

SINIŠA SRBLJIĆ

JEZIČNI PROCESORI

000010110
111000011
100001011
100000111
001001101

1



ELEMENT



Uporabu ovog sveučilišnog udžbenika odobrio je
Odbor za znanstveno-nastavnu literaturu Sveučilišta u Zagrebu
rješenjem broj 02-310/3-2000 od 13. lipnja 2000.

Intelektualno je vlasništvo, poput svakog drugog vlasništva, neotuđivo, zakonom
zaštićeno i mora se poštivati. Nijedan dio ove knjige ne smije se preslikati niti
umnažati na bilo koji način, bez pismenog dopuštenja nakladnika.

CIP – Katalogizacija u publikaciji
Nacionalna i sveučilišna knjižnica, Zagreb

UDK 004.4'4 (075.8)

SRBLJIĆ, Siniša
Jezični procesori 1 : uvod u teoriju formalnih jezika,
automata i gramatika / Siniša Srbljić – Zagreb
: Element, 2000

Bibliografija: str. 208 – 209. – Kazalo.

ISBN 953-197-129-3

400619030

ISBN 953-197-129-3

Siniša Srbljić

JEZIČNI PROCESORI 1

Uvod u teoriju formalnih jezika, automata i gramatika

Zagreb, 2000.

©Prof. dr. sc. Siniša Srbljić

Urednik
Prof. dr. sc. Neven Elezović

Recenzenti
Prof. dr. sc. Leo Budin, FER, Zagreb
Prof. dr. sc. Ignac Lovrek, FER, Zagreb
Prof. dr. sc. Damir Kalpić, FER, Zagreb
Prof. dr. sc. Nikola Rožić, FESB, Split

Lektorica
Marija Bednjanec, prof.

Nakladnik
Element, Zagreb

Tehnički urednik
Sandra Gračan, dipl. inž.

Crteži
Autor

Design ovitka
Julija Vojković

Tisk
Ekološki glasnik, Donja Lomnica

SADRŽAJ

UVOD	2
1.1 ZNAKOVLJE I OZNAKE	9
1.2 PRIMJER FORMALNOG JEZIKA, PRIPADAJUĆEG AUTOMATA I GRAMATIKE	14
REGULARNI JEZICI	15
2.1 KONAČNI AUTOMATI	15
2.1.1 Deterministički konačni automat (<i>DKA</i>)	15
2.1.3 Nedeterministički konačni automat (<i>NKA</i>)	29
2.1.4 Nedeterministički konačni automat s ϵ prijelazima (ϵ - <i>NKA</i>)	34
2.1.5 Konačni automati s izlazom	39
2.2 REGULARNI IZRAZI	44
2.2.1 Definicija regularnih izraza	44
2.2.2 Konstrukcija ϵ - <i>NKA</i> na temelju zadanih regularnih izraza	46
2.2.3 Generator konačnog automata	50
2.3 SVOJSTVA REGULARNIH JEZIKA	51
2.3.1 Svojstva zatvorenosti regularnih jezika	51
2.3.2 Regularne definicije	53
2.3.3 Svojstvo napuhavanja	54
2.4 GRAMATIKA	56
2.4.1 Formalna gramatika	56
2.4.2 Regularna gramatika	61
KONTEKSTNO NEOVISNI JEZICI	69
3.1 KONTEKSTNO NEOVISNA GRAMATIKA	69
3.1.1 Nejednoznačnost gramatike, jezika i niza	69
3.1.2 Pojednostavljenje gramatike	76
3.1.3 Parsiranje niza	88
3.2 POTISNI AUTOMAT (PA)	103
3.2.1 Model potisnog automata	103
3.2.2 Definicija potisnog automata	106
3.2.3 Potisni automat i kontekstno neovisna gramatika	110
3.3 SVOJSTVA KONTEKSTNO NEOVISNIH JEZIKA	117
3.3.1 Svojstva zatvorenosti kontekstno neovisnih jezika	118
3.3.2 Svojstvo napuhavanja	124
REKURZIVNO PREBROJIVI JEZICI	126
4.1 TURINGOV STROJ (TS)	126
4.1.1 Osnovni model Turingovog stroja	126
4.1.2 Metode izrade Turingovog stroja	133
4.1.3 Prošireni modeli Turingovog stroja	138
4.1.4 Pojednostavljeni modeli Turingovog stroja	146
4.1.5 Generiranje jezika Turingovim strojem	150
4.2 GRAMATIKA NEOGRANIČENIH PRODUKCIJA	152
4.2.1 Konstrukcija TS za jezik zadan gramatikom neograničenih produkcija	153
4.2.2 Konstrukcija gramatike za jezik zadan TS	154

4.3 SVOJSTVA REKURZIVNIH I REKURZIVNO PREBROJIVIH JEZIKA.....	156
4.3.1 <i>Svojstva zatvorenosti rekurzivnih i rekurzivno prebrojivih jezika.....</i>	156
4.3.2 <i>Izračunljivost.....</i>	158
4.3.3 <i>Odlučivost</i>	162
KONTEKSTNO OVISNI JEZICI.....	165
5.1 KONTEKSTNO OVISNA GRAMATIKA.....	165
5.2 LINEARNO OGRANIČEN AUTOMAT (LOA)	167
5.2.1 <i>Konstrukcija LOA za jezik zadan kontekstno ovisnom gramatikom.....</i>	168
5.2.2 <i>Konstrukcija kontekstno ovisne gramatike za jezik zadan LOA</i>	168
5.3 SVOJSTVA KONTEKSTNO OVISNIH JEZIKA.....	170
5.3.1 <i>Unija, nadovezivanje i Kleeneov operator</i>	170
5.3.2 <i>Presjek i komplement</i>	171
5.3.3 <i>Odlučivost kontekstno ovisnih jezika.....</i>	172
5.3.4 <i>Primjer rekurzivnog jezika koji nije kontekstno ovisni jezik</i>	174
RAZREDBA JEZIKA, AUTOMATA I GRAMATIKA.....	177
6.1 STRUKTURNΑ SLOŽENOST JEZIKA	177
6.1.1 <i>Chomskyjeva hijerarhija jezika.....</i>	177
6.1.2 <i>Hijerarhija gramatika i automata.....</i>	179
6.2 SLOŽENOST PRIHVAĆANJA JEZIKA	180
6.2.1 <i>Definicija prostorne i vremenske složenosti prihvaćanja jezika</i>	181
6.2.2 <i>Svojstva prostorne i vremenske složenosti prihvaćanja jezika</i>	183
6.2.3 <i>Klase jezika s obzirom na složenost prihvaćanja jezika.....</i>	187
6.2.4 <i>Klase jezika polinomne složenosti.....</i>	194

1 UVOD

Uporabu elektroničkih digitalnih računala nemoguće je zamisliti bez primjene jezičnih procesora i programskih jezika kao što su C, C++, Java, FORTRAN, itd. Rješavanje problema primjenom računala zahtijeva izgradnju algoritma pomoću jednog od programskega jezika. Danas postoji mnoštvo programskih jezika i jezičnih procesora različitih mogućnosti i područja primjene. Sastavnice viših programskih jezika, kao što su variable, iterativni i rekurzivni algoritmi, polja, datoteke i apstraktne tipove podataka omogućuju korištenje računala, a da korisnik pri tome ne mora poznavati sklopovsku građu računala koja je zasnovana na registrima, binarnim zapisima, numeričkim adresama, memorijskoj hijerarhiji, itd. Zadatak jezičnog procesora je da svojim sučeljem približi uporabu računala različitim područjima primjene.

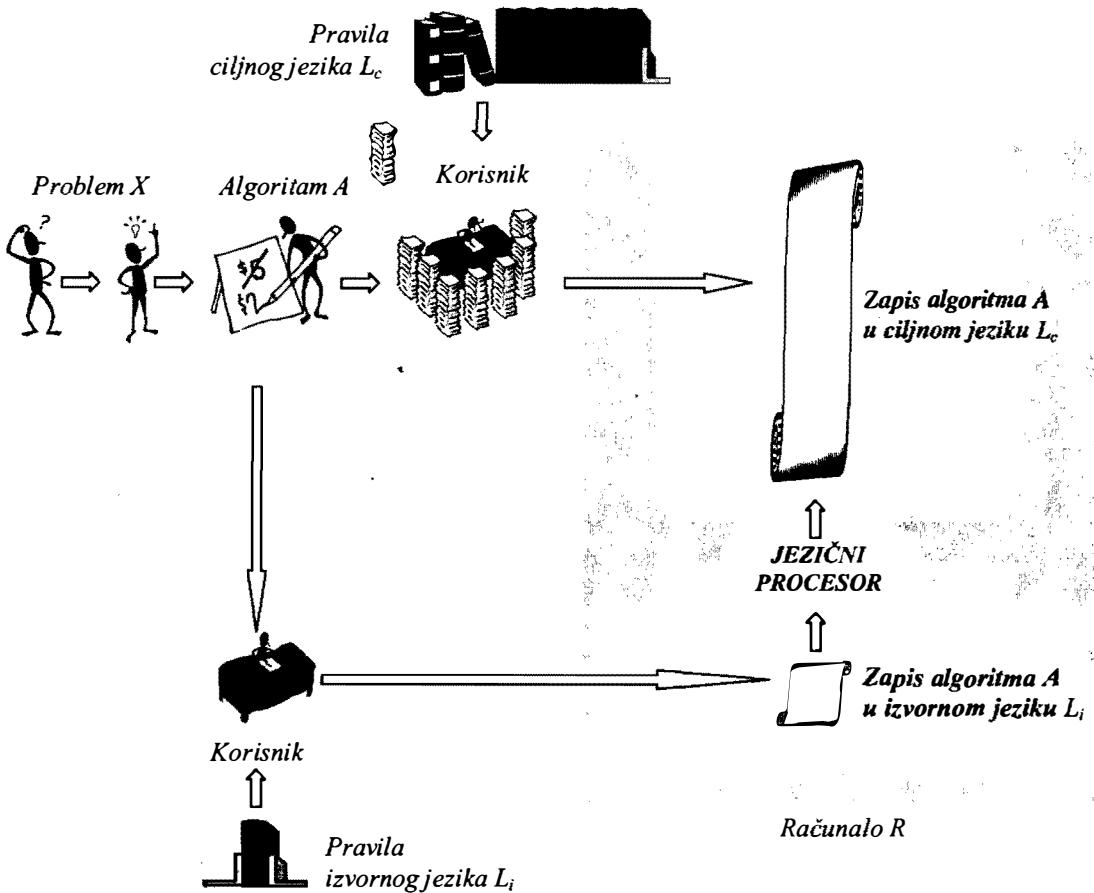
Slika 1.1 opisuje rad jezičnog procesora. Za rješavanje problema X želi se koristiti računalo R . Neka je programski jezik L_c ugrađen u računalo R i neka je programe zapisane tim programskim jezikom moguće izravno izvoditi na računalu R . A je algoritamski zapis rješenja problema X . Pretpostavimo da postoji više razloga zbog kojih programski jezik L_c nije prikladan za zapis algoritma A : neprilagođenost području primjene, složena pravila koja često prepostavljaju poznavanje sklopovske grade računala, zapis algoritma A pomoću jezika L_c je velik i nepregledan, itd. Potrebno je definirati novi programski jezik L_i koji je prikladniji za rješavanje problema X . Međutim, da bi se omogućilo izvođenje programa zapisanih pomoću programskega jezika L_i na računalu R , potrebno je u računalo R ugraditi jezični procesor koji prevodi zapis algoritma iz jezika L_i u zapis algoritma u jeziku L_c . Dakle, osnovni zadat jezičnog procesora je:

Jezični procesor prevodi zapis algoritma iz izvornog jezika L_i u zapis algoritma u ciljnog jeziku L_c koji se može izvesti na zadatom računalu.

Jezični procesor jest u pravilu program koji na ulazu čita program napisan u izvornom jeziku (*izvorni program*), te ga prevodi u istovjetni program u ciljnem jeziku (*ciljni program*). Definicija jezičnog procesora zasnovana je na tri jezika: *izvorni jezik*, *ciljni jezik* i *jezik izgradnje*. Budući da je u većini slučajeva jezični procesor program, jezik izgradnje jest jezik pomoću kojeg je ostvaren sam jezični procesor. Na temelju dane definicije jezični procesor prikazuje se na sljedeći način:

$$JP_{L_g}^{L_i \rightarrow L_c}$$

gdje je JP jezični procesor, L_i je izvorni jezik, L_c je ciljni jezik, te L_g je jezik izgradnje. Pored tri spomenuta jezika vezana uz definiciju jezičnog procesora, razvijaju se i posebni jezici definiranja jezika. Jezik definiranja jezika omogućuje jednostavni i jednoznačni zapis pravila programskega jezika.



Slika 1.1: Osnovni zadatak jezičnog procesora

Budući da se rad jezičnog procesora zasniva na prevođenju programa iz jednog jezika u drugi jezik, potrebno je pobliže odrediti što su to program i jezik. Program se definira kao niz znakova. Znakovi su slova abecede, dekadske znamenke, operatori, posebni znakovi i pravopisni znakovi. Na primjer, sljedeći niz znakova:

```
import java.awt.*;public class Example{public static void main(String args[ ]){}}
```

jest najkraći ispravno napisan program pomoću programskog jezika Java:

```
import java.awt.*;
public class Example
{
    public static void main(String args[])
    {
    }
}
```

Niz znakova:

```
import java.awt.*;public class Example{ public static void main(String args[ ]) {int
a=2;int b=3;int c;c=a+b;}}
```

jest izvorni program napisan programskim jezikom Java koji zbraja dva cijela broja:

```

import java.awt.*;

public class Example
{
    public static void main(String args[])
    {
        int a=2;
        int b=3;
        int c;

        c=a+b;
    }
}

```

Sljedeći niz znakova:

```

import java.awt.*;public class Example{public static void main(String args[]){int
a=2;int b=3;int c;c=a-b;} }

```

jest izvorni program koji oduzima dva cijela broja:

```

import java.awt.*;

public class Example
{
    public static void main(String args[])
    {
        int a=2;
        int b=3;
        int c;

        c=a-b;
    }
}

```

Neka se prethodno dani primjeri programa označe oznakama p_1 , p_2 i p_3 . Dani primjeri jasno pokazuju da je moguće navesti beskonačno mnogo primjera ispravno napisanih programa u programskom jeziku Java. Programi se promatraju kao konačni nizovi znakova. Neka L_{Java} označava skup svih ispravno napisanih programa u programskom jeziku Java. Skup L_{Java} jest beskonačni skup konačnih nizova i bilo koji niz u skupu L_{Java} je ispravno napisan program u programskom jeziku Java. Na primjer, programi p_1 , p_2 i p_3 , odnosno nizovi znakova p_1 , p_2 i p_3 , su elementi skupa L_{Java} . Neka se sa L_{svi} označi skup svih nizova koji se mogu napisati pomoću zadanih znakova abecede, dekadskih znamenaka, operatora, posebnih znakova i pravopisnih znakova. Sljedeći primjer niza znakova p_4 :

```

import java.awt.*;public class Example{public static void main(String args[ ]) {int a=2;int
b=3;int c;c((((=a-b;)} }

```

nije ispravno napisan program u programskom jeziku Java:

```

import java.awt.*;

public class Example
{
    public static void main(String args[])
    {
        int a=2;
        int b=3;
        int c;
        c((((=a-b;
    }
}

```

Budući da dani niz p_4 nije ispravno napisan program u programskom jeziku Java, zaključuje se da je skup svih ispravno napisanih programa L_{Java} pravi podskup skupa svih nizova L_{svi} . Na primjer, niz p_4 je u skupu L_{svi} , ali nije u skupu L_{Java} .

U dalnjem opisu rada jezičnog procesora formalni jezici definiraju se skupovima. Na primjer, skup L_{Java} je formalna definicija programskog jezika Java. Prethodno je opisano da je najopćenitiji model jezičnog procesora zasnovan na procesu prevođenja jezika:

Jezični procesor prevodi program (niz znakova) p_x koji je u jeziku L_i (koji je u skupu L_i) u program (u niz znakova) p_y koji je u jeziku L_c (koji je u skupu L_c).

Jezični procesori su složeni programski sustavi koji za potrebe prevođenja izvornog programa u ciljni program koriste različite procese. U nastavku odjeljka opisuju se dva osnovna procesa koji čine okosnicu postupka prevođenja: *prihvaćanje izvornog programa i generiranje ciljnog programa*.

Neka jezični procesor $JP_{L_g}^{L_i \rightarrow L_c}$ prevodi program p_x iz jezika L_i u program p_y iz jezika L_c .

Prihvaćanje izvornog programa p_x iz jezika L_i izvodi se na sljedeći način:

Tijekom procesa prihvaćanja čita se znak po znak izvornog programa p_x (odnosno čita se znak po znak niza p_x). Ako je program (niz) p_x u jeziku L_i (u skupu L_i), onda nakon posljednjeg pročitanog znaka ispisuje se da je niz p_x ispravno napisani program u jeziku L_i . Nije li program (niz) p_x u jeziku L_i (u skupu L_i), nakon posljednjeg pročitanog znaka ispisuje se da niz p_x nije ispravno napisani program u jeziku L_i .

Proces prihvaćanja ispituje ispravnost izvornog programa. Nije li niz p_x ispravno napisani program u jeziku L_i , daljnji proces prevođenja izvornog jezika se zaustavlja. Jezični procesor ispisuje poruke o učinjenim pogreškama u izvornom programu. Za potrebe prihvaćanja programa (nizova) iz jezika L_i gradi se *formalni automat* M . Formalni automat M je matematički model automata koji čita ulazni niz znak po znak i koji kao izlaz ispisuje je li pročitani niz znakova u zadanim jezicima (odnosno u zadanim skupima). Različite jezike (odnosno skupove) prihvaćaju različiti formalni automati. U odjeljaku 1.2 dan je primjer jednog jednostavnog formalnog automata.

Ako je niz p_x ispravno napisani program u jeziku L_i , jezični procesor pokreće proces generiranja ciljnog programa. *Generiranje ciljnog programa* p_y u jeziku L_c izvodi se na sljedeći način:

Ciljni program generira se tako da se ispisuje znak po znak programa p_y (niza znakova p_y) koji je u jeziku L_c i koji je je prijevod izvornog programa p_x .

Za potrebe generiranja ciljnog programa p_y u jeziku L_c gradi se *formalna gramatika* G . Formalna gramatika G je matematički model koji ispisom znak po znak gradi, odnosno generira, nizove znakova. Gramatika se zadaje tako da generira sve programe (sve nizove znakova) iz jezika L_c (iz skupa L_c). Budući da gramatika ne generira niti jedan program (niz znakova) koji nije u jeziku L_c , bilo koji generirani niz jest ispravno napisani program u cilnjom jeziku L_c (odnosno u skupu L_c). Istodobnim čitanjem znakova izvornog programa p_x i generiranjem znakova ciljnog programa p_y osigura se da gramatika G ne generira bilo koji program iz jezika L_c (bilo koji niz iz skupa L_c), već da gramatika G generira ciljni program p_y koji je upravo prijevod izvornog programa p_x . Različite jezike (odnosno skupove) generiraju različite formalne gramatike. U odjeljaku 1.2 dan je primjer jedne jednostavne formalne gramatike.

Zbog složenosti jezičnih procesora, proces prevođenja nije moguće ostvariti kao jedinstveni dvostupanjski proces prihvaćanja izvornog programa i generiranja ciljnog programa. U stvarnim izvedbama jezičnih procesora ugrađen je višerazinski proces postupnog prevođenja izvornog programa u ciljni program. Proces prevođenja na svakoj razini ostvaruje se kao dvostupanjski proces *prihvaćanja* programa generiranog od procesa prevođenja prethodne razine, te *generiranje programa* za proces prevođenja naredne razine. Početni proces prevođenja prihvata izvorni program dok završni proces prevođenja generira ciljni program.

Različite razine prevodenja podijeljene su u dvije osnovne faze rada jezičnog procesora: *faza analize izvornog programa i faza sinteze ciljnog programa*. Tijekom faze analize izvornog programa izvode se procesi prevodenja na dvije razine. Osnovni zadatak procesa prevodenja tijekom faze analize je pojednostavljenje procesa prihvaćanja izvornog jezika. U većini današnjih jezičnih procesora, tijekom faze sinteze ciljnog programa proces prevodenja izvodi se na tri razine. Osnovni zadatak procesa prevodenja tijekom faze sinteze je pojednostavljenje procesa generiranja ciljnog programa.

Tijekom faze analize izvornog programa izvode se dva procesa prevodenja. Jedan proces prevodenja izvodi se tijekom *leksičke analize*, a drugi proces prevodenja izvodi se tijekom *sintaksne i semantičke analize*. Leksička analiza grupira znakove izvornog programa u osnovne elemente jezika, koji se nazivaju *leksičke jedinke*. Na primjer, leksičke jedinke su: *variabla* (npr. niz od više slova i brojaka koji započinje slovom), *ključne riječi* (npr. ključna riječ za naredbu grananja je **IF**), *konstante* (npr. realne konstante, cjelobrojne konstante, znakovne konstante), operatori i pravopisni znakovi. Leksička pravila određuju dozvoljene oblike leksičkih jedinki.

Leksička jedinka formalno se definira kao niz znakova. Leksička pravila određuju skup svih pravilno napisanih leksičkih jedinki (nizova) za zadani programski jezik. Skup pravilno napisanih leksičkih jedinki može biti beskonačan i u formalnom smislu taj skup definira jezik leksičkih jedinki. Za potrebe prihvaćanja leksičkih jedinki gradi se zasebni formalni automat koji je osnovica leksičkog analizatora. Za svaku leksičku jedinku napisanu u izvornom programu provjerava se putem automata je li u skupu pravilno napisanih leksičkih jedinki (odnosno je li u jeziku leksičkih jedinki). Ako su sve leksičke jedinke ispravno napisane, pristupa se njihovom prevodenju u niz jedinstvenih znakova. Svaka leksička jedinka zamijeni se jednim jedinstvenim znakom.

Na primjer, neka je zadana sljedeća naredba (niz znakova) u izvornom programu programskog jezika Java:

```
PPP=jura[i][j+7+24];
```

Naredba se sastoji od četrnaest leksičkih jedinki: varijabla PPP, operator pridruživanja =, varijabla jura, otvorena zagrada [, varijabla i, zatvorena zagrada], otvorena zagrada [, varijabla j, operator zbrajanja +, cjelobrojna konstanta 7, operator zbrajanja +, cjelobrojna konstanta 24, zatvorena zagrada] i pravopisni znak ;. Budući da su sve leksičke jedinke pravilno napisane (sve su u skupu pravilno napisanih jedinki), leksički analizator pristupa prevodenju naredbe PPP=jura[i][j+7+24];. Generira se niz jedinstvenih znakova:

```
V←V[V][V+K+K];
```

gdje je V jedinstveni znak leksičke jedinke varijabla, ← je jedinstveni znak leksičke jedinke znaka pridruživanja =, K je jedinstveni znak leksičke jedinke konstante, dok je uloga ostalih jedinstvenih znakova lako uočljiva. Iako su sve variable predstavljenje jedinstvenim znakom V, a konstante jedinstvenim znakom K, leksički analizator gradi posebnu podatkovnu strukturu, nazvanu *tablica znakova*, u koju se spremaju svi ostali podaci važni za variable i konstante. Dok se rad sintaksnog analizatora temelji samo na informaciji sadržanoj u nizu jedinstvenih znakova leksičkih jedinki V←V[V][V+K+K];, ostale faze rada jezičnog procesora koriste i ostale podatke spremljene u tablici znakova.

Nakon leksičke analize, izvorni program preveden je u niz jedinstvenih znakova koji je osnovica za sljedeći proces prevodenja. Proses prevodenja raspodijeljen je između sintaksne i semantičke analize. Tijekom sintaksne analize izvodi se proces prihvaćanja nizova jedinstvenih znakova leksičkih jedinki, a tijekom semantičke analize izvodi se proces generiranja višeg međukoda. Sintaksna pravila definiraju: način gradnje izraza pomoću leksičkih jedinki (npr. aritmetički izrazi, logički izrazi), način gradnje ispravnih naredbi programskog jezika pomoću leksičkih jedinki i izraza (npr. naredbe pridruživanja, naredbe bezuvjetnog grananja, naredbe uvjetnog grananja), način gradnje bloka naredbi, te strukturu cjelokupnog izvornog programa. Na primjer, uobičajeno je da naredbe grananja imaju sljedeći oblik: **IF<izraz>THEN<naredba>**. Primjer pokazuje da sintaksna pravila pored samih leksičkih jedinki (u primjeru su leksičke jedinke **IF**, **THEN** i ;) uključuju i složenije strukture, kao što je logički izraz <izraz>, koji se zasebno definiraju. Nadalje, pravila se zadaju rekurzivno, tj. struktura naredbe definira se pomoću naredbe (u primjeru označeno kao <naredba>).

Sintaksna pravila definiraju skup (jezik) svih nizova jedinstvenih znakova leksičkih jedinki koji su sintaksno ispravni izvorni programi. Za potrebe prihvaćanja nizova jedinstvenih znakova leksičkih jedinki gradi se formalni automat koji je okosnica sintaksnog analizatora.

Nakon uspješno provedene sintaksne analize, odnosno nakon što formalni automat sintaksnog analizatora utvrdi da je niz jedinstvenih leksičkih jedinki u skupu sintaksno ispravnih izvornih programa, semantički analizator pokreće proces generiranja višeg međukoda. Prije generiranja višeg međukoda, provjeravaju se semantička pravila. Semantička pravila su interpretacijska pravila koja povezuju izvođenje programa s ponašanjem računala. Na primjer, ako se jednoj varijabli pridruži vrijednost zbroja cijelobrojne i realne varijable, onda zbroj poprima realnu vrijednost. Nadalje, semantika jezika određuje skup dozvoljenih značenja. Na primjer, neki jezici ne dozvoljavaju množenje cijelobrojnih i realnih varijabli. Potrebno je prethodno pretvoriti obje varijable u realne ili cijelobrojne. U tim jezicima operator množenja ima značenje cijelobrojnog ili realnog množenja, ali nešta značenje množenja cijelih brojeva s realnim.

Tijekom procesa generiranja višeg međukoda izračunaju se konstantne vrijednosti i pojednostavi se struktura naredbe. U višem međukodu niz $7+24$ zamijeni se s nizom 31 koji ima značenje cijelobrojnog zbroja brojeva 7 i 24. Struktura naredbi se pojednostavi na temelju informacija spremljenih u tablici znakova. Tablica znakova popunjava se tijekom svih faza rada jezičnog procesora. Pretpostavimo da su u dijelu deklaracija varijabli u Java programu bile zadane sljedeće definicije:

```
int i, j;
int PPP;
int jura[][]=new int[20][100];
```

Na temelju datih definicija zaključuje se da su i , j i PPP cijelobrojne varijable, a $jura$ je dvodimenzionalno polje cijelobrojnih vrijednosti. Označavanje dvodimenzionalnog polja se pojednostavi u cilju smanjivanja broja znakova potrebnih za zapis naredaba višeg međukoda, te u cilju učinkovitije sinteze ciljnog programa. Iako je uobičajeno da naredbe višeg međukoda sadrže kazaljke na mjesta u tablici znakova gdje su spremljeni svi podaci o varijablama i konstantama, zbog jednostavnosti prikaza u sljedećem primjeru generirane naredbe višeg međukoda navedena su izvorna imena varijabli i izračunata vrijednost cijelobrojne konstante 31. Niz jedinstvenih znakova leksičkih jedinki $V \leftarrow V[V] [V+K+K]$; prevodi se u niz znakova naredbe višeg međukoda:

```
PPP ← jura[i, j+31]
```

Generiranjem višeg međukoda završava faza analize izvornog programa i započinje faze sinteze ciljnog programa. U cilju pojednostavljenja sveukupnog procesa generiranja i optimiranja ciljnog programa, u većini današnjih jezičnih procesora tijekom faze sinteze izvode se tri procesa prevodenja: *prevodenje višeg međukoda u srednji međukod*, *prevodenje srednjeg međukoda u niži međukod* i *prevodenje nižeg međukoda u ciljni program*.

Jedan od osnovnih zadataka procesa prevodenja višeg međukoda u srednji međukod jest pretvorba složenih podatkovnih struktura, kao što su polja podataka, i složenih naredbi koje upravljaju tijekom izvođenja programa u niz naredbi koje koriste samo varijable i jednostavne naredbe grananja. Na primjer, naredba dohvata vrijednosti elementa dvodimenzionalnog polja prevodi se u niz naredbi koje najprije izračunaju sljednu adresu zadanog elementa polja u memoriji računala, a zatim se generira naredba koja dohvaća vrijednost elementa polja na temelju izračunate adrese. Neka je dvodimenzionalno polje spremljeno u memoriju redak po redak, gdje su pojedini elementi polja cijelobrojne vrijednosti koje zauzimaju 4 okteta memorije. Na temelju vrijednosti indeksa dvodimenzionalnog polja:

```
jura[i, j+31]
```

i na temelju veličine maksimalnog indeksa dvodimenzionalnog polja:

```
int jura[][]=new int[20][100];
```

izračuna se indeks jednodimenzionalnog polja kao $(i*100)+(j+31)$. Vrijednost izračunatog indeksa jednodimenzionalnog polja pomnoži se brojem okteta potrebnih za spremanje jednog elementa polja, a to je 4, te se time dobiva adresa traženog elementa jednodimenzionalnog polja. Ako se izračunatoj adresi doda početna adresa polja adr_jura , dobiva se apsolutna adresa traženog podatka u memoriji računala. Sljedeće naredbe slijedno računaju traženu adresu pomoći privremenih varijabli $v1$ do $v6$, dok zadnja naredba

dohvaća podatak koji je spremlijen na adresi izračunatoj u privremenoj varijabli v6. Vrijednost dohvaćenog podatka pridruži se varijabli PPP:

```
v1 ← i*100
v2 ← j+31
v3 ← v1+v2
v4 ← 4*v3
v5 ← adr_jura
v6 ← v5+v4
PPP ← @v6
```

gdje oznaka @v6 ima značenje dohvata podatka s adrese koja je jednaka vrijednost variabile v6.

