisi_G ('Ekvivalentna gramatika u CNF-u', G2)
```

```
    elif i == 7:
      "Konverzija u GNF (Greibachina normalna forma)"
      Ok = True
      for p in P:
        for q in p[1:]:
          if p[0] == q[0]: Ok = False
      if Ok:
        print NL, 'Gramatika je rekurzivna slijeva!'
      else:
        G2 = GNF(G)
        print
        Ispisi_G ('Ekvivalentna gramatika u GNF-u', G2)
  else:
    print 'Ne postoji gramatika s danim imenom!'
else:
  print 'Niste izabrali nijedan algoritam!'
```

GRAMATIKA JEZIKA PRIRODNIH BROJEVA DJELJIVIH S k *(2)*

U prethodnom smo poglavlju pokazali kako se može izvesti linearna zdesna gramatika jezika prirodnih brojeva djeljivih s k, $0<k\leq 25$. Ovdje ćemo ih transformirati u ekvivalentne beskontekstne gramatike.

P R I M J E N E

Beskontekstni jezici nalaze svoju primjenu u jezicima za programiranje, matematici, bibliotekarstvu, kemiji, biologiji, botanici (cvjećarstvu), itd.

Beskontekstne gramatike se najviše koriste u opisu osnovne sintaksne strukture jezika za programiranje. Poznato je da su prvi jezici za programiranje, kao na primjer FORTRAN, razvijeni "empirijski", bez posebnih formalizama za prikaz njihove osnovne sintaksne strukture. Jezik ALGOL 60, publiciran 1963. godine, bio je prvi jezik za programiranje čiju su osnovnu sintaksnu strukturu Backus i Nauer prikazali u posebnoj notaciji, danas poznatoj kao Backus-Nauerova forma (BNF).

Primjene beskontekstnih gramatika u matematici, posebno diskretnoj matematici (teoriji skupova, matematičkoj logici, itd). Na primjer, Booleove formule (logički izrazi) jesu jezik koji bi se mogao opisati jednostavnom beskontekstnom gramatikom:

$$L \to \neg L \mid L \wedge L \mid L \vee L \mid L \Rightarrow L \mid L \equiv L \mid (L) \mid t \mid f \mid x \mid y$$

gdje su:

- $\neg$ negacija ("ne")
- $\wedge$ konjunkcija ("i" ili "et")
- $\vee$ disjunkcija ("ili")
- $\Rightarrow$ implikacija
- $\equiv$ ekvivalencija
- t, f logičke konstante ("istina" i "neistina")
- x, y logičke varijable

JEZIK REGULARNIH IZRAZA

Regularni izraz, kako smo ga definirali u drugom poglavlju, jest beskontekstni jezik! Na primjer, može biti generiran (opisan) sljedećom gramatikom:

```
E → ε|S|B|Z|(E)|E*|E?|E+|E|E|{E}|EE
S → 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|
    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
B → 0|1|2|3|4|5|6|7|8|9
Z → -|_|"|#|$|%|=|<|>|.|,|'|:|;|&|!|/|\|[|]
```

Nedostatak je opisa regularnih jezika regularnim izrazima što su pojedini znakovi bili rabljeni kao meta simboli i nije postojao način da budu uključeni kao znakovi rečenica regularnog jezika kojeg opisuju. Ako smo ih trebali, morali smo definirati regularnu gramatiku ili konačni generator.

PROŠIRENA PRAVILA PISANJA REGULARNIH IZRAZA

Svi jezici za programiranje koji se danas koriste (C++, Java Skript, Perl, PHP, Phyton, ...) imaju procedure i funkcije za rad s regularnim skupovima, odnosno, regularnim izrazima. Značenje regularnih izraza je sparivanje određenih nizova (riječi), u skladu s određenim pravilima. Koriste ih mnogi programi za uređenje teksta, programi za pretragu i manipuliranje nizovima. Može se reći da danas "regularni izraz", koji se još naziva "uzorak" (engl. *patern*), rabi za opis (označivanje) skupa nizova znakova bez davanja njegova precizna značenja. Još uvijek ne postoji standardna notacija pisanja regularnih jezika.

Ovdje ćemo pokušati dati nekoliko pravila koja su zajednička za većinu spomenutih jezika koji rade s regularnim izrazima. Evo tablice u kojoj smo prikazali notaciju (sintaksu) pisanja regularnih izraza u većini jezika za programiranje. Vidjet ćemo da neki metaznakovi imaju različito značenje od prvobitno definiranog. Osim toga, jezik regularnih izraza koji se koristi u jezicima za programiranje, nazovimo ga RE, sa svim svojim proširenjima i uvođenjem pojedinih svojstava više nije beskontekstan već jezik sa svojstvima.

Osnovna, beskontekstna gramatika definirana je kao

```
E →  .|S|B|Z|\Y|[N]|[-N]|[N-]|(E)|EE|E*|E?|E+|E|E|E{B}|E{B,B}|E$
S →  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|
     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|SS
B →  0|1|2|3|4|5|6|7|8|9|BB
Z →  -|_|"|#|%|=|
N →  X-X|NN|X
X →  S|B|Z|\Y
Y →  [|\|^|$|.||?|*|+|(|)|{|}|d|n|r|s|t|w|E|Q|W
```

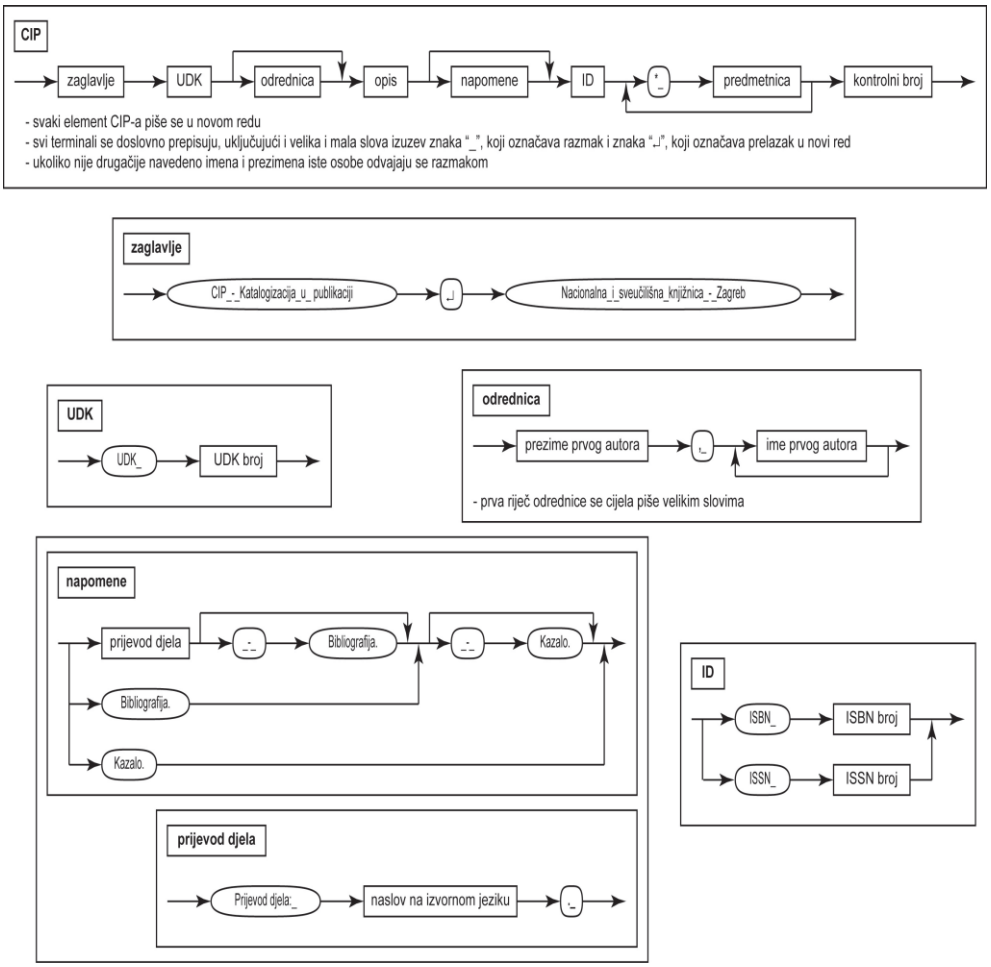
Ako s $\mathcal{A}$ označimo skup svih znakova prema nekom ASCII uređenju, u sljedećoj je tablici dana proširena sintaksa pisanja regularnih izraza u Pythonu.

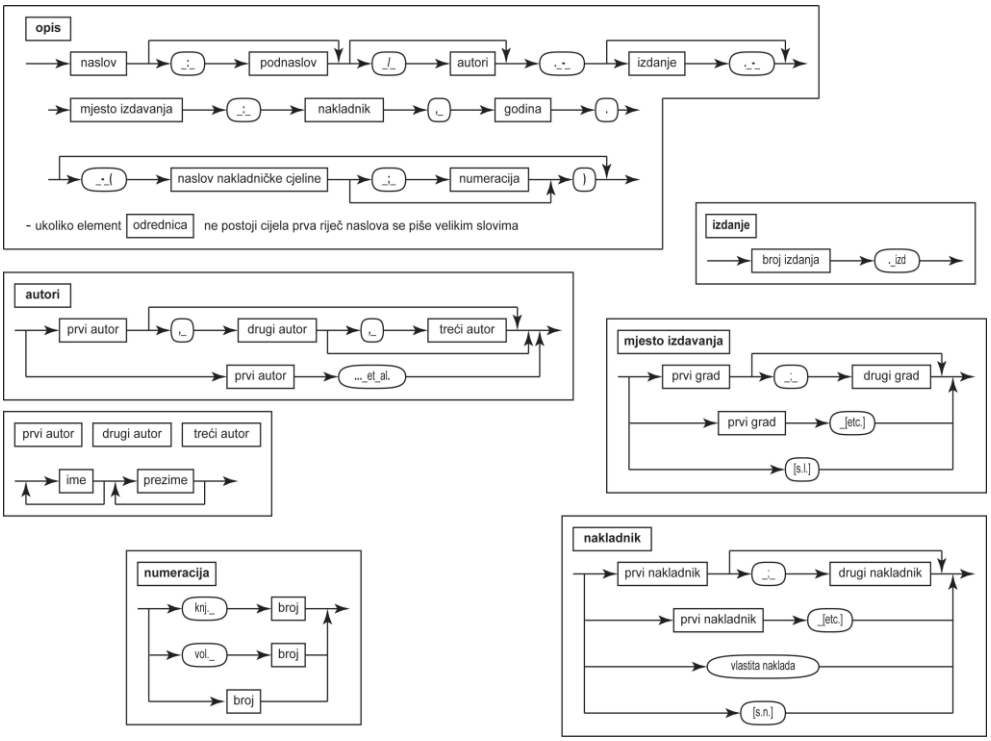
Metaznak	Značenje	Primjeri	Jezik
.	Bilo koji znak		
()	Podizraz (ili "blok")	(a)	{a}
|	Alternativa ("ili")	0|1|2|99 gr(a|e)y	{0,1,2,99} {gray,grey}
*	Ponavljanje prethodnog izraza 0, 1 ili bilo koji veći broj puta	(a)* ili a* (a|b)* go*gle	{ε,a,aa,aaa,...} {ε,a,b,aa,ab,ba,bb,...} {ggle,gogle,google,... }
?	Izostavljanje prethodnog izraza ili pojavljivanje jedanput.	colou?r M(i|a)r(k?)o ((great)?grand)? ((fa|mo)ther)	{color,colour} {Miro, Mirko, Maro, Marko} {father,mother,grand father,grand mother,great grand father,great grand mother}
+	Ponavljanje prethodnog izraza 1 ili bilo koji veći broj puta	go+gle	{gogle,google,gooogle,...}
[$a_1a_2...a_n$]	=(a_1|a_2|...|a_n)	[abc]	{a,b,c}
[a_i-a_j]	=(a_i| a_{i+1}| ...| a_j) Skup znakova od a_i do a_j prema *A* uređenju. Ako je redni broj a_i veći od rednog broja a_j, skup je prazan.	[A-F] (-|+)*[0-9]+.[0-9]+ ([0-9]| [1-9][0-9]| 1[0-9][0-9]| 2[0-4][0-9]| 25[0-5])	{A,B,C,D,E,F} Realni brojevi u Pascalu {0,1,2,...9, 10,...,99, 100,...,199, 200,...,249, 250,...,255}
[$a_1...a_nb_i$-b_j]	=(a_1|a_2|...|a_n|b_i|...|b_j) Ako je znak '-' dio izraza piše se na početku ili na kraju. Ako su uglate zagrade dio izraza pišu kao [][...]	[abcq-z] [a-fu-z] [-abc]=[abc-] [][0123]	{a,b,c,q,r,s,t,u,v,w,x,y,z} {a,b,c,d,e,f,u,v,w,x,y,z} {-,a,b,c} {],[,0,1,2,3}
{n}	Izostavljanje prethodnog podizraza ili ponavljanje najviše n puta.	(0|1){2}	{ε,0,1,00,01,10,11}
{n,m}	Ponavljanje prethodnog podizraza najmanje m, najviše n puta.	A{1,8}(.|A{0,3})? gdje je A=([0-9]| [a-z]| [A-Z]|-|_|"|#|$|%| &|=|+|^|'|@)	Imena DOS datoteke
\d	=[0-9] (skup brojki)		{0,1,2,3,4,5,6,7,8,9}
\s	Razmak (blank)		
\w	=[a-zA-Z0-9_] (alfanumerički znakovi)		{0,...,9,_,A,...,Z,a,...,z}
\W	Skup svih znakova bez alfanumeričkih znakova ([a-zA-Z0-9_])		
\xFF	Znak čiji je *A* kod jednak F, gdje je FF heksadecimalni broj	\xA9	Znak © ako je kodna stranica Latin-1

Prioritet izvršavanja operacija: (1) izraz u zagradi, (2) *, +, ? i (3)nastavljanje. Tamo gdje je prioritet očigledan bez zagrada mogu se izbrisati suvišne zagrade. Na primjer, `(a)` je jednako `a`, `(b)c` je jednako `bc` i `a(b)*` je jednako `ab*`.

KATALOGIZACIJA U PUBLIKACIJI

CIP (Cataloguing in publication = Katalogizacija u publikaciji) je program izradbe preliminarnih, skraćenih kataložnih zapisa za publikacije u pripremi za tisak. Izrađuje se na temelju točnog prijeloma publikacije. Kataloški opis izrađuje se u skladu s nacionalnim i međunarodnim pravilima za formalnu i sadržajnu obradu. Dakle, CIP je jezik! Ako bismo ga klasificirali bio bi tipa 1 te se ne bi mogao definirati beskontekstnom grmatikom. Ipak, njegova temeljna sintaksna struktura može biti prikazana sintaksnim dijagramima u kojima ćemo izborom naziva neterminala, određenih imena i opisa pojedinih dijelova jezika, uvesti određena značenja. Na nekim mjestima ćemo dodati i dodatna pravila riječima, i to je to! Pogledajmo:





PRIMJERI GRAMATIKA

Na kraju ovoga poglavlja evo nekoliko primjera ekvivalentnih gramatika. Najprije šest ekvivalentnih gramatika koje generiraju jezik "aritmetičkih izraza", potom još dva jezika.

Jezik {a,a+a,a+a+a+…,a*a,a*a*…, (a), ((a)),…}

1) Rekurzivna zdesna

```
E -> T+E| T    T -> F*T| F    F -> (E)| a
```

2) Bez jediničnih produkcija

```
E -> T+E| F*T| (E)| a    T -> F*T| (E)| a    F -> (E)| a
```

3) U CNF-u

```
E -> TA| FB| CD| a    T -> FB| CD| a    F -> CD| a    A -> GE    B -> HT
C -> (                D -> EI           G -> +        H -> *     I -> )
```

4) U GNF-u

```
E -> (EABTCE| aCE| (EACE| aBTCE| (EABT| a| (EA| aBT
T -> aBT| (EABT| a| (EA    F -> (EA| a    A -> )    B -> *    C -> +
```

5) Rekurzivna slijeva