U sljedećem koraku prevodi se srednji međukod u niži međukod. Naredbe nižeg međukoda koriste simboličke registre r1 do r8. Simbolička imena varijabli prevode se u stvarne memorejske adrese. Varijabla i spremljena je u prva četiri okteta počev od početne adrese početak, varijabla j spremljena je u druga četiri okteta, a prvi elemenet dvodimenzionalnog polja jura spremlijen je na adresi početak+8. Naredbe nižeg međukoda su:

```
r1 ← [početak]
r2 ← r1*100
r3 ← [početak+4]
r4 ← r3+31
r5 ← r4+r2
r6 ← 4*r5
r7 ← početak+8
r8 ← [r7+r6]
```

gdje oznaka [početak] ima značenje dohvata podatka s adrese početak.

Tijekom završnog procesa prevođenja generira se ciljni program na temelju nižeg međukoda. U cilnjom jeziku se umjesto simboličkih registara koriste stvarni registri procesora računala, simboličke memorejske adrese poprimaju numeričke vrijednosti, a naredbe ciljnog programa su strojne naredbe procesora sastavljene od nula i jedinica.

Pojednostavljeni i poopćeni opis rada jezičnog procesora zorno predviđa dva osnovna procesa koja su osnovica kako cijelokupnog rada jezičnog procesora, tako i svake pojedine faze njegovog rada: *proces prihvaćanja jezika* i *proces generiranja jezika*. Ti procesi su dijelovi jedinstvenog procesa prevođenja. U poglavljima koja slijede opisane su klase jezika različite složenosti, svojstva klase jezika, automati koji prihvaćaju danu klasu jezika i gramatike koje ih generiraju. Poznavanje formalnih jezika, automata i gramatika jest temelj bez kojeg nije moguće postići učinkovit rad jezičnog procesora, niti je moguće razumjeti rad današnjih jezičnih procesora.

U nastavku uvodnog poglavlja pored znakovlja i oznaka opisuju se primjeri jednostavnog formalnog jezika, automata i gramatike. U poglavlju 2 opisuju se regularni jezici, konačni automati i regularna gramatika. Regularni jezici su najjednostavniji u smislu da ih prihvaćaju najjednostavniji automati (konačni automati), odnosno da ih generira najjednostavnija gramatika. Konačni automati osnovica su leksičke analize većine jezičnih procesora, kao i različitih procesa prevođenja tijekom faze sinteze ciljnog programa. Kontekstno neovisni jezici, kontekstno neovisna gramatika i potisni automati opisani su u poglavlju 3. Kontekstno neovisna gramatika posebice je od značaja za sintaksnu analizu izvornog programa. U poglavlju 4 opisani su najsloženiji jezici, a to su rekurzivno prebrojivi jezici. Rekurzivno prebrojive jezike prihvaćaju Turingovi strojevi, koji predstavljaju najopćenitiji formalni model izračunljivosti. Budući da je primjenom Turingovih strojeva moguće simulirati rad bilo kojeg suvremenog digitalnog električkog računala, modeli Turingovih strojeva koriste se tijekom semantičke analize za interpretaciju značenja pojedinih dijelova izvornog programa. Linearno ograničeni automat predstavlja ograničeni model Turingovog stroja koji prihvaća kontekstno ovisne jezike. Kontekstno ovisni jezici opisani su u poglavlju 5. Završna razredba jezika prema njihovoj strukturnoj složenosti dana je u poglavlju 6. U drugom dijelu poglavlja 6 definirana je vremenska i prostorna funkcija složenosti prihvaćanja jezika.

1.1 Znakovlje i oznake

Znak i abeceda

Znak jest apstraktni pojam koji se ne definira formalno kao što se ne definira ni točka u geometriji. Kao primjer znaka obično se uzimaju brojke i slova.

Abeceda jest konačni skup znakova. Brojke 0 i 1 čine binarnu abecedu $B = \{0, 1\}$.

Niz

Niz jest konačni slijed znakova abecede pozicioniranih jedan do drugog. Na primjer, nizovi 011, 1011 i 1100011 zadani su nad binarnom abecedom $B=\{0, 1\}$.

Duljina niza jednaka je broju znakova od kojih se sastoje niz. Na primjer, niz $w=1001101$ ima duljinu $|w|=7$. Niz w sastoje se od sedam znakova binarne abecede $B=\{0, 1\}$.

Prazni niz označava se znakom ε i duljina praznog niza jest $|\varepsilon|=0$. To je niz koji ne sadrži niti jedan znak.

U tablici 1.1 navedeni su pojmovi vezani uz nizove. Prazni niz ε jest neutralni element za operaciju nadovezivanja $\varepsilon w = w\varepsilon = w$.

Nadovezivanje nizova označava se potencijama (npr. $w=y^2$ ili $w=xxxx=x^4$). Potencije definiraju sljedeće izraze:

$$w^0 = \varepsilon,$$

$$w^i = w^{i-1}w \text{ za } i > 0,$$

$$w^1 = w^0w = \varepsilon w = w.$$

Prefiks niza w dobije se odbacivanjem niti jednog, jednog ili više posljednjih znakova niza w . (npr. Niz ban jest prefiks niza banana)

Sufiks niza w dobije se odbacivanjem niti jednog, jednog ili više početnih znakova niza w . (npr. niz nana jest sufiks niza banana)

Podniz niza w dobije se odbacivanjem sufiksa i prefiksa niza w . (npr. niz ana jest podniz niza banana)

Pravi prefiks, sufiks ili podniz niza w je neprazan niz x koji je prefiks, sufiks ili podniz niza w i $x \neq w$.

Podslijed niza w dobije se odbacivanjem niti jednog, jednog ili više ne nužno uzastopnih znakova niza w . (npr. baaa je podslijed niza banana)

Nadovezivanje niza x i niza y dobije se dodavanjem znakova niza y iza niza x , $w = xy$. (npr. nadovezivanjem niza ban i niza ana nastaje niz banana).

Tablica 1.1: Pojmovi vezani uz nizove

Jezik

Formalni jezik je skup nizova nad abecedom. Na temelju dane formalne definicije kao primjeri jezika mogu se navesti: prazni skup $L_1 = \{\}$, skup koji ima jedan element - prazni niz $L_2 = \{\varepsilon\}$, skup nizova za koje vrijedi da je broj nula i broj jedinica paran broj $L_3 = \{00, 11, 0011, 0101, 1001, 1100, \dots\}$, itd. Jezik L_3 zadan je nad binarnom abecedom $B = \{0, 1\}$ i nije konačan. Primjer jezika koji nije konačan je i skup svih mogućih nizova nad nekom abecedom Σ . Taj jezik označava se oznakom Σ^* . Jezik $L_4 = \Sigma^*$ nad abecedom $\Sigma = \{0, 1\}$ je skup svih mogućih nizova znamenaka 0 i 1: $L_4 = \Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111, 0000, 0001, \dots\}$.

U tablici 1.2 navedene su operacije nad jezicima.

Operacija	Oznaka	Definicija
Unija jezika L i N	$L \cup N$	$L \cup N = \{w \mid w \in L \vee w \in N\}$
Presjek jezika L i N	$L \cap N$	$L \cap N = \{w \mid w \in L \wedge w \in N\}$
Razlika jezika L i N	$L - N$	$L - N = \{w \mid w \in L \wedge w \notin N\}$
Nadovezivanje jezika L i N	$L N$	$L N = \{xy \mid x \in L \wedge y \in N\}$
Kartezijev produkt	$L \times N$	$L \times N = \{(x, y) \mid x \in L \wedge y \in N\}$
Partitivni skup	2^L	skup svih podskupova od L
Kleeneov operator L^*	L^*	$L^* = \bigcup_{i=0}^{\infty} L^i$
Kleeneov operator L^+	L^+	$L^+ = \bigcup_{i=1}^{\infty} L^i$
Komplement jezika	L^c	$L^c = \{w \mid w \notin L\}$

Tablica 1.2: Operacije nad jezicima

Za potrebe definicije Kleeneovih operatora nadovezivanje jezika označeno je potencijama ($L^2 = LL$ označava nadovezivanje jezika L sa samim sobom). Definira se da je:

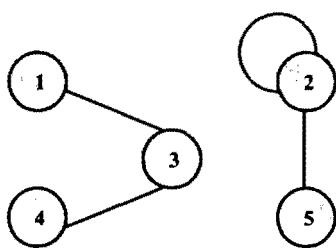
$$L^0 = \{\varepsilon\} \text{ i } L^i = L^{i-1}L.$$

Beskonačni skupovi

Kardinalni broj skupa jest broj elemenata skupa. Postoji li bijekcija između elemenata dvaju skupova (preslikavanje jedan u jedan), ta dva skupa imaju jednak kardinalne brojeve. Imaju li dva skupa konačni broj elemenata, a jedan skup je pravi podskup drugog, onda oni imaju različite kardinalne brojeve. Ako su oba skupa beskonačna, a jedan je pravi podskup drugog, to ne znači nužno da imaju različite kardinalne brojeve. Na primjer, skup parnih brojeva je pravi podskup skupa prirodnih brojeva N . Međutim, ta dva skupa imaju jednak kardinalne brojeve jer postoji bijekcija $f(2i) = i$, gdje je i prirodni broj.

Nemaju svi beskonačni skupovi jednak kardinalne brojeve. Za sve skupove za koje postoji bijekcija na skup prirodnih brojeva kažemo da su *prebrojivo beskonačni*. Skup cijelih brojeva Z (skup prirodnih brojeva uvećan za 0 i negativne cijele brojeve) i skup racionalnih brojeva Q (skup svih razlomaka gdje je brojnik cijeli broj, a nazivnik prirodni broj) su primjeri prebrojivo beskonačnih skupova.

Primjer beskonačnog skupa koji nema kardinalni broj jednak kardinalnom broju skupa prirodnih brojeva je skup realnih brojeva. Skup realnih brojeva je *neprebrojivo beskonačan* jer se ne može pronaći bijektivno preslikavanje između elemenata skupa prirodnih brojeva i skupa realnih brojeva.



Slika 1.2: Primjer grafa

Graf

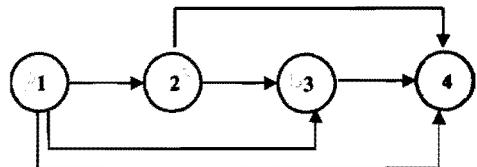
Graf $G = (V, E)$ sastoji se od konačnog skupa čvorova V i skupa parova čvorova E . Parovi čvorova su grane grafa. Slika 1.2 prikazuje primjer grafa s čvorovima $V = \{1, 2, 3, 4, 5\}$ i granama $G = \{(1, 3), (3, 4), (2, 5)\}$.

Put grafa jest niz čvorova $v_1, v_2, v_3, \dots, v_k$ ($k \geq 1$) za koji vrijedi da je (v_i, v_{i+1}) grana grafa, i to za bilo koji $i, 1 \leq i < k$. *Duljina puta* je $k - 1$. Primjer puta grafa na slici 1.2 je: 1, 3, 4. Primjer puta je i: 2, 2. Put je zatvoren ako je $v_1 = v_k$.

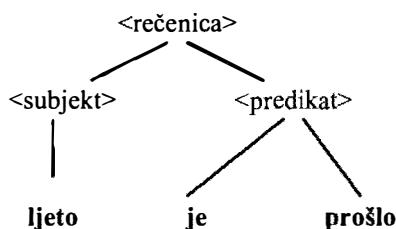
Usmjereni graf

Grane *usmjerenog grafa* su uređeni parovi čvorova koje nazivamo *usmjerenim granama*. Usmjerena grana od čvora v do čvora w označava se izrazom $v \rightarrow w$. Slika 1.3 prikazuje primjer usmjerenog grafa.

Put usmjerenog grafa jest niz čvorova $v_1, v_2, v_3, \dots, v_k$ ($k \geq 1$) za koji vrijedi da je $v_i \rightarrow v_{i+1}$ usmjereni grana grafa, i to za bilo koji $i, 1 \leq i < k$. Kaže se da put vodi od čvora v_1 do čvora v_k . Na slici 1.3 primjer puta je niz $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$. Ako su $v \rightarrow w$ i $w \rightarrow z$ usmjereni grane, onda su čvorovi v i w prethodnici čvora z , a w i z su sljedbenici čvora v . Čvor v je *neposredni prethodnik* čvora w , a w je *neposredni sljedbenik* čvora v .



Slika 1.3: Primjer usmjerenog grafa

Stablo

Slika 1.4: Primjer stabla

Stablo je usmjereni graf sa sljedećim svojstvima:

- 1) Postoji jedan čvor, nazvan *korijen* stabla, koji nema prethodnika i od njega postoji put do svih ostalih čvorova.
- 2) Bilo koji čvor, osim korijena stabla, ima točno jednog neposrednog prethodnika.

Stablo se crta tako da je korijen stabla na vrhu grafa, a usmjereni grane pokazuju prema dolje. Sljedbenici čvora poredaju se s lijeva na desno. Primjer stabla je prikazan na slici 1.4. Stablo prikazuje raščlambu rečenice "Ljeto je prošlo". Čvorovi su riječi ili dijelovi rečenice.

U nazivlju stabla neposredni prethodnik je *roditelj*, neposredni sljedbenik je *dijete*, prethodnici su *preci*, a sljedbenici su *potomci*. Čvor bez djece je *list*, a svi ostali čvorovi su *unutrašnji*. U primjeru danom na slici 1.4 čvor "<subjekt>" je roditelj čvora "ljeto", a čvor "ljeto" je dijete čvora "<subjekt>". Čvorovi "<subjekt>" i "<rečenica>" su preci čvora "ljeto", dok su čvorovi "<subjekt>" i "ljeto" potomci čvora "<rečenica>". Čvorovi "ljeto", "je" i "prošlo" su listovi dok su svi ostali čvorovi unutrašnji.

Dokaz tvrdnje matematičkom indukcijom

Dokazivanje tvrdnje $P(n)$ matematičkom indukcijom izvodi se u dva koraka. U prvom koraku dokaže se baza, tj. dokazuje se da je tvrdnja istinita za neki konačni broj n_0 . U većini slučajeva $n_0=0$:

$$a) P(n_0)$$

a zatim se na temelju pretpostavke da tvrdnja vrijedi za $P(n-1)$, što je induktivna hipoteza, dokaže da vrijedi tvrdnja:

$$b) P(n), \text{ za } n_0 \geq 1.$$

Primjer 1.1. Neka je $P(n)$ sljedeća tvrdnja:

$$\sum_{i=0}^n i^2 = \frac{n(n+1)(2n+1)}{6}$$

Uvjet (a) za $n_0=0$ može se lako dokazati, budući da su lijeva i desna strana jednake 0. Pretpostavi se da je tvrdnja $P(n-1)$ istinita, tj. da je:

$$\sum_{i=0}^{n-1} i^2 = \frac{(n-1)n(2n-1)}{6}$$

Gornji izraz uvrsti se u:

$$\sum_{i=0}^n i^2 = \sum_{i=0}^{n-1} i^2 + n^2$$

i dobije se:

$$\sum_{i=0}^n i^2 = \frac{(n-1)n(2n-1)}{6} + n^2 = \frac{n(n+1)(2n+1)}{6}$$

Relacije

Binarna *relacija* je skup parova elemenata skupova. Prvi element para je u skupu *domene* relacije, a drugi element para je u skupu *kodomene* relacije. U većini slučajeva domena i kodomena su isti skup S . Ako je par (a, b) element relacije R , onda se to kratko zapiše aRb .

Relacija R nad skupom S ima svojstvo:

- 1) *refleksivnosti* ako za sve a u skupu S vrijedi aRa ;
- 2) *nerefleksivnosti* ako za niti jedan a u skupu S ne vrijedi aRa ;
- 3) *tranzitivnosti* ako za sve a, b i c u skupu S za koje vrijedi aRb i bRc proizlazi da vrijedi aRc ;
- 4) *simetričnosti* ako sve a i b u skupu S za koje vrijedi aRb proizlazi da vrijedi bRa ;
- 5) *asimetričnosti* ako za sve a i b u skupu S za koje vrijedi aRb proizlazi da ne vrijedi bRa .

Asimetrična relacija je ujedno i nerefleksivna relacija. Relacija koja je refleksivna, simetrična i tranzitivna naziva se *relacija ekvivalencije*.

Neka je \mathcal{P} skup svojstava relacije. \mathcal{P} -okruženje relacije R je najmanja relacija R' koja uključuje sve parove relacije R i koja ima sva svojstva iz \mathcal{P} . Na primjer, tranzitivno okruženje relacije R označava se znakom R^+ i gradi se na sljedeći način:

- 1) Ako je (a, b) u skupu R , onda je (a, b) u skupu R^+ ;
- 2) Ako je (a, b) u skupu R^+ i ako je (b, c) u skupu R , onda je (a, c) u skupu R^+ ;
- 3) Niti jedan drugi element nije u R^+ .

Skup R^+ uključuje skup R i ima svojstvo tranzitnosti.

Refleksivno i tranzitivno okruženje relacije R definira se na sljedeći način:

$$R^* = R^+ \cup \{(a, a) \mid a \text{ je element skupa } S\}.$$

Primjer 1.2. Ako je $R = \{(1, 2), (2, 2), (2, 3)\}$ relacija nad skupom $S = \{1, 2, 3\}$, onda je:

$$\begin{aligned} R^+ &= \{(1, 2), (2, 2), (2, 3), (1, 3)\}, \\ R^* &= \{(1, 2), (2, 2), (2, 3), (1, 3), (1, 1), (3, 3)\}. \end{aligned}$$

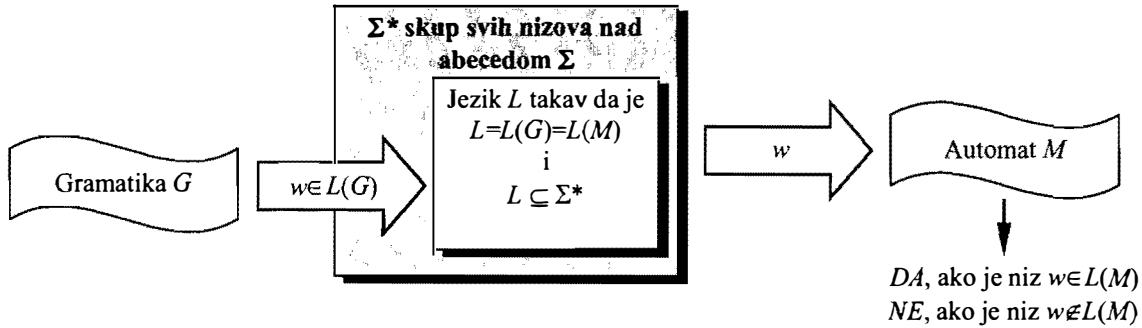
1.2 Primjer formalnog jezika, pripadajućeg automata i gramatike

Definiciji formalnog jezika pridružuju se dva matematička modela: automat i gramatika.

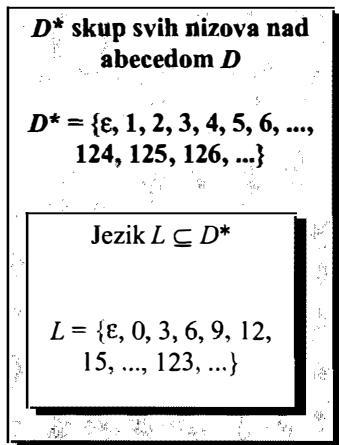
Automat jest model diskretnog matematičkog sustava koji čitanjem znak po znak odlučuje je li pročitani niz element zadanog jezika. Automat se sastoji od *stanja*. Prije čitanja prvog znaka automat se nalazi u *početnom stanju*. Čitanjem znakova niza automat mijenja stanja. Stanja se dijele na *prihvatljiva* i *neprihvatljiva*. Ako se nakon posljednjeg pročitanog znaka automat nalazi u jednom od prihvatljivih stanja, onda se niz prihvaca. Skup svih nizova koje prihvaca automat M je jezik za koji se kaže da ga prihvaca automat M . Jezik koji prihvaca automat M označava se s $L(M)$.

Gramatika jest model matematičkog sustava koji gradi, odnosno generira, nizove znakova. Gramatika se sastoji od skupa produkcija. Producije su pravila koja određuju na koji način se grade nizovi znakova. Producije se sastoje od *završnih* i *nezavršnih* znakova. Završni znakovi su ujedno i znakovi abecede jezika. Počev od *početnog nezavršnog* znaka primjenom produkcija nastoje se zamijeniti svi nezavršni znakovi završnim znakovima. Postupak zamjene se nastavlja sve dok se ne generira niz koji se sastoji isključivo od znakova abecede. Skup svih nizova koje generira gramatika G je jezik za koji se kaže da ga generira gramatika G . Jezik koji generira gramatika G označava se s $L(G)$.

Slika 1.5 prikazuje formalni jezik, gramatiku i automat. U primjeru 1.3 opisan je rad automata i gramatike za zadani jezik.



Slika 1.5: Formalan jezik, automat i gramatika

Slika 1.6: Formalan jezik L

Primjer 1.3. Neka je jezik L definiran nad skupom dekadskih znamenaka $D=\{0, 1, 2, 3, \dots, 9\}$. Niz se promatra kao cijelobrojna vrijednost. Niz je u jeziku L ako je njegova cijelobrojna vrijednost dijeljiva s tri $L=\{\epsilon, 0, 3, 6, 9, 12, 15, \dots, 123, \dots\}$. Slika 1.6 prikazuje skup svih nizova D^* nad abecedom D i jezik L .

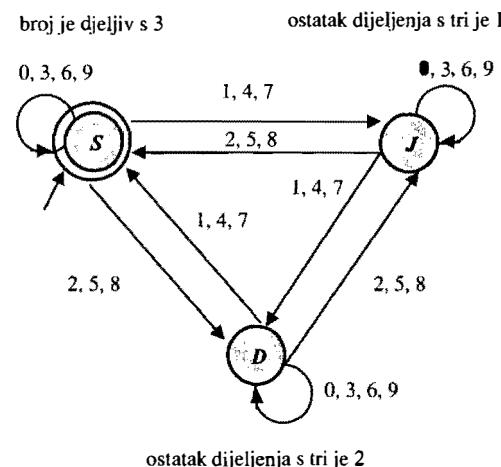
Postoje dva načina prikaza automata. Jedan od načina prikaza automata je pomoću usmjerenog grafa koji se naziva dijagram stanja. Čvorovi grafa su stanja automata, a usmjereni grane označavaju prijelaze. Početno stanje automata označeno je znakom S i usmjerenim granom bez izvođenog čvora. Prihvatljiva stanja su dvostruko okružena. Uz svaku granu označen je ulazni znak koji pokreće promjenu stanja automata. Prijede li automat u jedno od prihvatljivih stanja nakon što se pročita čitav niz, niz je element jezika.

Slika 1.7 prikazuje automat koji prihvata zadani jezik L . Automat ima tri stanja ovisno o tome je li ostatak dijeljenja s tri jednak 0 (stanje S), 1 (stanje J) ili 2 (stanje D). Početno stanje automata je S . Početno stanje S je ujedno i prihvatljivo stanje.

Rad automata može se pokazati na nizu $w=1404$. Niz w je djeljiv s 3 i prema tome $w \in L$. Na početku rada automat se nalazi u stanju S . Nakon pročitanog znaka 1 automat prelazi u stanje J , kao što se to vidi u dijagramu stanja koji je prikazan na slici 1.7. Automat je u stanju J i čita sljedeći znak 4. Automat prelazi u stanje D . Znak 0 ne mijenja stanje automata. Nakon pročitanog posljednjeg znaka 4 automat prelazi ponovno u stanje S , i budući da je to prihvatljivo stanje, automat prihvata niz w . Niz w je element jezika L .

Nakon pročitanog niza 400 automat završava u stanju J . Broj 400 nije djeljiv s 3, on nije element jezika L i automat ga ne prihvata (stanje J nije prihvatljivo stanje).

Drugi način prikaza automata je tablicom prijelaza. Reci tablice određuju stanja automata, a

Slika 1.7: Dijagram stanja automata jezika L

stupci označavaju ulazne znakove. Posljednji stupac ima oznaku 1 ili 0 ovisno o tome da li se stanje prihvata ili ne. Početno stanje stavlja se u prvi redak tablice. Automat koji prepozna zadani jezik L , a prikazan je dijagramom stanja na slici 1.7, prikazan je tablicom prijelaza 1.3. Elementi tablice određuju novo stanje. Na primjer, ako je automat u stanju J (stanje J određuje redak tablice), a na ulazu se pojavi znak 4 (ulazni znak 4 određuje stupac tablice), onda automat prelazi u stanje D .

Stanja	Ulagni znak										Oznaka prihvatljivosti
	0	1	2	3	4	5	6	7	8	9	
S	S	J	D	S	J	D	S	J	D	S	1
J	J	D	S	J	D	S	J	D	S	J	0
D	D	S	J	D	S	J	D	S	J	D	0

Tablica 1.3: Tablica prijelaza automata jezika L

Za zadani jezik L moguće je izgraditi gramatiku koja ga generira. Producije gramatike jezika L prikazuju se na sljedeći način:

$$\begin{aligned} <S> &\rightarrow 0 <S> \mid 3 <S> \mid 6 <S> \mid 9 <S> \mid 1 <J> \mid 4 <J> \mid 7 <J> \mid 2 <D> \mid 5 <D> \mid 8 <D> \\ <S> &\rightarrow \epsilon \\ <J> &\rightarrow 0 <J> \mid 3 <J> \mid 6 <J> \mid 9 <J> \mid 1 <D> \mid 4 <D> \mid 7 <D> \mid 2 <S> \mid 5 <S> \mid 8 <S> \\ <D> &\rightarrow 0 <D> \mid 3 <D> \mid 6 <D> \mid 9 <D> \mid 1 <S> \mid 4 <S> \mid 7 <S> \mid 2 <J> \mid 5 <J> \mid 8 <J> \end{aligned}$$

U zagradama “ $<>$ ” su nezavršni znakovi gramatike. $<S>$ je početni nezavršni znak. Znak “ \rightarrow ” označava da je nezavršni znak na lijevoj strani moguće zamijeniti nizom znakova na desnoj strani. Lijeva i desna strana znaka “ \rightarrow ” označavaju se kao lijeva i desna strana produkcije. Oznaka “ $|$ ” odjeljuje desne strane produkcija koje imaju istu lijevu stranu, odnosno predstavlja “ili” operator. Na primjer, nezavršni znak $<S>$ zamjeni se jednim od nizova: $0 <S>$, $3 <S>$, $6 <S>$, itd. Zamjena nezavršnih znakova desnim stranama produkcije nastavlja se sve dok u nizu ima nezavršnih znakova.

Operatorom \Rightarrow označava proces generiranja desnog međuniza znakova iz lijevog međuniza znakova primjenom točno jedne produkcije gramatike. U lijevom međunizu podcrtata se onaj nezavršni znak gramatike koji je lijeva strana produkcije koja se primjenjuje, a u desnom međunizu podcrtaju su znakovi koji su desna strana produkcije.

Prikazana gramatika generira nizove koji su elementi prethodno zadanog jezika L . Na primjer, produkcija $<S> \rightarrow \epsilon$ generira prazni niz koji je element jezika L :

$$\underline{<S>} \Rightarrow \underline{\epsilon}.$$

Niz 0 generira se primjenom dvije produkcije. Prvo se primjeni produkcija $<S> \rightarrow 0 <S>$:

$$\underline{<S>} \Rightarrow \underline{0} \underline{<S>}.$$

a zatim se primjeni produkcija $<S> \rightarrow \epsilon$ na znak $<S>$. Na desnoj strani ostaje samo 0 (ϵ je prazni niz i ne sadrži niti jedan znak):

$$0 \underline{<S>} \Rightarrow 0 \underline{\epsilon} = 0.$$

Gramatika generira niz 1404 na sljedeći način:

$$\begin{array}{llll} \underline{<S>} & \Rightarrow & \underline{1} \underline{<J>} & \text{(primjena produkcije } <S> \rightarrow 1 <J>) \\ \underline{1} \underline{<J>} & \Rightarrow & \underline{1} \underline{4} \underline{<D>} & \text{(primjena produkcije } <J> \rightarrow 4 <D>) \\ \underline{1} \underline{4} \underline{<D>} & \Rightarrow & \underline{1} \underline{4} \underline{0} \underline{<D>} & \text{(primjena produkcije } <D> \rightarrow 0 <D>) \\ \underline{1} \underline{4} \underline{0} \underline{<D>} & \Rightarrow & \underline{1} \underline{4} \underline{0} \underline{4} \underline{<S>} & \text{(primjena produkcije } <D> \rightarrow 4 <S>) \\ \underline{1} \underline{4} \underline{0} \underline{4} \underline{<S>} & \Rightarrow & \underline{1} \underline{4} \underline{0} \underline{4} \underline{\epsilon} = 1404. & \text{(primjena produkcije } <S> \rightarrow \epsilon) \end{array}$$

Uobičajeno je da se proces generiranja niza zapiše na sljedeći način:

$$<S> \Rightarrow 1 <J> \Rightarrow 14 <D> \Rightarrow 140 <D> \Rightarrow 1404 <S> \Rightarrow 1404.$$

2 REGULARNI JEZICI

Definicija regularnih jezika zasniva se na konačnom automatu: neki jezik jest regularan ako i samo ako postoji konačni automat koji ga prihvata. Time je definirana istovjetnost konačnih automata i regularnih jezika: za bilo koji regularni jezik moguće je izgraditi konačni automat koji ga prihvata, i obrnuto, bilo koji konačni automat prihvata jedan od regularnih jezika.

Regularni jezici su najjednostavniji jezici u smislu da je za njihovo prihvaćanje moguće izgraditi najjednostavniji automat: konačan automat. U odjeljku 2.1 prikazane su različite inačice konačnih automata. U odjeljku 2.2 definirani su regularni izrazi. Regularni izrazi omogućuju jednostavni način opisa regularnih jezika. Regularni jezici dobili su naziv upravo po regularnim izrazima. U istom odjeljku pokazana je istovjetnost regularnih izraza i konačnih automata. Odjeljak 2.3 opisuje svojstva regularnih jezika. Regularna gramatika opisana je u odjelu 2.4.

2.1 Konačni automati

Osnovni i najjednostavniji model konačnog automata jest deterministički konačni automat. Deterministički konačni automat je opisan u odjeljku 2.1.1. Postupci minimizacije konačnih automata objašnjeni su u odjeljku 2.1.2. Razne inačice konačnih automata i dokaz njihove istovjetnosti opisane su u odjelicima 2.1.3 i 2.1.4. U odjeljku 2.1.5 binarna funkcija izlaza proširena je općom funkcijom koja omogućuje široku primjenu konačnih automata.

2.1.1 Deterministički konačni automat (DKA)

Rad mnogih tehničkih i netehničkih sustava moguće je opisati konačnim automatima. Teorija konačnih automata ima veliku ulogu u procesu gradnje takvih sustava. Primjeri konačnih automata mogu se naći u digitalnoj elektronici i leksičkoj analizi skoro svih jezičnih procesora. Pojedini dijelovi računalnih sustava, pa i čitavo računalo, može se prikazati kao jedan složeni konačni automat. Konačni automati našli su široku primjenu i u modeliranju netehničkih sustava u medicini, biologiji, psihologiji, itd.

Deterministički konačni automat (DKA) sastoji se od konačnog skupa stanja i funkcije prijelaza. Funkcija prijelaza jednoznačno je određena znakom na ulazu i stanjem u kojem se nalazi automat. Za

pojedini znak i stanje postoji samo jedan prijelaz. Jedno od stanja, obično označeno znakom q_0 , jest početno stanje. Konačni automat započinje rad u početnom stanju. Prijede li konačni automat nakon pročitanih svih znakova niza x iz početnog stanja u jedno od prihvatljivih stanja, niz x se prihvata. Deterministički konačni automat prikazuje se dijagramom stanja ili tablicom prijelaza. U odjeljku 1.2 dan je primjer dijagrama stanja i tablice prijelaza za prihvatanje nizova dekadskih znamenaka čija je cijelobrojna vrijednost djeljiva s tri.

Deterministički konačni automat formalno se zadaje kao uređena petorka:

$$dka = (Q, \Sigma, \delta, q_0, F)$$

gdje je:

- | | |
|-----------------|--|
| Q | - konačan skup stanja; |
| Σ | - konačan skup ulaznih znakova; |
| δ | - funkcija prijelaza $Q \times \Sigma \rightarrow Q$; |
| $q_0 \in Q$ | - početno stanje; |
| $F \subseteq Q$ | - skup prihvatljivih stanja. |

U primjeru 1.3 čvorovi dijagrama stanja čine konačan skup stanja Q , brojevi od 0 do 9 čine skup ulaznih znakova Σ , čvor označen znakom S jest početno stanje q_0 , čvorovi dvostruko okruženi (odnosno, stanja čiji su reci označeni znakom 1 u tablici prijelaza) čine skup F , a funkcija prijelaza dana je usmjerenim granama grafa (odnosno, sadržaj tablice prijelaza određuje funkciju prijelaza).

Funkcija prijelaza jednoznačno određuje prijelaz u iduće stanje, što se zapiše na sljedeći način:

$$\text{novo_stanje} = \delta(\text{staro_stanje}, \text{ulazni_znak}).$$

Za bilo koji par *starog stanja* i *ulaznog znaka* jednoznačno je određeno u koje stanje prelazi DKA.

Definira se funkcija $\hat{\delta} : Q \times \Sigma^* \rightarrow Q$. Oznaka Σ^* označava skup svih mogućih nizova ulaznih znakova, uključujući i prazni niz. U dalnjem tekstu w, x, y i z označavaju nizove ulaznih znakova (w, x, y i z su elementi skupa Σ^*). Ulazni znakovi označeni su slovima a, b , ili brojkama (slova a, b i brojke su elementi skupa Σ). Stanja se označavaju slovima p i q (p i q su elementi skupa Q). Funkcija $\hat{\delta}(q, w)$ određuje stanje DKA nakon što je u stanju q pročitan niz ulaznih znakova $w \in \Sigma^*$. Funkcija $\hat{\delta}$ definira se na sljedeći način:

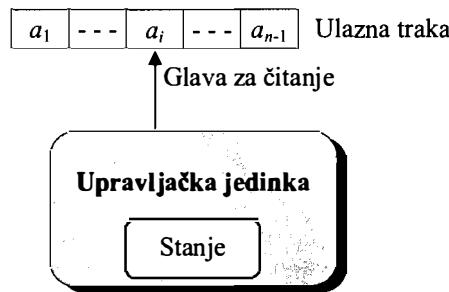
- (1) $\hat{\delta}(q, \epsilon) = q$, gdje je ϵ prazni niz;
- (2) za bilo koji niz ulaznih znakova w i za bilo koji ulazni znak a vrijedi:

$$\hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w), a), \text{ gdje je } w \in \Sigma^* \text{ i } a \in \Sigma.$$

U (1) zadano je da DKA ne može promijeniti stanje, a da ne pročita barem jedan ulazni znak. U (2) zadano je na koji način se računa stanje u koje prelazi DKA čitanjem znakova niza wa . Nakon što se pročita niz w , određuje se stanje $p = \hat{\delta}(q, w)$, a zatim se računa $\delta(p, a)$.

Uvrštavanjem u (2) $w=\epsilon$ dobije se $\hat{\delta}(q, a) = \delta(\hat{\delta}(q, \epsilon), a) = \delta(q, a)$. Prepostavi li se da se ne razlikuju ulazni znak $a \in \Sigma$ i niz $a \in \Sigma^*$ duljine $|a|=1$, funkcije imaju jednaku vrijednost za sve one argumente za koje su obje funkcije definirane. Zato se u dalnjem tekstu koristi samo jedna oznaka δ za obje funkcije.

Pojmovi prihvatanja i neprihvatanja ulaznog niza mogu se formalno definirati. Deterministički konačni automat $dka = (Q, \Sigma, \delta, q_0, F)$ prihvata niz x ako je $\delta(q_0, x) = p$ za neki $p \in F$. DKA prihvata skup $L(dka) = \{x \mid \delta(q_0, x) \in F\}$ koji je podskup skupa svih mogućih nizova ($L(dka) \subseteq \Sigma^*$). Za sve ostale nizove koji nisu u skupu $L(dka)$ kaže se da ih DKA ne prihvata.



Slika 2.1: Model konačnog automata

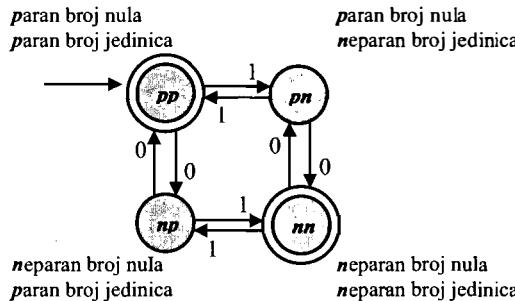
Rad konačnog automata opisuje se modelom prikazanim na slici 2.1. *Upravljačka jedinka*, koja se nalazi u jednom od konačnog broja stanja, čita niz znakova abecede Σ napisanih na ulaznoj traci. *Glava za čitanje* konačnog automata samo čita znakove trake, ali ne može pisati. Glava se miče u desno. Upravljačka jedinka pamti samo stanje u kojem se nalazi. Jednim pomakom glave upravljačka jedinka u stanju q pročita ulazni znak a , promjeni stanje u novo stanje $\delta(q, a)$ i pomakne glavu za čitanje za jedno mjesto u desno. Ako je novo stanje prihvatljivo, onda konačni automat prihvaca do tada pročitani podniz znakova, ali bez znaka na kojem je pozicionirana glava za čitanje. Pomakom glave izvan krajnje desne pozicije, prihvaca se ili ne prihvaca cijeli niz ispisani na traci.

Primjer determinističkog konačnog automata

Neka je zadan sljedeći dka: $Q=\{pp, np, pn, nn\}$, $\Sigma=\{0, 1\}$, $q_0=pp$, $F=\{pp, nn\}$ i funkcije prijelaza:

$$\begin{array}{llll} \delta(pp, 1) = pn & \delta(pn, 1) = pp & \delta(np, 1) = nn & \delta(nn, 1) = np \\ \delta(pp, 0) = np & \delta(pn, 0) = nn & \delta(np, 0) = pp & \delta(nn, 0) = pn \end{array}$$

Konačni automat prikazan je dijagramom stanja na slici 2.2 i tablicom 2.1. U tabličnom prikazu skup ulaznih znakova proširen je oznakom kraja ulaznog niza \perp . Oznaka kraja ulaznog niza uvodi se za potrebe zaustavljanja procesa čitanja znakova. Konačni automat sa proširem skupom ulaznih znakova naziva se *procesni konačni automat*. Konačni automat zadan na slici 2.2 prihvaca sve nizove koji imaju parni broj jedinica i parni broj nula, ili neparni broj jedinica i neparni broj nula.



Slika 2.2: Dijagram stanja DKA

Stanja	Ulagni znak		Oznaka prihvatljivosti
	0	1	
pp	pp	pn	1
pn	nn	pp	0
np	pp	nn	0
nn	pn	np	1

Tablica 2.1: Tablica prijelaza DKA

Postupak prihvaćanja niza konačnim automatom opisan je na primjeru sljedećih nizova:

Niz	Formalni postupak prihvaćanja niza	Da li se niz prihvaca?
ϵ	$\delta(pp, \epsilon) = pp$, $pp \in F$	DA
1	$\delta(pp, 1) = pn$, $pn \notin F$	NE

11	$\delta(pp, 11) =$ $\delta(\delta(pp, 1), 1) =$ $\delta(pn, 1) = pp,$ $pp \in F$	DA
1011	$\delta(pp, 1011) =$ $\delta(\delta(pp, 101), 1) =$ $\delta(\delta(\delta(pp, 10), 1), 1) =$ $\delta(\delta(\delta(pp, 1), 0), 1), 1) =$ $\delta(\delta(\delta(pn, 0), 1), 1) =$ $\delta(\delta(nn, 1), 1) =$ $\delta(np, 1) = nn,$ $nn \in F$	DA

Opisani formalni postupak prihvaćanja niza prilično je nepraktičan i nepregledan. Dijagram stanja ili tablica prijelaza pojednostavljuje postupak prihvaćanja niza. Prateći dijagram stanja ili tablicu prijelaza, te uzimajući slijedno jedan po jedan znak niza s lijeva na desno, mogu se odrediti prijelazi u nova stanja. Prijelaz u novo stanje prikazuje se na sljedeći način:

$$\begin{array}{ccc} & \text{ulazni znak} & \\ \text{prethodno stanje} & \xrightarrow{\quad\quad\quad} & \text{novo stanje} \end{array}$$

Tablica prijelaza i dijagram stanja olakšavaju postupak prihvaćanja niza:

Niz	Slijed prijelaza	Da li se niz prihvaca?
ϵ	pp	$pp \in F$ DA
1	1	$pn \notin F$ NE
11	$pp \rightarrow pn \rightarrow pp$	$pp \in F$ DA
1011	$pp \rightarrow pn \rightarrow nn \rightarrow np \rightarrow nn$	$nn \in F$ DA

DKA dijeli skup svih mogućih nizova ulaznih znakova u dva podskupa. Prijelazom u jedno od prihvatljivih stanja, niz je u skupu za koji se kaže da ga DKA prihvata. Ne prijeđe li DKA u jedno od prihvatljivih stanja nakon posljednjeg pročitanog ulaznog znaka, niz je u skupu za koji se kaže da se ne prihvata.

Zadani DKA prihvata sve nizove koji imaju parni broj nula i parni broj jedinica, ili neparni broj nula i neparni broj jedinica. DKA razlikuje prihvata li niz zato što je broj nula i jedinica paran, ili ga prihvata zato što je broj jedinica i nula neparan. Nadalje, DKA razlikuje da li se niz ne prihvata zato što u njemu ima parni broj nula i neparni broj jedinica, ili se niz ne prihvata zbog neparnog broja nula i parnog broja jedinica. Na slici 2.2 uz stanja su dani opisi da li pročitani niz ulaznih znakova ima parni ili neparni broj jedinica ili nula.

Primjer 2.1. Jezik $L(M_1) = \{\}$ nad abecedom $\Sigma = \{0, 1\}$ prihvata DKA M_1 :

$$M_1 = (\{q\}, \{0, 1\}, \{\delta(q, 0) = q, \delta(q, 1) = q\}, q, \{\}).$$

Jezik $L(M_2) = \{\epsilon\}$ nad abecedom $\Sigma = \{0, 1\}$ prihvata DKA M_2 :

$$M_2 = (\{q, p\}, \{0, 1\}, \{\delta(q, 0) = p, \delta(q, 1) = p, \delta(p, 0) = p, \delta(p, 1) = p\}, q, \{q\}).$$

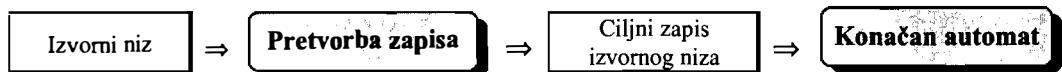
Jezik $L(M_3) = \Sigma^*$ nad abecedom $\Sigma = \{0, 1\}$ prihvaca DKA M_3 :

$$M_3 = (\{q\}, \{0, 1\}, \{\delta(q, 0) = q, \delta(q, 1) = q\}, q, \{q\}).$$

Programsko ostvarenje konačnog automata

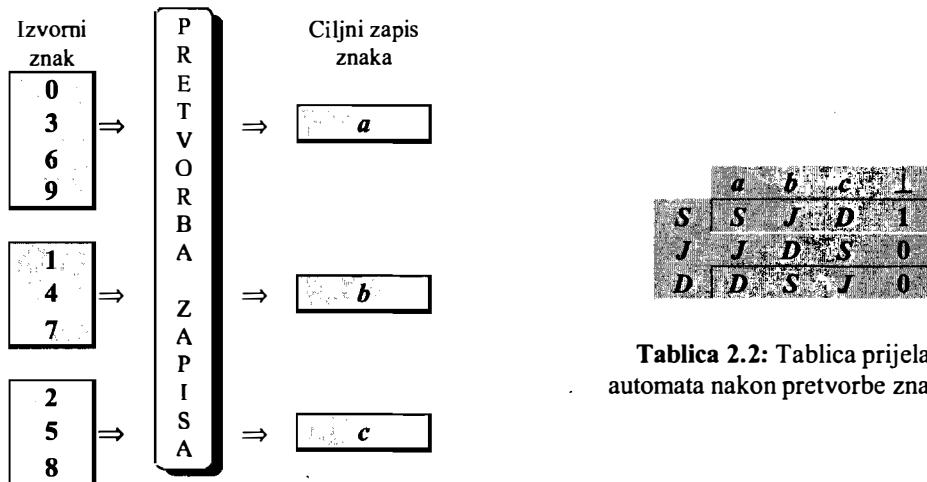
Konačni automat jest matematički model koji je moguće ostvariti programskim jezikom. Za potrebe učinkovitog programskog ostvarenja posebice se razmatraju načini zapisa ulaznih znakova, stanja i funkcije prijelaza.

Pažljivim izborom ciljnog *zapis* pojedinih *znakova* pojednostavi se funkcija prijelaza. Zapis ulaznog znaka pretvorci se u učinkovitiji ciljni zapis za potrebe programskog ostvarenja konačnog automata. Na slici 2.3 prikazana je međusobna povezanost pretvorbe zapisa znakova i konačnog automata.



Slika 2.3: Jednostavni model pretvorbe znakova

Slika 2.4 prikazuje pretvorbu dekadskih znakova $D = \{0, 1, 2, 3, \dots, 9\}$ za potrebe konačnog automata zadano u primjeru 1.3. Konačni automat za znakove 0, 3, 6 i 9 ima iste prijelaze i zato se ta cijela grupa zamjeni jedinstvenim znakom a . Druge dvije grupe znakova zamjene se jedinstvenim znakovima b i c . Tablica 2.2 prikazuje konačni automat definiran nad novim skupom znakova $D' = \{a, b, c\}$.



Slika 2.4: Primjer pretvorbe znakova

Predloženi ciljni zapis omogućuje učinkovito programskog ostvarenje konačnog automata. Tablica 2.2 jest značajno manja od tablice 2.1.

Postoje dva osnovna načina *zapis stanja*. Prvi način je da se stanje konačnog automata zapiše u varijablu. Takav način zapisa stanja naziva se *izravan način*. Ako se stanje određuje na temelju dijela programa koji se izvodi, onda kažemo da su stanja zapisana *posrednim načinom*. Na slici 2.5 prikazana su oba načina zapisa stanja konačnog automata koji je zadani na slici 2.2.

Tablica 2.2: Tablica prijelaza automata nakon pretvorbe znakova

Znak kraja niza \perp označava da je pročitan posljednji znak ulaznog niza. Nakon pročitane oznake kraja niza, ispiše se poruka o prihvaćanju ili neprihvaćanju niza. Program ispisuje i ostale dostupne podake o tome zašto se neki niz prihvaca ili ne.

```

-----  

tablica[PP, 0] = NP;  

tablica[PP, 1] = PN;  

tablica[PP,  $\perp$ ] = 1;  

tablica[NP, 0] = PP;  

tablica[NP, 1] = NN;  

tablica[NP,  $\perp$ ] = 0;  

tablica[PN, 0] = NN;  

tablica[PN, 1] = PP;  

tablica[PN,  $\perp$ ] = 0;  

tablica[NN, 0] = PN;  

tablica[NN, 1] = NP;  

tablica[NN,  $\perp$ ] = 1;  

stanje = PP;  

učitaj(znak);  

dok (znak !=  $\perp$ )
{
    stanje = tablica[stanje, znak];
    učitaj(znak);
}
ispisi(tablica[stanje,  $\perp$ ], stanje);
-----  

-----  

PP: učitaj(znak);
    ako (znak ==  $\perp$ )
    {
        ispiši("NIZ SE PRIHVAĆA, PARAN:1, PARAN:0");
    }
    ako (znak == 0)
    {
        skoči NP;
    }
    inače
    {
        skoči PN;
    }
    .

```

(a)

```

PN: učitaj(znak);
    ako (znak ==  $\perp$ )
    {
        ispiši("NIZ SE NE PRIHVAĆA, PARAN:1, NEPARAN:0");
    }
    ako (znak == 0)
    {
        skoči PP;
    }
    inače
    {
        skoči NN;
    }
    .

```

```

NN: učitaj(znak);
    ako (znak ==  $\perp$ )
    {
        ispiši("NIZ SE NE PRIHVAĆA, NEPARAN:1, PARAN:0");
    }
    ako (znak == 0)
    {
        skoči NN;
    }
    inače
    {
        skoči PN;
    }
    .

```

(b)

Slika 2.5: (a) Primjer izravnog načina zapisa stanja konačnog automata
(b) Primjer posrednog načina zapisa stanja konačnog automata

Funkcija prijelaza ostvaruje se na dva načina: vektorski i lista. *Vektorski pristup* uvodi za svako stanje konačnog automata jedan vektor. Vektor ima onoliko elemenata koliko ima različitih ulaznih znakova. U izvornom načinu zapisa elementi vektora su nova stanja, a u posrednom načinu zapisa elementi vektora određuju adresu potprograma ili mjesto od kojeg se nastavlja izvođenje programa. Vektorski pristup koristi naredbe viših programskih jezika case, switch ili izračunati goto.

Na slici 2.6 prikazana su oba pristupa za stanje *Bin* konačnog automata zadanog tablicom 2.3. Konačni automat provjerava da li je ulazni niz ispravno napisani binarni broj, oktalni broj ili dekadski broj. U ulaznom nizu binarni brojevi su označeni slovom *B*, oktalni brojevi slovom *O*, a dekadski brojevi slovom *D*. Stanje greške označava neispravnost ulaznog niza. Osnovna prednost vektorskog pristupa jest brzina računanja novog stanja. Ciljni kodovi znakova mogu se odrediti tako da pokazuju izravno na odgovarajući element vektora. Nedostatak vektorskog pristupa jest neučinkovito korištenje memorije. Bez obzira da li više različitih ulaznih znakova zahtijeva prijelaz u isto novo stanje, za sve njih je potrebno izgraditi zasebni element vektora. Svi vektori su jednake veličine i broj elemenata u svim vektorima jednak je broju različitih ulaznih znakova.

	<i>B</i>	<i>O</i>	<i>D</i>	0	1	2	3	4	5	6	7	8	9	1
<i>Početak</i>	<i>Bin</i>	<i>Okt</i>	<i>Dek</i>	<i>Gr</i>	0									
<i>Bin</i>	<i>Gr</i>	<i>Gr</i>	<i>Gr</i>	<i>Bin</i>	<i>Bin</i>	<i>Gr</i>	1							
<i>Okt</i>	<i>Gr</i>	<i>Gr</i>	<i>Gr</i>	<i>Okt</i>	1									
<i>Dek</i>	<i>Gr</i>	<i>Gr</i>	<i>Gr</i>	<i>Dek</i>	1									
<i>Gr</i>	<i>Gr</i>	<i>Gr</i>	<i>Gr</i>	<i>Gr</i>	<i>Gr</i>	<i>Gr</i>	<i>Gr</i>	<i>Gr</i>	<i>Gr</i>	<i>Gr</i>	<i>Gr</i>	<i>Gr</i>	<i>Gr</i>	0

Tablica 2.3: Tablica prijelaza DKA

ulazni znak	<i>B</i>	<i>O</i>	<i>D</i>	0	1	2	3	4	5	6	7	8	9
vektor izravnog načina zapisa stanja	<i>Gr</i>	<i>Gr</i>	<i>Gr</i>	<i>Bin</i>	<i>Bin</i>	<i>Gr</i>							
vektor posrednog načina zapisa stanja	<i>pot_Gr</i>	<i>pot_Gr</i>	<i>pot_Gr</i>	<i>pot_Bin</i>	<i>pot_Bin</i>	<i>pot_Gr</i>							

Slika 2.6: Primjer vektorskog zapisa funkcije prijelaza za stanje *Bin* automata zadanog tablicom 2.3

Učinkovito korištenje memorije postiže se *listom*. Za pojedine vektore uvodi se lista koja se sastoji od parova ulaznih znakova i stanja u koje konačni automat prelazi za dani ulazni znak (odnosno, adresa potprograma ili mesta od kojeg se nastavlja izvođenje programa). U listi se odredi stanje (odnosno, potprogram ili dio programa) koje se najviše puta ponavlja. Parovi u kojima se nalazi određeno stanje izbacuje se iz liste, a na kraj liste doda se oznaka izabranog stanja (odnosno, potprograma ili dijela programa). Na taj način smanji se memorijski prostor potreban za spremanje pojedinih vektori. Međutim, proces računanja novog stanja jest duži i izvodi se u dva koraka:

- 1) U prvom koraku provjerava se da li u listi postoji zapis za traženi ulazni znak. Postoji li zapis, uzme se podatak o novom stanju (odnosno, izvede se potprogram ili zadani dio programa).
- 2) Ne postoji li zapis, novo stanje je stanje koje je navedeno posljednje u listi (odnosno, izvodi se potprogram naveden na kraju liste).

<i>Lista za izravni način zapisa stanja</i>	(<i>Znak_01, Bin</i>)	(<i>Svi_ostali, Gr</i>)
<i>Lista za posredni način zapisa stanja</i>	(<i>Znak_01, pot_Bin</i>)	(<i>Svi_ostali, pot_Gr</i>)

Slika 2.7: Primjer liste za potrebe zapisa funkcije prijelaza za stanje *Bin* automata zadanog tablicom 2.3

Na slici 2.7 dat je primjer liste za stanje *Bin* konačnog automata zadanog tablicom 2.3. U primjeru je korištena pretvorba znakova 0 i 1 u jedinstveni znak *Znak_01*.

Lista zauzima manji memorijski prostor od vektorskog zapisa, ali je zato vrijeme pretraživanja liste duže od vremena računanja indeksa vektorskog zapisa.

2.1.2 Minimizacija konačnog automata

Za bilo koji DKA moguće je izgraditi beskonačno mnogo drugih DKA koji prihvataju isti jezik. Učinkovito programsko ostvarenje zahtijeva gradnju DKA sa što manjim brojem stanja:

Za regularni jezik L moguće je izgraditi DKA M koji ima manji ili jednak broj stanja od bilo kojeg drugog DKA M' koji prihvata isti taj jezik L .

Istovjetnost stanja

Stanje p DKA $M=(Q, \Sigma, \delta, q_0, F)$ je istovjetno stanju p' DKA $M'=(Q', \Sigma, \delta', q_0', F')$ ako i samo ako DKA M u stanju p prihvata isti skup nizova kao i DKA M' u stanju p' . Za bilo koji niz w skupa Σ^* mora vrijediti $\delta(p, w) \in F \wedge \delta'(p', w) \in F'$ ili $\delta(p, w) \notin F \wedge \delta'(p', w) \notin F'$.

Relacija istovjetnosti ima svojstvo tranzitivnosti: ako je su stanja p i q istovjetna, te ako su stanja q i r istovjetna, onda su istovjetna i stanja p i r . U cilju smanjivanja broja stanja zadanog DKA, grupa istovjetnih stanja zamjeni se jedinstvenim stanjem sljedećim postupkom. Najprije se istovjetna stanja označe zajedničkim imenom. Sve oznake istovjetnih stanja u funkciji prijelaza δ označe se izabranim zajedničkim imenom. U skupu stanja Q ostavi se samo jedno od istovjetnih stanja, a sva ostala istovjetna stanja se izbace. Postupak je opisan na slici 2.8, gdje su stanja p_4 i p_5 istovjetna.

	c	d	
p_1	p_1	p_4	0
p_2	p_3	p_5	1
p_3	p_5	p_1	0
p_4	p_2	p_3	1
p_5	p_2	p_3	1

	c	d	
p_1	p_1	X	0
p_2	p_3	X	1
p_3	X	p_1	0
X	p_2	p_3	1
X	p_2	p_3	1

	c	d	
p_1	p_1	X	0
p_2	p_3	X	1
p_3	X	p_1	0
X	p_2	p_3	1
X	p_2	p_3	1

Slika 2.8: Pojednostavljenje DKA odbacivanjem jednog od istovjetnih stanja

Definicija istovjetnosti stanja proširuje se na definiciju istovjetnosti DKA:

DKA M i N su istovjetni ako i samo ako su istovjetna njihova početna stanja.

Ispitivanje istovjetnosti stanja p i q svodi se na ispitivanje dva uvjeta:

- 1) *Uvjet podudarnosti:* Stanja p i q moraju biti oba prihvatljiva ($p \in F \wedge q \in F$) ili neprihvatljiva ($p \notin F \wedge q \notin F$).
- 2) *Uvjet napredovanja:* Za bilo koji ulazni znak a mora vrijediti da su stanja $\delta(p, a)$ i $\delta(q, a)$ istovjetna.

Pronalaženje istovjetnih stanja

Pronalaženje istovjetnih stanja ponekad nije jednostavan zadatak kao u prethodnom primjeru na slici 2.8. U nastavku odjeljka data su tri algoritma za pronalaženje istovjetnih stanja.

Algoritam I. Algoritam se zasniva na ispitivanju gore navedena dva uvjeta. Uvjeti se ispituju za sve parove stanja DKA. Predloženi algoritam je neučinkovit, jer je potrebno ispitati sve parove stanja. Neka se ispita istovjetnost stanja p_0 i p_7 DKA datog na slici 2.10. Algoritam se izvodi u sljedećim koracima:

- 1) Tablica za ispitivanje istovjetnosti parova stanja gradi se tako da se za sve ulazne znakove izgradi zasebni stupac. Izaberu se dva stanja za koja se želi ispitati istovjetnost, te se taj par upiše u prvi redak.



- 2) Uvjet podudarnosti ispituje se za sve parove novih stanja koji su upisani u tablicu u prethodnom koraku. Ako par stanja nije podudaran, onda par stanja označen u prvom retku nije istovjetan i algoritam se zaustavlja. Ako su stanja podudarna, odrede se novi parovi stanja za sve ulazne znakove, te se ti parovi upišu u odgovarajuće stupce tablice. Na primjer, za par stanja (p, q) i ulazni znak a , u stupac ulaznog znaka a upiše se novi par stanja $(\delta(p, a), \delta(q, a))$.

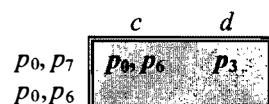
1. prolaz

p_0, p_7	c	d	$(p_0 \notin F \wedge p_7 \notin F)$	p_0, p_7	c	d	$\delta(p_0, c) = p_0, \delta(p_7, c) = p_6$	$\delta(p_0, d) = p_3, \delta(p_7, d) = p_3$
			\Rightarrow stanja p_0 i p_7 su podudarna					

- 3) Za novi par stanja u koraku (2) postoje tri mogućnosti:

Par stanja	Akcija
Dva ista stanja	Nema akcije
Dva različita stanja za koje postoji zapis u jednom od prethodnih redaka	Nema akcije
Dva različita stanja za koje ne postoji zapis u jednom od prethodnih redaka	Stvori novi redak u tablici i upiši u njega novi par stanja

1. prolaz



- 4) Ako se u koraku (3) ne pojavi niti jedan novi redak u tablici, onda je par stanja zapisan u prvom retku tablice istovjetan, kao i svi parovi stanja zapisani u ostalim redicama tablice. Ostali parovi stanja su istovjetni jer vrijedi uvjet napredovanja.