```
E -> E+T| T    T -> T*F| F    F -> (E)| a
```

6) Rekurzivna ("sa svih strana")

E -> E+E| E*E| (E)| a

Jezik $\{x: \mathbb{N}_a(x)=\mathbb{N}_b(x), x\in\{a,b\}^+\}$

1) S → aB| bA A → a| aS| bAA B → b| bS| aBB

2) S → ab| ba| aSb| bSa| SS

Jezik $\{x: \mathbb{N}_a(x)=\mathbb{N}_b(x) =\mathbb{N}_c(x),\ x\in\{a,b,c\}^*\} \setminus \{a^n b^n c^n: n>1\}$

1) S → ABC| ACB| BAC| BCA| CAB| CBA| ε A → a| AS| SA B → b| BS| SB
   C → c| CS| SC

2) S → aSbSc| aScSb| bSaSc| bScSa| cSaSb| cSbSa| SS| ε

Pitanja i zadaci

1) Koje jezike generiraju sljedeće gramatike:

G_1: S → cS| ε
G_2: S → cSd| ε
G_3: S → Sd| cS| c| d
G_4: S → ScS| c
G_5: S → cA, A → d| cA| Td, T → Td| d

2) Dane su tri gramatike:

G_1: S → A, A → a| aB, B → bC, C → cA
G_2: S → A, A → a| aB| abcA, B → bC, C → cA
G_3: S → a| abcS

Dokažite da su ekvivalentne!

3) Definirajte gramatiku koja generira jezik $\{x: x\in\{a,b\}^+, \mathbb{N}_a(x)>0, \mathbb{N}_b(x)=2n, n\geq 0\}$.

4) Definirajte gramatike sljedećih jezika:

$L_1 = \{ x^n y^n: x=ab, y=cd, n>0 \}$

$L_2 = \{ 0^m 1^n: m>n\geq 0 \}$

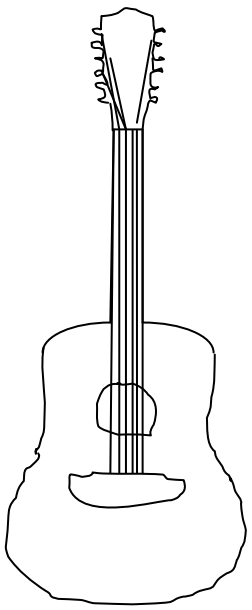
$L_3 = \{ 0^m 1^n:$ m – neparan i n - paran; m – paran i n – neparan $\}$

$L_4 = \{ 0^k 1^m 0^n:$ n = k+m $\}$

5) Kao što smo pokazali u primjeru 3.8 formalizam pisanja produkcija gramatike jest regularni jezik i može se opisati regularnim izrazom. Napišite pravila pisanja produkcija definirajući ih u proširenoj notaciji regularnih izraza Pythona.

7.
KONTEKSTNI JEZICI

1



U ovom ćemo poglavlju pokazati da je klasa beskontekstnih jezika podskup šire klase jezika – kontekstnih jezika. Drugim riječima, snaga kontekstnih gramatika kao generatora jezika veća je od generiranja jezika beskontekstnim gramatikama. Prije definiranja svojstva kontekstnih jezika, konteksnih gramatika i linearno ograničenih automata koji ih generiraju, dajemo proširenje beskonteksnih gramatika – indeksiranu gramatiku. Bavimo se problemom izvođenja kontekstnih gramatika, dajemo primjere izvođenja kontekstnih gramatika nekoliko jezika i, na kraju, primjene kontekstnih jezika.

7.1 INDEKSIRANE GRAMATIKE

Indeksirane gramatike, koje ćemo ovdje definirati, uveo je Aho 1968. godine dodajući produkcijama beskontekstne gramatike stogovni mehanizam indeksiranja.

♦ Indeksirana gramatika

Indeksirana gramatika je petorka $G=(N, T, P, S, I)$, gdje N, T i S imaju isto značenje kao kod "obične" gramatike, I je konačni skup indeksa, a skup produkcija P definiran je kao

$$A \rightarrow X_1\psi_1X_2\psi_2...X_n\psi_n,\ n\geq 0 \qquad A_f \rightarrow X_1\psi_1X_2\psi_2...X_n\psi_n,\ n\geq 0$$

gdje je $A\in N$, $X_i\in(N\cup T)$, $f\in I$ i $\psi_i\in I^*$, tako da ako je $X_i\in T$, tada je $\psi_i=\varepsilon$.

♣ Primjer 7.1

Evo jednog primjera indeksirane gramatike, $G=(\{S,T,A,B,C\},\ \{a,b,c\},\ P,\ S,\ \{f,g\})$, gdje je skup produkcija:

$$\begin{array}{llll} S \rightarrow T_g & A_f \rightarrow aA & B_f \rightarrow bB & C_f \rightarrow cC \\ T \rightarrow T_f \mid ABC & A_g \rightarrow a & B_g \rightarrow b & C_g \rightarrow c \end{array}$$

Indeksirana gramatika generira indeksirani jezik. Izvođenje rečeničnih formi definira se na sljedeći način:

♦ Rečenična forma indeksirane gramatike

Neka su $\alpha, \beta\in(NI^* \cup T)^*$, $A\in N$, $\theta\in I^*$ i $\alpha A\theta\beta$ rečenična forma. Ako je

$$A \rightarrow X_1\psi_1X_2\psi_2...X_n\psi_2$$

produkcija iz P, tada je izvođenje nove rečenične forme definirano kao

$$\alpha A\theta\beta \Rightarrow \alpha X_1\psi_1\theta'_1X_2\psi_2\theta'_2...X_n\psi_n\theta'_n\beta$$

gdje je $\theta'_1=\theta$ ako je $X_i\in N$, odnosno $\theta'_1=\varepsilon$ ako je $X_i\in T$. Drugim riječima, niz indeksa koji slijedi neterminal A distribuira se (dopisuje) svim neterminalima koje sadrži desna

strana produkcije A, ali ne terminalima, kojima nikad ne može biti pridružen indeks. Ako je $\alpha A_f \theta \beta$ rečenična forma i ako je

$$A_f \rightarrow X_1\psi_1 X_2\psi_2 \ldots X_n\psi_2$$

produkcija iz $\mathcal{P}$, tada je izvođenje nove rečenične forme definirano kao

$$\alpha A_f \theta \beta \Rightarrow \alpha X_1\psi_1\theta'_1 X_2\psi_2\theta'_2 \ldots X_n\psi_n\theta'_n\beta$$

Vidimo da je u ovom slučaju „pojeden" neterminal A sa svojim indeksom, ali je nova rečenična forma izvedena na isti način kao u prvom slučaju.

Rečenična forma koja ne sadrži nijedan neterminal (niti indeksirani neterminal) jest rečenica indeksiranog jezika. Dakle, jezik generiran indeksiranom gramatikom može se definirati kao

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

i tipa je 1 (kontekstan).

♣ Primjer 7.2

Gramatika iz primjera 7.1 generira jezik $\{a^n b^n c^n: n \geq 0\}$. Na primjer, rečenica aaabbbccc može se dobiti nizom izvođenja:

$$\begin{aligned} S \Rightarrow T_g \Rightarrow T_{fg} \Rightarrow T_{ffg} \Rightarrow A_{ffg}B_{ffg}C_{ffg} \Rightarrow aA_{fg}B_{ffg}C_{ffg} \Rightarrow aaA_gB_{ffg}C_{ffg} \\ \Rightarrow aaaB_{ffg}C_{ffg} \Rightarrow aaabB_{fg}C_{ffg} \Rightarrow aaabbB_gC_{ffg} \Rightarrow aaabbbC_{ffg} \Rightarrow aaabbbcC_{fg} \\ \Rightarrow aaabbbccC_g \Rightarrow \underline{aaabbbccc} \end{aligned}$$

♣ Primjer 7.3

Evo još jedne indeksirane gramatike koja generira (kontekstni) jezik $\{a^n b^n c^n: n \geq 0\}$.

$$S \rightarrow A_x \quad A \rightarrow aA_y \mid B \quad B_y \rightarrow bBc \quad B_x \rightarrow \varepsilon$$

Na primjer, rečenica aaabbbccc može se dobiti nizom izvođenja:

$$\begin{aligned} S \Rightarrow A_x \Rightarrow aA_{yx} \Rightarrow aaA_{yyx} \Rightarrow aaaA_{yyyx} \Rightarrow aaaB_{yyx}c \Rightarrow aaabB_{yx}cc \\ \Rightarrow aaabbB_xccc \Rightarrow \underline{aaabbbccc} \end{aligned}$$

♣ Primjer 7.4

Indeksirane gramatike koja generira kontekstni jezik $\{ww: w \in \{a,b\}^*\}$ definirana je produkcijama:

$$S \rightarrow A_x \quad A \rightarrow aA_y \mid bA_z \mid B \quad B_y \rightarrow Ba \quad B_z \rightarrow Bb \quad B_x \rightarrow \varepsilon$$

Na primjer, rečenica abbabb može se dobiti nizom izvođenja:

$$\begin{aligned} S \Rightarrow A_x \Rightarrow aA_{yx} \Rightarrow abA_{zyx} \Rightarrow abbA_{zzyx} \Rightarrow abbB_{zzyx} \Rightarrow abbB_{zyx}b \Rightarrow abbB_{yx}bb \\ \Rightarrow abbB_xabb \Rightarrow \underline{abbabb} \end{aligned}$$

7.2 KONTEKSTNE GRAMATIKE

U prethodnim smo potpoglavljima definirali indeksiranu gramatiku jezika

$$L_{abc} = \{a^n b^n c^n: n>0\}$$

koja je uvela "međuneterminale". S obzirom na to da je L_{abc} konteksni jezik moguće je definirati gramatiku koja će ga generirati. Na primjer, lako se može dokazati da lontekstna gramatika s produkcijama

$$\begin{aligned} S &\rightarrow abc \mid aSBc \\ cB &\rightarrow Bc \\ bB &\rightarrow bb \end{aligned}$$

generira jezik L_{abc}:

$$\begin{aligned} &i=1: \quad S \Rightarrow \underline{abc} \\ &i=2: \quad S \Rightarrow aSBc \Rightarrow aabcBc \Rightarrow aabBcc \Rightarrow \underline{aabbcc} \\ &\ldots \\ &i=n: \quad S \overset{n-1}{\underset{(S\rightarrow aSBc)}{\Rightarrow}} a^{n-1}S(Bc)^{n-1} \underset{(S\rightarrow abc)}{\Rightarrow} a^{n}b(cB)^{n-1}c \overset{n-1}{\underset{(cB\rightarrow Bc)}{\Rightarrow}} a^{n}bB^{n-1}c^{n} \\ &\qquad \overset{n-1}{\underset{(bB\rightarrow bb)}{\Rightarrow}} \underline{a^{n}b^{n}c^{n}} \end{aligned}$$

U poglavlju posvećenom gramatikama definirali smo kontekstnu gramatiku koja generira kontekstni jezik, jezik tipa 1. Bila je to gramatika $G=(\mathcal{N}, \mathcal{T}, \mathcal{P}, S)$ u kojoj je svaka produkcija oblika

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

Posljedica te definicije bila je da kontekstna gramatika ne smije sadržavati prazne produkcije. Drugim riječima, gramatika je kontekstna ako se sve produkcije mogu napisati kao

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

koja nam zorno predočuje svojstvo konteksnih gramatika da neterminal A postaje niz γ u kontekstu α i β. U novijim radovima iz teorije formalnih jezika može se naći definicija kontekstne gramatike koja je jednaka našoj, ali uz dodatak da je dopuštena produkcija

$$S \rightarrow \varepsilon$$

uz uvjet da se S ne smije pojavljivati na desnoj strani (alternativi) nijedne produkcije. Za kontekstnu gramatiku se još kaže da je s <u>povećavajućom duljinom</u> (length-increasing).

♦ Rečenična forma kontekstne gramatike

Rečenična forma kontekstnih gramatika definirana je na isti način kao u slučaju gramatike bilo kojeg tipa, ali je izvođenje uvijek slijeva:

$$S \overset{*}{\underset{lm}{\Rightarrow}} w$$

♣ Primjer 7.5

Dana je gramatika koja generira jezik Lanc:

S → ABSc| Abc
BA → CA
CA → CB
CB → AB
Bb → bb
A → a

Evo nekoliko nizova izvođenja:

(1) S⇒ **A**bc ⇒ a**b**c
(2) S⇒ **A**BSc ⇒ a**B**Sc ⇒ a**BA**bcc ⇒ a**CA**bcc ⇒ a**CB**bcc ⇒ a**AB**bcc ⇒ aa**Bb**cc ⇒ aabbcc
(3) S⇒ **A**BSc ⇒ a**B**Sc ⇒ a**BA**BScc ⇒ a**CA**Bcc ⇒ a**CB**B**Scc** ⇒ a**AB**Bcc ⇒ aaBB**S**cc
⇒ aaB**BA**bccc ⇒ aaB**CA**bccc ⇒ aaB**CB**bccc ⇒ aa**BA**Bbccc ⇒ aa**CA**Bbccc
⇒ aa**CB**Bbccc ⇒ aa**AB**Bbccc ⇒ aaaB**Bb**ccc ⇒ aaa**Bb**bccc ⇒ aaabbbccc

♦ Kurodina normalna forma (KNF)

Kaže se da je kontekstna gramatika $\mathcal{G}=(\mathcal{N}, \mathcal{T}, \mathcal{P}, \mathcal{S})$ u Kurodinoj normalnoj formi (KNF) ako joj produkcije imaju jedan od četiri oblika:

1) A → α
2) A → BC
3) AB → AC
4) AB → BA

gdje su: $\alpha \in \mathcal{N} \cup \mathcal{T}$, A, B, C $\in \mathcal{N}$. Ovim se oblicima dodaje i $\mathcal{S} \to \varepsilon$ pod uvjetom da se $\mathcal{S}$ ne smije pojaviti niti u jednoj alternativi produkcija gramatike.

Može se dokazati da svaka kontekstna gramatika ima barem jednu ekvivalentnu gramatiku s produkcijama u KNF.

♣ Primjer 7.6

Gramatika s produkcijama:

S → A
A → a| BC
C →)
B → DA| DE| BC
D → (
EC → FC
EH → DH
F → GA| HA
G → DH
H → +| *

jest kontekstna gramatika u KNF. Primijetiti da generira beskonteksni jezik i ekvivalentna je, na primjer, gramatici:

E → E+E| E*E| (E)| a

♣ Primjer 7.7

Gramatika s produkcijama:

S → abc| aXbc Xb → bX Xc → Ybcc bY → Yb aY → aaX| aa

može se transformirati u ekvivalentnu gramatiku u KNF:

S → AD| ED AY → AE| AA
D → BC BY → YB
E → AF A → a
FB → BF B → b
FC → XC C → c
X → YD

Na primjer, rečenica aabbcc može se dobiti nizom izvođenja:

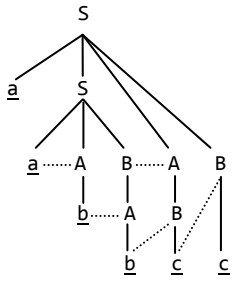
S ⇒ ED ⇒ AFD ⇒ AFBC ⇒ ABFC ⇒ ABXC ⇒ ABYDC ⇒ AYBDC ⇒AABDC
⇒ aABDC ⇒ aaBDC ⇒ aaBBCC *⇒ aabbcc

7.3 GRAF IZVOĐENJA

Kao što znamo, niz izvođenja linearnih i beskontekstnih gramatika može biti prikazan stablom, koje smo nazvali stablo izvođenja. Međutim, takav prikaz nije moguć u slučaju kontekstnih gramatika jer lijeve strane produkcija mogu sadržavati više od jednog neterminala. Na primjer, sljedeća kontekstna gramatika generira jezik L_{abc}

S → aSAB| aAB BA → AB aA → ab bA → bb bB→ bc cB→ cc

Graf izvođenja rečenice aabbcc možemo prikazati sljedećim "stablom" izvođenja.



Grane prikazane isprekidanim crtama pokazuju kontekstne aspekte u izvođenju rečenice. Da bismo niz izvođenja prikazali grafom, kojeg ćemo sada zvati graf izvođenja, trebamo svakoj produkciji iz $\mathcal{P}$ gramatike $\mathcal{G}=(\mathcal{N},\mathcal{T},\mathcal{P},\mathcal{S})$ pridružiti ime f

$f : \alpha\rightarrow\beta,\ \alpha\in(\mathcal{N}\cup\mathcal{T})^*\mathcal{N}(\mathcal{N}\cup\mathcal{T})^*,\ \beta\in(\mathcal{N}\cup\mathcal{T})^*,\ f\notin(\mathcal{N}\cup\mathcal{T})$

♣ Primjer 7.8

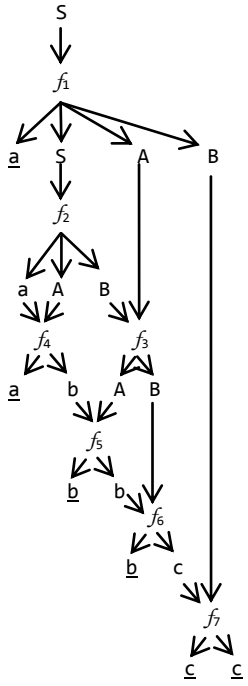
Ako produkcijama gramatike jezika L_{abc} iz prethodnog primjera pridružimo imena:

f_1: S → aSAB f_5: bA → bb
f_2: S → aAB f_6: bB → bc
f_3: BA → AB f_7: cB → cc
f_4: aA → ab

tada je, na primjer, niz izvođenja rečenice aabbcc:

$$S \overset{f_1}{\Rightarrow} aSAB \overset{f_2}{\Rightarrow} aaABAB \overset{f_4}{\Rightarrow} aabBAB \overset{f_3}{\Rightarrow} aabABB \overset{f_5}{\Rightarrow} aabbBB \overset{f_6}{\Rightarrow} aabbcB \overset{f_7}{\Rightarrow} aabbcc$$

što se može prikazati grafom izvođenja:



7.4 DVOSTRUKO-STOGOVNI AUTOMAT

Vidjeli smo da nam je u slučaju beskontekstnih jezika bio potreban automat koji ima pomoćnu memoriju sa strukturom stoga. Bio je to "stogovni" automat (generator). Veličina memorije stoga rabljena u generiranju rečenice x bila je linearno proporcionalna njezinoj duljini. Može se dokazati da takav automat ne može generirati (niti prepoznavati) rečenice kontekstnog jezika. Za to je potrebno rabiti automate s dva stoga ili "linearno ograničeni" automat, čiju definiciju dajemo u nastavku.

♦ Dvostruko-stogovni automat

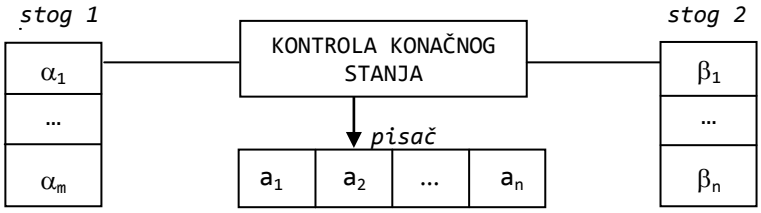
Dvostruko-stogovni automat definiran je kao uređena sedmorka

$$\mathcal{P}t = (Q, \Sigma, \Gamma_1, \Gamma_2, \Delta, s, \mathcal{F})$$

gdje su:

Q	konačan skup stanja (kontrole konačnog stanja)
Σ	ulazni alfabet
Γ_1, Γ_2	alfabet prvog i drugog stoga
Δ	funkcija prijelaza, definirana kao $\Delta: Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma_1 \times \Gamma_2 \rightarrow Q \times \Gamma_1^* \times \Gamma_2^*$
s	početno stanje, $s \in Q$
$\mathcal{F}$	skup konačnih stanja, $\mathcal{F} \subseteq Q$

Vidimo da se ovaj automat razlikuje od stogovnog automata u definiciji funkcije prijelaza Δ koja koristi dva stoga kao pomoćnu memoriju. Kaže se da je $\mathcal{P}t$ linearno-ograničen jer je duljina oba stoga linearno proporcionalna duljini generiranog niza. Dvostruko-stogovni generator, koji ima pisač i izlaznu vrpcu, prikazan je na sl. 7.1.



Sl. 7.1 – *Dvostruko-stogovni genenerator.*

♦ Konfiguracija dvostruko-stogovnog generatora

Konfiguracija dvostruko-stogovnog generatora $\mathcal{P}t$ jest (q,w,α,β) iz $Q\times\Sigma^*\times\Gamma_1^*\times\Gamma_2^*$, gdje su:

q tekuće stanje
w generirani niz znakova
α niz znakova koji predstavlja sadržaj prvog stoga; vrh je prvi znak niza
β niz znakova koji predstavlja sadržaj drugog stoga; vrh je prvi znak niza

Početna konfiguracija je $(s,\varepsilon,\varepsilon,\varepsilon)$, a završna konfiguracija $(q,w,\varepsilon,\varepsilon)$, $q\in\mathcal{F}$, $w\in\Sigma^*$.

♦ Pomak dvostruko-stogovnog generatora

Pomak dvostuko-stogovnog generatora $\mathcal{P}t$ jest relacija $\vdash_{\mathcal{P}t}$ (ili samo $\vdash$ ako se $\mathcal{P}t$ podrazumijeva):

$$(q, w, \alpha, \beta) \vdash (q', wa, \gamma_1\alpha, \gamma_2\beta)$$

ako $\delta(q, a, \gamma_1, \gamma_2)$ sadrži (q', γ_1, γ_2) za $q, q'\in Q$, $a\in\Sigma\cup\{\varepsilon\}$, $w\in\Sigma^*$, $\gamma_1\in\Gamma_1$, $\gamma_2\in\Gamma_2$. Kaže se da je niz w generiran s $\mathcal{P}t$ ako

$$(s,\varepsilon,\varepsilon,\varepsilon) \vdash^* (q,w,\varepsilon,\varepsilon),\ q\in\mathcal{F}$$

♣ Primjer 7.9

Dvostruko-stogovni automat gdje su:

$Q = \{q_0,q_1,f\}$ $\Sigma = \{a,b,c\}$ $\Gamma_1 = \{a\}$ $\Gamma_2 = \{b\}$ $s = q_0$ $\mathcal{F} = \{f\}$
$\Delta = \{ ((q_0,a,\varepsilon,\varepsilon),(q_0,a,\varepsilon)), ((q_0,b,a,\varepsilon),(q_1,\varepsilon,b)), ((q_1,b,a,\varepsilon),(q_1,\varepsilon,b)), ((q_1,c,\varepsilon,b),(f,\varepsilon,\varepsilon)),$
$((f,c,\varepsilon,b),(f,\varepsilon,\varepsilon)) \}$

generira (i prepoznaje) kontekstni jezik $\{a^nb^nc^n:\ n>0\}$. Na primjer, rečenica aabbcc može se generirati sa:

$(q_0,\varepsilon,\varepsilon,\varepsilon) \vdash (q_0,a,a,\varepsilon) \vdash (q_0,aa,aa,\varepsilon) \vdash (q_1,aab,a,b) \vdash (q_1,aabb,\varepsilon,bb) \vdash (f,aabbc,\varepsilon,b)$
$\vdash (f,aabbcc,\varepsilon,\varepsilon)$

7.5 EKVIVALENTNOST DVOSTRUKO-STOGOVNIH AUTOMATA I KONTEKSTNIH GRAMATIKA

Rečenice kontekstnih jezika mogu biti generirane kontekstnim gramatikama ili dvostruko-stogovnim automatom (generatorom), pa očekujemo da će kontekstna gramatika zadanog kontekstnog jezika imati ekvivalentni dvostruko-stogovni (ili linearno-ograničeni) automat. Da bismo to pokazali najprije definiramo dvostruko-stogovni automat s jednim stanjem.

♦ Dvostruko-stogovni automat s jednim stanjem

Ako je $\mathcal{G}=(\mathcal{N},\mathcal{T},\mathcal{P},\mathcal{S})$ kontekstna gramatika u Kurodinoj normalnoj formi može se definirati dvostruko-stogovni automat s jednim stanjem kao uređena sedmorka

$$\mathcal{P}q=(q,\Sigma,\Gamma_1,\Gamma_2,\Delta,z_1,z_2)$$

gdje su:

q	stanje
Σ	ulazni alfabet
Γ_1	alfabet prvog stoga, $\Gamma_1=\mathcal{N}\cup\mathcal{T}\cup\{z_2,z_\theta\}$
Γ_2	alfabet drugog stoga, $\Gamma_2=\mathcal{N}\cup\mathcal{T}\cup\{z_1\}$
Δ	funkcija prijelaza, definirana kao $\Delta: Q\times(\Sigma\cup\{\varepsilon\})\times\Gamma_1\times\Gamma_2 \rightarrow Q\times\Gamma_1^*\times\Gamma_2^*$
z_1,z_2	početni simobli prvog i drugog stoga

Funkcija prijelaza definirana je kao

$\Delta(q,\varepsilon,z_1,z_2) = (q,z_1,\mathcal{S}z_\theta)$ $\quad$ $\Delta(q,\varepsilon,z_1,z_\theta) = (q,\varepsilon,\varepsilon)$

$\Delta(q,\varepsilon,z_1,\mathcal{S}) = (q,\varepsilon,\varepsilon)$ ako je $\mathcal{S}\rightarrow\varepsilon\in\mathcal{P}$

$\Delta(q,\varepsilon,A,B) = (q,C,D)$ ako je $AB\rightarrow CD\in\mathcal{P}$

$\Delta(q,\varepsilon,A,\varepsilon) = (q,\gamma,\varepsilon)$, $\quad$ $\Delta(q,\varepsilon,\varepsilon,A) = (q,\varepsilon,\gamma)$ ako je $A\rightarrow\gamma\in\mathcal{P}$

$\Delta(q,\varepsilon,A,\varepsilon) = (q,BC,\varepsilon)$, $\quad$ $\Delta(q,\varepsilon,\varepsilon,A) = (q,\varepsilon,BC)$ ako je $A\rightarrow BC\in\mathcal{P}$

$\Delta(q,a,a,\varepsilon) = (q,\varepsilon,\varepsilon)$, $\quad$ $\Delta(q,a,\varepsilon,a) = (q,\varepsilon,\varepsilon)$ ako je $A\rightarrow a\in\mathcal{P}$