Postoji li novi redak u tablici, algoritam se nastavlja od koraka (2).

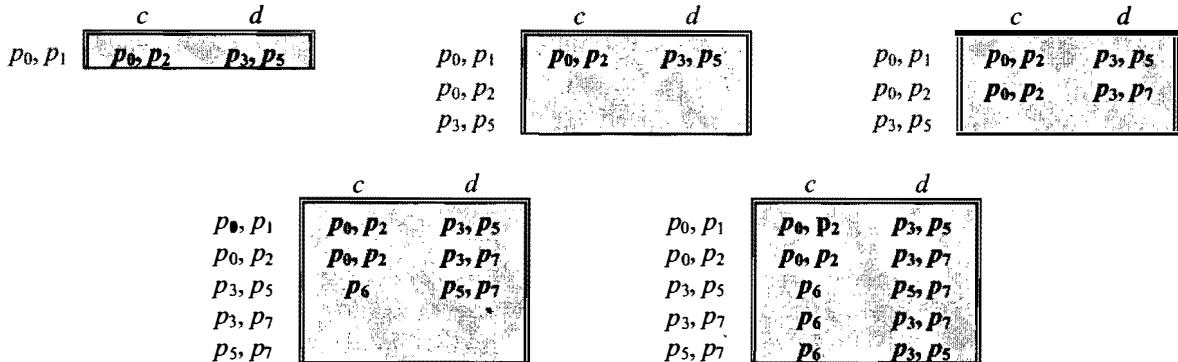
U drugom koraku drugog prolaza algoritam ispituje podudarnost stanja p_0 i p_6 . Budući da stanja p_0 i p_6 nisu podudarna, stanja p_0 i p_7 nisu istovjetna i algoritam se zaustavlja.

2. prolaz

p_0, p_7	c	d	$(p_0 \notin F \wedge p_6 \in F) \Rightarrow$ stanja p_0 i p_6 nisu podudarna \Rightarrow stanja p_0 i p_7 nisu istovjetna \Rightarrow ALGORITAM SE ZAUSTAVLJA
p_0, p_6	p_0, p_6	p_3	

Na slici 2.9 dokazana je istovjetnost sljedećih parova stanja: $(p_0, p_1), (p_0, p_2), (p_3, p_5), (p_3, p_7)$ i (p_5, p_7) . Budući da za istovjetnost vrijedi tranzitivnost, na temelju istovjetnosti para stanja (p_0, p_1) i para stanja (p_0, p_2) , par stanja (p_1, p_2) također je istovjetan. DKA prikazan na slici 2.10 pojednostavi se na sljedeći način:

stanja p_0, p_1 i p_2 zamijene se stanjem A , a stanja p_3, p_5 i p_7 zamijene se stanjem B . Pojednostavljeni DKA prikazan je na slici 2.11.



Slika 2.9: Postupak dokazivanja istovjetnosti stanja p_0 i p_1 DKA datog na slici 2.10

	<i>c</i>	<i>d</i>	
p_0	p_0	p_3	0
p_1	p_2	p_5	0
p_2	p_2	p_7	0
p_3	p_6	p_7	0
p_4	p_1	p_6	1
p_5	p_6	p_5	0
p_6	p_6	p_3	1
p_7	p_6	p_3	0

	<i>c</i>	<i>d</i>	
A	A	B	0
B	p_6	B	0
p_4	A	p_6	1
p_6	p_6	B	1

Slika 2.11: DKA je istovjetan s DKA sa slike 2.10, ali bez istovjetnih stanja

Slika 2.10: DKA s istovjetnim stanjima

Algoritam 2. Algoritam dijeli skup stanja DKA $M=(Q, \Sigma, \delta, q_0, F)$ u podskupove na temelju uvjeta podudarnosti. Algoritam se izvodi u tri koraka:

- 1) Skup stanja podijeli se u dvije grupe. U jednoj grupi su sva stanja koja su prihvatljiva (sva stanja $p \in F$), a u drugoj grupi su sva stanja koja nisu prihvatljiva (sva stanja $q \notin F$). Neka se znakom Π označi podjela skupa stanja u dvije grupe.
- 2) Primjeni se algoritam na slici 2.12 na podjelu Π .
- 3) Ako je podjela na grupe ostala ista, tj. ako je $\Pi_{nova}=\Pi$, onda se algoritam zaustavlja. Stanja u istim grupama su istovjetna. Ako je $\Pi_{nova} \neq \Pi$, onda se algoritam nastavlja od koraka (2) sa $\Pi:=\Pi_{nova}$.

za (sve grupe stanja G_j u podjeli Π)

podijeli grupu stanja G_j na podskupove tako da su dva stanja p i q iz grupe G_j u istom podskupu ako i samo ako za svaki ulazni znak a vrijedi:

```
( $\delta(p, a) \in G_i \wedge \delta(q, a) \in G_i$ ), gdje je  $G_i$  jedna od grupe stanja podjele  $\Pi$ ;
// za različite ulazne znakove  $a$  grupe  $G_i$  mogu biti različite
označi novu podjelu skupa stanja  $\Pi_{nova}$ ;
```

Slika 2.12: Algoritam računanja nove podjele Π_{nova}

Za opis rada algoritma koristi se DKA zadan na slici 2.13. Različite podjele Π označene su indeksima. Prva podjela skupa stanja je:

$\Pi_1 : (G_{11}=\{p_1, p_2, p_3, p_4\}, G_{12}=\{p_5, p_6, p_7\})$, jer je:

$$(p_1 \notin F \wedge p_2 \notin F \wedge p_3 \notin F \wedge p_4 \notin F) \text{ i } (p_5 \in F \wedge p_6 \in F \wedge p_7 \in F)$$

U prvom prolazu algoritam daje sljedeću podjelu:

$\Pi_2 : (G_{21}=\{p_1, p_2\}, G_{22}=\{p_3, p_4\}, G_{23}=\{p_5\}, G_{24}=\{p_6, p_7\})$, jer je:

$$\begin{array}{l} \delta(p_1, c) \in G_{12} \\ \delta(p_2, c) \in G_{12} \end{array}$$

$$\begin{array}{l} \delta(p_3, c) \in G_{11} \\ \delta(p_4, c) \in G_{11} \end{array}$$

$$\begin{array}{l} \delta(p_5, c) \in G_{12} \\ \delta(p_6, c) \in G_{11} \end{array}$$

$$\begin{array}{l} \delta(p_7, c) \in G_{11} \\ \delta(p_5, d) \in G_{11} \end{array}$$

$$\begin{array}{l} \delta(p_1, d) \in G_{11} \\ \delta(p_2, d) \in G_{11} \end{array}$$

$$\begin{array}{l} \delta(p_3, d) \in G_{12} \\ \delta(p_4, d) \in G_{12} \end{array}$$

$$\begin{array}{l} \delta(p_5, d) \in G_{11} \\ \delta(p_6, d) \in G_{11} \end{array}$$

$$\begin{array}{l} \delta(p_7, d) \in G_{11} \\ \delta(p_7, d) \in G_{11} \end{array}$$

U drugom prolazu algoritam daje podjelu:

$\Pi_3 : (G_{31}=\{p_1, p_2\}, G_{32}=\{p_3\}, G_{33}=\{p_4\}, G_{34}=\{p_5\}, G_{35}=\{p_6, p_7\})$, jer je:

$$\begin{array}{l} \delta(p_1, c) \in G_{24} \\ \delta(p_2, c) \in G_{24} \end{array}$$

$$\delta(p_3, c) \in G_{21}$$

$$\delta(p_4, c) \in G_{22}$$

$$\delta(p_5, c) \in G_{24}$$

$$\begin{array}{l} \delta(p_6, c) \in G_{22} \\ \delta(p_7, c) \in G_{22} \end{array}$$

$$\begin{array}{l} \delta(p_1, d) \in G_{22} \\ \delta(p_2, d) \in G_{22} \end{array}$$

$$\delta(p_3, d) \in G_{23}$$

$$\delta(p_4, d) \in G_{24}$$

$$\delta(p_5, d) \in G_{22}$$

$$\begin{array}{l} \delta(p_6, d) \in G_{21} \\ \delta(p_7, d) \in G_{21} \end{array}$$

U trećem prolazu algoritam daje podjelu:

$\Pi_4 : (G_{41}=\{p_1, p_2\}, G_{42}=\{p_3\}, G_{43}=\{p_4\}, G_{44}=\{p_5\}, G_{44}=\{p_6, p_7\})$, jer je:

$$\begin{array}{l} \delta(p_1, c) \in G_{35} \\ \delta(p_2, c) \in G_{35} \end{array}$$

$$\delta(p_3, c) \in G_{31}$$

$$\delta(p_4, c) \in G_{33}$$

$$\delta(p_5, c) \in G_{35}$$

$$\begin{array}{l} \delta(p_6, c) \in G_{33} \\ \delta(p_7, c) \in G_{33} \end{array}$$

$$\begin{array}{l} \delta(p_1, d) \in G_{32} \\ \delta(p_2, d) \in G_{32} \end{array}$$

$$\delta(p_3, d) \in G_{34}$$

$$\delta(p_4, d) \in G_{35}$$

$$\delta(p_5, d) \in G_{32}$$

$$\begin{array}{l} \delta(p_6, d) \in G_{31} \\ \delta(p_7, d) \in G_{31} \end{array}$$

Nakon trećeg prolaza ne dolazi do nove podjele i algoritam se zaustavlja. Algoritmom je određeno da su istovjetna stanja p_1 i p_2 , te stanja p_6 i p_7 . Istovjetna stanja nalaze se u istim grupama. Na slici 2.14 prikazan je novi DKA u kojem su istovjetna stanja zamjenjena novim stanjima. Nova stanja označena su uglatim zagradama u kojima su navedena stara istovjetna stanja.

	c	d	
p_1	p_6	p_3	0
p_2	p_7	p_3	0
p_3	p_1	p_5	0
p_4	p_4	p_6	0
p_5	p_7	p_3	1
p_6	p_4	p_1	1
p_7	p_4	p_2	1

	c	d	
$[p_1, p_2]$	$[p_6, p_7]$	$[p_3]$	0
$[p_3]$	$[p_1, p_2]$	$[p_5]$	0
$[p_4]$	$[p_4]$	$[p_6, p_7]$	0
$[p_5]$	$[p_6, p_7]$	$[p_3]$	1
$[p_6, p_7]$	$[p_4]$	$[p_1, p_2]$	1

Slika 2.14: DKA je istovjetan s DKA sa slike 2.13, ali bez istovjetnih stanja

Slika 2.13: DKA s istovjetnim stanjima

Algoritam 3. Algoritam pronalazi neistovjetna stanja. Označi li se par stanja (p, q) DKA $M=(Q, \Sigma, \delta, q_0, F)$ algoritmom koji je prikazan na slici 2.15, dva stanja p i q su neistovjetna.

označi sve parove (p, q) za koje vrijedi da je $p \in F$ i $q \in (Q - F)$;

za (svaki par različitih stanja takvih da je $(p, q) \in (F \times F)$ ili $(p, q) \in (Q - F \times Q - F)$)

{
 ako (za neki znak a par $(\delta(p, a), \delta(q, a))$ je označen)
 označi (p, q) ;

rekurzivno označi sve neoznačene parove u listi koja je pridružena paru (p, q) i sve ostale parove u listama koje su pridružene parovima označenim u ovom koraku;

}
inače
{

za (svi znakovi a)

{
 ako ($\delta(p, a) \neq \delta(q, a)$)
 stavi (p, q) u listu koja je pridružena paru $(\delta(p, a), \delta(q, a))$

}

}

}

Slika 2.15: Algoritam označavanja neistovjetnih stanja

Rad algoritma opisuje se na primjeru traženja neistovjetnih stanja DKA zadanog na slici 2.16. Traženje neistovjetnih stanja opisano je u šest koraka.

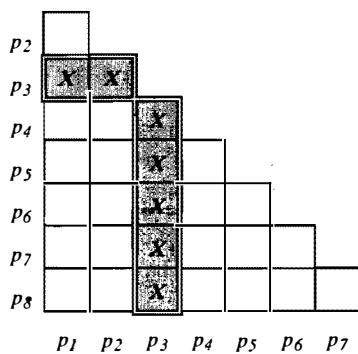
	0	1	
p_1	p_2	p_6	0
p_2	p_7	p_3	0
p_3	p_1	p_3	1
p_4	p_3	p_7	0
p_5	p_8	p_6	0
p_6	p_3	p_7	0
p_7	p_7	p_5	0
p_8	p_7	p_3	0

	0	1	
$[p_1, p_5]$	$[p_2, p_8]$	$[p_4, p_6]$	0
$[p_2, p_8]$	p_7	p_3	0
$[p_3]$	$[p_1, p_5]$	p_3	1
$[p_4, p_6]$	p_3	p_7	0
p_7	p_7	$[p_1, p_5]$	0

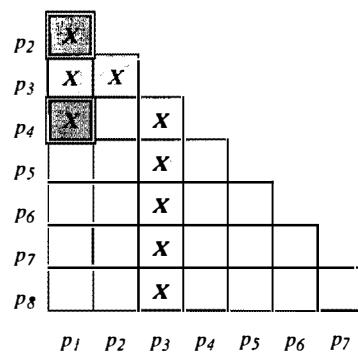
Slika 2.17: DKA je istovjetan s DKA sa slike 2.16, ali bez istovjetnih stanja

Slika 2.16: DKA s istovjetnim stanjima

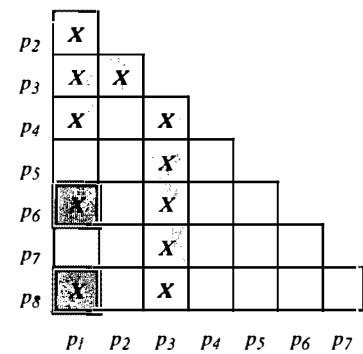
U prvom koraku algoritam označi sve parove stanja u kojima je stanje p_3 : (p_1, p_3) , (p_2, p_3) , (p_4, p_3) , (p_5, p_3) , (p_6, p_3) i (p_8, p_3) . Stanje p_3 je prihvativivo, a sva ostala stanja su neprihvativiva.



1. korak



lista uz $(p_2, p_8) = \{(p_1, p_5)\}$



lista uz $(p_2, p_8) = \{(p_1, p_5)\}$

lista uz $(p_2, p_7) = \{(p_1, p_7)\}$

lista uz $(p_6, p_5) = \{(p_1, p_7)\}$

2. korak

3. korak

U drugom koraku označe se parovi (p_1, p_2) i (p_1, p_4) , jer su prethodno označeni parovi $(\delta(p_1, 1), \delta(p_2, 1))=(p_6, p_3)$ i $(\delta(p_1, 0), \delta(p_4, 0))=(p_2, p_3)$. Par (p_1, p_5) stavlja se u listu koja je pridružena paru $(\delta(p_1, 0), \delta(p_5, 0))=(p_2, p_8)$. Za ulazni znak 1 par (p_1, p_5) se ne stavlja niti u jednu listu, jer oba stanja prelaze u stanje p_6 za znak 1.

U trećem koraku označe se parovi (p_1, p_6) i (p_1, p_8) , jer su prethodno označeni parovi $(\delta(p_1, 0), \delta(p_6, 0))=(p_2, p_3)$ i $(\delta(p_1, 1), \delta(p_8, 1))=(p_6, p_3)$. Par (p_1, p_7) se stavlja u listu koja je pridružena paru $(\delta(p_1, 0), \delta(p_7, 0))=(p_2, p_7)$ i u listu koja je pridružena paru $(\delta(p_1, 1), \delta(p_7, 1))=(p_6, p_5)$.

p_2	X						
p_3	X	X					
p_4	X	X	X				
p_5	X	X	X				
p_6	X	X	X				
p_7	X	X	X				
p_8	X	X					

$p_1 \quad p_2 \quad p_3 \quad p_4 \quad p_5 \quad p_6 \quad p_7$

lista uz $(p_2, p_6) = \{(p_1, p_5)\}$

lista uz $(p_6, p_5) = \{(p_1, p_7)\}$

4. korak

p_2	X						
p_3	X	X					
p_4	X	X	X				
p_5	X	X	X	X			
p_6	X	X	X	X			
p_7	X	X	X	X			
p_8	X	X	X	X			

$p_1 \quad p_2 \quad p_3 \quad p_4 \quad p_5 \quad p_6 \quad p_7$

lista uz $(p_2, p_8) = \{(p_1, p_5)\}$

lista uz $(p_6, p_5) = \{(p_1, p_7)\}$

5. korak

p_2	X						
p_3	X	X					
p_4	X	X	X				
p_5	X	X	X	X			
p_6	X	X	X	X	X		
p_7	X	X	X	X	X	X	
p_8	X						

$p_1 \quad p_2 \quad p_3 \quad p_4 \quad p_5 \quad p_6 \quad p_7$

lista uz $(p_2, p_8) = \{(p_1, p_5)\}$

6. korak

U četvrtom koraku označe se parovi (p_2, p_4) , (p_2, p_5) , (p_2, p_6) i (p_2, p_7) , jer su prethodno označeni parovi $(\delta(p_2, 0), \delta(p_4, 0))=(p_7, p_3)$, $(\delta(p_2, 1), \delta(p_5, 1))=(p_3, p_6)$, $(\delta(p_2, 0), \delta(p_6, 0))=(p_7, p_3)$ i $(\delta(p_2, 1), \delta(p_7, 1))=(p_3, p_5)$. Budući da je stanje (p_1, p_7) u listi koja je pridružena paru (p_2, p_7) , onda se označi i par (p_1, p_7) . Par (p_2, p_8) se ne označava i ne stavlja niti u jednu listu, jer za ulazni znak 0 oba stanja prelaze u stanje p_7 , a za ulazni znak 1 oba stanja prelaze u stanje p_3 .

U petom koraku označe se parovi (p_4, p_5) , (p_4, p_7) i (p_4, p_8) , jer su prethodno označeni parovi $(\delta(p_4, 0), \delta(p_5, 0))=(p_3, p_8)$, $(\delta(p_4, 0), \delta(p_7, 0))=(p_3, p_7)$ i $(\delta(p_4, 0), \delta(p_8, 0))=(p_3, p_7)$. Par (p_4, p_6) se ne označava i ne stavlja niti u jednu listu, jer za ulazni znak 0 oba stanja prelaze u stanje p_3 , a za ulazni znak 1 oba stanja prelaze u stanje p_7 .

Algoritam završava rad u šestom koraku označavanjem parova stanja (p_5, p_6) , (p_5, p_7) , (p_5, p_8) , (p_6, p_7) , (p_6, p_8) i (p_7, p_8) , jer ona neposredno prelaze u stanja koja su prethodno označena. Ostali su neoznačeni parovi stanja (p_1, p_5) , (p_2, p_8) i (p_4, p_6) , što znači da su ti parovi stanja istovjetni. DKA bez istovjetnih stanja prikazan je na slici 2.17.

Nedohvatljiva stanja

Stanje p DKA $M=(Q, \Sigma, \delta, q_0, F)$ jest nedohvatljivo ako ne postoji niti jedan niz $w \in \Sigma^*$ za koji vrijedi da je $p=\delta(q_0, w)$. Stanje p_4 DKA sa slike 2.10 je primjer nedohvatljivog stanja. Odbacivanjem nedohvatljivih stanja dobije se istovjetni DKA s manjim brojem stanja. Dohvatljiva stanja DKA $M=(Q, \Sigma, \delta, q_0, F)$ pronalaze se sljedećim algoritmom:

- 1) U listu dohvatljivih stanja DS upiše se početno stanje q_0 .
- 2) Lista DS proširi se skupom stanja $\{p \mid p=\delta(q_0, a), \text{ za sve } a \in \Sigma\}$.
- 3) Za sva stanja $q_i \in DS$ proširi se lista skupom stanja $\{p \mid p=\delta(q_i, a), \text{ stanje } p \text{ nije prethodno upisano u listu, za sve } a \in \Sigma\}$.

Ako se lista dohvatljivih stanja DS proširi upisom novog stanja, rad algoritma nastavlja se od koraka (3). Lista DS sadrži sva dohvatljiva stanja, dok su ostala stanja nedohvatljiva.

Rad algoritma opisuje se na primjeru DKA danog na slici 2.18.

1. korak $DS \quad q_0$

2. korak $DS \quad q_0, q_1, q_5$

jer je $\delta(q_0, c) = q_1$ i $\delta(q_0, d) = q_5$

3. korak $DS \quad q_0, q_1, q_5, q_2, q_7, q_3$

jer je $\delta(q_1, c) = q_2$, $\delta(q_1, d) = q_7$ i $\delta(q_5, c) = q_3$

4. korak $DS \quad q_0, q_1, q_5, q_2, q_7, q_3$

DS se ne mijenja, jer stanja q_2 , q_7 i q_3 ne prelaze niti u jedno stanje koje od prije nije upisano u listu, i rad algoritma se zaustavlja.

Nedohvatljiva stanja q_4 , q_6 i q_8 se odbacuju. Istovjetni DKA bez nedohvatljivih stanja prikazan je na slici 2.19.

	c	d
q_0	q_1	q_5
q_1	q_2	q_7
q_2	q_2	q_5
q_3	q_5	q_7
q_4	q_5	q_6
q_5	q_3	q_1
q_6	q_8	q_0
q_7	q_0	q_1
q_8	q_3	q_6

	c	d
q_0	q_1	q_5
q_1	q_2	q_7
q_2	q_2	q_5
q_3	q_5	q_7
q_5	q_3	q_1
q_7	q_0	q_1

Slika 2.19: DKA koji je istovjetan s DKA sa slike 2.18, ali bez nedohvatljivih stanja

Slika 2.18: DKA s nedohvatljivim stanjima

DKA s minimalnim brojem stanja

Odbacivanjem istovjetnih i nedohvatljivih stanja dobije se istovjetni DKA s minimalnim brojem stanja. Ne postoji niti jedan drugi DKA koji prihvaca isti jezik, a ima manji broj stanja. Buduci da je postupak odbacivanja nedohvatljivih stanja jednostavniji od postupka odbacivanja istovjetnih stanja, te buduci da se odbacivanjem nedohvatljivih stanja smanjuje ukupni broj stanja, sto pojednostavljuje postupak odbacivanja istovjetnih stanja, učinkovitije je prvo primijeniti postupak odbacivanja nedohvatljivih stanja, a zatim postupak odbacivanja istovjetnih stanja.

Na slikama 2.20 i 2.21 prikazani su istovjetni DKA. DKA na slici 2.21 izgrađen je odbacivanjem nedohvatljivih i istovjetnih stanja DKA sa slike 2.20. DKA sa slike 2.20 jest istovjetan s DKA sa slike 2.10. DKA sa slike 2.21 jest istovjetan s DKA sa slike 2.11, samo što je stanje p_4 odbačeno kao nedohvatljivo, a stanje p_6 je označeno znakom C.

	0	1	
p_0	p_0	p_3	0
p_1	p_2	p_5	0
p_2	p_2	p_7	0
p_3	p_6	p_7	0
p_4	p_1	p_6	1
p_5	p_6	p_5	0
p_6	p_6	p_3	1
p_7	p_6	p_3	0

	0	1	
A	A	B	0
B	C	B	0
C	C	B	1

Slika 2.21: DKA koji je istovjetan s DKA sa slike 2.20, ali s minimalnim brojem stanja

Slika 2.20: DKA s istovjetnim i nedohvatljivim stanjima

Na slici 2.22 dokazana je istovjetnost DKA prikazanih na slikama 2.20 i 2.21. Postupak se zasniva na dokazu istovjetnosti njihovih početnih stanja primjenom prvog algoritma.

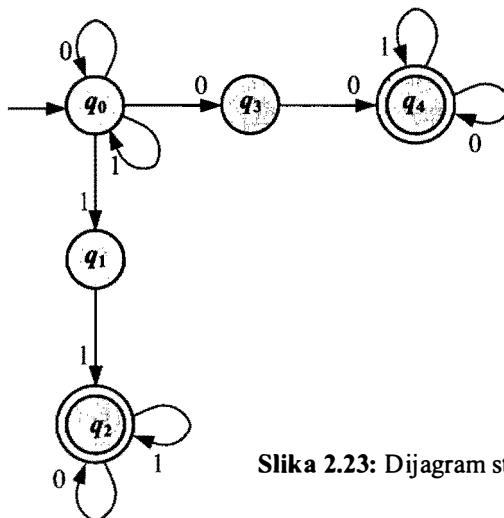
	0	1	
p_0, A	p_0, A	p_3, B	
p_3, B	p_0, A	p_3, B	
	p_6, C	p_7, B	
	p_6, C	p_7, B	
	p_6, C	p_7, B	
	p_6, C	p_7, B	

Slika 2.22: Postupak dokazivanja istovjetnosti DKA prikazanih na slikama 2.20 i 2.21

2.1.3 Nedeterministički konačni automat (NKA)

Definicija DKA proširuje se novom funkcijom prijelaza. Za razliku od funkcije prijelaza DKA koja za jedno stanje i jedan znak određuje prijelaz u jedinstveno stanje $\delta(q, a)=p$, nova funkcija prijelaza određuje prijelaz u skup stanja $\delta(q, a)=\{p_1, p_2, \dots\}$. Skup stanja može biti prazan. Konačni automat koji se zasniva na novoj funkciji prijelaza naziva se nedeterministički konačni automat (NKA). Za bilo koji NKA N moguće je izgraditi DKA D koji prihvata isti jezik $L(N)=L(D)$.

Za neke jezike lakše je izgraditi NKA nego DKA. Na temelju izgrađenog NKA, konstruira se DKA koristeći algoritam opisan u nastavku ovog odjeljka. DKA s minimalnim brojem stanja, koji omogućuje učinkovito programsko ostvarenje konačnog automata, izgradi se algoritmom opisanim u odjeljku 2.1.2.



Slika 2.23: Dijagram stanja NKA

Na slici 2.23 prikazan je NKA dijagramom stanja. NKA prihvata nizove koji imaju barem dvije uzastopne nule ili jedinice. Za ulazni znak 0 i stanje q_0 NKA ostaje u stanju q_0 , ili prelazi u stanje q_3 . Za ulazni znak 1 i stanje q_0 NKA ostaje u stanju q_0 , ili prelazi u stanje q_1 .

Nedeterminizam NKA ne označava pojavu slučajnih događaja. Nedeterminizam NKA opisuje se u odnosu na DKA na sljedeći način:

- 1) Za bilo koji niz w postoji samo jedan slijed prijelaza DKA iz početnog stanja q_0 u stanje $p=\delta(q_0, w)$, odnosno na temelju samo jednog slijeda prijelaza moguće je utvrditi da li se neki niz prihvata. Završi li slijed prijelaza prihvatljivim stanjem, niz w se prihvata.
- 2) Za neki niz w NKA može imati više od jednog slijeda prijelaza. Tijekom postupka utvrđivanja da li NKA prihvata niz w , provjeravaju se svi slijedovi prijelaza. Postoji li barem jedan slijed prijelaza iz početnog stanja u jedno od prihvatljivih stanja, NKA prihvata niz.

Na primjer, niz znakova 01001 se prihvata, jer postoji slijed prijelaza u prihvatljivo stanje q_4 :

$$\begin{array}{ccccccc} 0 & & 1 & & 0 & & 1 \\ \rightarrow q_0 & \rightarrow q_0 & \rightarrow q_3 & \rightarrow q_4 & \rightarrow q_4, & & q_4 \in F \end{array}$$

Za isti taj niz 01001 moguće je dati i drugi slijed prijelaza, koji završava neprihvatljivim stanjem q_1 :

$$\begin{array}{ccccccc} 0 & & 1 & & 0 & & 1 \\ q_0 & \rightarrow q_1, & q_1 \notin F \end{array}$$

Budući da postoji barem jedan slijed prijelaza koji vodi u jedno od prihvatljivih stanja, niz znakova 01001 se prihvata. Niz znakova 1010 se ne prihvata, jer ne postoji niti jedan slijed prijelaza u jedno od prihvatljivih stanja q_2 ili q_4 .

Na slici 2.24 prikazan je NKA tablicom prijelaza. U tablici prijelaza nalaze se skupovi stanja u koja prelazi NKA za određeni ulazni znak. Skup stanja jest prazan za stanje q_1 i ulazni znak 0, te za stanje q_3 i ulazni znak 1. Prazni skup označava da daljnji slijed prijelaza ne postoji bez obzira na ulazne znakove koji slijede u ulaznom nizu. U dalnjem tekstu prazni skup označava se znakom \emptyset .