$\Delta(q,\varepsilon,\varepsilon,Z) = (q,Z,\varepsilon)$, $\quad$ $\Delta(q,\varepsilon,Z,\varepsilon) = (q,\varepsilon,Z)$ ako je $Z\in\mathcal{N}\cup\mathcal{T}$

• Teorem 7.1

Neka je $\mathcal{G}=(\mathcal{N},\mathcal{T},\mathcal{P},\mathcal{S})$ kontekstna gramatika u Kurodinoj normalnoj formi i $\mathcal{P}q$ izvedeni dvostruko-stogovni automat s jednim stanjem tada je $\mathcal{L}(\mathcal{G})=\mathcal{L}(\mathcal{P}q)$ pa kažemo da su $\mathcal{G}$ i $\mathcal{P}q$ ekvivalentni.

Dakako, važno je upamtiti što teorem 7.1 tvrdi, dok nas njegov dokaz ne zanima. Dalje, valja primijetiti da za automat s praznom stogovnom listom ne trebamo imati nijedno završno stanje, jer je prazna potisna lista uvjet okončanja postupka generiranja rečenice jezika (ili prihvaćanja, ako je automat u ulozi prepoznavača).

♣ Primjer 7.10

Gramatika s produkcijama:

$S \rightarrow AD \mid ED$ $\quad$ $D \rightarrow BC$ $\quad$ $E \rightarrow AF$ $\quad$ $FB \rightarrow BF$ $\quad$ $FC \rightarrow XC$ $\quad$ $X \rightarrow YD$

$AY \rightarrow AE \mid AA$ $\quad$ $BY \rightarrow YB$ $\quad$ $A \rightarrow a$ $\quad$ $B \rightarrow b$ $\quad$ $C \rightarrow c$

jest kontekstna gramatika u KNF-u koja generira jezik L_{abc}. Na primjer, rečenica aabbcc može se dobiti nizom izvođenja:

$$\begin{aligned}S &\Rightarrow ED \Rightarrow AFD \Rightarrow AFBC \Rightarrow ABFC \Rightarrow ABXC \Rightarrow ABYDC \Rightarrow AYBDC \Rightarrow AABDC\\ &\Rightarrow aABDC \Rightarrow aaBDC \Rightarrow aaBBCC \Rightarrow aabBCC \Rightarrow aabbCC \Rightarrow aabbcC \Rightarrow \underline{aabbcc}\end{aligned}$$

Ekvivalentni dvostruko-stogovni automat s jednim stanjem definiran je kao

$$\mathcal{P}q = (q, \{a,b,c,\}, \{a,b,c,A,B,C,D,E,F,S,X,Y,z_0,z_1,z_2\}, \{a,b,c,A,B,C,D,E,F,S,X,Y,z_1,z_2\}, \Delta, z_1, z_2)$$

gdje je funkcija prijelaza definirana kao

$$\begin{array}{llll}
\Delta(q,\varepsilon,z_1,z_2) = (q,z_1,Sz_0) & \Delta(q,\varepsilon,z_1,z_0) = (q,\varepsilon,\varepsilon) & & \\
\Delta(q,\varepsilon,S,\varepsilon) = \{(q,AD,\varepsilon),(q,ED,\varepsilon)\} & \Delta(q,\varepsilon,\varepsilon,S) = \{(q,\varepsilon,AD),(q,\varepsilon,ED)\} & & \\
\Delta(q,\varepsilon,D,\varepsilon) = (q,BC,\varepsilon) & \Delta(q,\varepsilon,\varepsilon,D) = (q,\varepsilon,BC) & & \\
\Delta(q,\varepsilon,E,\varepsilon) = (q,AF,\varepsilon) & \Delta(q,\varepsilon,\varepsilon,E) = (q,\varepsilon,AF) & & \\
\Delta(q,\varepsilon,X,\varepsilon) = (q,YD,\varepsilon) & \Delta(q,\varepsilon,\varepsilon,X) = (q,\varepsilon,YD) & & \\
\Delta(q,\varepsilon,A,Y) = \{(q,A,E),(q,A,A)\} & \Delta(q,\varepsilon,B,Y) = (q,Y,B) & & \\
\Delta(q,\varepsilon,F,B) = (q,B,F) & \Delta(q,\varepsilon,F,C) = (q,X,C) & \Delta(q,\varepsilon,B,Y) = (q,Y,B) & \\
\Delta(q,\varepsilon,A,\varepsilon) = (q,a,\varepsilon) & \Delta(q,\varepsilon,B,\varepsilon) = (q,b,\varepsilon) & \Delta(q,\varepsilon,C,\varepsilon) = (q,c,\varepsilon) & \\
\Delta(q,a,a,\varepsilon) = (q,\varepsilon,\varepsilon) & \Delta(q,a,\varepsilon,a) = (q,\varepsilon,\varepsilon) & & \\
\Delta(q,b,b,\varepsilon) = (q,\varepsilon,\varepsilon) & \Delta(q,b,\varepsilon,b) = (q,\varepsilon,\varepsilon) & & \\
\Delta(q,c,c,\varepsilon) = (q,\varepsilon,\varepsilon) & \Delta(q,c,\varepsilon,c) = (q,\varepsilon,\varepsilon) & & \\
\Delta(q,\varepsilon,\varepsilon,Z) = (q,Z,\varepsilon) & \Delta(q,\varepsilon,Z,\varepsilon) = (q,\varepsilon,Z) & Z\in\{a,b,c,A,B,C,D,E,F,X,Y\} &
\end{array}$$

Na primjer, rečenica aabbcc bit će generirana nizom:

$$\begin{aligned}(q,\varepsilon,z_1,z_2) &\Rightarrow (q,\varepsilon,z_1,\ Sz_0) \Rightarrow (q,\varepsilon,z_1,\ EDz_0) \Rightarrow (q,\varepsilon,z_1,\ AFDz_0)\\ &\Rightarrow (q,\varepsilon,FAz_1,\ Dz_0) \Rightarrow (q,\varepsilon,FAz_1,\ BCz_0) \Rightarrow (q,\varepsilon,BAz_1,\ FCz_0)\\ &\Rightarrow (q,\varepsilon,FBAz_1,\ Cz_0) \Rightarrow (q,\varepsilon,XBAz_1,\ Cz_0) \Rightarrow (q,\varepsilon,BAz_1,\ XCz_0)\\ &\Rightarrow (q,\varepsilon,BAz_1,\ YDCz_0) \Rightarrow (q,\varepsilon,YAz_1,\ BDCz_0) \Rightarrow (q,\varepsilon,Az_1,\ YBDCz_0)\\ &\Rightarrow (q,\varepsilon,Az_1,\ ABDCz_0) \Rightarrow (q,\varepsilon,az_1,\ ABDCz_0) \Rightarrow (q,a,z_1,\ ABDCz_0)\\ &\Rightarrow (q,a,z_1,\ aBDCz_0) \Rightarrow (q,aa,z_1,\ BDCz_0) \Rightarrow (q,aa,z_1,\ bDCz_0)\\ &\Rightarrow (q,aab,z_1,\ DCz_0) \Rightarrow (q,aab,z_1,\ BCCz_0) \Rightarrow (q,aab,z_1,\ bCCz_0)\\ &\Rightarrow (q,aabb,z_1,\ CCz_0) \Rightarrow (q,aabb,z_1,\ cCz_0) \Rightarrow (q,aabbc,z_1,\ Cz_0)\\ &\Rightarrow (q,aabbc,z_1,\ cz_0) \Rightarrow (q,aabbcc,z_1,\ z_0) \Rightarrow (q,aabbcc,\varepsilon,\ \varepsilon)\end{aligned}$$

7.6 IZVOĐENJE GRAMATIKA KONTEKSTNIH JEZIKA

Problem izvođenja gramatika kontekstnih jezika mnogo je teži od izvođenja gramatika beskontekstnih jezika. Na primjer, izvedimo gramatiku jezika

$$L = \{w:\ w\in\{a,b\}^+,\ N_a>N_b\}$$

$$\begin{aligned}= \{&a,\ aa,\ aaa,\ aab,\ aba,\ baa,\ aaaa,\ aaab,\ aaba, abaa, baaa, aaaaa, aaaab,\\ &aaaba, aabaa, abaaa, baaaa, aaabb, aabab, abaab, baaab, aabba, ababa,\\ &baaba, abbaa, babaa, bbaaa, aaaaaa, ...\}\end{aligned}$$

Prvo treba provjeriti je li jezik beskontekstan. Ako jest, mora biti zadovoljeno svojstvo napuhavanja beskontekstnih jezika. No, da bismo izveli (kontekstnu) gramatiku danog jezika moramo uočiti svojstvo njegovih rečenica. Očigledno će podjezik čije rečenice sadrže samo znakove a zadovoljavati dano svojstvo, pa možemo napisati produkcije (beskontekstne) gramatike koja generira takve rečenice:

$S \rightarrow a \mid aS$

Rečenice w koje sadrže znakove b moraju zadovoljavati uvjet

$N_b < |w|/2$

što, na primjer, znači da će rečenice duljine 3 ili 4 moći sadržati jedan znak b, 5 ili 6 dva, itd. Taj uvjet će zadovoljavati beskontekstna gramatika:

$S \rightarrow a \mid aS \mid aSB$
$B \rightarrow b$

Jezik kojeg generira ova gramatika jest $\{a^m b^n: m>n,\ m>0,\ n\geq 0\}$. Sada preostaje premještanje znakova b ulijevo, što nije problem ako dodamo još dvije produkcije

$aB \rightarrow Ba$
$bB \rightarrow Bb$

pa je rezultirajuća gramatika koja generira dani konteksni jezik:

$S \rightarrow a \mid aS \mid aSB$
$aB \rightarrow Ba$
$bB \rightarrow Bb$
$B \rightarrow b$

Pri izvođenju gramatika koristit ćemo supstituciju i faktorizaciju da bismo dobili ekvivalentnu „kompaktnu“ rezultirajuću gramatiku. Na primjer, ako smo inicijalno izveli gramatiku jezika L_{abc}

$S \rightarrow ABSc \mid Abc \qquad CB \rightarrow AB$
$BA \rightarrow CA \qquad Bb \rightarrow bb$
$CA \rightarrow CB \qquad A \rightarrow a$

poslije supstitucija $CB \rightarrow AB$ i $CA \rightarrow AB$ imamo

$S \rightarrow ABSc \mid Abc$
$BA \rightarrow AB$
$Bb \rightarrow bb$
$A \rightarrow a$

i supstituciju $A \rightarrow a$, pa je rezultirajuća ekvivalentna gramatika:

$S \rightarrow aBSc \mid abc \qquad Ba \rightarrow aB \qquad Bb \rightarrow bb$

U nastavku dajemo primjere izvođenja gramatika za još četiri kontekstna jezika.

Jezik $\{a^i b^j c^k: i,j,k>0,\ i=j+k\}$

$S \rightarrow aaSBc \mid aabc$?? a aaabbc, aaabcc
$cB \rightarrow Bc$
$bBc \rightarrow bcc \mid bbc \mid bbb$

Rečenica aaaaaabccccc:

S ⇒ aaSBC ⇒ aaaaSBcBc ⇒ aaaaaabcBcBc
⇒ aaaaaabBccBc ⇒ aaaaaabcccBc ⇒ aaaaaabccBcc
⇒ aaaaaabcBccc ⇒ aaaaaabBcccc ⇒ aaaaaabccccc

S → aaSBc| aabc
bçB → bcc| bbc| bbb
ccB → ccc

S ⇒ aaSBC ⇒ aaaaSBcBc ⇒ aaaaaabcBcBc
⇒ aaaaaabcccBc ⇒ aaaaaabccccc

S → aaSBc| ε
aB → ab
bçB → bcc| bbc| bbb
ccB → ccc

S ⇒ aaSBC ⇒ aaaaSBcBc ⇒ aaaaaaSBcBcBc ⇒ aaaaaaBcBcBc
⇒ aaaaaabcBcBc ⇒ aaaaaabcccBc ⇒ aaaaaabccccc

Jezik $\{a_1^n a_2^n \ldots a_k^n: k>1, n>0\}$

U prethodnom smo poglavlju pokazali kako se jednostavno može izvesti gramatika (beskontekstnog) jezika $\{a^n b^n: n \geq 1\}$. Međutim, gramatiku (kontekstnog) jezika $\{a^n b^n c^n: n \geq 1\}$, ili jezika L_{abc} kako smo ga ranije nazvali, mnogo je teže izvesti. Mnoge knjige koje se bave teorijom formalnih jezika sadrže kontekstne gramatike koje generiraju taj jezik. Neke su „jednostavne", druge složenije. Na primjer, u Shallit, J., 2009. imamo dvije takve gramatike:

$\mathcal{G}_1$: S → abc| aBSc Ba → aB Bb → bb

$\mathcal{G}_2$: S → ABSc| Abc BA → CA CA → CB CB → AB Bb → bb A → a

Autor je ove knjige u svom tridesetogodišnjem bavljenju teorijom formalnih jezika često davao zadatak studentima da izvedu gramatiku danog jezika. Rijetki su bili oni koji su to mogli riješiti, ali, ako bi i došli do rješenja, najčešće je to bilo intuitivno, bez posebnih objašnjenja. Evo nekoliko rješenja:

$\mathcal{G}_3$: S → abc| aSQ bQc → bbcc cQ → Qc

$\mathcal{G}_4$: S → abc| aSBC CB → BC bB → bb bC → bc cC → cc cB → Bc

$\mathcal{G}_5$: S → abc| aAbc Ab → bA Ac → Bbcc bB → Bb aB → aaA| aa

$\mathcal{G}_6$: S → abc| aSBc cB → Bc bB → bb

Analizirajmo kako je izvedena gramatika $\mathcal{G}_3$. Prva alternativa prve produkcije, S→abc, generira prvu rečenicu jezika: 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: S→aSQ. Iz niza izvođenja

$S \Rightarrow aSQ \Rightarrow aabcQ$

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

$$cQ \rightarrow Qc$$
$$bQc \rightarrow bbcc$$

Dakle, gramatika $\mathcal{G}_3$ generira jezik $\{a^n b^n c^n : n \geq 1\}$. Na primjer:

$$S \Rightarrow aSQ \Rightarrow aaSQQ \Rightarrow aaaSQQQ \Rightarrow aaaabcQQQ$$
$$\Rightarrow aaaabQcQQ \Rightarrow aaaabbccQQ \Rightarrow aaaabbcQcQ$$
$$\Rightarrow aaaabbQccQ \Rightarrow aaaabbbcccQ \Rightarrow aaaabbbccQc$$
$$\Rightarrow aaaabbbcQcc \Rightarrow aaaabbbQccc \Rightarrow aaaabbbbcccc$$

Sada se postavlja pitanje može li se izvesti gramatika jezika $\{a_1^n a_2^n \ldots a_k^n : k>1, n>0, a_i \neq a_j, \text{ za } i \neq j\}$? Do potvrdnog odgovora bi teško došli ako bismo promatrali produ-kcije većine danih gramatika. No, ako bismo analizirali gramatiku $\mathcal{G}_1$,

$\mathcal{G}_1$: $S \rightarrow abc \mid aBSc$
$Ba \rightarrow aB$
$Bb \rightarrow bb$

zaključili bismo da je izvedena na temelju ideje da se u svakoj novoj rečeničnoj formi mora osigurati jednak broj znakova a, b i c, pri čemu će krajnji znak rečenice, c, biti već na svom mjestu, preostaje da se znak a premjesti zdesna ulijevo, i to je to! Dakle, lako se može doći do produkcija gramatike za a_k znakova:

$S \rightarrow a_1a_2 \ldots a_k \mid a_1A_2 \ldots A_{k-1}Sa_k$
$A_{k-1}a_1 \rightarrow a_1A_{k-1}$
$A_{k-1}a_2 \rightarrow a_2A_{k-1}$
…
$A_{k-1}a_{k-1} \rightarrow a_{k-1}a_{k-1}$
…
$A_{k-2}a_1 \rightarrow a_1A_{k-2}$
$A_{k-2}a_2 \rightarrow a_2A_{k-2}$
…
$A_{k-2}a_{k-2} \rightarrow a_{k-2}a_{k-2}$
…
$A_2a_1 \rightarrow a_1A_2$
$A_2a_2 \rightarrow a_2a_2$

U sljedećoj su tablici dane gramatike za k=2 do k=5:

k	a_1	a_2	a_3	a_4	a_5	A_2	A_3	A_4	Gramatika
2	a	b							$S \rightarrow ab \mid aSb$
3	a	b	c			B			$S \rightarrow abc \mid aBSc$ $Ba \rightarrow aB$ $Bb \rightarrow bb$
4	a	b	c	d		B	C		$S \rightarrow abcd \mid aBCSd$

									Ca → aC Cb → bC Cc → cc Ba → aB Bb → bb
5	a	b	c	d	e	B	C	D	S → abcde| aBCDSe Da → aD Db → bD Dc → cD Dd → dd Ca → aC Cb → bC Cc → cc Ba → aB Bb → bb

Vidimo da je za $k=2$ gramatika beskontekstna i jednaka je gramatici koju smo već izveli promatrajući nizove a^nb^n. Za $k=4$, $a_1=a$, $a_2=b$, $a_3=c$, $a_4=d$, $A_2=B$ i $A_3=C$, imamo gramatiku:

$S \rightarrow abcd \mid aBCSd$
$Ca \rightarrow aC$
$Cb \rightarrow bC$
$Cc \rightarrow cc$
$Ba \rightarrow aB$
$Bb \rightarrow bb$

koja će generirati rečenice jezika $\{a^nb^nc^nd^n,\ n>0\}$. Na primjer, za $n=3$, imamo niz izvođenja:

$$S \Rightarrow aBCSd \Rightarrow aBCaBCSdd \Rightarrow aBCaBCabcddd$$
$$\Rightarrow aBaCBCabcddd \Rightarrow aaBCBCabcddd \Rightarrow aaBCBaCbcddd \Rightarrow aaBCaBCbcddd$$
$$\Rightarrow aaBaCBCbcddd \Rightarrow aaaBCBCbcddd \Rightarrow aaaBCBbCcddd \Rightarrow aaaBCbbCcddd$$
$$\Rightarrow aaaBbCbCcddd \Rightarrow aaabbCbCcddd \Rightarrow aaabbbCCcddd \Rightarrow aaabbbCccddd$$
$$\Rightarrow \underline{aaabbbcccddd}$$

ili

$$S \overset{16}{\Rightarrow} \underline{aaabbbcccddd}$$

Opisani postupak za izvođenje gramatike danog jezika ima „nedostatak“ u slučaju velikog broja znakova alfabeta, k, i velikog broja, n. Tada će postojati veliki broj rečeničnih formi u kojima će krajnji neterminali u „putovati“ slijeva udesno. Ako bismo rekurzivnu alternativu produkcije startnog simbola,

$$S \rightarrow a_1A_2 \ldots A_{k-1}Sa_k$$

napisali tako da je rekurzivna i postavili startni simbol u „sredinu“ neterminala A_2 i A_{k-1}:

$$S \rightarrow a_1A_2 \ldots A_{(k+1)/2}S \ldots A_{k-1}a_k$$

dobili bismo efikasniju gramatiku. Tada bi pomicanje neterminala ispred i iza startnog simola bilo najviše do sredine niza. Neterminali lijevo ispred startnog

simbola pomicat će se desno, a neterminali iza startnog simbola, lijevo. Na primjer, za n=3, imamo niz izvođenja:

S → abcd| aBSCd
cC → cc
dC → Cd
Ba → aB
Bb → bb

Na primjer, za n=3, imamo niz izvođenja:

S ⇒ aBSCd ⇒ aBaBSCCdd ⇒ aBaBabcdCCdd ⇒ aaBBabcdCCdd ⇒ aaBaBbcdCCdd
⇒ aaaBBbcdCCdd ⇒ aaaBbbcdCCdd ⇒ aaabbbcdCCdd ⇒ aaabbbcCdCdd ⇒ aaabbbccdCdd
⇒ aaabbbccCddd ⇒ aaabbbcccddd

ili

S ^{12}⇒ aaabbbcccddd

Jezik $\{a^n b^m c^n d^m,\ n>0,\ m>0\}$

Evo još jednog primjera kontekstnog jezika:

$\{a^n b^m c^n d^m,\ n>0,\ m>0\}$

Ideja je da se prvo osigura generiranje jednakog broja znakova a i c, što se postiže sa:

S → aXcd
X → aXc

U drugom koraku treba osigurati generiranje jednakog broja znakova b i d, bez obzira na njihovu poziciju u nizu:

X → bY
Y → ε| bYZd

i na kraju treba osigurati premještanje generiranih znakova d iza c:

Zdc → cZd
Zdd → dd

Dakle, gramatika koja generira jezik $\{a^n b^m c^n d^m,\ n>0,\ m>0\}$ definirana je sa:

S → aXcd
X → aXc| bY
Y → ε| bYZd
Zdc → cZd
Zdd → dd

Evo nekoliko primjera rečenica toga jezika:

S ⇒ aXcd ⇒ abYcd ⇒ abcd

S ⇒ aXcd ⇒ aaXccd ⇒ aabYccd ⇒ aabccd

S ⇒ aXcd ⇒ aaXccd ⇒ aabYccd ⇒ aabbYZdccd ⇒ aabbZdccd
⇒ aabbcZdcd ⇒ aabbccZdd ⇒ aabbccdd

S ⇒ aXcd ⇒ aaXccd ⇒ aabYccd ⇒ aabbYZdccd ⇒ aabbbYZdZdccd
⇒ aabbbZdZdccd ⇒ aabbbZdcZdcd ⇒ aabbbcZdZdcd
⇒ aabbbcZdcZdd ⇒ aabbbccZdZdd ⇒ aabbbccZddd ⇒ aabbbccddd

Jezik $\{a^{n^2},\ n>0\}$

Dakle, treba definirati gramatiku jezika {a, aaaa, aaaaaaaaa, aaaaaaaaaaaaaaaa, …}, tj. nizova a^{n^2}, $n>0$. U literaturi (G. E. Révész) se može naći rješenje:

S → a| CD
C → ACB| AB
AB → aBA
Aa → aA
Ba → aB
AD → Da
BD → Ea
BE → Ea
E → a

Pitanja i zadaci

1) Pokažite da gramatika dana s

S → aAB
Aa → SaB
Ab → SBb
B → SA| ab
bB → a

generira prazan jezik!

2) Koje jezike generiraju gramatike

G_1:
S → ACA
AC → AACA| ADc| AcD
D → AD| A
A → 0| 1

G_2:
S → AcB| BcA

Ac → XAcX
Bc → XBcX
A → 1Y| 1
B → 0Y| 0
X → 0| 1
Y → X| XY

3) Dokažite da su gramatike G_1 i G_2

G_1:	G_2:
S → EF| ESF	S → abcd| XSY
E → aB	Xa → aX
F → Cd	Xb → abb
Ba → aB	dY → Yd
Bb → bb	cY → ccd
BC → bc	
cC → Cc	
dC → Cd	

ekvivalentne (generiraju jezik $\{a^n b^n c^n d^n, n>0\}$).

8.
JEZICI BEZ OGRANIČENJA

0

8.1 GRAMATIKE BEZ OGRANIČENJA

- Proširena Kurodina normalna forma (EKNF)

8.2 TURINGOV STROJ

Turingov generator

- Turingov generator
- Konfiguracija i pomak Turingova generatora

8.3 IZVOĐENJE GRAMATIKA BEZ OGRANIČENJA

Jezik $L_1=\{ww: w\in\{a_1,a_2,\ldots,a_n\}^*, n>1\}$

Jezik $L_2=\{a^k: k=1+2+\ldots+n=(1+n)n/2,\ n\geq 1\}$

Jezik $L_3=\{a^{2^n}: n>0\}$

Jezik $L_4=\{a^{m^n}: m>0, n>0\}$

Jezik $L_5=\{a^{n^2}: n>0\}$

Jezik $L_6=\{a^{n^p}: n>0, p>2\}$

Jezik prirodnih brojeva djeljivih sa zadanim brojem

Jezici bez ograničenja ili jezici tipa 0 najšira su klasa jezika. Mogu biti generirani gramatikama bez ograničenja i Turingovim stojevima.

8.1 GRAMATIKE BEZ OGRANIČENJA

Gramatika tipa 0 ili gramatika bez ograničenja najšira je klasa gramatika prema Chomkyevoj hijerarhiji gramatika. Općenito, gramatika bez ograničenja sadrži produkcije oblika

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

♣ Primjer 8.1

Gramatika koja generira jezik $\{a^{n^2} : n \geq 1\}$

$S \rightarrow a \mid CD \quad C \rightarrow ACB \mid AB \quad AB \rightarrow aBA \quad Aa \rightarrow aA \quad Ba \rightarrow aB \quad AD \rightarrow Da$

$BD \rightarrow Ea \quad BE \rightarrow Ea \quad E \rightarrow a$

jest kontekstna, a ekvivalentna gramatika s produkcijama

$(1)\ A \rightarrow BRAE \quad (2)\ B \rightarrow BRAA \quad (3)\ RA \rightarrow aAR \quad (4)\ Ra \rightarrow aR \quad (5)\ RE \rightarrow E$
$(6)\ B \rightarrow X \quad (7)\ XA \rightarrow X \quad (8)\ Xa \rightarrow aX \quad (9)\ XE \rightarrow \varepsilon$

jest gramatika bez ograničenja, jer sadrži produkcije, (5), (7) i (9), u kojima je $|\alpha| > |\beta|$.

♣ Primjer 8.2

Jezik $\{ww: w \in \{0,1\}^*\}$ može biti generiran gramatikom bez ograničenja s produkcijama:

$S \rightarrow AS0 \mid BS1 \mid C \quad AC \rightarrow 0C \quad BC \rightarrow 1C \quad A0 \rightarrow 0A \quad B0 \rightarrow 0B \quad A1 \rightarrow 1A \quad B1 \rightarrow 1B \quad C \rightarrow \varepsilon$

♦ Proširena Kurodina normalna forma (EKNF)

U prethodnom smo poglavlju uveli Kurodinu normalnu formu (KNF) konteksnih gramatika. Definira se i proširena Kurodina normalna forma (EKNF) za gramatiku bez ograničenja u kojoj produkcije mogu biti napisane da zadovoljavaju slijedeće oblike:

$1)\ A \rightarrow \alpha \quad 2)\ A \rightarrow BC \quad 3)\ AB \rightarrow CD \quad 4)\ A \rightarrow \varepsilon \quad 5)\ AB \rightarrow \varepsilon$

gdje su: $\alpha \in \mathcal{N} \cup \mathcal{T}$, $A,B,C,D \in \mathcal{N}$.

● Teorem 8.1

Svaka gramatika bez ograničenja ima barem jednu ekvivalentnu gramatiku, također bez ograničenja, u kojoj produkcije mogu biti napisane u EKNF.

♣ **Primjer 8.3**

Jezik $\{ww: w \in \{0,1\}^*\}$ može biti generiran gramatikom bez ograničenja s produkcijama:

$S \to ABC \quad AB \to 0AD \mid 1AE \mid \varepsilon \quad D0 \to 0D \quad D1 \to 1D \quad E0 \to 0E \quad E1 \to 1E$
$C \to \varepsilon \quad DC \to B0C \quad EC \to B1C \quad 0B \to B0 \quad 1B \to B1$

Ekvivalentna gramatika s produkcijama u EKNF jest:

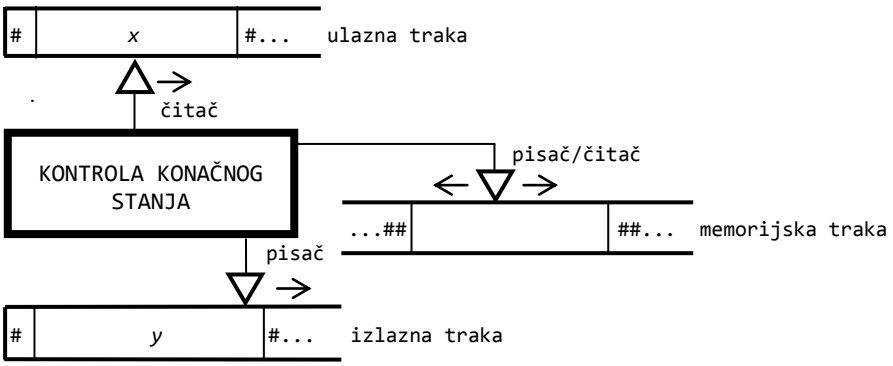
$S \to FG \quad F \to A \quad G \to BC \quad AB \to HI \mid JK \mid \varepsilon \quad H \to 0 \quad I \to AD$
$J \to 1 \quad K \to AE \quad DM \to MD \quad DL \to LD \quad EL \to LE \quad EM \to ME$
$C \to \varepsilon \quad DC \to PQ \quad P \to B \quad L \to 0 \quad M \to 1 \quad Q \to LC$
$EC \to RT \quad R \to B \quad T \to MC \quad LB \to BL \quad MB \to BM$

8.2 TURINGOV STROJ

Koncept Turingovog stroja ("Turing machine") jedan je od najvažniji matematičkih koncepata razvijen tridesetih godina dvadesetog stoljeća (Alan Turing, 1936.). Važno je napomenuti da su Turing u Engleskoj i istovremeno, ali ne u tako očitom obliku, E. Post u SAD, formulirali pojam apstraktnog računskog stroja i postavili osnovne principijelne karakteristike nekoliko godina prije pojave prvog prototipa računala s automatskim upravljanjem.

Turingov stroj nije stvarno sagrađen. Danas se koristi kao osnovni model za razrješavanje biti pojma "izračunivosti računalnih algoritama", "teorije složenosti" itd, a za nas je od posebnog značenja njegova primjena u teoriji i praksi formalnih jezika.

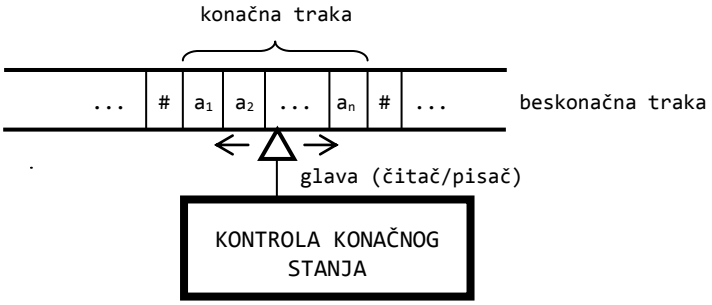
Postoji više varijanti Turingovog stroja, od onih koji imaju samo jednu traku, pa do onih s neograničenim brojem traka. U našim primjenama općenita shema Turingovog stroja prikazana je na sl. 8.1.



Sl. 8.1 – *Turingov stroj.*

Turingov generator

Prema hijerarhijskoj strukturi jezika i automata Turingov stroj jest automat koji generira i prihvaća jezike tipa 0 ili jezike bez ograničenja. Turingov stroj u svojstvu generatora, zvat ćemo ga Turingov generator, može se prikazati kao na sl. 8.2.



Sl. 8.2 – *Turingov generator.*

Dakle, Turingov generator ima samo jednu traku koja je istodobno ulazna, izlazna i memorijska. Kontrola konačnog stanja premješta glavu (čitač/pisač) od stanja do stanja, lijevo ili desno u svakom koraku, te čita i upisuje znakove u ćelije trake. Inicijalno je beskonačna traka prazna (sadrži beskonačno ćelija praznih znakova ili blankova). Glava se nalazi na proizvoljnoj poziciji. Beskonačna će traka na kraju u jednom svom konačnom dijelu sadržavati znakove alfabeta (rečenicu) jezika kojeg Turingov generator generira. Poslije ovog neformalnog opisa Turingovog generatora evo i njegove formalne definicije.

♦ Turingov generator

Turingov generator jest uređena šestorka, $\mathcal{T}g=(Q, \Sigma, \Gamma, \delta, q_0, F)$, gdje su:

- Q konačan skup stanja (kontrole konačnog stanja)
- Σ ulazni alfabet
- Γ konačni skup alfabeta znakova ulazno-izlazne trake, koji sadrži ulazni alfabet i posebni simbol # nazvan blank (razmak)
- δ funkcija prijelaza, definirana kao

  $$\delta: Q\times\Sigma\cup\{\#\} \rightarrow Q\times\{\Gamma\setminus\{\#\}\}\times\{-1,0,1\}$$

  gdje je -1 pomak glave ulijevo za jedno mjesto, 0 mirovanje glave i 1 pomak glave udesno za jedno mjesto
- q_0 početno stanje, $q_0\in Q$
- F skup završnih stanja, $F\subseteq Q$

♦ Konfiguracija i pomak Turingova generatora

Konfiguracija Turingova generatora $\mathcal{T}g=(Q, \Sigma, \Gamma, \delta, q_0, F)$ jest $(\alpha,q,x\beta)$ iz $\Gamma^*\times Q\times\Gamma^*$, gdje su:

- α niz znakova konačne trake lijevo od pozicije glave, $\alpha\in\Gamma^*$
- q tekuće stanje, $q\in Q$
- x tekući znak (na poziciji glave), $x\in\Gamma$
- β niz znakova desno od pozicije glave, $\beta\in\Gamma^*$

Početna konfiguracija je $(\varepsilon,q_0,\#)$, a završna konfiguracija (ε,q,w), $q\in F$, $w\in\Sigma^*$.

Pomak Turingova generatora $\mathcal{T}g=(Q, \Sigma, \Gamma, \delta, q_0, \mathcal{F})$ jest relacija $\vdash_{\mathcal{T}g}$ (ili samo $\vdash$ ako se $\mathcal{T}g$ podrazumijeva) definirana kao što slijedi:

$(\alpha, q_1, a\beta) \vdash (\alpha, q_2, b\beta)$ ako je $\delta(q_1, a) = (q_2, b, 0)$
$(\alpha, q_1, a\beta) \vdash (\alpha b, q_2, \beta)$ ako je $\delta(q_1, a) = (q_2, b, 1)$
$(\alpha, q_1, \varepsilon) \vdash (\alpha, q_2, b)$ ako je $\delta(q_1, \#) = (q_2, b, 0)$
$(\alpha, q_1, \varepsilon) \vdash (\alpha b, q_2, \varepsilon)$ ako je $\delta(q_1, \#) = (q_2, b, 1)$
$(\alpha a, q_1, b\beta) \vdash (\alpha, q_2, acb)$ ako je $\delta(q_1, b) = (q_2, c, -1)$
$(\varepsilon, q_1, a\beta) \vdash (\varepsilon, q_2, \#b\beta)$ ako je $\delta(q_1, a) = (q_2, b, -1)$
$(\alpha a, q_1, \varepsilon) \vdash (\alpha, q_2, ab)$
$(\varepsilon, q_1, \varepsilon) \vdash (\varepsilon, q_2, \#b)$
} ako je $\delta(q_1, \#) = (q_2, b, -1)$

gdje su $q_1, q_2 \in Q$, $a, b \in \Gamma$. Jezik generiran Turingovim generatorom $\mathcal{T}g$ definiran je kao

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

U primjerima koji slijede pokazat ćemo kako se jednostavno mogu definirati Turingovi generatori, najprije poznatog nam kontekstnog jezika L_{abc}, potom triju jezika bez ograničenja.

♣ Primjer 8.4

Jezik L_{abc} može biti generiran Turingovih generatorom u kojem je tablica prijelaza dana sa:

	#	A	B	a	b	c
→0	(1,A, 1)					
1	(1,A, 1) (2,ε,-1)					
2	(4,ε, 1)	(3,a, 1)	(2,B,-1)	(2,a,-1)		
3	(2,B,-1)		(3,B, 1)	(3,a, 1)		
4			(5,b, 1)	(4,a, 1)		
5	(6,c,-1)		(5,B, 1)		(5,b, 1)	(5,c, 1)
6	(7,ε, 1)		(5,b, 1)	(6,a,-1)	(6,b,-1)	(6,c,-1)
⊗7						

Ideja je sljedeća:

1) Generira se n znakova A: $(\varepsilon,0,\varepsilon) \vdash (A,1,\varepsilon) \vdash \ldots \vdash (A^n,1,\varepsilon)$.
2) Prelaskom u stanje 2, pa u 3 i ponovo u 2: $\vdash (A^{n-1},2,A) \vdash (A^{n-1}a,3,\varepsilon) \vdash (A,2,aB) \vdash^* (\varepsilon,2,\#a^nB^n)$ na kraju je dosegnuta konfiguracija $(\varepsilon,2,\#a^nB^n)$, odnosno, niz A^n "pretvoren" je u a^n i dodan niz B^n.
3) Prelaskom u stanje 4 i konfiguraciju $(\varepsilon,4,a^nB^n)$ poslije n pomaka i konfiguracije $(a^n,4,B^n)$ prelazi se u konfiguraciju

   a) $(a^nb,5,B^{n-1}) \vdash^{n-1} (a^nbB^{n-1},5,\varepsilon) \vdash (a^nbB^{n-2},6,Bc) \vdash (a^nbB^{n-2},5,bc) \vdash^* (\varepsilon,6,\#a^nb^nc^n) \vdash (\varepsilon,7,\underline{a^nb^nc^n})$, ako je $n>1$

   b) $(ab,5,\varepsilon) \vdash (a,6,bc) \vdash^* (\varepsilon,6,\#abc) \vdash (\varepsilon,7,\underline{abc})$, ako je $n=1$

Na primjer, rečenice abc i aaabbbccc bit će generirane nizom pomaka:

$(\varepsilon,0,\varepsilon)$
$\vdash (A,1,\varepsilon) \vdash (\varepsilon,2,A) \vdash (a,3,\varepsilon) \vdash (\varepsilon,2,aB) \vdash (\varepsilon,2,\#aB) \vdash (\varepsilon,4,aB)$
$\vdash (a,4,B) \vdash (ab,5,\varepsilon) \vdash (a,6,bc) \vdash (\varepsilon,6,abc) \vdash (\varepsilon,6,\#abc) \vdash (\varepsilon,7,\underline{abc})$

(ε,0,ε)
⊢ (A,1,ε) ⊢ (AA,1,ε) ⊢ (AAA,1,ε) ⊢ (AA,2,A)
⊢ (AAa,3,ε) ⊢ (AA,2,aB) ⊢ (A,2,AaB) ⊢ (Aa,3,aB)
⊢ (Aaa,3,B) ⊢ (AaaB,3,ε) ⊢ (Aaa,2,BB) ⊢ (Aa,2,aBB)
⊢ (A,2,aaBB) ⊢ (ε,2,AaaBB) ⊢ (a,3,aaBB) ⊢ (aa,3,aBB)
⊢ (aaa,3,BB) ⊢ (aaaB,3,B) ⊢ (aaaBB,3,ε) ⊢ (aaaB,2,BB)
⊢ (aaa,2,BBB) ⊢ (aa,2,aBBB) ⊢ (a,2,aaBBB) ⊢ (ε,2,aaaBBB)
⊢ (ε,2,#aaaBBB) ⊢ (ε,4,aaaBBB) ⊢ (a,4,aaBBB) ⊢ (aa,4,aBBB)
⊢ (aaa,4,BBB) ⊢ (aaab,5,BB) ⊢ (aaabB,5,B) ⊢ (aaabBB,5,ε)
⊢ (aaabB,6,Bc) ⊢ (aaabBb,5,c) ⊢ (aaabBbc,5,ε) ⊢ (aabBb,6,cc)
⊢ (aaabB,6,bcc) ⊢ (aaab,6,Bbcc) ⊢ (aaabb,5,bcc) ⊢ (aaabbb,5,cc)
⊢ (aaabbbc,5,c) ⊢ (aaabbbcc,5,ε) ⊢ (aaabbbc,6,cc) ⊢ (aaabbb,6,ccc)
⊢ (aaabb,6,bccc) ⊢ (aaab,6,bbccc) ⊢ (aaa,6,bbbccc) ⊢ (aa,6,abbbccc)
⊢ (a,6,aabbbccc) ⊢ (ε,6,aaabbbccc) ⊢ (ε,6,#aaabbbccc) ⊢ (ε,7,aaabbbccc)

♣ Primjer 8.5

Jezik {*ww*: *w*∈{a,b,c}$^+$} može biti generiran Turingovih generatorom u kojem je tablica prijelaza dana sa:

	#	a	b	c	A	B	C
→0	(1,A, 1),(1,B, 1), (1,C, 1)						
1	(1,A, 1),(1,B, 1), (1,C, 1),(7,ε,-1)						
2	(8,ε, 1)	(2,a,-1)	(2,b,-1)	(2,c,-1)	(7,A,-1)	(7,B,-1)	(7,C,-1)
3		(3,a, 1)	(3,b, 1)	(3,c, 1)	(4,a, 1)	(5,b, 1)	(6,c, 1)
4	(2,a,-1)	(4,a, 1)	(4,b, 1)	(4,c, 1)	(4,A, 1)	(4,B, 1)	(4,C, 1)
5	(2,b,-1)	(5,a, 1)	(5,b, 1)	(5,c, 1)	(5,A, 1)	(5,B, 1)	(5,C, 1)
6	(2,c,-1)	(6,a, 1)	(6,b, 1)	(6,c, 1)	(6,A, 1)	(6,B, 1)	(6,C, 1)
7	(3,ε, 1)	(3,a, 1)	(3,b, 1)	(3,c, 1)	(7,A,-1)	(7,B,-1)	(7,C,-1)
⊗8							

Svaki znak alfabeta označen je istim takvim velikim slovom. Prvo se generira niz $A_1A_2...A_n$∈{A,B,C}$^+$, prelazi u stanje 7, vraća se na početak generiranog niza i prelazi u stanje 3:

$$(\varepsilon,0,\#) \vdash (A_1,1,\#) \vdash^{n-1} (A_1...A_n,1,\#) \vdash (A_1...A_{n-1},7,A_n) \vdash^{n} (\varepsilon,7,\#A_1...A_n) \vdash (\varepsilon,3,A_1...A_n)$$

Sada se, u stanju 3, pretražuje generirani niz i nailaskom na A_i∈{A,B,C}, a prvi put će to biti A_1, A_i se zamjenjuje s jednakim takvim malim slovom i prelazi u stanje 4, ako je A_i=a, stanje 5, ako je A_i=b ili stanje 6, ako je A_i=c. Glava se pomiče na kraj generiranog niza i dopisuje ("kopira") znak A_i na poziciju n+i. Prelazi se u stanje 2 i vraća ulijevo sve dok tekući znak ne bude veliko slovo. Ako generirani niz ne sadrži veliko slovo doći će se do njegova početka, što znači da je postupak generiranja završen i stroj se zaustavlja. Nailaskom na veliko slovo prelazi se u stanje 7 i nastavlja pomicanje glave sve do nailaska na malo slovo. Tada se prelazi u stanje 3 i ponavlja opisani postupak. Pogledajmo, na primjer, kako će biti generirana rečenica abcabc:

(ε,0,ε)
⊢ (A,1,ε) ⊢ (AB,1,ε) ⊢ (ABC,1,ε) ⊢ (AB,7,C) ⊢ (A,7,BC)
⊢ (ε,7,ABC) ⊢ (ε,7,#ABC) ⊢ (ε,3,ABC) ⊢ (a,4,BC) ⊢ (aB,4,C)
⊢ (aBC,4,ε) ⊢ (aB,2,Ca) ⊢ (a,7,BCa) ⊢ (ε,7,aBCa) ⊢ (a,3,BCa)
⊢ (ab,5,Ca) ⊢ (abC,5,a) ⊢ (abCa,5,ε) ⊢ (abC,2,ab) ⊢ (ab,2,Cab)
⊢ (a,7,bCab) ⊢ (ab,3,Cab) ⊢ (abc,6,ab) ⊢ (abca,6,b) ⊢ (abcab,6,ε)
⊢ (abca,2,bc) ⊢ (abc,2,abc) ⊢ (ab,2,cabc) ⊢ (a,2,bcabc) ⊢ (ε,2,abcabc)
⊢ (ε,2,#abcabc) ⊢ (ε,8,abcabc)

♣ Primjer 8.6

Jezik $\{a^{2^n}: n>0\}$ može biti generiran Turingovih generatorom u kojem je tablica prijelaza dana sa:

	#	a	A
0	(1,A, 1)		
1	(2,A,-1), (3,A,-1)		
2	(9,ε, 1)		(2,a,-1)
3			(4,a,-1)
4	(5,A,-1)	(4,a, 1)	(4,A, 1)
5		(6,a,-1)	(5,A,-1)
6	(7,ε, 1), (8,ε, 1)	(6,a,-1)	(4,a, 1)
7	(2,ε,-1)	(7,A, 1)	(7,A, 1)
8	(3,ε,-1)	(8,A, 1)	(8,A, 1)
⊗9			

Ideja je da se prva rečenica a^2 (n=1) generira kao AA potom se, prelaskom u stanje 2, svako A zamijeni s a. Generiranje nizova za n>1 postiže se prelaskom u stanje 3 i dosezanja konfiguracije

$(\varepsilon,6,\#a^{2^{n-1}}A^{2^{n-1}})$

Potom će se niz a^{2^n} dobiti nizom pomaka

$(\varepsilon,6,\#a^{2^{n-1}}A^{2^{n-1}}) \vdash (\varepsilon,7,a^{2^{n-1}}A^{2^{n-1}}) \vdash^* (\varepsilon,9,a^{2^n})$

a niz $a^{2^{n+1}}$ nizom pomaka

$(\varepsilon,6,\#a^{2^{n-1}}A^{2^{n-1}}) \vdash (\varepsilon,8,a^{2^{n-1}}A^{2^{n-1}}) \vdash^* (\varepsilon,6,\#a^{2^n}A^{2^n}) \vdash (\varepsilon,7,a^{2^n}A^{2^n}) \vdash^* (\varepsilon,9,a^{2^{n+1}})$

Prve tri rečenice, a^2, a^4 i a^8, mogu biti generirane sa:

(ε,0,#)
⊢ (A,1,#)

⊢ (A,2,A)
⊢ (ε,2,Aa) ⊢ (ε,2,#aa)⊢ (ε,9,aa)

⊢ (A,3,A)
⊢ (Aa,4,#) ⊢ (A,5,aA) ⊢ (ε,6,AaA) ⊢ (a,4,aA) ⊢(aa,4,A) ⊢ (aaA,4,#)
⊢ (aa,5,AA) ⊢ (a,5,aAA) ⊢ (ε,6,aaAA)
⊢ (ε,6,#aaAA)

⊢ (ε,7,aaAA)
⊢ (A,7,aAA) ⊢ (AA,7,AA) ⊢ (AAA,7,A) ⊢ (AAAA,7,#) ⊢ (AAA,2,A)
⊢ (AA,2,Aa) ⊢ (A,2,Aaa) ⊢ (ε,2,Aaaa) ⊢ (ε,2,#aaaa) ⊢ (ε,9,aaaa)

⊢ (ε,8,aaAA)
⊢ (A,8,aAA) ⊢ (AA,8,AA) ⊢ (AAA,8,A) ⊢ (AAAA,8,#)
⊢ (AAA,3,A) ⊢ (AAAa,4,#) ⊢ (AAA,5,aA) ⊢ (AA,6,AaA)
⊢ (AAa,4,aA) ⊢ (AAaa,4,A) ⊢ (AAaaA,4,#) ⊢ (AAaa,5,AA)
⊢ (AAa,5,aAA) ⊢ (AA,6,aaAA) ⊢ (A,6,AaaAA) ⊢ (Aa,4,aaAA)
⊢ (Aaa,4,aAA) ⊢ (Aaaa,4,AA) ⊢ (AaaaA,4,A) ⊢ (AaaaAA,4,#)
⊢ (AaaaA,5,AA) ⊢ (Aaaa,5,AAA) ⊢ (Aaa,5,aAAA) ⊢ (Aa,6,aaAAA)
⊢ (A,6,aaAAA) ⊢ (ε,6,AaaaAAA) ⊢ (ε,6,AaaaAAA) ⊢ (a,4,aaaAAA)
⊢ (aa,4,aaAAA) ⊢ (aaa,4,aAAA) ⊢ (aaaa,4,AAA) ⊢ (aaaaA,4,AA)
⊢ (aaaaAA,4,A) ⊢ (aaaaAAA,4,#) ⊢ (aaaaAA,5,AA) ⊢ (aaaaA,5,AAA)
⊢ (aaaa,5,AAAA) ⊢ (aaa,5,aAAAA) ⊢ (aaa,5,aAAAA) ⊢ (aa,6,aaAAAA)
⊢ (a,6,aaaAAAA) ⊢ (ε,6,aaaaAAAA) ⊢ (ε,6,#aaaaAAAA) ⊢ (ε,7,aaaaAAAA)
⊢ (ε,7,aaaaAAAA) ⊢*(AAAAAAAA,7,#) ⊢ (AAAAAAA,2,A) ⊢*(ε,2,#aaaaaaaa)
⊢ (ε,9,aaaaaaaa)

♣ Primjer 8.7

Jezik $\{a^{n^2}: n>0\}$ može biti generiran Turingovih generatorom u kojem je tablica prijelaza dana sa:

	#	a	A
0	(1,A, 1)		
1	(2,ε,-1), (3,A,-1)		
2	(9,ε, 1)	(2,a,-1)	(2,a,-1)
3			(4,a, 1)
4	(5,A, 1)		(4,A, 1)
5	(6,A,-1)	(5,a, 1)	(5,A, 1)
6		(7,a,-1)	(6,A,-1)
7	(8,ε, 1)	(7,a,-1)	(5,a, 1)
8	(2,ε,-1), (3,A,-1)	(8,a, 1)	(8,A, 1)
⊗9			

Do nje se došlo na temelju promatranja rečenica a^{n^2} i $a^{(n-1)^2}$. Razlika njihovih duljina jednaka je

$$|a^{n^2}| - |a^{(n-1)^2}| = n^2 - (n-1)^2 = (n-n+1)(n+n-1) = 2n-1$$

Dakle, vrijedi $a^{n^2} = a^{(n-1)^2}a^{2n-1}$. Za $n=1$ imamo $a^1=a^0a^1=a$, a za $n=2$ je $a^4=a^1a^3$ itd. Općenito će rečenica a^{n^2} biti dobivena dopisujući nizove a^{2i-1} od $i=1$ do n, tj $a^{n^2}=a^1a^3a^5...a^{2n-1}$. Pogledajmo kako će biti generirani rečenice a, a^4 i a^9:

(ε,0,#)
 ⊢ (A,1,#)
  ⊢ (ε,2,A) ⊢ (ε,2,#a) ⊢ (ε,9,a)

  ⊢ (ε,3,AA) ⊢ (a,4,A) ⊢ (aA,4,#) ⊢ (aAA,5,#) ⊢ (aA,6,AA)
  ⊢ (a,6,AAA) ⊢ (ε,7,#aAAA) ⊢ (ε,8,aAAA) ⊢ (a,8,AAA) ⊢ (aA,8,AA) ⊢ (aAA,8,A)
  ⊢ (aAAA,8,#)
   ⊢ (aAA,2,A) ⊢ (aA,2,Aa) ⊢ (a,2,Aaa) ⊢ (ε,2,aaaa) ⊢ (ε,2,#aaaa) ⊢ (ε,9,a^4)

   ⊢ (aAA,3,AA) ⊢ (aAAa,4,A) ⊢ (aAAaA,4,#)
    ⊢ (aAAaAA,5,#) ⊢ (aAAaA,6,AA) ⊢ (aAA,6,aAAA) ⊢ (aA,7,AaAAA)
    ⊢ (aAa,5,aAAA) ⊢ (aAaa,5,AAA) ⊢ (aAaaA,5,AA) ⊢ (aAaaAA,5,A)
    ⊢ (aAaaAAA,5,#) ⊢ (aAaaAA,6,AA) ⊢ (aAaaA,6,AAA) ⊢ (aAaa,6,AAAA)
    ⊢ (aAa,6,aAAAA) ⊢ (aA,7,aaAAAA) ⊢ (a,7,AaaAAAA) ⊢ (aa,5,aaAAAA)
    ⊢ (aaa,5,aAAAA) ⊢ (aaaa,5,AAAA) ⊢ (aaaaA,5,AAA) ⊢ (aaaaAA,5,AA)
    ⊢ (aaaaAAA,5,A) ⊢ (aaaaAAAA,5,#) ⊢ (aaaaAAA,6,AA) ⊢*(aaa,6,aAAAAA)
    ⊢ (aa,7,aaAAAAA) ⊢ (aa,7,aaAAAAA) ⊢*(ε,7,#aaaaAAAAA) ⊢*(ε,8,aaaaAAAAA)
    ⊢*(aaaaAAAAA,8,#) ⊢ (aaaaAAAA,2,A) ⊢*(ε,2,#aaaaaaaaa) ⊢ (ε,9,a^9)

8.3 IZVOĐENJE GRAMATIKA BEZ OGRANIČENJA

Primjeri 8.4, 8.5 i 8.6 svaki na svoj način u dovoljnoj mjeri ilustriraju generiranje nekih jezika uz pomoć Turingovog generatora. Međutim, problem izvođenja gramatika jezika bez ograničenja sigurno je najkompleksniji u teoriji formalnih jezika, dijelu koji se bavi izvođenjem gramatika. Ovdje ćemo pokazati kako riješiti taj problem na primjeru nekoliko jezika bez ograničenja. Veliki doprinos svemu tomu dao je student Željko Brcković u svom završnom radu na studiju Informatike Filozofskog fakulteta u Zagrebu.

Jezik $L_1=\{ww: w\in\{a_1,a_2, ..., a_n\}^*, n>1\}$

Poznat je problem izvođenja gramatike jezika $\{ww: w\in\{a_1,a_2, ..., a_n\}^*, n>1\}$, tj. jezika u kojem rečenice imaju kontekstno svojstvo da su sačinjene od dva jednaka niza *w*, nad alfabetom $\{a_1,a_2, ..., a_n\}$. Općenito rješenje je sljedeće:

1) Pridružimo svakom znaku alfabeta a_i neterminal A_i, $i=1,..., n$.
2) Nizom izvođenja izvesti rečeničnu formu $w\alpha$, $w=w_1...w_k$, $w_k\in\{a_1,a_2, ..., a_n\}$, gdje je β "reverzni" niz neterminala, $\beta=W_k...W_1$, a W_i je neterminal pridružen znaku w_i. To će se postići produkcijom:

   $$Y \rightarrow a_1YA_1 \mid a_2YA_2 \mid ... \mid a_nYA_n \mid \varepsilon$$

3) Nastaviti izvoditi rečenične forme tako da se prvo svaki W_i, od $i=1$ do $i=k$, zamijeniti znakom w_i, potom premjesti ispred W_k:

   $$wW_k...W_1 \Rightarrow wW_k... W_2w_1 \ ^*\!\!\Rightarrow ww_1W_k... W_2$$

   Da bismo to postigli, gramatika mora saržavati produkcije:

   $$A_1Z \rightarrow a_1Z \ ... \ A_nZ \rightarrow a_nZ$$

   koje će osigurati da se uvijek krene od posljednjeg neterminala reverznog niza, te produkcije:

   $$\begin{array}{llll} A_1a_1 \rightarrow a_1A_1 & A_2a_1 \rightarrow a_1A_2 & ... & A_na_1 \rightarrow a_1A_n \\ A_1a_2 \rightarrow a_2A_1 & A_2a_2 \rightarrow a_2A_2 & ... & A_na_2 \rightarrow a_2A_n \\ ... & ... & ... & ... \\ A_1a_n \rightarrow a_nA_1 & A_2a_n \rightarrow a_nA_2 & ... & A_na_n \rightarrow a_nA_n \end{array}$$

   koje će osigurati premještanje tekućeg znaka na početak reverznog niza neterminala.

Dakle, gramatika jezika $\{ww: w\in\{a_1,a_2, ..., a_n\}^*, n>1\}$, definirana je skupom produkcija:

$$G_1: \ S \rightarrow YZ \qquad Y \rightarrow a_1YA_1 \mid a_2YA_2 \mid ... \mid a_nYA_n \mid \varepsilon$$

$$\begin{array}{llll} A_1Z \rightarrow a_1Z & A_1a_1 \rightarrow a_1A_1 & A_2a_1 \rightarrow a_1A_2 & ... \quad A_na_1 \rightarrow a_1A_n \\ A_2Z \rightarrow a_2Z & A_1a_2 \rightarrow a_2A_1 & A_2a_2 \rightarrow a_2A_2 & ... \quad A_na_2 \rightarrow a_2A_n \\ ... & ... & ... & ... \quad ... \\ A_nZ \rightarrow a_nZ & A_1a_n \rightarrow a_nA_1 & A_2a_n \rightarrow a_nA_2 & ... \quad A_na_n \rightarrow a_nA_n \\ Z \rightarrow \varepsilon & & & \end{array}$$

Na primjer, za $n=2$ i $a_1=a$, $a_2=b$, $A_1=A$ i $A_2=B$, gramatika jezika $\{ww: w\in\{a,b\}^*\}$ definirana je produkcijama:

$$\begin{array}{llllll} S \rightarrow YZ & Y \rightarrow aYA \mid bYB \mid \varepsilon & AZ \rightarrow aZ & BZ \rightarrow bZ & Aa \rightarrow aA \\ Ab \rightarrow bA & Ba \rightarrow aB & Bb \rightarrow bB & Z \rightarrow \varepsilon & \end{array}$$

a za $n=3$ i $a_1=a$, $a_2=b$, $a_3=c$, $A_1=A$, $A_2=B$ i $A_3=C$, gramatika jezika $\{ww: w\in\{a,b,c\}^*\}$ definirana je produkcijama:

$$\begin{array}{llllll} S \rightarrow YZ & Y \rightarrow aYA \mid bYB \mid bYC \mid \varepsilon & AZ \rightarrow aZ & BZ \rightarrow bZ & CZ \rightarrow cZ \\ Aa \rightarrow aA & Ab \rightarrow bA & Ac \rightarrow cA & Ba \rightarrow aB & Bb \rightarrow bB & Bc \rightarrow cB \\ Ca \rightarrow aC & Cb \rightarrow bC & Cc \rightarrow cC & Z \rightarrow \varepsilon & & \end{array}$$