Za niz znakova 1010 mogu se dati svi slijedovi prijelaza:

$$\begin{array}{ccccc} 1 & & 0 \\ q_0 & \rightarrow q_1 & \rightarrow & \text{nema daljnog slijeda prijelaza} \end{array}$$

$$\begin{array}{ccccc} 1 & & 0 & & 1 \\ q_0 & \rightarrow q_0 & \rightarrow q_3 & \rightarrow & \text{nema daljnog slijeda prijelaza} \end{array}$$

$$\begin{array}{ccccc} 1 & & 0 & & 1 & 0 \\ q_0 & \rightarrow q_0 & \rightarrow q_0 & \rightarrow q_1 & \rightarrow & \text{nema daljnog slijeda prijelaza} \end{array}$$

$$\begin{array}{ccccc} 1 & & 0 & & 1 & 0 \\ q_0 & \rightarrow q_0 & \rightarrow q_0 & \rightarrow q_0 & \rightarrow q_3, & q_3 \notin F \end{array}$$

$$\begin{array}{ccccc} 1 & & 0 & & 1 & 0 \\ q_0 & \rightarrow q_0 & \rightarrow q_0 & \rightarrow q_0 & \rightarrow q_0, & q_0 \notin F \end{array}$$

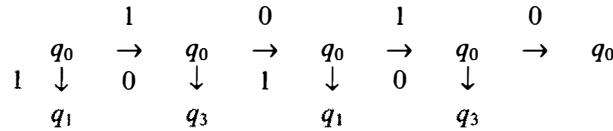
	0	1	
q_0	$\{q_0, q_3\}$	$\{q_0, q_1\}$	0
q_1	$\{\}$	$\{q_2\}$	0
q_2	$\{q_2\}$	$\{q_2\}$	1
q_3	$\{q_4\}$	$\{\}$	0
q_4	$\{q_4\}$	$\{q_4\}$	1

Slika 2.24: Tablica prijelaza NKA

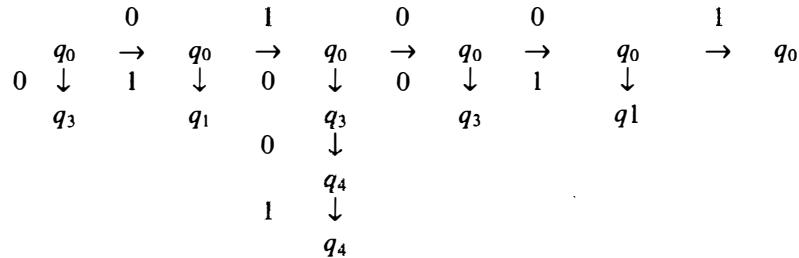
Budući da za niz 1010 ne postoji niti jedan slijed prijelaza iz početnog stanja u jedno od prihvatljivih stanja, NKA ne prihvata niz.

Rad NKA opisuje se na sljedeći način: svaki puta kada postoji mogućnost prijelaza u više različitih stanja za jedan ulazni znak, stvoriti se upravo toliko DKA koliko ima prijelaza u različita stanja. Svi nastali

DKA paralelno obrađuju ostatak ulaznog niza. Izgradi li barem jedan DKA slijed prijelaza koji završava prihvatljivim stanjem, niz se prihvaća. Slijed prijelaza za niz 1010 simbolički se opisuje na sljedeći način:



Prihvaćanje niza 01001 simbolički se opisuje na sljedeći način:



Za niz 01001 postoji slijed prijelaza koji završava prihvatljivim stanjem q_4 i zato se niz prihvaća.

Nedeterministički konačni automat (NKA) formalno se zadaje kao uređena petorka:

$$nka = (Q, \Sigma, \delta, q_0, F)$$

gdje je:

- Q - konačan skup stanja;
- Σ - konačan skup ulaznih znakova;
- δ - funkcija prijelaza $Q \times \Sigma \rightarrow 2^Q$, gdje 2^Q jest skup svih podskupova skupa stanja Q ;
- $q_0 \in Q$ - početno stanje;
- $F \subseteq Q$ - skup prihvatljivih stanja.

Funkcija $\hat{\delta} : Q \times \Sigma^* \rightarrow 2^Q$ definira se na sljedeći način:

- 1) $\hat{\delta}(q, \varepsilon) = \{q\}$,
- 2) $\hat{\delta}(q, wa) = P = \{p \mid \text{za neko stanje } r \text{ iz } \hat{\delta}(q, w), p \text{ jest u } \delta(r, a)\},$ gdje je $w \in \Sigma^*$,
 $a \in \Sigma$ i $P \subseteq Q$.

U (1) jest zadano da NKA ne može promijeniti stanje, a da ne pročita barem jedan ulazni znak. U (2) jest zadano na koji način se računa skup stanja u koja prelazi NKA čitanjem znakova niza wa . Prvo se izračunaju stanja $r = \hat{\delta}(q, w)$, gdje su r stanja u koja prelazi NKA iz stanja q čitanjem znakova niza w . Nakon toga se računa skup $\delta(r, a)$ za znak a i sva stanja r .

Uvrštavanjem $w = \varepsilon$ u (2) dobije se:

$$\hat{\delta}(q, a) = P = \{p \mid \text{gdje je } p \text{ iz } \delta(q, a)\} = \delta(q, a),$$

jer je prema (1) $\hat{\delta}(q, \varepsilon) = \{q\}$. Time se omogućuje korištenje jedinstvene oznake δ za obje funkcije prijelaza.

Definira se da NKA $M = (Q, \Sigma, \delta, q_0, F)$ prihvaća jezik $L(M) = \{w \mid \delta(q_0, w) \text{ sadrži barem jedno stanje iz skupa } F\}$.

Funkcija δ proširuje se na argumente $2^Q \times \Sigma^*$ na sljedeći način:

- 3) $\delta(P, w) = \bigcup_{q \in P} \delta(q, w)$ za bilo koji skup stanja $P \subseteq Q$.

NKA prikazan na slici 2.24 prihvata niz 01001, jer je prihvatljivo stanje q_4 u skupu $\delta(q_0, 01001) = \{q_0, q_1, q_4\}$:

$$\begin{aligned}\delta(q_0, 0) &= \{q_0, q_3\}, \\ \delta(q_0, 01) &= \delta(\delta(q_0, 0), 1) = \delta(\{q_0, q_3\}, 1) = \delta(q_0, 1) \cup \delta(q_3, 1) = \{q_0, q_1\} \cup \{\} = \{q_0, q_1\}, \\ \delta(q_0, 010) &= \delta(\delta(q_0, 01), 0) = \delta(\{q_0, q_1\}, 0) = \delta(q_0, 0) \cup \delta(q_1, 0) = \{q_0, q_3\} \cup \{\} = \{q_0, q_3\}, \\ \delta(q_0, 0100) &= \delta(\delta(q_0, 010), 0) = \delta(\{q_0, q_3\}, 0) = \delta(q_0, 0) \cup \delta(q_3, 0) = \{q_0, q_3\} \cup \{q_4\} = \{q_0, q_3, q_4\}, \\ \delta(q_0, 01001) &= \delta(\delta(q_0, 0100), 1) = \delta(\{q_0, q_3, q_4\}, 1) = \delta(q_0, 1) \cup \delta(q_3, 1) \cup \delta(q_4, 1) = \\ &\quad \{q_0, q_1\} \cup \{\} \cup \{q_4\} = \{q_0, q_1, q_4\}, \quad (q_4 \in \delta(q_0, 01001) \wedge q_4 \in F) \Rightarrow \text{niz } 01001 \text{ se prihvata.}\end{aligned}$$

Konstrukcija DKA iz zadanog NKA

Za bilo koji NKA $M=(Q, \Sigma, \delta, q_0, F)$ moguće je izgraditi istovjetni DKA $M'=(Q', \Sigma, \delta', q_0', F')$. NKA i DKA su istovjetni ako prihvataju isti jezik. Izgradi se skup stanja Q' koji ima onoliko stanja koliko skup Q ima podskupova. Neka je skup stanja NKA $Q=\{q_0, q_1, q_2, \dots, q_i\}$. Stanja DKA mogu se označiti na sljedeći način $Q'=\{[\emptyset], [q_0], [q_1], \dots, [q_i], [q_0, q_1], [q_0, q_2], \dots, [q_{i-1}, q_i], [q_0, q_1, q_2], \dots, [q_0, q_1, q_2, \dots, q_i]\}$. Za početno stanje DKA q_0' uzima se početno stanje NKA $q_0'=[q_0]$. Funkcija prijelaza DKA δ' odredi se tako da za neki niz $w \in \Sigma^*$ vrijedi da je $\delta'([q_0], w)=[p_0, p_1, \dots, p_j]$ ako i samo ako je $\delta(q_0, w)=\{p_0, p_1, \dots, p_j\}$. Stanje DKA jest prihvatljivo ako u oznaci tog stanja postoji barem jedno prihvatljivo stanje NKA iz skupa F . Budući da je skup Q' konačan, izgrađeni automat ima konačni broj stanja.

Neka je zadan NKA $M=(Q, \Sigma, \delta, q_0, F)$. Dokazuje se da DKA $M'=(Q', \Sigma, \delta', q_0', F')$ prihvata isti jezik kao i NKA M ako je zadovoljeno:

- 1) Skup stanja DKA jest $Q'=2^Q$, tj. skup svih podskupova skupa stanja NKA Q . Stanja su označena uglatim zagradama $[p_0, p_1, \dots, p_j] \in Q'$, gdje su $p_k \in Q$.
- 2) Skup prihvatljivih stanja DKA F' jest skup svih stanja $[p_0, p_1, \dots, p_j]$ gdje je barem jedan $p_k \in F$.
- 3) Početno stanje DKA jest $q_0'=[q_0]$.
- 4) Funkcija prijelaza DKA jest $\delta'([p_0, p_1, \dots, p_l], a)=[r_0, r_1, \dots, r_j]$ ako i samo ako je $\delta(\{p_0, p_1, \dots, p_l\}, a)=\{r_0, r_1, \dots, r_j\}$, gdje je $a \in \Sigma$.

Mnoga stanja DKA mogu biti nedohvatljiva. Zato se izgrađeni DKA minimizira kako bi se omogućilo učinkovito programsko ostvarenje.

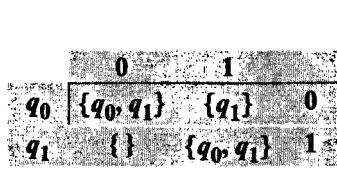
Primjer 2.2. Neka je zadan NKA $M=(\{q_0, q_1\}, \{0, 1\}, \delta, q_0, \{q_1\})$. Funkcija prijelaza NKA M prikazana je tablicom na slici 2.25. DKA $M'=(Q', \Sigma, \delta', q_0', F')$ gradi se na sljedeći način:

- 1) $Q'=\{[\emptyset], [q_0], [q_1], [q_0, q_1]\}$,
- 2) $F'=\{[q_1]\}$,
- 3) $q_0'=[q_0]$,
- 4) funkcija prijelaza δ' je:

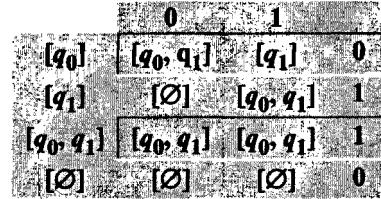
$$\begin{aligned}\delta'([q_0], 0) &= [q_0, q_1], & \text{jer je } \delta(q_0, 0) = \{q_0, q_1\} \\ \delta'([q_0], 1) &= [q_1], & \text{jer je } \delta(q_0, 1) = \{q_1\}\end{aligned}$$

$\delta'([q_1], 0) = [\emptyset]$,	jer je $\delta(q_1, 0) = \{ \}$
$\delta'([q_1], 1) = [q_0, q_1]$,	jer je $\delta(q_1, 1) = \{q_0, q_1\}$
$\delta'([q_0, q_1], 0) = [q_0, q_1]$,	jer je $\delta(\{q_0, q_1\}, 0) = \delta(q_0, 0) \cup \delta(q_1, 0) = \{q_0, q_1\} \cup \{ \ } = \{q_0, q_1\}$
$\delta'([q_0, q_1], 1) = [q_0, q_1]$,	jer je $\delta(\{q_0, q_1\}, 1) = \delta(q_0, 1) \cup \delta(q_1, 1) = \{q_1\} \cup \{q_0, q_1\} = \{q_0, q_1\}$
$\delta'([\emptyset], 0) = [\emptyset]$,	jer je $\delta(\{ \ }, 0) = \{ \ }$
$\delta'([\emptyset], 1) = [\emptyset]$,	jer je $\delta(\{ \ }, 1) = \{ \ }$

Izgrađeni DKA prikazan je na slici 2.26. Oba konačna automata prihvataju jezik u kojemu je niz 1 i svi nizovi čiji prefiks je 0 ili 11.



Slika 2.25: NKA



Slika 2.26: DKA istovjetan s NKA sa slike 2.25

Istovjetnost DKA i NKA

Neka je DKA $M'=(Q', \Sigma, \delta', q_0', F')$ izgrađen na temelju zadanoog NKA $M=(Q, \Sigma, \delta, q_0, F)$. Želi se dokazati da automati M i M' prihvataju isti jezik $L(M)=L(M')$. Dokazuje se indukcijom da za bilo koji niz $w \in \Sigma^*$ vrijedi:

- i) $\delta'([q_0], w)=[r_0, r_1, \dots, r_j]$ ako i samo ako je $\delta(q_0, w)=\{r_0, r_1, \dots, r_j\}$.

Indukcija se zasniva na duljini niza x :

- a) U prvom dijelu dokazuje se da (i) vrijedi za $|w|=0$, odnosno za $w=\epsilon$. Ako se uzme u obzir činjenica da DKA i NKA ne mogu promijeniti stanje a da ne pročitaju barem jedan znak, te ako se uzme u obzir da je $q_0'=[q_0]$ na temelju koraka (3) izgradnje DKA, onda se lako dokazuje da (i) vrijedi za $w=\epsilon$.
- b) Pretpostavi se da (i) vrijedi za niz $x \in \Sigma^*$, a zatim se dokazuje se da (i) vrijedi za niz xa , gdje je a ulazni znak $a \in \Sigma$.

Na temelju pretpostavke da vrijedi:

- b1) $\delta'([q_0], x)=[p_0, p_1, \dots, p_l]$ ako i samo ako je $\delta(q_0, x)=\{p_0, p_1, \dots, p_l\}$,

i na temelju definicije (4) konstrukcije funkcije δ' u ovom odjeljku:

- b2) $\delta'([p_0, p_1, \dots, p_l], a)=[r_0, r_1, \dots, r_j]$ ako i samo ako je $\delta(\{p_0, p_1, \dots, p_l\}, a)=\{r_0, r_1, \dots, r_j\}$,

zaključuje se da vrijedi:

- b3) $\delta'([q_0], xa)=[r_0, r_1, \dots, r_j]$ ako i samo ako je $\delta(q_0, xa)=\{r_0, r_1, \dots, r_j\}$.

Uzevši u obzir da je $\delta'([q_0], w) \in F'$ ako i samo ako $\delta(q_0, w)$ sadrži barem jedno stanje iz F , zaključuje se da NKA M i DKA M' prihvataju isti jezik.

2.1.4 Nedeterministički konačni automat s ϵ prijelazima (ϵ -NKA)

Funkcija prijelaza NKA proširuje se sljedećim svojstvom: konačni automat može promijeniti stanje a da ne pročita niti jedan ulazni znak. NKA zasnovan na proširenoj funkciji prijelaza naziva se nedeterministički konačni automat s ϵ -prijelazima (ϵ -NKA). Promjena stanja ϵ -NKA, a da se ne pročita ulazni znak, naziva se ϵ -prijelaz. U dijagramu stanja i u tablici prijelaza ϵ -prijelazi označeni su oznakom ϵ .

U nastavku odjeljka pokazana je istovjetnost ϵ -NKA i NKA. Opisan je algoritam pretvorbe zadalog ϵ -NKA u NKA. Povezivanjem algoritma pretvorbe ϵ -NKA u NKA i algoritma pretvorbe NKA u DKA, dobije se algoritam pretvorbe ϵ -NKA u DKA. Algoritam pretvorbe ϵ -NKA u DKA i algoritmi minimizacije omogućuju učinkovito programsko ostvarenje konačnog automata.

Na slici 2.27 dan je dijagram stanja ϵ -NKA koji prihvaca jezik $L = \{0^n 1^m 2^l \mid n, m, l \geq 0\}$. Jezik L sastoji se od nizova koji započinju proizvoljnim brojem znakova 0, iza kojih slijedi proizvoljan broj znakova 1, a završavaju proizvoljnim brojem znakova 2. Postoji li barem jedan slijed prijelaza iz početnog stanja u jedno od prihvatljivih stanja, uz uvjet da se procitaju svi znakovi niza, niz se prihvata. U slijed prijelaza uključuju se i ϵ -prijelazi.

Prazni niz ϵ se prihvata, jer postoji slijed prijelaza u prihvatljivo stanje q_2 :

$$\begin{array}{ccccccc} \epsilon & \epsilon \\ q_0 & \rightarrow & q_1 & \rightarrow & q_2, & q_2 \in F. \end{array}$$

Primjenom ϵ -prijelaza ϵ -NKA promjeni stanje u q_2 . Za danu promjenu stanja nije potrebno pročitati niti jedan ulazni znak. Treba biti pažljiv, jer su istom oznakom označeni prazni niz $\epsilon \in \Sigma^*$ i oznaka ϵ -prijelaza.

Niz 002 se prihvata, jer postoji slijed prijelaza:

$$\begin{array}{ccccccc} 0 & 0 & \epsilon & \epsilon & 2 \\ q_0 & \rightarrow & q_0 & \rightarrow & q_1 & \rightarrow & q_2 \rightarrow q_2, \quad q_2 \in F. \end{array}$$

Nakon drugog pročitanog znaka 0, ϵ -NKA primjenom ϵ -prijelaza promjeni stanje u q_2 , te nastavi čitanje znaka 2.

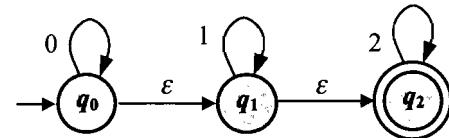
Niz 122 se prihvata, jer postoji slijed prijelaza:

$$\begin{array}{ccccccc} \epsilon & 1 & \epsilon & 2 & 2 \\ q_0 & \rightarrow & q_1 & \rightarrow & q_1 & \rightarrow & q_2 \rightarrow q_2, \quad q_2 \in F. \end{array}$$

Nakon što pročita znak 1, ϵ -NKA je u stanju q_1 . Iz stanja q_1 moguće je ϵ -prijelazom pijeći u stanje q_2 . Međutim, iz stanja q_1 ne postoji prijelaz u stanje q_0 . Nakon pročitanog znaka 2, ϵ -NKA prelazi u stanje q_2 . Iz stanja q_2 nije moguće pijeći u stanja q_0 i q_1 .

Niz 01210 se ne prihvata jer ne postoji niti jedan slijed prijelaza u prihvatljivo stanje q_2 koji omogućuje čitanje svih znakova niza:

$$\begin{array}{ccccccc} 0 & \epsilon & 1 & \epsilon & 2 & 1 \\ q_0 & \rightarrow & q_0 & \rightarrow & q_1 & \rightarrow & q_1 \rightarrow q_2 \rightarrow q_2 \rightarrow, \quad \text{nema daljnog slijeda prijelaza.} \end{array}$$



Slika 2.27: Dijagram stanja ϵ -NKA

Nakon što pročita znak 2, ϵ -NKA jest u stanju q_2 . U stanju q_2 nije definiran prijelaz za znak 1 i ne postoji mogućnost primjene ϵ -prijelaza i povratka u stanje q_1 . Budući da za znak 1 ne postoji mogućnost primjene niti jednog prijelaza, ϵ -NKA ne prihvata niz 01210.

Nedeterministički konačni automat s ϵ -prijelazima (ϵ -NKA) formalno se zadaje kao uređena petorka:

$$\epsilon\text{-nka} = (Q, \Sigma, \delta, q_0, F)$$

gdje je:

Q	- konačan skup stanja;
Σ	- konačan skup ulaznih znakova;
δ	- funkcija prijelaza $Q \times (\Sigma \cup \{\epsilon\}) \rightarrow 2^Q$;
$q_0 \in Q$	- početno stanje;
$F \subseteq Q$	- skup prihvatljivih stanja.

Funkcija prijelaza $\delta(q, a)$ određuje skup stanja P . U skupu stanja P su sva ona stanja p za koja postoji prijelaz iz stanja q . Oznaka a jest ϵ ili neki znak iz skupa ulaznih znakova Σ .

	0	1	2	ϵ	
0	$\{q_0\}$	\emptyset	\emptyset	$\{q_1\}$	0
1	\emptyset	$\{q_1\}$	\emptyset	$\{q_2\}$	0
2	\emptyset	\emptyset	$\{q_2\}$	\emptyset	1

Funkcija $\delta(q, a)$ prikazuje se tablicom prijelaza. Na slici 2.28 prikazana je funkcija prijelaza $\delta(q, a)$ za ϵ -NKA zadan dijagramom stanja na slici 2.27.

Slika 2.28: Funkcija prijelaza $\delta(q, a)$ za ϵ -NKA
zadan dijagramom stanja na slici 2.27

Proširenje funkcije prijelaza δ na argumente $Q \times \Sigma^*$ zahtijeva definiranje funkcije ϵ -OKRUŽENJE. Funkcija ϵ -OKRUŽENJE(q) pridružuje stanju q ($q \in Q$) skup stanja R ($R \subseteq Q$). Skup R jest skup svih stanja u koja ϵ -NKA prelazi iz stanja q primjenom samo ϵ -prijelaza:

$$\epsilon\text{-OKRUŽENJE}(q) = \{p \mid \text{stanje } p \text{ jest stanje } q \text{ ili } \epsilon\text{-NKA prelazi iz stanja } q \text{ u stanje } p \text{ isključivo primjenom } \epsilon\text{-prijelaza}\}$$

Obilaskom samo onih grana dijagrama stanja koje su ϵ -prijelazi moguće je odrediti skup ϵ -OKRUŽENJE(q). Stanjima ϵ -NKA, koji je zadan na slikama 2.27 i 2.28, funkcija ϵ -OKRUŽENJE pridružuje sljedeće skupove:

$$\begin{aligned}\epsilon\text{-OKRUŽENJE}(q_0) &= \{q_0, q_1, q_2\} \\ \epsilon\text{-OKRUŽENJE}(q_1) &= \{q_1, q_2\} \\ \epsilon\text{-OKRUŽENJE}(q_2) &= \{q_2\}\end{aligned}$$

Za skup stanja P funkcija ϵ -OKRUŽENJE računa se na sljedeći način:

$$\epsilon\text{-OKRUŽENJE}(P) = \bigcup_{q \in P} \epsilon\text{-OKRUŽENJE}(q), \text{ gdje je } P \subseteq Q.$$

Funkcija $\hat{\delta}(q, w)$ jest proširenje funkcije prijelaza na argumente $\hat{\delta} : Q \times \Sigma^* \rightarrow 2^Q$. Funkcija $\hat{\delta}$ određuje skup stanja za koje postoji slijed prijelaza (uključujući i ϵ -prijelaze) iz stanja q za zadani niz znakova w . Funkcija $\hat{\delta}$ definira se pomoću funkcije ϵ -OKRUŽENJE:

- 1) $\hat{\delta}(q, \epsilon) = \epsilon\text{-OKRUŽENJE}\{q\}$, gdje je ϵ prazni niz,
- 2) $\hat{\delta}(q, wa) = \epsilon\text{-OKRUŽENJE}(P)$, gdje je $P = \{p \mid \text{za neko stanje } r \text{ iz } \hat{\delta}(q, w), p \text{ je iz } \delta(r, a)\}, w \in \Sigma^* \text{ i } a \in \Sigma$.

Za skupove stanja, funkcije δ i $\hat{\delta}$ računaju se na sljedeći način:

$$3) \quad \delta(R, a) = \bigcup_{q \in R} \delta(q, a), \text{ gdje je } R \subseteq Q \text{ i } a \in \Sigma,$$

$$4) \quad \hat{\delta}(R, w) = \bigcup_{q \in R} \hat{\delta}(q, w), \text{ gdje je } R \subseteq Q \text{ i } w \in \Sigma^*.$$

Skup $\delta(q_0, \varepsilon)$ nije nužno jednak skupu $\hat{\delta}(q_0, \varepsilon)$. Na primjer, za ε -NKA zadan na slikama 2.27 i 2.28 vrijedi:

$\delta(q_0, \varepsilon) = \{q_1\}$ iz retka q_0 i stupca ε tablice prijelaza na slici 2.28;

$$\hat{\delta}(q_0, \varepsilon) = \varepsilon\text{-OKRUŽENJE}(q_0) = \{q_0, q_1, q_2\}.$$

U funkciji δ oznaka ε označava posebni znak, a u funkciji $\hat{\delta}$ oznaka ε označava prazni niz.

Nadalje, skup $\delta(q_0, a)$ nije nužno jednak skupu $\hat{\delta}(q_0, a)$, gdje je a ulazni znak skupa Σ . Zato je potrebno razlikovati funkcije δ i $\hat{\delta}$. Na primjer, za ε -NKA zadan na slikama 2.27 i 2.28 vrijedi:

$\delta(q_0, 1) = \emptyset = \{\}$ iz retka q_0 i stupca 1 tablice prijelaza na slici 2.28;

$$\begin{aligned} \hat{\delta}(q_0, 1) &= \hat{\delta}(q_0, \varepsilon 1) = \varepsilon\text{-OKRUŽENJE}(\delta(\hat{\delta}(q_0, \varepsilon), 1)) = \varepsilon\text{-OKRUŽENJE}(\delta(\{q_0, q_1, q_2\}, 1)) \\ &= \varepsilon\text{-OKRUŽENJE}(\delta(q_0, 1) \cup \delta(q_1, 1) \cup \delta(q_2, 1)) \\ &= \varepsilon\text{-OKRUŽENJE}(\emptyset \cup \{q_1\} \cup \emptyset) = \varepsilon\text{-OKRUŽENJE}(\{q_1\}) \\ &= \varepsilon\text{-OKRUŽENJE}(q_1) = \{q_1, q_2\}. \end{aligned}$$

U funkciji δ argument 1 je ulazni znak za koji ne postoji niti jedan prijelaz iz stanja q_0 . U funkciji $\hat{\delta}$ argument 1 je niz duljine jedan za koji postoje više prijelaza. Na primjer, mogu se navesti sljedeći prijelazi za niz 1:

$$\begin{array}{ccc} \varepsilon & 1 & \varepsilon \\ q_0 \rightarrow q_1 \rightarrow q_1, & \text{te slijed} & q_0 \rightarrow q_1 \rightarrow q_1 \rightarrow q_2. \end{array}$$

Definira se da ε -NKA $M = (Q, \Sigma, \delta, q_0, F)$ prihvata jezik $L(M) = \{w \mid \hat{\delta}(q_0, w) \text{ sadrži barem jedno stanje skupa } F\}$.

Niz 01 se prihvata, jer je:

$$\hat{\delta}(q_0, \varepsilon) = \varepsilon\text{-OKRUŽENJE}(q_0) = \{q_0, q_1, q_2\},$$

$$\begin{aligned} \hat{\delta}(q_0, 0) &= \hat{\delta}(q_0, \varepsilon 0) = \varepsilon\text{-OKRUŽENJE}(\delta(\hat{\delta}(q_0, \varepsilon), 0)) = \varepsilon\text{-OKRUŽENJE}(\delta(\{q_0, q_1, q_2\}, 0)) \\ &= \varepsilon\text{-OKRUŽENJE}(\delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_2, 0)) \\ &= \varepsilon\text{-OKRUŽENJE}(\{q_0\} \cup \emptyset \cup \emptyset) = \varepsilon\text{-OKRUŽENJE}(\{q_0\}) \\ &= \varepsilon\text{-OKRUŽENJE}(q_0) = \{q_0, q_1, q_2\}, \end{aligned}$$

$$\begin{aligned} \hat{\delta}(q_0, 01) &= \varepsilon\text{-OKRUŽENJE}(\delta(\hat{\delta}(q_0, 0), 1)) = \varepsilon\text{-OKRUŽENJE}(\delta(\{q_0, q_1, q_2\}, 1)) \\ &= \varepsilon\text{-OKRUŽENJE}(\delta(q_0, 1) \cup \delta(q_1, 1) \cup \delta(q_2, 1)) \\ &= \varepsilon\text{-OKRUŽENJE}(\emptyset \cup \{q_1\} \cup \emptyset) = \varepsilon\text{-OKRUŽENJE}(\{q_1\}) \\ &= \varepsilon\text{-OKRUŽENJE}(q_1) = \{q_1, q_2\} \end{aligned}$$

Skup $\hat{\delta}(q_0, 01) = \{q_1, q_2\}$ sadrži prihvatljivo stanje q_2 i zato se niz 01 prihvata.

Niz 10 se ne prihvata, jer je:

$$\hat{\delta}(q_0, \varepsilon) = \varepsilon\text{-OKRUŽENJE}(q_0) = \{q_0, q_1, q_2\},$$

$$\begin{aligned}\hat{\delta}(q_0, 1) &= \hat{\delta}(q_0, \varepsilon 1) = \varepsilon\text{-OKRUŽENJE}(\hat{\delta}(q_0, \varepsilon), 1) = \varepsilon\text{-OKRUŽENJE}(\delta(\{q_0, q_1, q_2\}, 1)) \\ &= \varepsilon\text{-OKRUŽENJE}(\delta(q_0, 1) \cup \delta(q_1, 1) \cup \delta(q_2, 1)) \\ &= \varepsilon\text{-OKRUŽENJE}(\emptyset \cup \{q_1\} \cup \emptyset) = \varepsilon\text{-OKRUŽENJE}(\{q_1\}) \\ &= \varepsilon\text{-OKRUŽENJE}(q_1) = \{q_1, q_2\},\end{aligned}$$

$$\begin{aligned}\hat{\delta}(q_0, 10) &= \varepsilon\text{-OKRUŽENJE}(\delta(\hat{\delta}(q_0, 1), 0)) = \varepsilon\text{-OKRUŽENJE}(\delta(\{q_1, q_2\}, 0)) \\ &= \varepsilon\text{-OKRUŽENJE}(\delta(q_1, 0) \cup \delta(q_2, 0)) \\ &= \varepsilon\text{-OKRUŽENJE}(\emptyset \cup \emptyset) \\ &= \varepsilon\text{-OKRUŽENJE}(\emptyset) = \emptyset\end{aligned}$$

Skup $\hat{\delta}(q_0, 10) = \emptyset$ jest prazni skup i ne sadrži niti jedno prihvatljivo stanje. Zato se niz 10 ne prihvata.

Konstrukcija NKA iz zadanog ε -NKA

Za bilo koji ε -NKA $M = (Q, \Sigma, \delta, q_0, F)$ moguće je izgraditi istovjetni NKA $M' = (Q', \Sigma, \delta', q_0', F')$ na sljedeći način:

- 1) $Q' = Q$,
- 2) $q_0' = q_0$,
- 3) $F' = F \cup q_0$ ako $\varepsilon\text{-OKRUŽENJE}(q_0)$ sadrži barem jedno stanje skupa F , inače $F' = F$,
- 4) $\delta'(q, a) = \hat{\delta}(q, a)$, $\forall a \in \Sigma$ i $\forall q \in Q$.

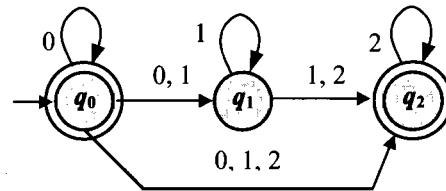
Primjer 2.3. Za ε -NKA zadan na slici 2.28 opisani postupak daje istovjetni NKA koji je prikazan tablicom prijelaza na slici 2.29, a dijagramom stanja na slici 2.30. NKA se gradi u četiri koraka:

- 1) $Q' = Q = \{q_0, q_1, q_2\}$,
 - 2) $q_0' = q_0$,
 - 3) $F' = F \cup \{q_0\} = \{q_2\} \cup \{q_0\} = \{q_0, q_2\}$, jer je $\varepsilon\text{-OKRUŽENJE}(q_0) \cap F = \{q_0, q_1, q_2\} \cap \{q_2\} = \{q_2\}$,
 - 4) funkcija δ' određuje se na sljedeći način:
- | | |
|--|--|
| $\delta'(q_0, 0) = \{q_0, q_1, q_2\}$, jer je $\hat{\delta}(q_0, 0) = \varepsilon\text{-OKRUŽENJE}(\delta(\hat{\delta}(q_0, \varepsilon), 0)) = \varepsilon\text{-OKRUŽENJE}(\delta(\varepsilon\text{-OKRUŽENJE}(q_0), 0))$ | $= \varepsilon\text{-OKRUŽENJE}(\delta(\{q_0, q_1, q_2\}, 0)) = \varepsilon\text{-OKRUŽENJE}(\{q_0\} \cup \emptyset \cup \emptyset) = \{q_0, q_1, q_2\}$ |
| $\delta'(q_0, 1) = \{q_1, q_2\}$, jer je $\hat{\delta}(q_0, 1) = \varepsilon\text{-OKRUŽENJE}(\delta(\hat{\delta}(q_0, \varepsilon), 1)) = \varepsilon\text{-OKRUŽENJE}(\delta(\varepsilon\text{-OKRUŽENJE}(q_0), 1))$ | $= \varepsilon\text{-OKRUŽENJE}(\delta(\{q_0, q_1, q_2\}, 1)) = \varepsilon\text{-OKRUŽENJE}(\emptyset \cup \{q_1\} \cup \emptyset) = \{q_1, q_2\}$ |
| $\delta'(q_0, 2) = \{q_2\}$, jer je $\hat{\delta}(q_0, 2) = \varepsilon\text{-OKRUŽENJE}(\delta(\hat{\delta}(q_0, \varepsilon), 2)) = \varepsilon\text{-OKRUŽENJE}(\delta(\varepsilon\text{-OKRUŽENJE}(q_0), 2))$ | $= \varepsilon\text{-OKRUŽENJE}(\delta(\{q_0, q_1, q_2\}, 2)) = \varepsilon\text{-OKRUŽENJE}(\emptyset \cup \emptyset \cup \{q_2\}) = \{q_2\}$ |
| $\delta'(q_1, 0) = \emptyset$, jer je $\hat{\delta}(q_1, 0) = \varepsilon\text{-OKRUŽENJE}(\delta(\hat{\delta}(q_1, \varepsilon), 0)) = \varepsilon\text{-OKRUŽENJE}(\delta(\varepsilon\text{-OKRUŽENJE}(q_1), 0))$ | $= \varepsilon\text{-OKRUŽENJE}(\delta(\{q_1, q_2\}, 0)) = \varepsilon\text{-OKRUŽENJE}(\emptyset \cup \emptyset) = \emptyset$ |

- $\delta'(q_1, 1) = \{q_1, q_2\}$, jer je $\hat{\delta}(q_1, 1) = \varepsilon\text{-OKRUŽENJE}(\delta(\hat{\delta}(q_1, \varepsilon), 1)) = \varepsilon\text{-OKRUŽENJE}(\delta(\varepsilon\text{-OKRUŽENJE}(q_1), 1)) = \varepsilon\text{-OKRUŽENJE}(\delta(\{q_1, q_2\}, 1)) = \varepsilon\text{-OKRUŽENJE}(\{q_1\} \cup \emptyset) = \{q_1, q_2\}$
- $\delta'(q_1, 2) = \{q_2\}$, jer je $\hat{\delta}(q_1, 2) = \varepsilon\text{-OKRUŽENJE}(\delta(\hat{\delta}(q_1, \varepsilon), 2)) = \varepsilon\text{-OKRUŽENJE}(\delta(\varepsilon\text{-OKRUŽENJE}(q_1), 2)) = \varepsilon\text{-OKRUŽENJE}(\delta(\{q_1, q_2\}, 2)) = \varepsilon\text{-OKRUŽENJE}(\emptyset \cup \{q_1\}) = \{q_2\}$
- $\delta'(q_2, 0) = \emptyset$, jer je $\hat{\delta}(q_2, 0) = \varepsilon\text{-OKRUŽENJE}(\delta(\hat{\delta}(q_2, \varepsilon), 0)) = \varepsilon\text{-OKRUŽENJE}(\delta(\varepsilon\text{-OKRUŽENJE}(q_2), 0)) = \varepsilon\text{-OKRUŽENJE}(\delta(\{q_2\}, 0)) = \varepsilon\text{-OKRUŽENJE}(\emptyset) = \emptyset$
- $\delta'(q_2, 1) = \emptyset$, jer je $\hat{\delta}(q_2, 1) = \varepsilon\text{-OKRUŽENJE}(\delta(\hat{\delta}(q_2, \varepsilon), 1)) = \varepsilon\text{-OKRUŽENJE}(\delta(\varepsilon\text{-OKRUŽENJE}(q_2), 1)) = \varepsilon\text{-OKRUŽENJE}(\delta(\{q_2\}, 1)) = \varepsilon\text{-OKRUŽENJE}(\emptyset) = \emptyset$
- $\delta'(q_2, 2) = \{q_2\}$, jer je $\hat{\delta}(q_2, 2) = \varepsilon\text{-OKRUŽENJE}(\delta(\hat{\delta}(q_2, \varepsilon), 2)) = \varepsilon\text{-OKRUŽENJE}(\delta(\varepsilon\text{-OKRUŽENJE}(q_2), 2)) = \varepsilon\text{-OKRUŽENJE}(\delta(\{q_2\}, 2)) = \varepsilon\text{-OKRUŽENJE}(\{q_2\}) = \{q_2\}$

	0	1	2	
q ₀	{q ₀ , q ₁ , q ₂ }	{q ₁ , q ₂ }	{q ₂ }	1
q ₁	∅	{q ₁ , q ₂ }	{q ₂ }	0
q ₂	∅	∅	{q ₂ }	1

Slika 2.29: Tablica prijelaza NKA konstruiranog na temelju ε -NKA zadano na slici 2.28



Slika 2.30: Dijagram stanja NKA konstruiranog na temelju ε -NKA zadano na slici 2.28

Istovjetnost NKA i ε -NKA

Dokazuje se indukcijom s obzirom na duljinu niza x . Dokaz se izvodi u dva koraka. U *koraku (i)* dokazuje se da vrijedi $\delta'(q_0, x) = \hat{\delta}(q_0, x)$, a u *koraku (ii)* dokazuje se da $\delta'(q_0, x)$ sadrži stanje iz skupa F' ako i samo ako $\hat{\delta}(q_0, x)$ sadrži stanje iz skupa F .

Korak (i):

Budući da je $\delta'(q_0, \varepsilon) = \{q_0\}$, a $\hat{\delta}(q_0, \varepsilon) = \varepsilon\text{-OKRUŽENJE}(q_0)$, tvrdnja $\delta'(q_0, x) = \hat{\delta}(q_0, x)$ nije nužno istinita za $x=\varepsilon$. Zato indukcija kreće od $|x|=1$:

- a) Za niz $|x|=1$, koji se sastoji samo od jednog znaka a , vrijedi da je $\delta'(q_0, a) = \hat{\delta}(q_0, a)$ na temelju same definicije (4) konstrukcije NKA.
- b) Neka je niz $x=wa$, gdje je $a \in \Sigma$ i $w \in \Sigma^*$. Želi se dokazati da je $\delta'(q_0, wa) = \hat{\delta}(q_0, wa)$. Prepostavi se da vrijedi induktivna hipoteza:

$$P = \delta'(q_0, w) = \hat{\delta}(q_0, w).$$

Dokaz započinje definicijom funkcije δ' NKA:

$$\delta'(q_0, wa) = \delta'(\delta'(q_0, w), a).$$

U gornji izraz uvrsti se jednakost induktivne hipoteze $\delta'(q_0, w)=P$:

$$\delta'(\delta'(q_0, w), a) = \delta'(P, a).$$

Na temelju definicije (3) funkcije δ' NKA vrijedi jednakost:

$$\delta'(P, a) = \bigcup_{q \in P} \delta'(q, a).$$

Na temelju definicije (4) konstrukcije NKA iz zadanog ε -NKA vrijedi jednakost:

$$\bigcup_{q \in P} \delta'(q, a) = \bigcup_{q \in P} \hat{\delta}(q, a).$$

Sada se uzme u obzir jednakost induktivne hipoteze $P = \hat{\delta}(q_0, w)$. P jest skup svih stanja u koja prelazi ε -NKA čitanjem znakova niza w . Na temelju definicije funkcije $\hat{\delta}$ ε -NKA vrijedi jednakost:

$$\bigcup_{q \in P} \hat{\delta}(q, a) = \hat{\delta}(P, a) = \hat{\delta}(\hat{\delta}(q_0, w), a) = \hat{\delta}(q_0, wa),$$

što dokazuje da je $\delta'(q_0, wa) = \hat{\delta}(q_0, wa)$.

Korak (ii):

Konačni automati prihvataju niz ako i samo ako u $\delta'(q_0, x)$, odnosno u $\hat{\delta}(q_0, x)$, postoji barem jedno stanje iz skupa dozvoljenih stanja. Zato se u nastavku dokazuje da $\delta'(q_0, x)$ sadrži stanje iz F' ako i samo ako $\hat{\delta}(q_0, x)$ sadrži stanje iz F .

Budući da je po definiciji $F' = F$, ili $F' = F \cup \{q_0\}$ ako ε -OKRUŽENJE(q_0) sadrži barem jedno stanje iz F , želi se dokazati da za bilo koji niz x za koji je q_0 iz $\delta'(q_0, x)$ vrijedi da je ε -OKRUŽENJE(q_0) $\subseteq \hat{\delta}(q_0, x)$.

Gornja tvrdnja jest istinita za prazni niz $x = \varepsilon$, budući da za prazni niz vrijedi da je $\hat{\delta}(q_0, \varepsilon) = \varepsilon$ -OKRUŽENJE(q_0).

Za niz $x \neq \varepsilon$, neka je $x = wa$, gdje je $a \in \Sigma$ i $w \in \Sigma^*$. Pretpostavimo da $\delta'(q_0, x)$ sadrži stanje q_0 . Želi se dokazati da $\hat{\delta}(q_0, x)$ sadrži sva stanja iz ε -OKRUŽENJE(q_0). Prethodno je dokazano da je $\delta'(q_0, x) = \hat{\delta}(q_0, x)$. Ako je q_0 iz $\delta'(q_0, x)$, onda je q_0 sigurno i u $\hat{\delta}(q_0, x)$. Budući da je q_0 iz $\hat{\delta}(q_0, x)$, a po definiciji $\hat{\delta}(q_0, x) = \varepsilon$ -OKRUŽENJE($\hat{\delta}(q_0, w), a$)) uključuje za svako stanje p iz $\hat{\delta}(q_0, x)$ ujedno i ε -OKRUŽENJE(p), onda je i ε -OKRUŽENJE(q_0) $\subseteq \hat{\delta}(q_0, x)$.

2.1.5 Konačni automati s izlazom

Izlaz konačnog automata ograničen je binarnom funkcijom koja označava da li se niz, ili bilo koji prefiks niza, prihvata ili ne prihvata. Funkcija izlaza konačnog automata proširuje se na dva osnovna načina: izlaz je funkcija stanja automata, ili izlaz je funkcija stanja i ulaznog znaka. Pridruži li se izlazna funkcija stanju konačnog automata, dobije se *Mooreov automat*. Pridruži li se izlazna funkcija stanju i ulaznom znaku, dobije se *Mealyev automat*. Za bilo koji Mealyev automat moguće je izgraditi istovjetni Mooreov automat, te je za bilo koji Mooreov automat moguće izgraditi istovjetni Mealyev automat. Dva automata su istovjetna ako za bilo koji ulazni niz daju isti izlazni niz.

Mooreov automat

Mooreov automat formalno se zadaje kao uređena šestorka:

$$MoDka = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$$

gdje je:

- | | |
|----------|----------------------------------|
| Q | - konačan skup stanja; |
| Σ | - konačan skup ulaznih znakova; |
| Δ | - konačan skup izlaznih znakova; |

- δ - funkcija prijelaza $Q \times \Sigma \rightarrow Q$;
 λ - funkcija izlaza $Q \rightarrow \Delta$;
 $q_0 \in Q$ - početno stanje.

Funkcija izlaza $\lambda(q)$ stanju q pridružuje jedan od izlaznih znakova iz skupa Δ .

Neka za ulazni niz $a_1 a_2 \dots a_n$ ($n \geq 0$ i $a_i \in \Sigma$) Mooreov automat daje slijed prijelaza:

$$\begin{array}{ccccccc} a_1 & a_2 & a_3 & & a_n \\ q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow \dots \rightarrow q_{n-1} \rightarrow q_n \end{array}$$

Za dani slijed prijelaza Mooreov automat daje sljedeći niz izlaznih znakova:

$$\lambda(q_0) \lambda(q_1) \lambda(q_2) \dots \lambda(q_{n-1}) \lambda(q_n).$$

Za prazni niz ϵ Mooreov automat daje izlaz $\lambda(q_0)$.

DKA $M' = (Q', \Sigma', \delta', q_0', F')$ jest poseban slučaj Mooreovog automata. DKA je Mooreov automat koji ima skup izlaznih znakova $\Delta = \{0, 1\}$. Ako je $q' \in F'$, onda je vrijednost funkcije izlaza $\lambda(q') = 1$. Ako $q' \notin F'$, vrijednost funkcije izlaza je $\lambda(q') = 0$. Niz se prihvata ako za stanje q' nakon posljednjeg pročitanog ulaznog znaka vrijedi $\lambda(q') = 1$.

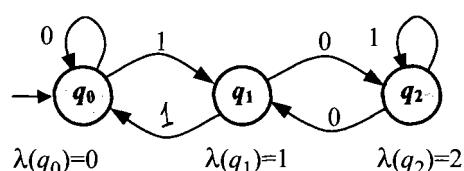
Primjer 2.4. Zadan je konačni skup ulaznih znakova $\Sigma = \{0, 1\}$. Niz je binarno zapisan cijeli broj. Potrebno je izgraditi Mooreov automat koji za pročitani prefiks ulaznog niza na izlazu daje ostatak dijeljenja tog prefiksa s tri.

Ako je niz $w \in \Sigma^*$ binarni zapis cijelog broja i , onda je niz $w0$ binarni zapis cijelog broja $2i$. Niz $w1$ je binarni zapis cijelog broja $2i+1$. U tablici 2.4 prikazani su ostaci dijeljenja niza w , $w0$ i $w1$ s 3.

w	$w0$	$w1$
$i/3$	$(2i)/3$	$(2i+1)/3$
0	0	1
1	2	0
2	1	2

Tablica 2.4: Ostatak dijeljenja niza w , $w0$ i $w1$ s 3
(w je binarni zapis cijelog broja i)

Dovoljno je izgraditi Mooreov automat s tri stanja: q_0 , q_1 i q_2 . Ako je ostatak dijeljenja 0, Mooreov automat je u stanju q_0 . Stanje q_1 označava da je ostatak dijeljenja 1, dok stanje q_2 označava da je ostatak dijeljenja 2. Izlazni znakovi skupa $\Delta = \{0, 1, 2\}$ označavaju ostatak dijeljenja. Funkcija izlaza je $\lambda(q_j) = j$, za $j = 0, 1, 2$. Slika 2.31 prikazuje dijagram stanja Mooreovog automata.



Slika 2.31: Dijagram stanja Mooreovog automata

Slika 2.32 prikazuje slijed prijelaza i niz izlaznih znakova za ulazni niz 1010.

<i>Slijed prijelaza</i>	q_0	\rightarrow	1	q_1	\rightarrow	0	q_2	\rightarrow	1	q_2	\rightarrow	0	q_1
<i>Izlazni niz</i>	$\lambda(q_0)=0$			$\lambda(q_1)=1$			$\lambda(q_2)=2$			$\lambda(q_2)=2$			$\lambda(q_1)=1$
<i>Pročitani prefiks ulaznog niza</i>	ϵ			1			10			101			1010
<i>Cjelobrojna vrijednost prefiksa po bazi 10</i>				1			2			5			10

Slika 2.32: Izlazni niz znakova za ulazni niz 1010 za Mooreov automat zadan na slici 2.31

Mealyev automat

Mealyev automat formalno se zadaje kao uređena šestorka:

$$MeDka = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$$

gdje je:

- Q - konačan skup stanja;
- Σ - konačan skup ulaznih znakova;
- Δ - konačan skup izlaznih znakova;
- δ - funkcija prijelaza $Q \times \Sigma \rightarrow Q$;
- λ - funkcija izlaza $Q \times \Sigma \rightarrow \Delta$;
- $q_0 \in Q$ - početno stanje.

Funkcija izlaza $\lambda(q, a)$ prijelazu $\delta(q, a)$ pridružuje jedan od izlaznih znakova iz skupa Δ .

Neka za ulazni niz $a_1a_2 \dots a_n$ ($n \geq 1$ i $a_i \in \Sigma$) Mealyev automat daje slijed prijelaza:

$$\begin{array}{ccccccc} a_1 & a_2 & a_3 & & a_n \\ q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow \dots \rightarrow q_{n-1} \rightarrow q_n \end{array}$$

Za dani slijed prijelaza Mealyev automat daje sljedeći niz izlaznih znakova:

$$\lambda(q_0, a_1) \ \lambda(q_1, a_2) \ \lambda(q_2, a_3) \ \dots \ \lambda(q_{n-1}, a_n).$$

Za prazni niz ϵ Mealyev automat ne daje nikakav izlaz.

Konstrukcija Mealyevog automata iz zadanog Mooreovog automata

Za prazni niz ϵ postoji izlaz za Mooreov automat, ali ne postoji izlaz za Mealyev automat. Zato se istovjetnost Mealyevog automata M' i Mooreovog automata M definira na sljedeći način:

$$b T_{M'}(w) = T_M(w),$$

gdje je w niz ulaznih znakova, b je izlaz Mooreovog automata za prazni niz $b = \lambda(q_0)$, $T_{M'}(w)$ je izlazni niz Mealyevog automata i $T_M(w)$ je izlazni niz Mooreovog automata.

Neka je zadan Mooreov automat $M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$. Istovjetni Mealyev automat $M' = (Q, \Sigma, \Delta, \delta', \lambda', q_0)$ gradi se promjenom funkcije izlaza:

- 1) $\lambda'(q, a) = \lambda(\delta(q, a))$, za sve q iz Q i za sve a iz Σ .

Oba automata imaju istu funkciju prijelaza δ . Za isti ulazni niz w , oba automata prolaze istim slijedom prijelaza. Budući da je funkcija prijelaza Mealyevog automata definirana tako da se prijelazu iz stanja q u stanje $\delta(q, a)$ dodijeli upravo izlaz $\lambda(\delta(q, a))$ koji ima Mooreov automat u stanju $\delta(q, a)$, zaključuje se da su ta dva automata istovjetna. Uvezši u obzir način gradnje funkcije λ' Mealyevog automata M' , i uvezši u obzir da oba automata prolaze istim slijedom stanja q_0, q_1, \dots, q_n , izgrađeni Mealyev automat M' za ulazni niz $w = a_1 a_2 \dots a_n$ ($n \geq 1$ i $a_i \in \Sigma$) daje sljedeći izlaz:

$$\begin{aligned}\lambda'(q_0, a_1) \lambda'(q_1, a_2) \dots \lambda'(q_{n-1}, a_n) &= \lambda(\delta(q_0, a_1)) \lambda(\delta(q_1, a_2)) \dots \lambda(\delta(q_{n-1}, a_n)) \\ &= \lambda(q_1) \lambda(q_2) \dots \lambda(q_n).\end{aligned}$$

Dodavanjem $\lambda(q_0)$ gornjem nizu dobije se upravo izlazni niz izvornog Moorevog automata M .

Primjer 2.5. Za Mooreov automat:

$$\begin{aligned}M = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, 1, 2\}, \\ \{\delta(q_0, 0)=q_0, \delta(q_0, 1)=q_1, \delta(q_1, 0)=q_2, \delta(q_1, 1)=q_0, \delta(q_2, 0)=q_1, \delta(q_2, 1)=q_2\}, \\ \{\lambda(q_0)=0, \lambda(q_1)=1, \lambda(q_2)=2\}, q_0)\end{aligned}$$

koji je izgrađen u primjeru 2.4 potrebno je izgraditi Mealyev automat $M'=(Q, \Sigma, \Delta, \delta, \lambda', q_0)$. Funkcija izlaza λ' računa se na sljedeći način:

$$\begin{aligned}\lambda'(q_0, 0)=0, \quad \text{jer je } \lambda'(q_0, 0)=\lambda(\delta(q_0, 0))=\lambda(q_0)=0 \\ \lambda'(q_0, 1)=1, \quad \text{jer je } \lambda'(q_0, 1)=\lambda(\delta(q_0, 1))=\lambda(q_1)=1 \\ \lambda'(q_1, 0)=2, \quad \text{jer je } \lambda'(q_1, 0)=\lambda(\delta(q_1, 0))=\lambda(q_2)=2 \\ \lambda'(q_1, 1)=0, \quad \text{jer je } \lambda'(q_1, 1)=\lambda(\delta(q_1, 1))=\lambda(q_0)=0 \\ \lambda'(q_2, 0)=1, \quad \text{jer je } \lambda'(q_2, 0)=\lambda(\delta(q_2, 0))=\lambda(q_1)=1 \\ \lambda'(q_2, 1)=2, \quad \text{jer je } \lambda'(q_2, 1)=\lambda(\delta(q_2, 1))=\lambda(q_2)=2\end{aligned}$$

Na slici 2.33 prikazan je slijed prijelaza i niz izlaznih znakova za ulazni niz 1010.

Slijed prijelaza	q_0	$\xrightarrow{1}$	q_1	$\xrightarrow{0}$	q_2	$\xrightarrow{1}$	q_2	$\xrightarrow{0}$	q_1
Izlazni niz		$\lambda(q_0, 1)=1$		$\lambda(q_1, 0)=2$		$\lambda(q_2, 1)=2$		$\lambda(q_2, 0)=1$	
Pročitani prefiks ulaznog niza	ϵ		1		10		101		1010
Cjelobrojna vrijednost prefiksa po bazi 10			1		2		5		10

Slika 2.33: Izlazni niz znakova za ulazni niz 1010 za Mealyev automat izgrađen u primjeru 2.5
(Mealyev automat istovjetan je Moorevom automatu koji je zadan slikom 2.31)

Konstrukcija Mooreovog automata iz zadanog Mealyevog automata

Neka je zadan Mealyev automat $M=(Q, \Sigma, \Delta, \delta, \lambda, q_0)$. Moguće je izgraditi istovjetni Mooreov automat $M'=(Q', \Sigma, \Delta, \delta', \lambda', q_0')$ na sljedeći način:

- 1) $Q'=Q \times \Delta$, gdje je stanje $[q, b] \in Q'$, $q \in Q$ i $b \in \Delta$,
- 2) $q_0'=[q_0, b_0]$, gdje je b_0 proizvoljni element skupa Δ ,
- 3) $\delta'([q, b], a)=[\delta(q, a), \lambda(q, a)]$, gdje je $q \in Q$, $b \in \Delta$ i $a \in \Sigma$,
- 4) $\lambda'([q, b])=b$, gdje $q \in Q$ i $b \in \Delta$.

Ako Mealyev automat M za ulazni niz $w = a_1 a_2 \dots a_n$ ($n \geq 1$ i $a_i \in \Sigma$) prolazi slijedom prijelaza $q_0, \delta(q_0, a_1), \delta(q_1, a_2), \dots, \delta(q_{n-1}, a_n) = q_0, q_1, \dots, q_n$ i daje izlazni niz:

$$\lambda(q_0, a_1) \lambda(q_1, a_2) \dots \lambda(q_{n-1}, a_n) = b_1 b_2 \dots b_n,$$

onda izgrađeni Mooreov automat M' prolazi slijedom prijelaza:

$$\begin{aligned} [q_0, b_0], \delta'([q_0, b_0], a_1), \dots, \delta'([q_{n-1}, b_{n-1}], a_n) &= \\ [q_0, b_0], [\delta([q_0, a_1], \lambda(q_0, a_1)], \dots, [\delta([q_{n-1}, a_n], \lambda(q_{n-1}, a_n)] &= \\ [q_0, b_0], [q_1, b_1], \dots, [q_n, b_n] \end{aligned}$$

i daje isti niz izlaznih znakova kao i Mealyev automat uz dodatak b_0 :

$$\lambda'([q_0, b_0] \lambda'([q_1, b_1] \dots \lambda'([q_n, b_n] = b_0 b_1 b_2 \dots b_n.$$

Primjer 2.6. Za Mealyev automat:

$$\begin{aligned} M = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, 1, 2\}, \\ \{\delta(q_0, 0)=q_0, \delta(q_0, 1)=q_1, \delta(q_1, 0)=q_2, \delta(q_1, 1)=q_0, \delta(q_2, 0)=q_1, \delta(q_2, 1)=q_2\}, \\ \{\lambda(q_0, 0)=0, \lambda(q_0, 1)=1, \lambda(q_1, 0)=2, \lambda(q_1, 1)=0, \lambda(q_2, 0)=1, \lambda(q_2, 1)=2\}, q_0) \end{aligned}$$

koji je izgrađen u primjeru 2.5 potrebno je konstruirati Mooreov automat $M'=(Q, \Sigma, \Delta, \delta, \lambda', q_0)$. Skup stanja Q' je:

k1) $Q'=\{[q_0, 0], [q_0, 1], [q_0, 2], [q_1, 0], [q_1, 1], [q_1, 2], [q_2, 0], [q_2, 1], [q_2, 2]\},$

početno stanje q_0' je:

k2) $q_0'=[q_0, 0],$

funkcija prijelaza δ' je:

k3)

$\delta'([q_0, 0], 0)=[q_0, 0]$	jer je	$\delta'([q_0, 0], 0)=[\delta(q_0, 0), \lambda(q_0, 0)]=[q_0, 0]$
$\delta'([q_0, 0], 1)=[q_1, 1]$	jer je	$\delta'([q_0, 0], 1)=[\delta(q_0, 1), \lambda(q_0, 1)]=[q_1, 1]$
$\delta'([q_0, 1], 0)=[q_0, 0]$	jer je	$\delta'([q_0, 1], 0)=[\delta(q_0, 0), \lambda(q_0, 0)]=[q_0, 0]$
$\delta'([q_0, 1], 1)=[q_1, 1]$	jer je	$\delta'([q_0, 1], 1)=[\delta(q_0, 1), \lambda(q_0, 1)]=[q_1, 1]$
$\delta'([q_0, 2], 0)=[q_0, 0]$	jer je	$\delta'([q_0, 2], 0)=[\delta(q_0, 0), \lambda(q_0, 0)]=[q_0, 0]$
$\delta'([q_0, 2], 1)=[q_1, 1]$	jer je	$\delta'([q_0, 2], 1)=[\delta(q_0, 1), \lambda(q_0, 1)]=[q_1, 1]$
$\delta'([q_1, 0], 0)=[q_2, 2]$	jer je	$\delta'([q_1, 0], 0)=[\delta(q_1, 0), \lambda(q_1, 0)]=[q_2, 2]$
$\delta'([q_1, 0], 1)=[q_0, 0]$	jer je	$\delta'([q_1, 0], 1)=[\delta(q_1, 1), \lambda(q_1, 1)]=[q_0, 0]$
$\delta'([q_1, 1], 0)=[q_2, 2]$	jer je	$\delta'([q_1, 1], 0)=[\delta(q_1, 0), \lambda(q_1, 0)]=[q_2, 2]$
$\delta'([q_1, 1], 1)=[q_0, 0]$	jer je	$\delta'([q_1, 1], 1)=[\delta(q_1, 1), \lambda(q_1, 1)]=[q_0, 0]$
$\delta'([q_1, 2], 0)=[q_2, 2]$	jer je	$\delta'([q_1, 2], 0)=[\delta(q_1, 0), \lambda(q_1, 0)]=[q_2, 2]$
$\delta'([q_1, 2], 1)=[q_0, 0]$	jer je	$\delta'([q_1, 2], 1)=[\delta(q_1, 1), \lambda(q_1, 1)]=[q_0, 0]$
$\delta'([q_2, 0], 0)=[q_1, 1]$	jer je	$\delta'([q_2, 0], 0)=[\delta(q_2, 0), \lambda(q_2, 0)]=[q_1, 1]$
$\delta'([q_2, 0], 1)=[q_2, 2]$	jer je	$\delta'([q_2, 0], 1)=[\delta(q_2, 1), \lambda(q_2, 1)]=[q_2, 2]$
$\delta'([q_2, 1], 0)=[q_1, 1]$	jer je	$\delta'([q_2, 1], 0)=[\delta(q_2, 0), \lambda(q_2, 0)]=[q_1, 1]$
$\delta'([q_2, 1], 1)=[q_2, 2]$	jer je	$\delta'([q_2, 1], 1)=[\delta(q_2, 1), \lambda(q_2, 1)]=[q_2, 2]$
$\delta'([q_2, 2], 0)=[q_1, 1]$	jer je	$\delta'([q_2, 2], 0)=[\delta(q_2, 0), \lambda(q_2, 0)]=[q_1, 1]$
$\delta'([q_2, 2], 1)=[q_2, 2]$	jer je	$\delta'([q_2, 2], 1)=[\delta(q_2, 1), \lambda(q_2, 1)]=[q_2, 2]$

i funkcija izlaza λ' je:

k4)

$$\begin{array}{lll} \lambda'([q_0, 0])=0 & \lambda'([q_1, 0])=0 & \lambda'([q_2, 0])=0 \\ \lambda'([q_0, 1])=1 & \lambda'([q_1, 1])=1 & \lambda'([q_2, 1])=1 \\ \lambda'([q_0, 2])=2 & \lambda'([q_1, 2])=2 & \lambda'([q_2, 2])=2 \end{array}$$

Izbacivanjem nedohvatljivih stanja $[q_0, 1]$, $[q_0, 2]$, $[q_1, 0]$, $[q_1, 2]$, $[q_2, 0]$ i $[q_2, 1]$ dobije se Mooreov automat koji je isti kao i Mooreov automat na slici 2.31. Okvirene su one funkcije prijelaza δ' koje su zadane nad dohvatiljivim stanjima.