Evo nekoliko primjera izvođenja rečenica jezika $L(G_2)$:

$S \Rightarrow YZ \Rightarrow aYAZ \Rightarrow aAZ \Rightarrow aaZ \Rightarrow \underline{aa}$

$S \Rightarrow YZ \Rightarrow aYAZ \Rightarrow abYBAZ \Rightarrow abBAZ \Rightarrow abBaZ \Rightarrow abaBZ \Rightarrow ababZ$
$\Rightarrow \underline{abab}$

$S \Rightarrow YZ \Rightarrow aYAZ \Rightarrow abYBAZ \Rightarrow abbYBBAZ \Rightarrow abbBBAZ \Rightarrow abbBBaZ$
$\Rightarrow abbBaBZ \Rightarrow abbaBBZ \Rightarrow abbaBbZ \Rightarrow abbabBZ \Rightarrow abbabbZ$
$\Rightarrow \underline{abbabb}$

i nekoliko primjera izvođenja rečenica jezika $L(G_3)$:

$S \Rightarrow YZ \Rightarrow aYAZ \Rightarrow acYCAZ \Rightarrow acCAZ \Rightarrow acCaZ \Rightarrow acaCZ \Rightarrow acacZ$
$\Rightarrow \underline{acac}$

$S \Rightarrow YZ \Rightarrow cYCZ \Rightarrow cbYBCZ \Rightarrow cbBCZ \Rightarrow cbBcZ \Rightarrow cbcBZ \Rightarrow cbcbZ$
$\Rightarrow \underline{cbcb}$

$S \Rightarrow YZ \Rightarrow bYBZ \Rightarrow bbYBBZ \Rightarrow bbcYCBBZ \Rightarrow bbcCBBZ \Rightarrow bbcCBbZ$
$\Rightarrow bbcCbBZ \Rightarrow bbcbCBZ \Rightarrow bbcbCbZ \Rightarrow bbcbbCZ \Rightarrow bbcbbcZ$
$\Rightarrow \underline{bbcbbc}$

Jezik $L_2=\{a^k: k=1+2+\ldots+n=(1+n)n/2,\ n\geq 1\}$

Ako napišemo nekoliko prvih rečenica ovoga jezika:

$w_1 = a$
$w_2 = w_1aa = aaa = a^3$
$w_3 = w_2aaa = a^3aaa = a^6$
$w_4 = w_3aaaa = a^6aaaa = a^{10}$
...

zaključit ćemo da je

$$w_n = w_{n-1}a^{|w_{n-1}|-|w_{n-2}|}a$$

odnosno, za $n>2$ razlika duljine niza između uzastopnih članova se povećava za 1:

$$|w_n| = |w_{n-1}|+(|w_{n-1}|-|w_{n-2}|)+1$$

Za svaki član a_{m+2} vrijedi:

$$a_{m+2} = a_{m+1} + (a_{m+1} - a_m) + 1$$

(Valjanost navedene formule možda nije očigledna, te zahtjeva obrazloženje. Poznato je da je zbroj $1+2+\ldots+n$ jednak $n(n+1)/2$. Ako za svaki član niza a_m napišemo da je jednak $m(m+1)/2$ možemo se uvjeriti u njezinu valjanost.)

G_2: (1) $S \rightarrow EDF$ (2,3) $D \rightarrow B|CD$ (4) $Ca \rightarrow aC$ (5) $CB \rightarrow aBC$
(6) $CF \rightarrow BF$ (7) $Ea \rightarrow aE$ (8) $EB \rightarrow aE$ (9) $EF \rightarrow \varepsilon$

Izvođenje mora započeti produkcijom (1). Ukoliko nakon toga slijedi produkcija (2) izvest ćemo prvu rečenicu jezika: a. Ako pak nakon prve produkcije koristimo produkciju (3) doći ćemo do niza EC^kBF gdje je k broj korištenja produkcije (3). Nastavak izvođenja nas dovodi do rečenice a^p gdje je $p=1+2+...+k+k+1$. Ako je $k=1$, dobit ćemo drugu rečenicu jezika: a^3. Ako je $k>1$ i nastavljamo izvođenje samo produkcijama (4) do (6). Redoslijed produkcija u izvođenju sada više ne može utjecati na konačan ishod. Broj korištenja produkcije (3) je već odredio rečenicu koja će se izvesti. Nije nužno da se izvođenje nastavi izvršavanjem produkcija (4) do (6). Taj redoslijed koristimo kako bi se lakše uočio princip gramatike G_2. Isto vrijedi i za obrazloženja koja stoje uz ostale gramatike. Dakle, dolazimo do niza $E\alpha F$ gdje je $|\alpha|=a_m+2$ i α se sastoji od a_{m+1} znakova a i a_m znakova B (ovdje se vidi kako smo formulu primjenili na gramatiku). Produkcije (7) do (9) pretvaraju sve znakove iz α u a i na kraju dobivamo a^p. Primjeri izvođenja u G_2:

[n=1] S ⇒ EDF⇒ EBF⇒ aEF⇒ a
[n=2] S ^{3}⇒ ECBF⇒ EaBCF⇒ EaBBF⇒ aEBBF⇒ aaEBF⇒ aaaEF⇒ a^3
[n=3] S ^{4}⇒ ECCBF ^{2}⇒ ECaBBF ^{4}⇒ EaaBaBBF ^{7}⇒ a^6

Jezik $L_3=\{a^{2^n}: n>0\}$

Dakle, treba definirati gramatiku jezika $\{a^2, a^4, a^8, a^{16}, ...\}$. U [16] ("Automata and Formal Languages", Juhani Karhumäki, Spring 2005) dana je gramatika toga jezika bez objašnjenja kako je izvedena:

S → ATaA
Ta → aaT
TA → QA| Q
aQ → Qa
AQ → AT| ε

Na primjer, rečenice a^2, a^4 i a^8 dobit ćemo izvođenjima:

S ⇒ ATaA ⇒ AaaTA ⇒ AaaQ ⇒ AaQa ⇒ AQaa ⇒ aa
S ⇒ ATaA ⇒ AaaTA ⇒ AaaQA ^{2}⇒ AQaaA ⇒ ATaaA ⇒ AaaTaA
⇒ AaaaaTA ⇒ AaaaaQ ^{4}⇒ AQaaaa ⇒ aaaa
S ⇒ ATaA ⇒ AaaTA ⇒ AaaQA ^{2}⇒ AQaaA ⇒ ATaaA ⇒ AaaTaA
⇒ AaaaaTA ⇒ AaaaaQA ^{4}⇒ AQaaaaA ⇒ ATaaaaA ⇒ AaaTaaaA
⇒ AaaaaTaaA ⇒ AaaaaaaTaA ⇒ AaaaaaaaaTA ⇒ AaaaaaaaaQ
^{8}⇒ AQaaaaaaaa ⇒ aaaaaaaa

Ali, što će se dogoditi ako, na primjer, poslije

S *⇒ AQaa

imamo niz izvođenja:

AQaa ⇒ ATaa ⇒ AaaTa ⇒ AaaaaT

Pogledajmo gramatiku:

S → QXaQ Xa → aaX QX → A Aa → aaA AQ → ε
XQ → YQ| B aY → Yaa aB → Baa QB → ε QY → QX| A

S → aa| QaaTA
aT → Taa
TA → TTA
QT → Q
aA → Aa
QA → ε

S ⇒ aa
S ⇒ QaaTA ⇒ QaTaaA ⇒ QTaaaaA ⇒ QaaaaA ⇒ QaaaAa ⇒ QaaAaa
⇒ QaAaaa ⇒ QAaaaa ⇒ aaaa
S ⇒ QaaTA ⇒ QaaTTA ⇒ QaTaaTA ⇒ QTaaaaTA ⇒ QaaaaTA
⇒ QaaaTaaA ⇒ QaaTaaaaA ⇒ QaTaaaaaaA ⇒ QTaaaaaaaaA
⇒ QaaaaaaaaA $^{8}\Rightarrow$ QAaaaaaaaa ⇒ aaaaaaaa

Možda ćemo dati prednost gramatici $\mathcal{G}_2$ u odnosu na gramatiku $\mathcal{G}_1$. Prije svega, gramatika $\mathcal{G}_2$ je korektna (nema nedokučivih produkcija). Možda bismo tu stali. Ali, može se izvesti ekvivalentnu gramatika koja je sigurno najbolje rješenje u generiranju danog jezika!

Zamislimo niz definiran sa $a_n=a^{2^n}$. Sljedeći je član $a_{n+1}= a^{2^{n+1}} = (a^{2^n})^2$. Dakle, ako imamo neku rečenicu iz zadanog jezika, sljedeću možemo dobiti udvostručavanjem, odnosno tako da svaki znak a u prethodnoj rečenici zamijenimo s aa. Sada nije problem napisati gramatiku koja će to činiti:

G_3: (1,2) S → aa| CSB
(3) Ca → aaC
(4) CB → ε

Produkcijom (1) možemo izravno izvesti prvu rečenicu jezika a^2 (za n=1). Ako izvođenje započnemo produkcijom (2), doći ćemo do niza $C^k aaB^k$ gdje je k broj uporabe produkcije (2). U nastavku izvođenja primjenom produkcije (3) neterminali C "prelaze" slijeva nadesno i udvostručavaju nizove znakova a. Produkcija (4) poništava preostale neterminale i na kraju izvođenja ostaje rečenica $a^{2^{k+1}}$. Primjeri izvođenja u G_3:

[n=1] S ⇒ a^2
[n=2] S ⇒ CSB ⇒ CaaB ⇒ aaCaB ⇒ aaaaCB ⇒ a^4
[n=3] S $^{3}\Rightarrow$ CCaaBB $^{3}\Rightarrow$ CaaaaB $^{5}\Rightarrow$ a^8

Jezik $L_4=\{a^{m^n} : m>0, n>0\}$

Po istom principu možemo izvesti i gramatike jezika $\{a^{3^n}: n>0\}$, $\{a^{4^n}: n>0\}$, … Općenito, ako uzmemo neki prirodni broj m, gramatiku jezika $\{a^{m^n}: m>0, n>0\}$ možemo napisati kao:

S → a^m| BSC

$Ba \rightarrow a^mB$
$BC \rightarrow \varepsilon$

Ako $m=1$ imamo jezik {a} i gramatiku

$S \rightarrow a \mid BSC$
$Ba \rightarrow aB$
$BC \rightarrow \varepsilon$

odnosno, ekvivalentnu gramatiku

$S \rightarrow a$

Jezik $L_5=\{a^{n^2} : n>0\}$

Dakle, treba definirati gramatiku jezika $\{a, a^4, a^9, a^{16}, ...\}$, tj. nizova a^{n^2}, $n>0$. U literaturi (G. E. Révész) može se naći rješenje (kontekstna gramatika):

$S \rightarrow a \mid CD \quad C \rightarrow ACB \mid AB \quad AB \rightarrow aBA \quad Aa \rightarrow aA \quad Ba \rightarrow aB \quad AD \rightarrow Da$
$BD \rightarrow Ea \quad BE \rightarrow Ea \quad E \rightarrow a$

Ideja je da se prvo izvede A^nB^nD, potom $a^{n^2}B^nA^nD$ što predstavlja $a^{(n+1)^2}$. Na primjer:

$S \Rightarrow \underline{a}$

$S \Rightarrow CD \Rightarrow ABD \Rightarrow aBAD \Rightarrow aBDa \Rightarrow aEaa \Rightarrow \underline{aaaa} = a^{(1+1)^2}$

$S \Rightarrow CD \Rightarrow ACBD \Rightarrow AABBAD \Rightarrow AaBABAD \Rightarrow AaBaBAAD \Rightarrow aABaBAAD$
$\Rightarrow aaBaaBAAD \Rightarrow aaaBaBAAD \Rightarrow aaaaBBAAD = a^{2^2}B^2A^2D$
$*\Rightarrow \underline{aaaaaaaaa} = a^{(2+1)^2}$

Evo našeg rješenja. Analizirajući rečenice jezika (v. primjer 8.7), uočit ćemo da se za zadani n, $n>0$, mogu napisati kao:

$a^{(n-1)^2}a^{2n-1}$

iz čega slijedi da se rečenice $x_1, x_2, ..., x_n$ danog jezika mogu napisati i kao:

$x_1 = a$
$x_2 = x_1a^3$
$x_3 = x_2a^5 = x_1a^3a^5$
...
$x_n = x_{n-1}a^{2n-1} = x_1a^3a^5...a^{2n-1}$

Dakle, prvo treba izvesti rečeničnu formu

$aTaaTaa...Taa = a(Taa)^{n-1}$

potom iza svakog T, krenuvši slijeva udesno, „prebaciti“ znakove a koji mu prethode. Ako je Q oznaka početka, a C kraja rečenične forme, možemo napisati prve produkcije gramatike:

$$S \rightarrow QaX$$
$$X \rightarrow TaaX \mid C$$

što će generirati rečenične forme:

$$[n=1]\ S \Rightarrow QaX \Rightarrow QaC$$
$$[n=2]\ S \Rightarrow QaX \Rightarrow QaTaaX \Rightarrow QaTaaC$$
$$\ldots$$
$$[n>2]\ S \Rightarrow^{n+1} Qa(Taa)^{n-1}TaaC$$

Sada preostaje da definiramo produkcije koje će osigurati prebacivanje znakova a:

$$aT \rightarrow ATa$$
$$QA \rightarrow aQ$$
$$QT \rightarrow Q$$

i produkcije koje na kraju brišu neterminale Q i C:

$$aC \rightarrow Ca$$
$$QC \rightarrow \varepsilon$$

Dakle, imamo gramatiku G_5:

$$G_5:\ S \rightarrow QaX \quad X \rightarrow TaaX \mid C \quad aT \rightarrow ATa \quad QA \rightarrow aQ \quad QT \rightarrow Q \quad aC \rightarrow Ca \quad QC \rightarrow \varepsilon$$

Evo nekoliko primjera izvođenja rečenica:

$$[n=1]\ S \Rightarrow QaX \Rightarrow QaC \Rightarrow aQC \Rightarrow \underline{a}$$
$$[n=2]\ S \Rightarrow QaX \Rightarrow QaTaaX \Rightarrow QaTaaC \Rightarrow QATaaaC \Rightarrow aQTaaaC \Rightarrow aQaaaC \Rightarrow aQaaCa$$
$$\Rightarrow aQaCaa \Rightarrow aQCaaa \Rightarrow aaaa = \underline{a^4}$$
$$[n=3]\ S \Rightarrow QaX \Rightarrow QaTaaX \Rightarrow QaTaaTaaX \Rightarrow QaTaaTaaC \Rightarrow QATaaaTaaC$$
$$*\Rightarrow aQA^3Ta^5C *\Rightarrow a^4QTa^5C \Rightarrow a^4Qa^5C \Rightarrow^* a^4QCa^5 \Rightarrow a^4a^5 \Rightarrow \underline{a^9}$$

Ako rečenice jezika L_5 napišemo kao $\{(a^n)^n : n>0\}$, tj rečenica $(a^n)^n$ sadrži n nizova a^n, dolazimo do još jedne ekvivalentne gramatike, označimo je s G_5'. Ideja je da za dani n prvo generiramo niz

$$QB^nA^nC$$

što ćemo dobiti produkcijama

$$S \rightarrow QXC \quad X \rightarrow BXA \mid BA$$

potom ćemo „pomnožiti“ n-1 puta B s A, što ćemo postići produkcijama $BA \rightarrow aAB$ i $Ba \rightarrow aB$:

$$QB^nA^nC *\Rightarrow Q(a^nA)^nB^nC$$

Još preostaje da se produkcijama $Qa \rightarrow aQ$ i $QA \rightarrow Q$ n puta zamijeni A u ε:

$$Q(a^nA)^nB^nC \;*\Rightarrow\; (a^n)^nQB^nC$$

i, na kraju, produkcijama $BC \rightarrow C$ i $QC \rightarrow \varepsilon$ izbriše sufiks QB^nC rečenične forme, čime se dobila rečenica danoga jezika. Evo kompletne gramatike

$$G_5^1: \; S \rightarrow QXC \quad X \rightarrow BXA \mid BA \quad BA \rightarrow aAB \quad Ba \rightarrow aB$$
$$BC \rightarrow C \quad Qa \rightarrow aQ \quad QA \rightarrow Q \quad QC \rightarrow \varepsilon$$

i nekoliko primjera izvođenja rečenica:

$$[n=1]\;\; S \Rightarrow QXC \Rightarrow QBAC \Rightarrow QaABC \Rightarrow QaAC \Rightarrow aQAC \Rightarrow aQC \Rightarrow \underline{a}$$
$$[n=2]\;\; S \Rightarrow QXC \Rightarrow QBXAC \Rightarrow QBBAAC \;*\Rightarrow\; QaaAaaAC \;*\Rightarrow\; \underline{a^4}$$
$$[n=3]\;\; S \Rightarrow QXC \Rightarrow QBXAC \Rightarrow QBBXAAC \Rightarrow QBBBAAAAC \;*\Rightarrow\; QaaaAaaaAaaaAC \;*\Rightarrow\; \underline{a^9}$$

I to nije sve! Pogledajmo niz definiran s $a_n=n^2$. Znamo da se razlika između uzastopnih članova niza povećava za 2, odnosno može se reći da za svaki član a_{n+2} vrijedi

$$a_{n+2} = a_{n+1} + (a_{n+1} - a_n) + 2$$

Očito je da će se rješenje temeljiti na istom principu kao i u primjeru izvođenja gramatike jezika L_2:

$$G_5^2: \; (1)\; S \rightarrow EDF \quad (2,3)\; D \rightarrow B \mid CD \quad (4)\; Ca \rightarrow aC \quad (5)\; CB \rightarrow aBC$$
$$(6)\; CF \rightarrow BBF \quad (7)\; Ea \rightarrow aE \quad (8)\; EB \rightarrow aE \quad (9)\; EF \rightarrow \varepsilon$$

Jedina razlika između G_5^2 i G_2 jest neterminal B u produkciji (6). Pogledajmo nekoliko primjera izvođenja u G_5^2:

$$[n=1]\; S \Rightarrow EDF \Rightarrow EBF \Rightarrow aEF \Rightarrow \underline{a}$$
$$[n=2]\; S \overset{3}{\Rightarrow} ECBF \Rightarrow EaBCF \Rightarrow EaBBBF \overset{5}{\Rightarrow} \underline{a^4}$$
$$[n=3]\; S \overset{4}{\Rightarrow} ECCBF \overset{2}{\Rightarrow} ECaBBBF \overset{5}{\Rightarrow} EaaBaBaBBBF \overset{10}{\Rightarrow} \underline{a^9}$$

Jezik $L_6=\{a^{n^p}: n>0, p>2\}$

Možemo se pitati: Postoje li i gramatike jezika $\{a^{n^3}: n>0\}$, $\{a^{n^4}: n>0\}$, ...? Odgovor je potvrdan. Pokazat ćemo postupak kojim za neki prirodni broj $p>2$ možemo dobiti gramatiku jezika $\{a^{n^p}: n>0\}$:

$$(1)\; S \rightarrow EG^{p-1}DBF \qquad (9)\; CF \rightarrow F$$
$$(2,3)\; D \rightarrow C_1 \mid DC_1 \qquad (10)\; GB \rightarrow G$$
$$(4)\; GC_1 \rightarrow C_1GC \qquad (11)\; Ga \rightarrow BG$$
$$(5)\; CC_1 \rightarrow C_1C \qquad (12)\; GF \rightarrow F$$
$$(6)\; EC_1 \rightarrow EC \qquad (13)\; EB \rightarrow E$$

(7) $CB \rightarrow aBC$ (14) $Ea \rightarrow aE$
(8) $Ca \rightarrow aC$ (15) $EF \rightarrow \varepsilon$

Produkcije (1) do (3) će generirati nizove $EG^{p-1}C_1{}^kBF$. Ako uključimo i produkcije (4) do (6) dobivamo nizove $E(C^kG)^{p-1}C_1{}^kBF$. Nastavak izvođenja dovodi nas do a^{k^p}. Demonstrirat ćemo to na primjeru gramatike jezika $\{a^{n^3} : n>0\}$:

(1) $S \rightarrow EGGDBF$ (9) $CF \rightarrow F$
(2,3) $D \rightarrow C_1 | DC_1$ (10) $GB \rightarrow G$
(4) $GC_1 \rightarrow C_1GC$ (11) $Ga \rightarrow BG$
(5) $CC_1 \rightarrow C_1C$ (12) $GF \rightarrow F$
(6) $EC_1 \rightarrow EC$ (13) $EB \rightarrow E$
(7) $CB \rightarrow aBC$ (14) $Ea \rightarrow aE$
(8) $Ca \rightarrow aC$ (15) $EF \rightarrow \varepsilon$

$[n=1]\ S \Rightarrow EGGDBF \Rightarrow EGGC_1BF \Rightarrow EGC_1GCBF \Rightarrow EC_1GCGCBF \Rightarrow ECGCGCBF$
$\Rightarrow ECGCGaBCF \Rightarrow ECGCGaBF \Rightarrow ECGCBGBF \Rightarrow ECGCBGF \Rightarrow ECGCBF$
$\Rightarrow ECGaBCF \Rightarrow ECGaBF \Rightarrow ECBGBF \Rightarrow ECBGF \Rightarrow ECBF \Rightarrow EaBCF$
$\Rightarrow EaBF \Rightarrow aEBF \Rightarrow aEF \Rightarrow \underline{a}$
$[n=2]\ S \overset{3}{\Rightarrow} EGGC_1C_1BF \overset{9}{\Rightarrow} ECCGCCGCCBF \overset{5}{\Rightarrow} ECCGCCGaaBF \overset{4}{\Rightarrow} ECCGCCBBF$
$\overset{8}{\Rightarrow} ECCGaaBaaBF \overset{7}{\Rightarrow} ECCBBBBF \overset{14}{\Rightarrow} EaaBaaBaaBaaBF \overset{13}{\Rightarrow} \underline{a^8}$
$[n=3]\ S \overset{4}{\Rightarrow} EGGC_1C_1C_1BF \overset{18}{\Rightarrow} ECCCGCCCGCCCBF \overset{9}{\Rightarrow} ECCCGCCCGaaaBF$
$\overset{5}{\Rightarrow} ECCCGCCCBBBF \overset{21}{\Rightarrow} ECCCGaaaBaaaBaaaBF \overset{13}{\Rightarrow} CCCBBBBBBBBBF$
$\overset{57}{\Rightarrow} E(aaaB)^9F \overset{37}{\Rightarrow} \underline{a^{27}}$

Ako rečenice jezika L_5 napišemo kao $\{(a^n)^n : n>0\}$, tj rečenica $(a^n)^n$ sadrži n nizova a^n, dolazimo do još jedne ekvivalentne gramatike, označimo je s G_5'. Ideja je da za dani n prvo generiramo niz

$$QB^{n-1}(YB^{n-1}A^n)C$$

što ćemo dobiti produkcijama

$$S \rightarrow QXC \qquad X \rightarrow DBXA | DA$$

potom ćemo „pomnožiti“ $n-1$ puta B s A, što ćemo postići produkcijama $BA \rightarrow aAB$ i $Ba \rightarrow aB$:

$$QB^{n-1}(YB^{n-1}A^n)C \Rightarrow^* Q(a^{n-1}A)^nB^{n-1}C$$

Još preostaje da se produkcijama $Qa \rightarrow aQ$ i $QA \rightarrow aQ$ n puta zamijeni A u a:

$$Q(a^{n-1}A)^{n-1}B^{n-1}C \Rightarrow^* (a^n)^nQB^{n-1}C$$

i, na kraju, produkcijama $BC \rightarrow C$ i $QC \rightarrow \varepsilon$ izbriše sufiks $QB^{n-1}C$ rečenične forme, čime se dobila rečenica danoga jezika. Evo kompletne gramatike

G_5': $S \rightarrow QXC$
$X \rightarrow DBXA | DA$

BA → aAB
Ba → aB
BC → C
BD → DB
Qa → aQ
QA → aQ
QD → QY
YD → BY
Ya → AY
YA → AY
YC → C
QC → ε

i nekoliko primjera izvođenja rečenica:

[n=1] S ⇒ QXC ⇒ QDAC ⇒ QYAC ⇒ QAYC ⇒ QAC ⇒ aQC ⇒ a
[n=2] S ⇒ QXC ⇒ QDBDAAC ⇒ QYBDAAC ⇒ QYDBAAC ⇒ QBYBAAC *⇒ QBYaAaAC
*⇒ QBAAAAC *⇒ QaAaAaAaAC *⇒ $\underline{a^8}$
[n=3] S ⇒ QDBXAC ⇒ QDBDBXAAC ⇒ QDBDBDAAAC ⇒ QDBDBDAAAC *⇒ QDDDBBAAAC
⇒ QYDDBBAAAC *⇒ QBBYBBAAAC *⇒ QBBYaaAaaAaaAC *⇒ QBBAAAAAAAAAC
⇒ Q(aaA)^{9}C ⇒ $\underline{a^{27}}$

Jezik prirodnih brojeva djeljivih sa zadanim brojem

Ako odaberemo neki prirodni broj *m*, možemo zamisliti jezik čije su rečenice zapisi brojeva djeljivih s *m*. Postavlja se pitanje: Kako napisati gramatiku takvog jezika? Odgovor na to pitanje dali smo na nekoliko mjesta u ovoj knjizi. Znamo da je taj jezik regularan pa smo pokazali kako se može izvesti njegova gramatika iz konačnog generatora. Pogledajmo sada sljedeću gramatiku:

$Y \rightarrow X^m | YX^m$

Rezultati izvođenja su rečenice $X^{k\times m}$. Ako zamislimo unarni brojevni sustav gdje je znamenka znak X, imamo gramatiku zapisa brojeva djeljivih s *m*. Sada se možemo zapitati: Kako to proširiti na brojevne sustave s višom bazom? Evo rješenja za binarni brojevni sustav:

(1) S → GYH (2,3) Y → X^m| YX^m (4) GX → G1 (5) 0X → 1
(6) 1X → X0 (7) GH → ε (8) 0H → H0 (9) 1H → H1

Produkcije od (1) do (3) generiraju niz $GX^{k\times m}H$. Produkcijama od (4) do (6) se niz $X^{k\times m}$ pretvara u zapis broja $k\times m$ u binarnom brojevnom sustavu. Produkcije od (7) do (9) eliminiraju preostale neterminale G i H. Slijedi primjer gramatike brojeva djeljivih s tri:

(1) S → GYH (2,3) Y → XXX| YXXX (4) GX → G1 (5) 0X → 1
(6) 1X → X0 (7) GH → ε (8) 0H → H0 (9) 1H → H1

S ⇒ GYH⇒ GXXXH⇒ G1XXH⇒ GX0XH⇒ G10XH⇒ G11H⇒ G1H1⇒ GH11
⇒ $\underline{11}$
S ^{3}⇒ GXXXXXXH ^{10}⇒ G110H ^{4}⇒ $\underline{110}$
S ^{4}⇒ GXXXXXXXXXH ^{16}⇒ G1001H ^{5}⇒ $\underline{1001}$

Što ako želimo rješenja u nekim višim brojevnim sustavima (na primjer, decimalnom)? Općenito rješenje je da na odgovarajući način preoblikujemo zadnju produkciju u lijevom stupcu i dodamo nekoliko novih produkcija. Na primjer, za ternarni brojevni sustav imamo gramatiku:

```
S  → GYH          GH → ε
Y  → X‴|YX‴       0H → H0
GX → G1           1H → H1
0X → 1            2H → H2
1X → 2
2X → X0
```

Evo i rješenja za decimalni i heksadecimalni brojevni sustav:

Decimalni brojevni sustav:

```
S  → GYH          GH → ε
Y  → X‴|YX‴       0H → H0
GX → G1           1H → H1
0X → 1            2H → H2
1X → 2            3H → H3
2X → 3            4H → H4
3X → 4            5H → H5
4X → 5            6H → H6
5X → 6            7H → H7
6X → 7            8H → H8
7X → 8            9H → H9
8X → 9
9X → X0
```

Heksadecimalni brojevni sustav:

```
S  → GYH          GH → ε
Y  → X‴|YX‴       0H → H0
FX → G1           1H → H1
0X → 1            2H → H2
1X → 2            3H → H3
2X → 3            4H → H4
3X → 4            5H → H5
4X → 5            6H → H6
5X → 6            7H → H7
6X → 7            8H → H8
7X → 8            9H → H9
8X → 9            AH → HA
9X → A            BH → HB
AX → B            CH → HC
BX → C            DH → HD
CX → D            EH → HE
DX → E            FH → HF
EX → F
FX → X0
```

Svi jezici L_2 do L_6 bili su definirani nad istim alfabetom, {a}. Dakle, njihove rečenice su bile nizovi znakova a. Na rečenice tih jezika možemo gledati i kao na zapise brojeva (koji zadovoljavaju određeno svojstvo) unarnim brojevnim sustavom gdje je znamenka znak a. Njihove gramatike također možemo preoblikovati tako da dobijemo gramatike koje generiraju zapise istih brojeve u primjerice, decimalnom brojevnom sustavu. Krenimo od gramatike G_1. Prvo uzimamo sljedeći niz produkcija:

7.1