2.2 Regularni izrazi

Regularni izrazi koriste se za opis regularnih jezika. U odjeljku 2.2.1 definirani su regularni izrazi, a u odjeljku 2.2.2 dat je algoritam konstrukcije konačnog automata za jezik zadan regularnim izrazima. Odjeljak 2.2.3 opisuje osnovne procese generatora konačnog automata.

2.2.1 Definicija regularnih izraza

Regularni jezik moguće je opisati regularnim izrazima. Dva su osnovna razloga uporabe regularnih izraza:

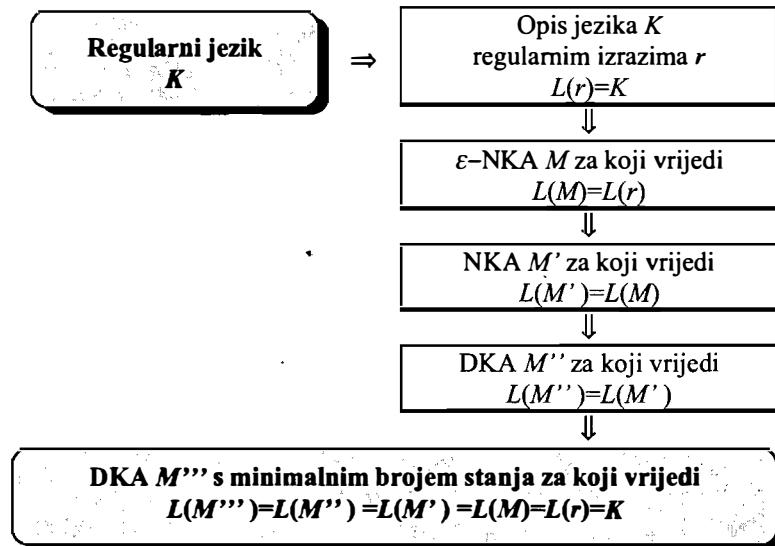
- 1) Ako je neki jezik moguće opisati regularnim izrazima, onda je taj jezik sigurno regularan. Neka $L(r)$ označava jezik definiran regularnim izrazima r . Za bilo koji jezik $L(r)$ zadan regularnim izrazima r moguće je izgraditi DKA M za koji vrijedi $L(M)=L(r)$.

Na primjer, jezik $N_1=\{aaca, abcba, bacab, bbcb\}$ jest skup nizova oblika wcw^R , gdje su w i w^R nizovi znakova a i b . Podniz w jednak je aa , ab , ba ili aa . Znakovi podniza w^R jednaki su znakovima podniza w , ali su napisani obrnutim redoslijedom. Jezik N_1 jest regularan i za njega je moguće izgraditi konačni automat. Čitanjem znakova podniza w konačni automat prelazi u jedno od četiri stanja. Stanje konačnog automata određuju pročitani znakovi podniza w koji mogu biti aa , ab , ba ili aa . Ovisno o kojem je stanju, konačni automat provjerava da li iza znaka c slijedi odgovarajući podniz znakova w^R .

Međutim, jezik $N=\{wcw^R \mid w \text{ i } w^R \text{ su podnizovi znakova } a \text{ i } b, w \text{ i } w^R \text{ su proizvoljne duljine}\}$ nije regularan. Ograničenja modela konačnog automata, opisana u odjeljku 2.1.1, onemogućuju gradnju DKA koji prihvata jezik N . Konačni skup stanja onemogućuje definiranje zasebnog stanja za svaki pojedini podniz w , budući da su podnizovi w proizvoljne duljine. Nadalje, glava za čitanje miče se samo u desno i nije moguće pisati po traci, što onemogućuje usporedbu znakova podnizova w i w^R na samoj ulaznoj traci. Automati, kao što su potisni automat i Turingov stroj, proširuju model konačnog automata funkcijama koje omogućuju prihvatanje jezika N .

Budući da jezik N nije regularan, zaključuje se da su regularni jezici pravi podskup skupa svih jezika. Za regularne jezike moguće je izgraditi DKA i oni se mogu opisati regularnim izrazima. Za ostale jezike, koji nisu regularni, nije moguće izgraditi konačni automat i oni se ne mogu opisati regularnim izrazima.

- 2) Postoji algoritam za pretvorbu regularnih izraza u ϵ -NKA. Na slici 2.34 prikazan je postupak gradnje DKA s minimalnim brojem stanja na temelju zadanih regularnih izraza.



Slika 2.34: Konstrukcija DKA s minimalnim brojem stanja za regularni jezik K

Neka je Σ abeceda. Regularni izrazi definiraju se rekurzivno nad abecedom Σ . Uz svako rekurzivno pravilo naveden je i jezik određen tim pravilom. Rekurzivna pravila za regularne izraze su:

- 1) \emptyset jest regularni izraz i označava jezik $L(\emptyset)=\{\}$.
- 2) ϵ jest regularni izraz i označava jezik $L(\epsilon)=\{\epsilon\}$.
- 3) $\forall a \in \Sigma, a$ jest regularni izraz i označava jezik $L(a)=\{a\}$. Treba biti pažljiv, jer je istom oznakom a označen znak abecede Σ , regularni izraz i niz jedinične duljine koji je element jezika $L(a)$.
- 4) Ako su r i s regularni izrazi koji označavaju jezike $L(r)$ i $L(s)$, onda:
 - a) $(r)+(s)$ jest regularni izraz koji označava jezik $L((r)+(s))=L(r) \cup L(s)$. Jezik $L((r)+(s))$ nastaje unijom jezika $L(r)$ i $L(s)$. Često se koristi i oznaka $(r) | (s)$. Druga oznaka koristi se za definiranje aritmetičkih izraza kako bi se ta oznaka razlikovala od oznake aritmetičkog operatora zbrajanja $+$.
 - b) $(r)(s)$ jest regularni izraz koji označava jezik $L((r)(s))=L(r)L(s)$. Jezik $L((r)(s))$ nastaje nadovezivanjem jezika $L(r)$ i $L(s)$.
 - c) $(r)^*$ jest regularni izraz koji označava jezik $L((r)^*)=L(r)^*$. Jezik $L((r)^*)$ nastaje primjenom Kleeneovog operatora nad jezikom $L(r)$.

Pravila asocijativnosti i prednosti za operatore definiraju se na sljedeći način:

- 1) unarni operator $*$ jest lijevo asocijativan i ima najveću prednost;
- 2) operator nadovezivanja jest lijevo asocijativan i ima veću prednost od operatora $+$;
- 3) operator $+$ jest lijevo asocijativan i ima najmanju prednost.

Primjer 2.7. Za binarnu abecedu $\Sigma=\{0, 1\}$ moguće je dati sljedeće primjere regularnih izraza i jezika:

- 1) Regularni izraz 01 definira jezik: $L(01)=\{01\}$.
- 2) Regularni izraz 0+1 definira jezik: $L(0+1)=\{0, 1\}$.
- 3) Regularni izraz (0+1)(0+1) definira jezik: $L((0+1)(0+1))=\{00, 01, 10, 11\}$.
- 4) Regularni izraz 1* definira jezik u kojem su prazni niz i svi nizovi koji se sastoje od proizvoljnog broja znakova 1: $L(1^*)=\{\epsilon, 1, 11, 111, \dots, 11111111, \dots\}$.
- 5) Regularni izraz (0+1)* definira jezik u kojem su prazni niz i svi nizovi znakova 0 i 1: $L(0+1)^*=\{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, \dots, 01111101, \dots\}$.
- 6) Regularni izraz (0+1)*00(0+1)* definira jezik u kojem bilo koji niz barem na jednom mjestu ima najmanje dva uzastopna znaka 0: $L((0+1)^*00(0+1)^*)=\{00, 000, 100, 001, 0000, 0100, 1000, 1100, 0001, 0010, 0011, \dots, 01001111001, \dots\}$.
- 7) Regularni izraz 0*1* definira jezik u kojem su prazni niz i nizovi u kojima iza proizvoljnog broja znakova 0 slijedi proizvoljni broj znakova 1: $L(0^*1^*)=\{\epsilon, 0, 1, 00, 01, 000, 001, 011, 111, \dots, 0001111111, \dots\}$.

Dva regularna izraza r i s su istovjetna ako označavaju iste jezike, tj. ako vrijedi $L(r)=L(s)$. Istovjetnost dvaju regularnih izraza r i s zapiše se na sljedeći način: $r=s$. Uzmu li se u obzir prednosti operatora i zakoni asocijativnosti, regularni izraz $(a)+(b)*(c)$ označava isti jezik kao i regularni izraz $a+b*c$, tj. $(a)+(b)*(c)=a+b*c$. Izrazi označavaju jezik koji se sastoji od niza a , niza c i nizova čiji je prefiks proizvoljni broj znakova b iza kojih slijedi jedan znak c : $L((a)+(b)*(c)))=L(a+b*c)=\{a, c, bc, bbc, bbb\dots, b\dots bc, \dots\}$.

U tablici 2.5 opisani su algebarski zakoni koji vrijede za regularne izraze.

$r+s=s+r$	+ jest komutativno
$r+(s+t) = (r+s)+t$	+ jest asocijativno
$(rs)t = r(st)$	nadovezivanje jest asocijativno
$r(s+t) = rs+rt$ $(s+t)r = sr+tr$	distributivnost nadovezivanja nad +
$\epsilon r = r\epsilon = r$	ϵ jest neutralni element za nadovezivanje
$r^* = (r+\epsilon)^*$	relacija između + i *
$r^{**} = r^*$	idempotentnost

Tablica 2.5: Algebarski zakoni koji vrijede za regularne izraze (r, s i t su regularni izrazi)

2.2.2 Konstrukcija ϵ -NKA na temelju zadanih regularnih izraza

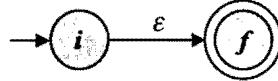
Za bilo koji regularni izraz r moguće je izgraditi ϵ -NKA M tako da vrijedi $L(M)=L(r)$:

- p1) Za regularni izraz \emptyset koji definira jezik $L(\emptyset)=\{\}$ konstruira se ϵ -NKA $M=(\{i, f\}, \Sigma, \{\}, i, \{f\})$. Početno stanje jest i , a prihvatljivo stanje jest $f \in F$. Dijagram stanja izgrađenog ϵ -NKA je:



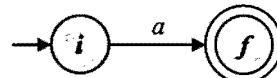
Napomena. Za bilo koji $b \in (\Sigma \cup \{\epsilon\})$, skup $\delta(i, b)$ jest prazan skup. Ne postoji niti jedan prijelaz u prihvatljivo stanje f . Budući da je početno stanje i neprihvatljivo, ϵ -NKA M ne prihvata niti jedan niz, uključujući i prazni niz ϵ .

- p2) Za regularni izraz ϵ koji definira jezik $L(\epsilon)=\{\epsilon\}$ konstruira se ϵ -NKA $M=(\{i, f\}, \Sigma, \{\delta(i, \epsilon)=f\}, i, \{f\})$. Početno stanje jest i , a prihvatljivo stanje jest $f \in F$. Dijagram stanja izgrađenog ϵ -NKA je:



Napomena. Za bilo koji $b \in \Sigma$, skup $\delta(f, b)$ jest prazan skup. Prijelaz $\delta(i, \epsilon)=\{f\}$ omogućuje prihvatanje praznog niza. ϵ -NKA M ne prihvata niti jedan niz, osim praznog niza ϵ .

- p3) Za regularni izraz a koji definira jezik $L(a)=\{a\}$ konstruira se ϵ -NKA $M=(\{i, f\}, \Sigma, \{\delta(i, a)=f\}, i, \{f\})$. Početno stanje jest i , a prihvatljivo stanje jest $f \in F$. Dijagram stanja izgrađenog ϵ -NKA je:

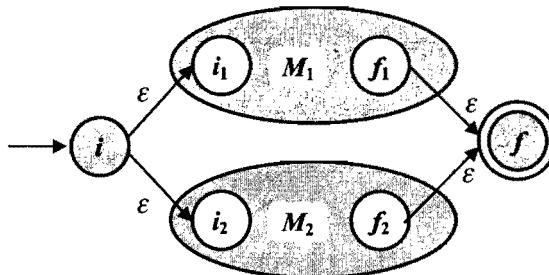


Napomena. Za bilo koji $b \in (\Sigma \cup \{\epsilon\})$ za koji vrijedi $b \neq a$, skup $\delta(f, b)$ jest prazan skup. Prijelaz $\delta(i, a)=\{f\}$ omogućuje prihvatanje niza a . ϵ -NKA M ne prihvata niti jedan niz, osim niza a . Ne prihvata se ni prazni niz ϵ .

- p4) Za regularni izraz r_1+r_2 koji definira jezik $L(r_1+r_2)=L(r_1) \cup L(r_2)$ konstruira se ϵ -NKA M na sljedeći način. Pretpostavimo da su prethodno izgrađeni ϵ -NKA $M_1=(Q_1, \Sigma_1, \delta_1, i_1, \{f_1\})$ i $M_2=(Q_2, \Sigma_2, \delta_2, i_2, \{f_2\})$ takvi da vrijedi $L(M_1)=L(r_1)$ i $L(M_2)=L(r_2)$ i da nema prijelaza iz stanja f_1 i f_2 niti za jedan ulazni znak (tj. $\delta_1(f_1, a)=\emptyset, \forall a \in (\Sigma_1 \cup \{\epsilon\})$ i $\delta_2(f_2, b)=\emptyset, \forall b \in (\Sigma_2 \cup \{\epsilon\})$). Promjenom imena stanja u Q_1 i Q_2 postiže se da je $Q_1 \cap Q_2 = \emptyset$. Konstruira se ϵ -NKA $M=(Q_1 \cup Q_2 \cup \{i, f\}, \Sigma_1 \cup \Sigma_2, \delta, i, \{f\})$. Novo početno stanje jest i , a novo prihvatljivo stanje jest f . Stanja i_1 i i_2 nisu više početna stanja, te stanja f_1 i f_2 nisu više prihvatljiva. Funkcija δ određuje se na sljedeći način:

- $\delta(i, \epsilon)=\{i_1, i_2\}$,
- $\delta(q, a)=\delta_1(q, a), \forall q \in (Q_1 - \{f_1\})$ i $\forall a \in (\Sigma_1 \cup \{\epsilon\})$,
- $\delta(q, b)=\delta_2(q, b), \forall q \in (Q_2 - \{f_2\})$ i $\forall b \in (\Sigma_2 \cup \{\epsilon\})$,
- $\delta(f_1, \epsilon)=\delta(f_2, \epsilon)=\{f\}$

Dijagram stanja izgrađenog ϵ -NKA M koji prihvata jezik $L(M)=L(r_1+r_2)$ je:

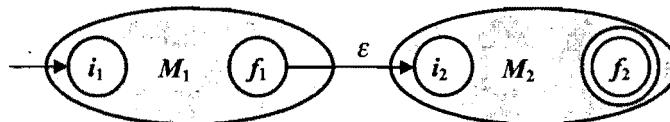


- p5) Za regularni izraz r_1r_2 koji definira jezik $L(r_1r_2)=L(r_1)L(r_2)$ konstruira se ϵ -NKA M na sljedeći način. Pretpostavimo da su prethodno izgrađeni ϵ -NKA $M_1=(Q_1, \Sigma_1, \delta_1, i_1, \{f_1\})$ i $M_2=(Q_2, \Sigma_2, \delta_2, i_2, \{f_2\})$ takvi da vrijedi $L(M_1)=L(r_1)$ i $L(M_2)=L(r_2)$ i da

nema prijelaza iz stanja f_1 i f_2 niti za jedan ulazni znak (tj. $\delta_1(f_1, a)=\emptyset$, $\forall a \in (\Sigma_1 \cup \{\epsilon\})$ i $\delta_2(f_2, b)=\emptyset$, $\forall b \in (\Sigma_2 \cup \{\epsilon\})$). Promjenom imena stanja u Q_1 i Q_2 postiže se da je $Q_1 \cap Q_2 = \emptyset$. Konstruira se ϵ -NKA $M=(Q_1 \cup Q_2, \Sigma_1 \cup \Sigma_2, \delta, i_1, \{f_2\})$. Novo početno stanje jest i_1 , a novo prihvativivo stanje jest f_2 . Stanje i_2 nije više početno stanje, te stanje f_1 nije više prihvativivo. Funkcija δ određuje se na sljedeći način:

- $\delta(q, a) = \delta_1(q, a)$, $\forall q \in (Q_1 - \{f_1\})$ i $\forall a \in (\Sigma_1 \cup \{\epsilon\})$,
- $\delta(q, b) = \delta_2(q, b)$, $\forall q \in Q_2$ i $\forall b \in (\Sigma_2 \cup \{\epsilon\})$,
- $\delta(f_1, \epsilon) = \{i_2\}$

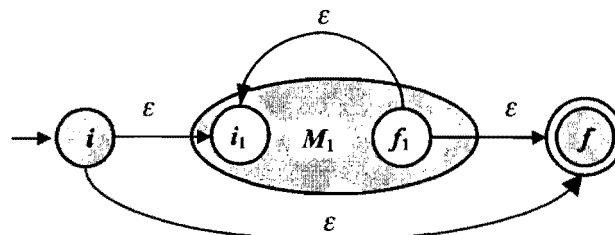
Dijagram stanja izgrađenog ϵ -NKA M koji prihvata jezik $L(M)=L(r_1 r_2)$ je:



- p6) Za regularni izraz r_1^* koji definira jezik $L(r_1^*)=L(r_1)^*$ konstruira se ϵ -NKA M na sljedeći način. Pretpostavimo da je prethodno izgrađeni ϵ -NKA $M_1=(Q_1, \Sigma_1, \delta_1, i_1, \{f_1\})$ za koji vrijedi $L(M_1)=L(r_1)$ i nema prijelaza iz stanja f_1 niti za jedan ulazni znak (tj. $\delta_1(f_1, a)=\emptyset$, $\forall a \in (\Sigma_1 \cup \{\epsilon\})$). Konstruira se ϵ -NKA $M=(Q_1 \cup \{i, f\}, \Sigma_1, \delta, i, \{f\})$. Novo početno stanje jest i , a novo prihvativivo stanje jest f . Stanje i_1 nije više početno stanje, te stanje f_1 nije više prihvativivo. Funkcija δ određuje se na sljedeći način:

- $\delta(i, \epsilon) = \delta(f_1, \epsilon) = \{i_1, f\}$,
- $\delta(q, a) = \delta_1(q, a)$, $\forall q \in (Q_1 - \{f_1\})$ i $\forall a \in (\Sigma_1 \cup \{\epsilon\})$,

Dijagram stanja izgrađenog ϵ -NKA M koji prihvata jezik $L(M)=L(r_1^*)$ je:

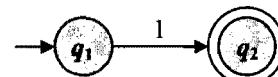


- p7) Budući da je $L((r))=L(r)$, za ϵ -NKA M regularnog izraza (r) uzima se ϵ -NKA M_1 regularnog izraza r , jer je $L(M_1)=L(r)=L((r))$.

Primjer 2.8. Neka se izgradi ϵ -NKA za regularni izraz $r=01^*+1$:

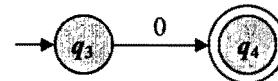
- 1) Regularni izraz 01^*+1 rastavi se na izraze $r=r_1+r_2$, gdje su $r_1=01^*$ i $r_2=1$.

- 2) Za $r_2=1$ konstruira se sljedeći ϵ -NKA:



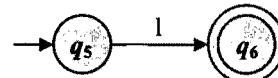
- 3) Regularni izraz $r_1=01^*$ rastavi se na izraze $r_1=r_3 r_4$, gdje su $r_3=0$ i $r_4=1^*$.

- 4) Za $r_3=0$ konstruira se sljedeći ϵ -NKA:

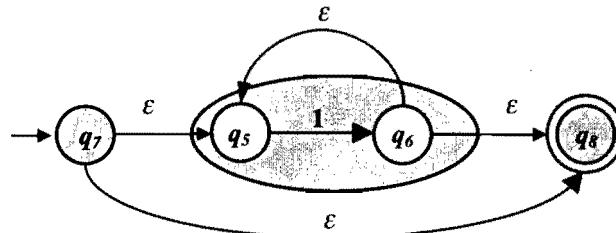


- 6) Regularni izraz $r_4=1^*$ rastavi se na izraze $r_4=r_5^*$, gdje je $r_5=1$.

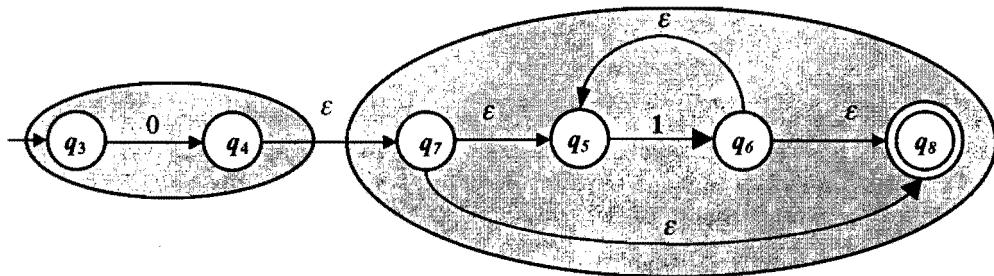
- 7) Za $r_5=1$ konstruira se sljedeći ϵ -NKA:



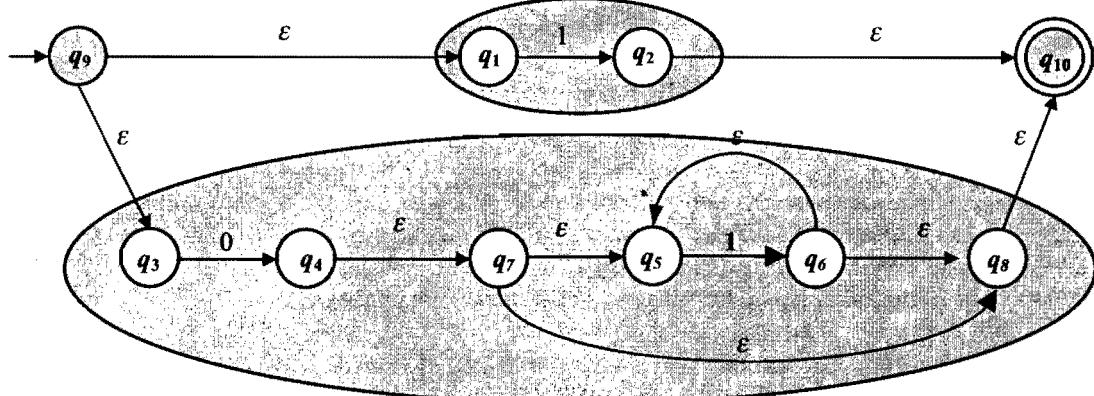
- 8) Za konstrukciju r_4 koristi se pravilo (p6):



- 9) Za konstrukciju $r_1=r_3r_4$ koristi se pravilo (p5):



- 10) Za konstrukciju $r=r_1+r_2$ koristi se pravilo (p4):

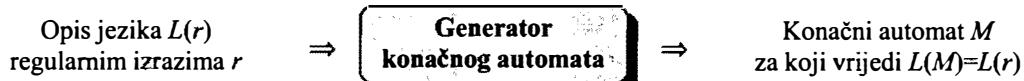


Bilo koji ϵ -NKA M izgrađen opisanim algoritmom ima sljedeća svojstva:

- 1) Broj stanja ϵ -NKA M nikad nije veći od broja $2|r|$, gdje je $|r|$ broj znakova u regularnom izrazu r . U pojedinim koracima konstrukcije ϵ -NKA ne stvara se više od dva nova stanja.
- 2) ϵ -NKA $M=(Q, \Sigma, \delta, i, \{f\})$ ima samo jedno početno stanje i , te samo jedno završno stanje f . Za završno stanje f vrijedi da je $\delta(f, a)=\emptyset, \forall a \in (\Sigma \cup \{\epsilon\})$, odnosno nema niti jedne usmjerene grane iz stanja f u neko drugo stanje.
- 3) Skup $\delta(q, a)$ sadrži najviše jedno stanje za ulazni znak a iz skupa Σ , dok skup $\delta(q, \epsilon)$ sadrži najviše dva stanja. Ima li čvor dvije izlazne grane, obje grane označene su ϵ -prijelazima.

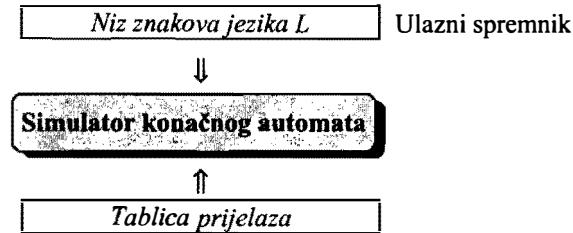
2.2.3 Generator konačnog automata

Slika 2.35 prikazuje generator konačnog automata. Konačni automat gradi se za jezik zadani regularnim izrazima.



Slika 2.35: Generator konačnog automata

Generator konačnog automata ostvaruje cijelokupni ili dio procesa pretvorbe regularnih izraza u DKA. Proces pretvorbe prikazan je na slici 2.34. Ovisno o vrsti konačnog automata koji se konstruira (DKA, NKA ili ϵ -NKA), izgradi se odgovarajući program simulator. Na slici 2.36 prikazan je konačni automat ostvaren izravnim načinom zapisa stanja. Načini zapisa stanja DKA opisani su u odjeljku 2.1.1.

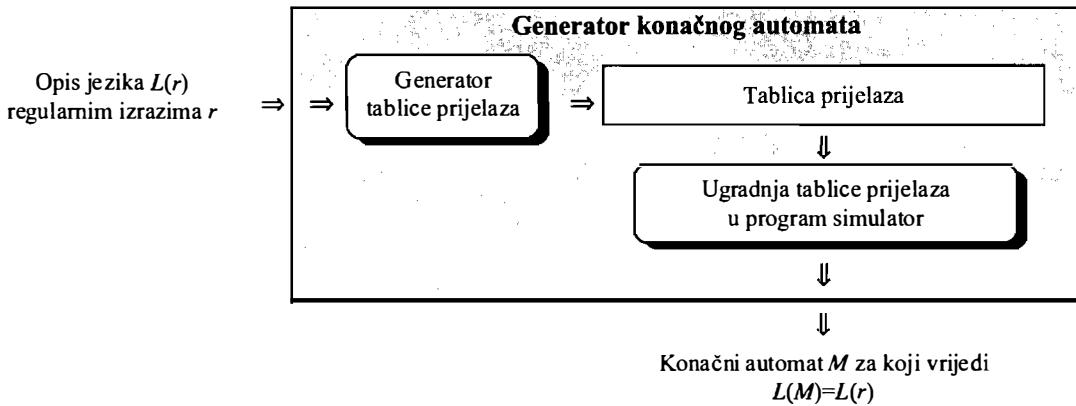


Slika 2.36: Konačni automat ostvaren izravnim načinom zapisa stanja

Simulator konačnog automata jest program koji slijedno čita znakove ulaznog spremnika. Nakon pročitanog znaka, simulator računa prijelaz u novo stanje (ili u nova stanja u slučaju simulacije NKA i ϵ -NKA) na temelju podataka iz tablice prijelaza. Stanje (odnosno skup stanja u slučaju simulacije NKA i ϵ -NKA) nakon posljednjeg pročitanog znaka određuje da li se ulazni niz prihvata.

Izravni način zapisa stanja pogodan je za gradnju generatora konačnog automata. Želi li se konstruirati konačni automat za neki drugi jezik, potrebno je izgraditi novu tablicu prijelaza. Program simulator nije potrebno mijenjati.

Generator konačnog automata gradi tablicu prijelaza na temelju zadanih regularnih izraza. Tablica prijelaza ugraditi se u program simulator. Procesi generiranja konačnog automata prikazani su na slici 2.37.