```
S  → GYH          4X → 5
GX → G1           5X → 6
0X → 1            6X → 7
1X → 2            7X → 8
2X → 3            8X → 9
3X → 4            9X → X0
```

Sada startni simbol S zamjenjujemo s Y. Zatim znak a zamjenjujemo znakom X. Ono što smo dobili pridružujemo produkcijama u 7.1. Rezultat je gramatika jezika koji se sastoji od zapisa brojeva u decimalnom sustavu i ti brojevi pripadaju skupu $\{2^n : n \geq 1\}$

$S \rightarrow GYH$	$4X \rightarrow 5$	$Y \rightarrow XX \mid CSB$
$GX \rightarrow G1$	$5X \rightarrow 6$	$CX \rightarrow XXC$
$0X \rightarrow 1$	$6X \rightarrow 7$	$CB \rightarrow \varepsilon$
$1X \rightarrow 2$	$7X \rightarrow 8$	
$2X \rightarrow 3$	$8X \rightarrow 9$	
$3X \rightarrow 4$	$9X \rightarrow X0$	

Isto možemo učiniti i s ostalim gramatikama. Jedino, valja pripaziti s onim gramatikama koje sadrže neterminale koji već jesu u 7.1. U tom slučaju moramo ih zamijeniti s nekim neterminalima koji se ne pojavljuju u 7.1. Na primjer $\mathcal{G}_6$ sadrži neterminal G. Zamijenit ćemo ga s G_1. Ako nastavimo već opisani postupak, dobivamo gramatiku koja generira zapise u decimalnom sustavu za brojeve iz skupa $\{n! : n>1\}$.

$S \rightarrow GYH$	$4X \rightarrow 5$	$Y \rightarrow EDBF$	$CF \rightarrow F$
$GX \rightarrow G1$	$5X \rightarrow 6$	$D \rightarrow C_1 \mid C_1G_1D$	$G_1B \rightarrow G_1$
$0X \rightarrow 1$	$6X \rightarrow 7$	$GC_1 \rightarrow C_1G_1C$	$G_1X \rightarrow BG_1$
$1X \rightarrow 2$	$7X \rightarrow 8$	$CC_1 \rightarrow C_1C$	$G_1F \rightarrow F$
$2X \rightarrow 3$	$8X \rightarrow 9$	$EC_1 \rightarrow EC$	$EB \rightarrow E$
$3X \rightarrow 4$	$9X \rightarrow X0$	$CB \rightarrow XBC$	$EX \rightarrow XE$
		$CX \rightarrow XC$	$EF \rightarrow \varepsilon$

9.
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 han)

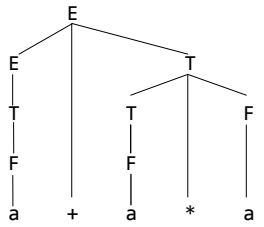
U ovom je završnom poglavlju opisana posebna klasa konteksnih jezika nazvana "jezici sa svojstvima" ili "atributni jezici". Prvo smo definirali atributne gramatike dobivene proširenjem beskontekstnih gramatika uvođenjem atributa, potom generator jezika sa svojstvima koji je utemeljen na dvije tablice: tablicu prijelaza i tablicu akcija.

9.1 SINTAKSA I SEMANTIKA

Formalni jezik smo definirali kao podskup bilo kojega skupa nizova znakova ili rečenica nad nekim alfabetom. U svim poglavljima ove knjige dali smo mnoge primjere netrivijalnih jezika, jezika u kojima su rečenice imale određenu strukturu ili pravila pisanja. Takvi su bili regularni skupovi koje smo označivali (generirali) regularnim izrazima. Ako smo koristili gramatiku za definiciju jezika ta su pravila imala značenje sintakse jezika. Dakle, rečenicama jezika nismo pridruživali nikakvo značenje već samo pravila, oblik ili formu (otud i naziv "formalni" jezik), što se zorno moglo prikazati stablom izvođenja. Na primjer, rečenicu `a+a*a` jezika definiranog gramatikom

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

možemo dobiti izvođenjem prikazanim stablom izvođenja



Znakovi alfabeta {a,+,*,(,)} ovoga jezika nemaju nikakvo značenje. Jedino što mi možda "znamo" da je + zbrajanje, * množenje, a (i) su zagrade u jeziku koji bi mogao predstavljati osnovnu strukturu pojednostavljenog aritmetičkog izraza.

Ponekad smo uvodili značenje u rečenice jezika, kao na primjer u jeziku prirodnih brojeva djeljivih s nekim zadanim brojem, da bismo lakše došli do rješenja problema izvođenja gramatika. To smo posebno činili u izvođenju svih kontekstnih gramatika i gramatika bez ograničenja.

Od početka se teorija formalnih jezika počela primjenjivati u opisu osnovne sintaksne strukture jezika za programiranje (BNF i ALGOL 60, 1963. godine). Međutim, rečenice (programi) jezika za programiranje imaju određena kontekstna svojstva koja se ne mogu prikazati gramatikom. Na primjer, niz

```
X := A + B
```

napisan kao dio programa u Pascalu zadovoljava opću sintaksu naredbe za dodjeljivanje Pascala:

```
<ime varijable> := <izraz>
```

Ta je naredba napisana korektno ali pod uvjetom da je X ime varijable cjelobrojnog, realnog, nizovnog ili skupovnog tipa, a A i B imena varijabli ili konstanti istog tipa. Ako je X realnog tipa, A i B mogu biti cjelobrojnog ili realnog tipa. Ako je X cjelobrojnog tipa, A i B moraju također biti cjelobrojnog tipa. Dakle, imena imaju neko značenje ili semantiku, koja može biti skup pridruženih svojstava kao što su, na primjer, tip i vrijednost. I naredba za dodjeljivanje ima semantiku: izračuna se izraz i vrijednost pridruži imenu varijable. Semantika izraza jest vrijednost koja se dobije njegovim izračunavanjem. Ako je izraz cjelobrojan ili realan, rezultat izračunavanja u našem primjeru bit će zbroj vrijednosti pridruženih imenima A i B; ako je nizovni, niz sadržan u A kojem je dopisan niz sadržan u B, a ako je skupovni, unija skupova A i B. Vidimo da znak + u ovom primjeru može imati pridruženo jedno od tri moguća svojstva: operacija zbrajanja u brojčanom izrazu, nastavljanje nizova ili unija skupova.

9.2 ATRIBUTNE GRAMATIKE

Probajmo napisati gramatiku jezika $L_{abc}=\{a^n b^n c^n : n>0\}$. Nizovi abc, aabbcc i aaabbbccc jesu rečenice tog jezika, dok aaabbbbcc i aabbbcc nisu. Pokušajmo opisati taj jezik beskontekstnom gramatikom s produkcijama:

$S \rightarrow ABC \quad A \rightarrow a|\ aA \quad B \rightarrow b|\ bB \quad C \rightarrow c|\ cC$

Ova gramatika generira sve rečenice danog jezika, na primjer rečenicu aaabbbccc:

$S \Rightarrow ABC \Rightarrow aABC \Rightarrow aaABC \Rightarrow aaaBC \Rightarrow aaabBC \Rightarrow aaabbBC$

Ali, generira i nizove koji nisu rečenice danog jezika. Na primjer, aabc ili abbc, itd. Znamo da, s obzirom da je dani jezik kontekstan, ne postoji beskontekstna gramatika koja bi mogla generirati takav jezik. Kao što ćemo pokazati u sljedećem potpoglavlju, postoji kontekstna gramatika koja generira rečenice danog jezika. Atributne gramatike, koje ćemo ovdje opisati, predstavljaju drugi način za definiranje kontekstne ovisnosti rečenica jezika.

♦ Atributna gramatika

Atributna gramatika $\mathcal{AG}=(\mathcal{G}, \mathcal{At}, \mathcal{R})$ jest beskontekstna gramatika $\mathcal{G}=(\mathcal{N}, \mathcal{T}, \mathcal{P}, \mathcal{S})$, proširena konačnim skupom atributa (ili svojstava), $\mathcal{At}$, i konačnim skupom semantičkih pravila, $\mathcal{R}$. Ako je $\mathcal{V}=\mathcal{N}\cup\mathcal{T}$ skup simbola gramatike, produkcije u $\mathcal{P}$ parovi odnosno $X\rightarrow\alpha, X\in\mathcal{N}, \alpha\in\mathcal{V}^*$, svakom simbolu gramatike pridružen je skup atributa, $\mathcal{At}(X)$, koji je

podijeljen na dva disjunktna skupa: $inh(X)$ (*inherited* – naslijeđen), za $X \in \mathcal{N}, X \neq S$, i $syn(X)$ (*synthesized* – sintetizirani). Vidimo da startni simbol i terminalni simboli nemaju definirane atribute nasljeđivanja. Sada se skup atributa $\mathcal{A}t$ može napisati kao $\mathcal{A}t = \cup \mathcal{A}t(X)$. Produkcije $p \in \mathcal{P}$ atributne gramatike pišu se u obliku

$$p: \; X_0 \rightarrow X_1 X_2 \ldots X_{n_p} \quad X_0 \in \mathcal{N}, X_i \in \mathcal{V}, n_p \geq 1, 1 \leq i \leq n_p$$

Simbol X_i ima pridružen *atribut* $X_i.a$ ako je $a \in \mathcal{A}t(X_i)$. Konačni skup sematičkih pravila $\mathcal{R}_p$ pridružen je produkciji p, po jedno pravilo za svaki sintetizirani atribut $X_0.a$ i po jedno pravilo za svaki atribut nalježivanja $X_i.a$. Dakle, $\mathcal{R}_p$ je kolekcija pravila oblika

$$X_i.a = f(y_1, \ldots, y_k), \; k \geq 0$$

gdje je

(1) $a \in syn(X_i)$, za $i=0$, ili $a \in inh(X_i)$, za $1 \leq i \leq n$,
(2) svaki y_j, $1 \leq j \leq k$, je atribut u p i
(3) f je funkcija, nazvana semantička funkcija, koja preslikava vrijednosti $y_1, \ldots, y_k$ u vrijednost iz $X_i.a$. U pravilu $X_i.a = f(y_1, \ldots, y_k)$ pojavljivanje $X_i.a$ ovisi o svakom pojavljivanju y_j, $1 \leq j \leq k$.

Iz dane definicije atributne gramatike slijedi da su sintetizirani atributi izlaz simbola (neterminala) na lijevim stranama produkcija, a atributi nasljeđivanja su izlazi simbola na desnim stranama (alternativama) produkcija. Pretpostavka je da su sintetizirani atributi terminala definirani izvana.

Ako proširimo našu gramatiku s atributom (svojstvom) koji opisuje duljinu prefiksnog niza, tj. broj znakova **a** niza, možemo ga iskoristiti da bismo generirali isti broj znakova b, potom znakova c.

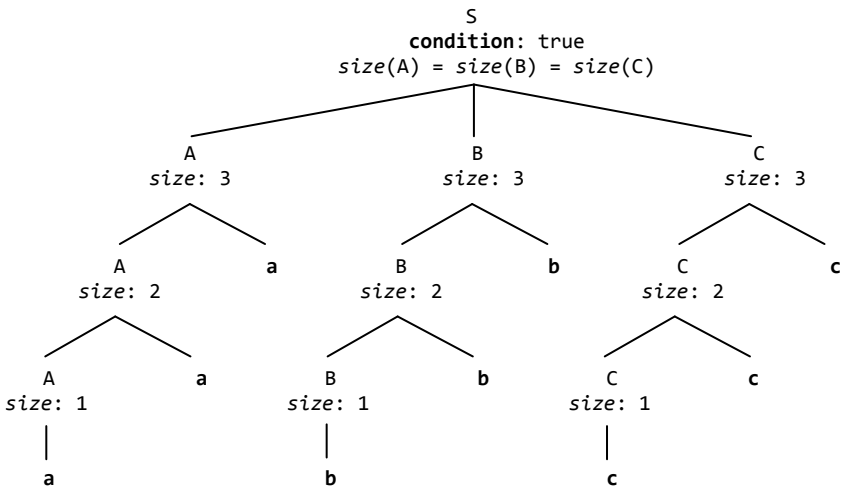
Prvo rješenje sadrži sintetizirani atribut *size* pridružen neterminalima A, B i C. Dodan je i uvjet (**condition**) u korijenu stabla izvođenja koji je ispunjen (jednak True) ako *size* od svih neterminala ima istu vrijednost. Ako se podniz znakova a sastoji od samo jednog znaka a, atribut *size* jednak je 1; ako sadrži niz znakova a iza kojeg slijedi jedan znak a, *size* pridružen prethodnom čvoru za jedan je veći od *size* pridruženog tekućem čvoru. Pridružujući atribute i uvjete prethodnoj gramatici rezultirajuća atributna gramatika može se napisati kao

S → ABC [**condition**: *size*(A) = *size*(B) = *size*(C)]

A → a [*size*(A) ← 1] | Aa [*size*(A) ← *size*(A) +1]
B → b [*size*(B) ← 1] | Bb [*size*(B) ← *size*(B) +1]
C → c [*size*(C) ← 1] | Cc [*size*(C) ← *size*(C) +1]

♣ Primjer 9.1

Pogledajmo primjer stabla izvođenja u slučaju generiranja niza (rečenice) **aaabbbccc**. Generirani niz je u jeziku jer su zadovoljeni svi uvjeti dane atributne gramatike.



Dakle, dana atributna gramatika generira nizove (rečenice) jezik L_{abc}. Lako se može pokazati da generirani nizovi, na primjer **aaabbcc**, **abbc** ili **aaabbbbccc**, ne zadovoljavaju postavljeni uvjet pa, prema tome, ne mogu biti rečenice L_{abc} jezika.

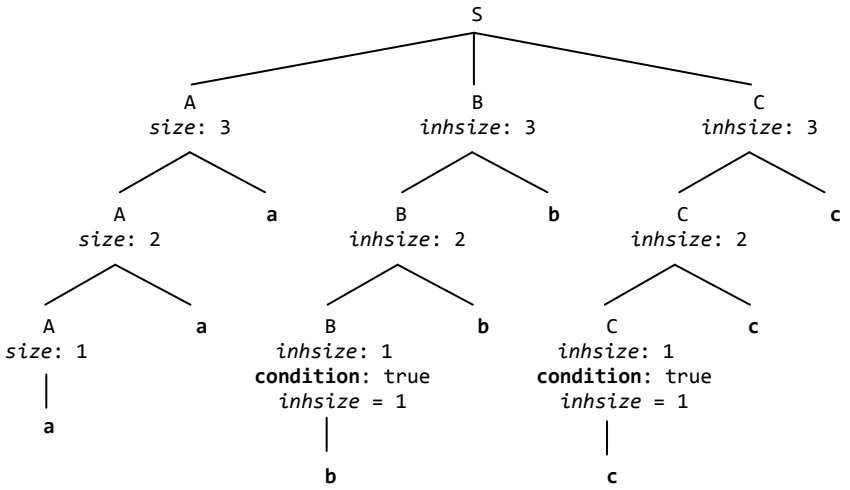
Nedostatak je sintetiziranih atributa, kao u prethodnom primjeru, što sve relevantne informacije moraju biti prenesene u korijen stabla izvođenja gdje će biti provjeren konačni rezultat (uvjet). Bilo bi bolje kad se ne bi trebalo vraćati na korijen već prenijeti informacije s jednog dijela stabla na posebno specificirani čvor otkud mogu biti naslijeđene i prenesene dalje, na drugi dio stabla.

Pokažimo kako bi to izgledalo primijenjeno na jezik L_{abc}. U našem ćemo rješenju rabiti atribut *size* kao sintetizirani atribut za niz znakova `a`. Za nizove znakova `b` i `c` imamo atribut *inhsize* („inherited size" - naslijeđena duljina). Kao što smo već vidjeli, moguće je sintetizirati duljinu niza znakova `a` u korijen stabla. U ovom ćemo rješenju postaviti *inhsize* atribut za nizove `b`-ova i `c`-ova na tu vrijednost i proslijediti ih niz stablo umanjujući vrijednost za jedan u svakom aktivnom čvoru. Kad aktivni čvor postane terminal (`b`, odnosno `c`), provjeravamo je li vrijednost *inhsize* atributa jednaka `1`. Ako jest, duljina niza `b`-ova i `c`-ova mora biti jednaka duljini niza znakova `a`. Evo sada nove atributne gramatike jezika L_{abc}:

```
S → ABC [inhsize (B) ← size(A);  inhsize (C) ← size(A)]
A → a [size(A) ← 1] | Aa [size(A) ← size(A) +1]
B → b [contition: inhsize(B) = 1] | Bb [inhsize(B) ← inhsize(B) -1]
C → c [contition: inhsize(C) = 1] | Cc [inhsize(C) ← inhsize(C) -1]
```

♣ Primjer 9.2

Pogledajmo primjer stabla izvođenja atributne gramatike koja koristi atribut nasljeđivanja, a u generiranja niza (rečenice) **aaabbbccc**:



Generirani niz je u jeziku jer su zadovoljeni svi uvjeti dane atributne gramatike.

9.2 GENERATOR JEZIKA SA SVOJSTVIMA

U prethodnim smo potpoglavljima definirali atributnu gramatiku jezika

$$L_{abc} = \{a^n b^n c^n : n>0\}$$

koja je uvela dodatna (kontekstna) svojstva koji nisu dio produkcija. Za isti smo jezik u sedmom poglavlju definirali indeksiranu i konteksnu gramatiku. Ovdje ćemo definirati jezike sa svojstvima koji obuhvaćaju sve jezike. Također ćemo definirati generator tih jezika.

Generator u kojem je kontrola konačnog stanja u potpunosti zadana funkcijom prijelaza generira regularne jezike. Drugim riječima, jezik je regularan ako sadrži rečenice:

$$w=a_1...a_n \qquad w \in \Sigma^+$$

za koje vrijedi:

$$q_1=\delta(q_0,a_1) \qquad q_2=\delta(q_1,a_2) \quad ... \quad q_k=\delta(q_j,a_n)$$

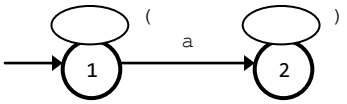
odnosno

$$q_k =\delta(\delta(...,\delta(q_0,a_1),a_2),...,a_n)$$

gdje su q_1, $i=1, ..., k$, stanja iz Q, q_0 jest početno i q_k jedno od završnih stanja, $q_k \in \mathcal{F}$.

♣ **Primjer 9.3**

Generator zadan dijagramom prijelaza:



gdje je $q_0=1$, $\mathcal{F}=\{2\}$, 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:

♦ Jezik sa svojstvima

Jezik sa svojstvima jest jezik čiji je generator zadan kao uređena osmorka:

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

gdje su:

- Q konačan *skup stanja* (kontrole završnog stanja)
- Σ *alfabet*
- $\mathcal{V}$ *rječnik*, $\mathcal{V} \subseteq \Sigma^+$
- δ *funkcija prijelaza*, definirana kao δ:

  $$Q \times \mathcal{V} \rightarrow P(Q)$$

  gdje je $P(Q)$ particija od Q
- q_0 *početno stanje*, $q_0 \in Q$
- $\mathcal{F}$ *skup završnih stanja*, $\mathcal{F} \subseteq Q$
- α *skup akcija* pridruženih svakom paru (q_i, s_j), $q_i \in Q$, $s_j \in \mathcal{V}$, za koji je definirana funkcija prijelaza
- $\mathcal{M}$ *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 9.4

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 gramatiku $\mathcal{G}_2$:

$A \rightarrow aB \mid (C$
$B \rightarrow *A \mid +A \mid \varepsilon$
$C \rightarrow A)$

Ova je gramatika linearna zdesna (prve dvije produkcije) i slijeva (treća produkcija). Zbog linearnosti slijeva nije moguće definirati ekvivalentni konačni generator koji bi imao konačan broj stanja. Ako je zadan maksimalan broj ugnježđenja podizraza (podizraza u zagradama napisanih unutar izraza sa zagradama), konstrukcija konačnog generatora je moguća. Na primjer, ako imamo zagrade samo na osnovnoj razini, tada bi gramatika linearna zdesna imala skup produkcija:

$A \rightarrow aB \mid (C$
$B \rightarrow *A \mid +A \mid \varepsilon$
$C \rightarrow aD$
$D \rightarrow)B \mid +C \mid *C$

Evo nekoliko rečenica generiranih ovom gramatikom:

$A \Rightarrow aB \Rightarrow a$

$A \Rightarrow (C \Rightarrow (aD \Rightarrow (a)B \Rightarrow (a)$
$A \Rightarrow (C \Rightarrow (aD \Rightarrow (a+C \Rightarrow (a+aD \Rightarrow (a+a)B \Rightarrow (a+a)$
$A \Rightarrow (C \Rightarrow (aD \Rightarrow (a+C \Rightarrow (a+aD \Rightarrow (a+a)B \Rightarrow (a+a)*A \Rightarrow (a+a)*(C$
$\Rightarrow (a+a)*(aD \Rightarrow (a+a)*(a+C \Rightarrow (a+a)*(a+aD \Rightarrow (a+a)*(a+a)B \Rightarrow (a+a)*(a+a)$

Da bismo proširili danu gramitiku za više ugnježđenja, najprije je napišimo kao

$A_0 \rightarrow aA_1 \mid (A_2$
$A_1 \rightarrow *A_0 \mid +A_0 \mid \varepsilon$
$A_2 \rightarrow aA_3$
$A_3 \rightarrow)A_1 \mid +A_2 \mid *A_2$

Ako se doda još jedna razina, imamo dodatak produkcija:

$C \rightarrow (E$
$E \rightarrow aF$
$F \rightarrow)D \mid +E \mid *E$

Na primjer tablica prijelaza za četiri razine ugnježđenja je:

	(	)	+	*	a
→A	C				B
⊗B			A	A	
C	E				D
D		B	C	C	
E	G				F
F		D	E	E	
G	I				H
H		F	G	G	
I	K				J
J		H	I	I	

♣ Primjer 9.5

No, generator jezika $\mathcal{L}(\mathcal{G}_1)$, odnosno $\mathcal{L}(\mathcal{G}_2)$, iz prethodnog primjera, 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 jednaka 0 i izostavljena je):

	(	)	+	*	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 Prepoznavac_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); ReadKey
END.
```

Evo nekoliko primjera prihvaćenih ulaznih nizova 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)
```

Pitanja i zadaci

1) 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

2) 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

1. AHO, A. *Indexed grammars - an extension of context-free grammars*, J. ACM 15, 647-671, 1968.

2. AHO, V. A.; ULLMAN, D. J.:
The Theory of Parsing, Translation, and Compiling, vol. I: *Parsing*, Prentice-Hall, 1972.

3. AHO, V. A.; ULLMAN, D. J.:
Principles of Compiler Design, Addison-Wesley Publishing Company, 1979.

4. AHO; SETHI; ULLMAN:
Compilers: Principles, Techiques, and Tools, Addison-Wesley Publishing Company, 1986.

5. ANDERSON, A. J.:
Automata Theory with Modern Applications, CAMBRIDGE UNIVERSITY PRESS, 2006.

6. BACKHOUSE, C. R.:
Sintax of Programming Languages: Theory and Practice, Prentice-Hall, 1979.

7. BERRY, E. R.:
Programming Language Translation, Ellis Horwood Limited, 1981.

8. CHRISWELL, I.:
A Course in Formal Languages, Automata and Groups, Springer-Verlag, London, 2009.

9. DENNING, J. P.; DENNIS, B. J.; QUALITZ, E. J.:
Machines, Languages, and Computation, Prentice-Hall, 1978.

10. DOVEDAN, Z.:
Jedan model sintaktičke analize jezika za programiranje, disertacija, Filozofski fakultet, Zagreb, 1992.

11. DOVEDAN, Z.:
Pascal i programiranje (1), don, Zagreb, 1995.

12. DOVEDAN, Z.:
Sintaktička analiza jezika sa svojstvima, Ljubljana, Informatica 82/3, 1983.

13. DOVEDAN, Z.:
FORMALNI JEZICI • sintaksna analiza, Zavod za informacijske studije, Filozofski fakultet, Zagreb, 2003.

14. DOVEDAN HAN, Z.:
Pascal s tehnikama programiranja (1), VVG, Velika Gorica, 2011.

15. EIJCK, J. van:
Sequentially Indexed Grammars, CWI and ILLC, Amsterdam, Uil-OTS, Utrecht, 2005.

16. GOOS, G.; HARTMANIS, J., editors:
Compiler Construction, An Advanced Course, Springer-Verlag, 1976.

17. GRUNE, D.:
Parsing Techniques – A Practical Guide, Ellis-Horwood, 1990.

18. HOPCROFT, E. J.; MOTWANI, R.; ULLMAN, D. J.:
Introduction to Automata Theory, Languages, and Computation, second edition, Addison-Wesley, 2001.

19. KALUŽNIN, A. L.:
Što je matematička logika, Zagreb, Školska knjiga 1975.

20. KARHUMÄKI, J.:
Automata and Formal Languages, Spring, 2005.

21. KUREPA, S.:
Uvod u matematiku, Zagreb, Tehnička knjiga, 1970.

22. PAAKKI, J.:
Atribute Grammar Paradigms – A High-Level Methodology in Language Implementation, ACM Computing Surveys, Vol. 27, No. 2, June 1995.

23. SHALLIT, J.:
A Second Course in Formal Languages and Automata Theory, CAMBRIDGE UNIVERSITY PRESS, Cambridge, 2009.

24. SLONNEGER, K.; KURTZ, B. L. *Formal Syntax and Semantics of Programming Languages*, Addison-Wesley Publishing Company, 1995.

25. TOMITA, M., editor:
Current Issues in Parsing Technology, Kluwer Academic Publishers, 1991.

26. WAITE, M. W.; GOOS, G.:
Compiler Construction, Springer-Verlag, 1984.

27. WIRTH, N.:
Algorithms + Data Structures = Programs, Prentice-Hall, 1976.

28. YEH, T. R., editor:
Applied Computation Theory: Analysis, Design, Modeling, Prentice-Hall, 1976.