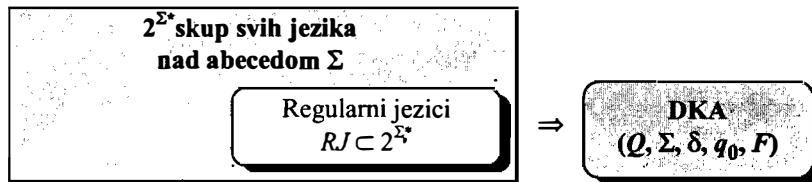


Slika 2.37: Osnovni procesi generatora konačnog automata

2.3 Svojstva regularnih jezika

U odjeljku 2.2.1 naveden je primjer jezika N u kojem su nizovi oblika wcw^R . Pokazano je da za jezik N nije moguće izgraditi konačni automat, te zato jezik nije regularan. Za jezik $K=\{0^{i^2} \mid i \text{ jest cijeli broj, } i \geq 1\}$ također nije moguće izgraditi konačni automat koji ga prihvata. Jezik K jest skup nizova koji se sastoje samo od znakova nula i duljina im je kvadrat cijelog broja.

Neka 2^{Σ^*} označava skup svih jezika L definiranih nad abecedom Σ ($L \subseteq \Sigma^*$), a klasa regularnih jezika neka označava skup svih regularnih jezika RJ . Na slici 2.38 prikazano je da je klasa regularnih jezika nad abecedom Σ pravi podskup skupa svih jezika 2^{Σ^*} ($RJ \subset 2^{\Sigma^*}$). U odjeljcima koji slijede opisana su svojstva klase regularnih jezika.



Slika 2.38: Regularni jezici i DKA

2.3.1 Svojstva zatvorenosti regularnih jezika

Zatvorenost klase jezika definira se s obzirom na pojedine operacije nad jezicima. Klasa jezika zatvorena je s obzirom na neku operaciju ako primjenom te operacije na bilo koji jezik iz te klase dobijemo jezik koji je u toj istoj klasi. Regularni jezici zatvoreni su s obzirom na više operacija. Na primjer, regularni jezici zatvoreni su s obzirom na operaciju unije dvaju regularna jezika. Unija dva regularna jezika L i N jest regularni jezik za koji je moguće izgraditi DKA M takav da vrijedi $L(M)=L \cup N$.

Istovjetnost regularnih jezika, konačnih automata i regularnih izraza koristi se za opis svojstava regularnih jezika. Regularni jezici i konačni automati istovjetni su na temelju definicije regularnih jezika, dok je njihova istovjetnost s regularnim izrazima pokazana u odjeljku 2.2.2.

Unija, nadovezivanje i Kleeneov operator

Regularni jezici zatvoreni su s obzirom na sljedeće operacije: unija dva regularna jezika, nadovezivanje dvaju regularnih jezika i Kleeneov operator. Svojstva zatvorenosti slijede neposredno iz definicije regularnih izraza.

Komplement

Regularni jezici zatvoreni su s obzirom na operaciju komplementa. Neka DKA $M=(Q, \Sigma, \delta, q_0, F)$ prihvaca regularni jezik $L(M)$. Za komplement jezika $L(M)^c$ moguce je izgraditi DKA $M'=(Q, \Sigma, \delta, q_0, Q-F)$ koji prihvaca jezik $L(M')=\{w \mid \delta(q_0, w) \in (Q-F)\} = \{w \mid \delta(q_0, w) \notin F\} = \Sigma^* - \{w \mid \delta(q_0, w) \in F\} = \Sigma^* - L(M) = L(M)^c$. Buduci da je za jezik koji je komplement regularnog jezika moguce izgraditi DKA, pokazano je da su regularni jezici zatvoreni s obzirom na operaciju komplementa.

Presjek

Regularni jezici zatvoreni su s obzirom na operaciju presjeka. Svojstvo zatvorenosti s obzirom na presjek slijedi izravno iz svojstva zatvorenosti s obzirom na uniju i komplement, sto se dokazuje De Morganovim pravilom: $L \cap N = ((L \cap N)^c)^c = (L^c \cup N^c)^c$.

Ako su zadani DKA $M_1=(Q_1, \Sigma, \delta_1, q_1, F_1)$ i DKA $M_2=(Q_2, \Sigma, \delta_2, p_1, F_2)$, onda se DKA $M=(Q, \Sigma, \delta, q_0, F)$ koji prihvaca regularni jezik $L(M)=L(M_1) \cap L(M_2)$ gradi na sljedeći način:

- 1) $Q = Q_1 \times Q_2$, stanje $[q, p] \in Q$, gdje je $q \in Q_1$ i $p \in Q_2$,
- 2) $q_0 = [q_1, p_1]$,
- 3) $F = F_1 \times F_2$, stanje $[q, p] \in F$, gdje je $q \in F_1$ i $p \in F_2$,
- 4) $\delta([q, p], a) = [\delta_1(q, a), \delta_2(p, a)]$, $\forall q \in Q_1$, $\forall p \in Q_2$ i $\forall a \in \Sigma$.

Primjer 2.9. Na slikama 2.39 i 2.40 prikazane su tablice prijelaza DKA M_1 i DKA M_2 . DKA M_1 prihvaca jezik $L(M_1)=\{w \mid w \text{ jest niz koji se sastoji od znakova } a \text{ i } b\}$, a DKA M_2 prihvaca jezik $L(M_2)=\{w \mid w \text{ jest niz koji se sastoji od znakova } a \text{ i } c\}$. Na slici 2.41 prikazan je DKA M koji prihvaca jezik $L(M)=L(M_1) \cap L(M_2) = \{w \mid w \text{ jest niz koji se sastoji od znakova } a\}$.

	a	b	c	
a	1	0	0	1
b	0	1	0	1
c	0	0	1	0
	q ₁	q ₂	q ₁	q ₂
q ₁	q ₁	q ₂	q ₁	1
q ₂	q ₂	q ₁	q ₂	0

Slika 2.39: Tablica prijelaza DKA M_1

	a	b	c	
a	1	0	0	1
b	0	1	0	1
c	0	0	1	0
	p ₁	p ₂	p ₁	p ₂
p ₁	p ₁	p ₂	p ₁	1
p ₂	p ₂	p ₁	p ₂	0

Slika 2.40: Tablica prijelaza DKA M_2

	<i>a</i>	<i>b</i>	<i>c</i>	
[q ₁ , p ₁]	[q ₁ , p ₁]	[q ₁ , p ₂]	[q ₂ , p ₁]	1
[q ₁ , p ₂]	[q ₁ , p ₂]	[q ₁ , p ₂]	[q ₂ , p ₂]	0
[q ₂ , p ₁]	[q ₂ , p ₁]	[q ₂ , p ₂]	[q ₂ , p ₁]	0
[q ₂ , p ₂]	0			

Slika 2.41: Tablica prijelaza DKA M koji prihvata jezik $L(M)=L(M_1)\cap L(M_2)$
(DKA M_1 i DKA M_2 su zadani na slikama 2.39 i 2.40)

Supstitucija

Regularni jezici zatvoreni su s obzirom na operaciju supstitucije. Neka je $R \subseteq \Sigma^*$ regularni jezik i neka se znaku a iz skupa Σ pridruži regularni jezik $R_a \subseteq \Delta^*$, gdje je Δ određena abeceda. Ako se niz $a_1a_2 \dots a_n$ ($a_i \in \Sigma$) regularnog jezika R zamijeni nizom $w_1w_2 \dots w_n$ ($w_i \in R_{a_i}$), gdje je w_i proizvoljni niz jezika R_{a_i} , onda je dobiveni jezik $f(R)$ regularan. Svojstvo zatvorenosti s obzirom na operaciju supstitucije dokazuje se tako da se jezici R i R_{a_i} opišu regularnim izrazima.

Svojstvo supstitucije omogućuje da se regularni jezici opišu na jednostavan i pristupačan način regularnim definicijama.

Primjer 2.10. Neka je R regularni jezik zadan regularnim izrazom $r=0*(0+1)1^*$. Ako se zada supstitucija koja znaku 0 pridružuje regularni jezik definiran regularnim izrazom $f(0)=a$, a znaku 1 pridružuje jezik definiran regularnim izrazom $f(1)=b^*$, onda nastaje regularni jezik $f(R)$ definiran regularnim izrazom:

$$\begin{aligned} f(R) &= f(0*(0+1)1^*) = (f(0))*(f(0)+f(1))(f(1))^* = a*(a+b^*)(b^*)^* \\ &= a^*(a+b^*)b^* = (a^*a+a^*b^*)b^* \end{aligned}$$

Budući da je $b^*b^*=b^*$, $a^*a=a^*$, i $a^++a^*=a^*$, dati regularni izraz moguće je napisati na sljedeći način:

$$(a^*a+a^*b^*)b^* = a^*ab^*+a^*b^*b^* = a^*ab^*+a^*b^* = a^*b^*+a^*b^* = (a^++a^*)b^* = a^*b^*$$

2.3.2 Regularne definicije

Regularne definicije su oblika:

$$\begin{aligned} d_1 &\rightarrow r_1 \\ d_2 &\rightarrow r_2 \\ \cdots & \\ d_n &\rightarrow r_n \end{aligned}$$

gdje su d_i znakovi označeni različitim imenima, a r_i su regularni izrazi nad abecedom $\Sigma \cup \{d_1, d_2, \dots, d_{i-1}\}$. Abeceda regularnog izraza sastoji se od osnovnih znakova skupa Σ i znakova označenih imenima d_1, d_2, \dots, d_{i-1} koji su prethodno definirani regularnim izrazima.

Regularni izraz r_j moguće je definirati nad abecedom Σ tako da se svi znakovi d_1, d_2, \dots, d_{j-1} zamijene regularnim izrazima.

Primjer 2.11. Programske varijable definiraju se regularnim izrazima nad abecedom $\Sigma=\{A, B, \dots, Z, a, b, \dots, z, 0, 1, 2, \dots, 9\}$ na sljedeći način:

- i) $\underline{\text{slово}} \rightarrow A+B+\dots+Z+a+b+\dots+z$
- ii) $\underline{\text{brojka}} \rightarrow 0+1+\dots+9$
- iii) $\underline{\text{varijabla}} \rightarrow \underline{\text{slovo}}(\underline{\text{slovo}}+\underline{\text{brojka}})^*$

Imena znakova d_1, d_2, \dots, d_{j-1} podcrtaća su kako bi se izbjegla nejednoznačnost s definicijama regularnih izraza. Regularni izrazi koji definiraju imena slovo i brojka zadani su nad abecedom $\Sigma=\{A, B, \dots, Z, a, b, \dots, z, 0, 1, 2, \dots, 9\}$, dok je regularni izraz koji definira varijablu zadan nad abecedom $\Sigma \cup \{\underline{\text{slovo}}, \underline{\text{brojka}}\}$.

Regularni izraz pridružen varijabli moguće je definirati isključivo nad abecedom Σ tako da se u (iii) umjesto znakova slovo i brojka uvrste regularni izrazi definirani u (i) i (ii):

$$\underline{\text{varijabla}} \rightarrow (A+B+\dots+Z+a+b+\dots+z)((A+B+\dots+Z+a+b+\dots+z)+(0+1+\dots+9))^*$$

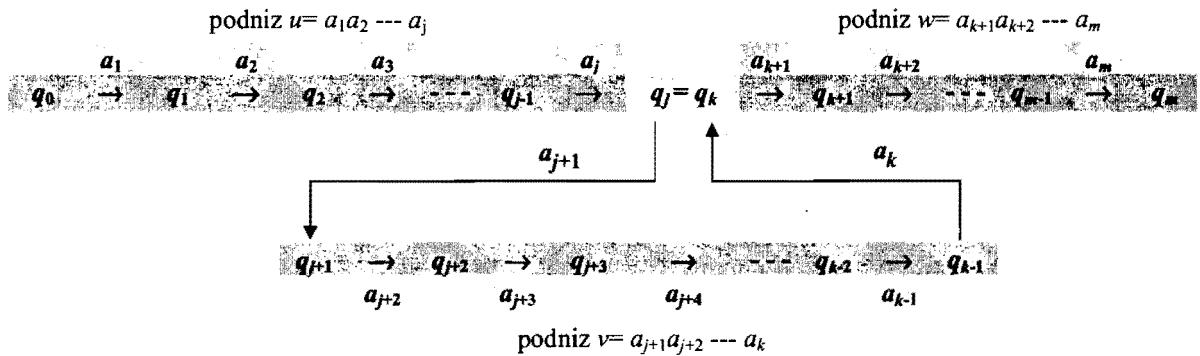
Primjer 2.12. Konstante bez predznaka definiraju se regularnim definicijama nad abecedom $\Sigma=\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, E, ., +, -\}$. Za operator + u regularnim izrazima koristi se oznaka $|$.

- $\underline{\text{broj}} \rightarrow 0|1|2|3|4|5|6|7|8|9$
 - $\underline{\text{brojke}} \rightarrow \underline{\text{broj}} \ \underline{\text{broj}}^*$
 - $\underline{\text{decimale}} \rightarrow . \ \underline{\text{brojke}} | \epsilon$
 - $\underline{\text{eksponent}} \rightarrow (E (+|-|\epsilon)) \ \underline{\text{brojke}} | \epsilon$
 - $\underline{\text{konstanta}} \rightarrow \underline{\text{brojke}} \ \underline{\text{decimale}} \ \underline{\text{eksponent}}$
-

2.3.3 Svojstvo napuhavanja

Svojstvo napuhavanja pogodno je za dokazivanje neregularnosti nekih jezika, kao i za dokazivanje ispravnosti raznih algoritama kojima se utvrđuje nepraznost regularnog jezika, beskonačnost regularnog jezika, itd..

Ako je jezik regularan, onda postoji DKA $M=(Q, \Sigma, \delta, q_0, F)$ koji ga prihvaca. Neka DKA M ima n stanja. Promatra se ulazni niz znakova $a_1a_2 \dots a_m$ koji je duži od broja stanja n , tj. $m > n$. Za bilo koji $i=1, 2, \dots, m$ neka je $\delta(q_0, a_1a_2 \dots a_i)=q_i$. Budući da DKA ima samo n različitih stanja, nije moguće da je svih $n+1$ stanja različito u nizu stanja q_0, q_1, \dots, q_n . Postoji barem jedno stanje koje se pojavi dva puta, odnosno postoje dva indeksa j i k za koje vrijedi da je $0 \leq j < k \leq n$ i $q_j=q_k$. Slijed prijelaza za niz $a_1a_2 \dots a_m$ prikazan je na slici 2.42. Budući da je $j < k$ i budući da je $k \leq n$, za duljinu niza $a_{j+1}a_{j+2} \dots a_k$ vrijedi $1 \leq |a_{j+1}a_{j+2} \dots a_k| \leq n$. Neka se niz $a_1a_2 \dots a_m$ označi znakom z . Niz z napiše se kao niz $z=uvw$, gdje je $u=a_1a_2 \dots a_j$, $v=a_{j+1}a_{j+2} \dots a_k$ i $w=a_{k+1}a_{k+2} \dots a_m$. Niz z prikazan je na slici 2.42.



Slika 2.42: Slijed prijelaza DKA

Ako je q_m stanje skupa F , onda se niz $z=uvw$ prihvata. Niz uw se također prihvata. Budući da je $q_j=q_k$, postoji slijed prijelaza iz početnog stanja q_0 u stanje q_m :

$$\delta(q_0, uw) = \delta(\delta(q_0, u), w) = \delta(q_j, w) = \delta(q_k, w) = q_m$$

Prihvata se niz $uvvw$:

$$\begin{aligned} \delta(q_0, uvvw) &= \delta(\delta(q_0, u), vvw) = \delta(q_j, vvw) = \delta(\delta(q_j, v), vw) = \delta(q_k, vw) \\ &= \delta(q_j, vw) = \delta(\delta(q_j, v), w) = \delta(q_k, w) = q_m. \end{aligned}$$

Prihvaćaju se svi nizovi oblika uv^iw , gdje je $i \geq 0$. Bilo koji dovoljno dugački niz $z \in L(M)$ moguće je rastaviti na podnizove $z=uvw$. Podniz v može se proizvoljan broj puta ponoviti ("napuhati"). Na taj način izgrađen niz uv^iw jest element jezika $L(M)$, gdje je M odgovarajući DKA.

Sadrži li regularni jezik dovoljno dugački niz $z=uvw$, taj jezik sadrži i beskonačni skup nizova oblika uv^iw . Svojstvo napuhavanja ne znači da je bilo koji dugački niz z oblika uv^iw za neki veliki cijeli broj i . Na primjer, može se pokazati da jezik $(0+1)^*$ sadrži proizvoljno dugački niz u kojem se niti jedan podniz ne pojavi više od tri puta uzastopno.

Za dokazivanje neregularnosti jezika koristi se sljedeće pravilo:

- i) Ako je L regularni jezik, onda postoji cijelobrojna konstanta n takva da je moguće bilo koji niz z iz jezika L za koji vrijedi $|z| > n$ rastaviti na podnizove $z=uvw$ na način da je $|uv| \leq n$ i $1 \leq |v|$, te za bilo koji $i \geq 0$, niz uv^iw je također element jezika L . Pokazuje se da n nije veći od broja stanja minimalnog DKA koji prihvata jezik L .

Primjer 2.13. Pomoću svojstva napuhavanja pokazuje se da jezik $K = \{0^{l^2} \mid l \text{ jest cijeli broj } l \geq 1\}$ nije regularan. Jezik sadrži nizove znakova 0 čija duljina jest kvadrat nekog cijelog broja.

- 1) Prepostavi se da je L regularni jezik.
- 2) Neka je n proizvoljna cijelobrojna konstanta definirana prema gore navedenom svojstvu (i) i neka je $z=0^{n^2}$ niz jezika L za koji vrijedi $|z|=n^2$ i $|z| > n$.
- 3) Prema svojstvu (i) niz z rastavlja se na podnizove uvw gdje je $1 \leq |v| \leq |uv| \leq n$. Potrebno je utvrditi da li niz uv^iw jest element jezika L za bilo koji cijeli broj i .

- 4) Ako se uzme da je $i=2$ i ako se uzme da je $|v| \leq |uv| \leq n$, onda je $|uvw| = |z| = n^2 < |uv^2w| = (n^2 + |v|) \leq (n^2 + n)$. Budući da je $(n^2 + n) < (n+1)^2$, onda vrijedi $n^2 < |uv^2w| < (n+1)^2$, tj. duljina niza uv^2w nije kvadrat niti jednog cijelog broja.

- 5) Bez obzira na veličinu broja n , bez obzira na duljinu niza $z \in L$ i bez obzira na podjelu na podnizove uvw , niz uv^2w nije element jezika L . Prema tome, pretpostavka da je jezik L regularan, nije istinita.
-

Algoritmi odlučivanja za regularne jezike

Nepraznost i beskonačnost regularnog jezika. Neka DKA M prihvata jezik $L(M)$ i neka DKA M ima n stanja. Svojstvo napuhavanja primjenjuje se za dokazivanje sljedećih tvrdnji:

- a) Regularni jezik $L(M)$ jest neprazan ako i samo ako DKA M prihvata niz z duljine manje od n , tj. $|z| < n$.

Za utvrđivanje da li je jezik $L(M)$ koji prihvata DKA M neprazan, proširuje se algoritam za pronalaženje nedohvatljivih stanja. Postoji li u skupu dohvataljivih stanja barem jedno prihvataljivo stanje, regularni jezik $L(M)$ jest neprazan.

- b) Regularni jezik $L(M)$ jest beskonačan ako i samo ako M prihvata niz duljine l , gdje je $n \leq l < 2n$.

Za utvrđivanje da li je jezik $L(M)$ beskonačan, promatra se dijagram stanja DKA M . Izbacivanjem svih neprihvataljivih stanja za koja ne postoji niti jedan slijed prijelaza u jedno od prihvataljivih stanja, dobije se DKA M' koji prihvata isti jezik $L(M)=L(M')$. Postoji li u dobivenom dijagrameu stanja DKA M' barem jedna zatvorena petlja, regularni jezik $L(M)$ jest beskonačan.

2.4 Gramatika

Gramatika koja generira regularne jezike naziva se regularna gramatika. U odjeljku 2.4.1 definira se formalna gramatika. Formalna definicija gramatike zasniva se na kontekstno neovisnoj gramatici. Svojstva kontekstno neovisne gramatike detaljno su opisana u poglavљу 3 zajedno sa svojstvima kontekstno neovisnih jezika, dok se u odjeljku 2.4.2 detaljno opisuju svojstva regularne gramatike. Budući da su regularni jezici pravi podskup kontekstno neovisnih jezika, oblik produkcija regularne gramatike ograničen je u odnosu na kontekstno neovisnu gramatiku, čime se jamči generiranje regularnih jezika.

2.4.1 Formalna gramatika

Kao što se *gramatika* koristi u proučavanju *prirodnih* jezika, formalna *gramatika* koristi se u generiranju i analizi nizova znakova *formalnog* jezika. Neka formalna gramatika sliči gramatici hrvatskog jezika, a formalni jezik neka se sastoji se od 16 rečenica. Definira se sedam elemenata formalnog jezika sličnih elementima rečenice prirodnog jezika:

<rečenica>, <subjektni_skup>, <predikat>, <objektni_skup>, <subjekt>, <objekt>, <atribut>

Elementi rečenice (odnosno nezavršni znakovi u nazivlju formalne gramatike) stavljaju se u zgrade <> da bi se razlikovali od samih riječi (završnih znakova) od kojih se sastoji rečenica i koje čine rječnik jezika (odnosno skup završnih znakova). Treba biti pažljiv, jer se poistovjećuje riječ koja se sastoji od niza glasova s jedinstvenim završnim znakom formalne gramatike. Neka se naš rječnik (odnosno skup završnih znakova) sastoji od sljedećih šest riječi (odnosno skup završnih znakova čini šest završnih znakova):

DJEVOJKE, MAČKE, GLEDAJU, ZBUNJENE, UPLAŠENE, . (točka).

Gramatika se sastoji od pravila (odnosno produkcija) na koji način se grade rečenice (odnosno nizovi znakova u nazivlju formalne gramatike) iz gore navedenih riječi (odnosno završnih znakova). Jedno od pravila je:

1) <rečenica> → <subjektni_skup> <predikat> <objektni_skup>.

koje kaže da se rečenica gradi od subjektnog skupa, predikata, objektnog skupa i na kraju rečenice dolazi točka. Ostala pravila određuju kako se pojedini elementi (nezavršni znakovi) zamjenjuju nizovima elemenata rečenice (odnosno nezavršnim znakovima) i riječima (odnosno završnim znakovima). Znakovi se zamjenjuju sve dok se cijeli niz ne sastoji isključivo od riječi (odnosno završnih znakova):

- 2) <subjektni_skup> → <atribut> <subjekt>
- 3) <objektni_skup> → <atribut> <objekt>
- 4) <predikat> → **GLEDAJU**
- 5) <subjekt> → **DJEVOJKE**
- 6) <subjekt> → **MAČKE**
- 7) <atribut> → **ZBUNJENE**
- 8) <atribut> → **UPLAŠENE**
- 9) <objekt> → **DJEVOJKE**
- 10) <objekt> → **MAČKE**

Rečenice jezika (odnosno nizovi završnih znakova) grade se primjenom pravila gramatike. Pravilo (1) određuje da se rečenica jezika sastoji od:

<subjektni_skup> <predikat> <objektni_skup>.

Primjenom pravila (2), znak <subjektni_skup> zamijeni desna strana pravila (2), odnosno niz znakova <atribut> <subjekt>:

<atribut> <subjekt> <predikat> <objektni_skup>.

Primjenom pravila (3), znak <objektni_skup> zamijeni niz <atribut> <objekt>:

<atribut> <subjekt> <predikat> <atribut>*<objekt>.

Primjenom pravila (4), znak <predikat> zamijeni riječ **GLEDAJU**:

<atribut> <subjekt> GLEDAJU <atribut> <objekt>.

Znakove <subjekt> i <objekt> zamijene desne strane pravila (5) i (10):

<atribut> DJEVOJKE GLEDAJU <atribut> MAČKE.

Dva znaka <atribut> zamijene se desnim stranama pravila (7) i (8):

ZBUNJENE DJEVOJKE GLEDAJU UPLAŠENE MAČKE.

Dobiveni niz znakova jest potpuna rečenica jezika koja se sastoji isključivo od riječi (odnosno završnih znakova). Relacija \Rightarrow označava da se primjenom samo jednog pravila prelazi iz lijevog međuniza u desni međuniz. Generiranje rečenice (odnosno niza završnih znakova) opisuje se na sljedeći način:

```

<rečenica>
⇒ <subjektni_skup> <predikat> <objektni_skup> .
⇒ <atribut> <subjekt> <predikat> <objektni_skup> .
⇒ <atribut> <subjekt> <predikat> <atribut> <objekt>.
⇒ <atribut> <subjekt> GLEDAJU <atribut> <objekt>.
⇒ <atribut> DJEVOJKЕ GLEDAJU <atribut> <objekt>.
⇒ <atribut> DJEVOJKЕ GLEDAJU <atribut> MAČKE.
⇒ UPLAŠENE DJEVOJKЕ GLEDAJU <atribut> MAČKE.
⇒ UPLAŠENE DJEVOJKЕ GLEDAJU ZBUNJENE MAČKE.

```

Relacija $\stackrel{*}{\Rightarrow}$ označava višestruku primjenu pravila gramatike. Generiranje rečenica prikazuje se pomoću relacije $\stackrel{*}{\Rightarrow}$ na sljedeći način:

```

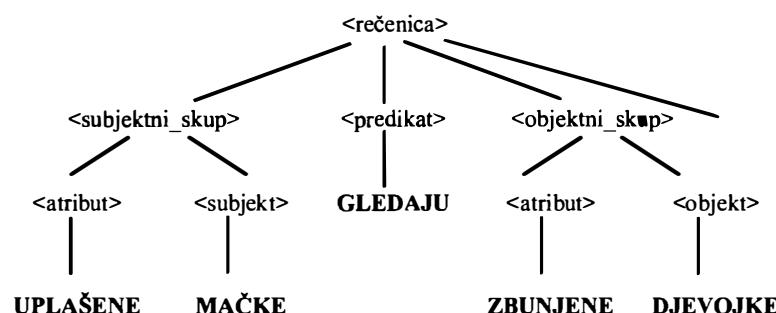
<rečenica>  $\stackrel{*}{\Rightarrow}$  <atribut> <subjekt> <predikat> <atribut> <objekt>.
 $\stackrel{*}{\Rightarrow}$  UPLAŠENE DJEVOJKЕ GLEDAJU UPLAŠENE MAČKE.

<rečenica>  $\stackrel{*}{\Rightarrow}$  <atribut> <subjekt> <predikat> <atribut> <objekt>.
 $\stackrel{*}{\Rightarrow}$  UPLAŠENE MAČKE GLEDAJU UPLAŠENE DJEVOJKЕ .

<rečenica>  $\stackrel{*}{\Rightarrow}$  <atribut> <subjekt> <predikat> <atribut> <objekt>.
 $\stackrel{*}{\Rightarrow}$  ZBUNJENE MAČKE GLEDAJU UPLAŠENE DJEVOJKЕ .

```

Generiranje rečenica jezika prikazuje se i stablom. Stablo na slici 2.43 prikazuje generiranje rečenice: "UPLAŠENE MAČKE GLEDAJU ZBUNJENE DJEVOJKЕ". Stablom su prikazana sva primjenjena pravila gramatike.



Slika 2.43: Stablo rečenice
UPLAŠENE MAČKE GLEDAJU ZBUNJENE DJEVOJKЕ .

Kontekstno neovisna gramatika je uređena četvorka:

$$G = (V, T, P, S)$$

gdje je:

- V - konačan skup nezavršnih znakova;
- T - konačan skup završnih znakova i $V \cap T = \emptyset$;

- P - konačan skup produkcija oblika $A \rightarrow \alpha$, A je nezavršni znak, α je niz znakova skupa $(V \cup T)^*$, α može biti prazni niz ϵ ;
- S - početni nezavršni znak.

Prethodni primjer gramatike formalno se zapiše na sljedeći način $G = (\{<\text{rečenica}>, <\text{subjektni_skup}>, <\text{predikat}>, <\text{objektni_skup}>, <\text{subjekt}>, <\text{atribut}>, <\text{objekt}>\}, \{\text{DJEVOJKE, MAČKE, GLEDAJU, ZBUNJENE, UPLAŠENE, }.\}, P, <\text{rečenica}>)$, gdje je P skup prethodno definiranih pravila od (1) do (10).

U formalnoj gramatici koriste se sljedeće oznake:

- 1) Velika slova A, B, C, D, E , i S su nezavršni znakovi gramatike. Uobičajeno je da se znak S koristi za početni nezavršni znak.
- 2) Mala slova a, b, c, d, e , brojke, i podebljani nizovi znakova su završni znakovi gramatike.
- 3) Velika slova X, Y , i Z su završni ili nezavršni znakovi.
- 4) Mala slova u, v, w, x, y, i, z označavaju nizove završnih znakova.
- 5) Mala grčka slova α, β, i, γ označavaju nizove koji se sastoje od završnih i nezavršnih znakova.

Ima li više produkcija isti znak na lijevoj strani, te se produkcije napišu u istom retku s jednim znakom na lijevoj strani, a različite desne strane odvoje se operatorom $|$. Na primjer, produkcije $A \rightarrow a$ i $A \rightarrow b$ zapišu se na sljedeći način: $A \rightarrow a | b$.

Primjer 2.14. Kontekstno neovisna gramatika koristi se za definiranje sintakse programskih jezika. Producije gramatike najčešće su u Backus-Naurovom obliku (BNF oblik). BNF jest formalizam sličan formalizmu kontekstno neovisne gramatike, osim malih razlika u formatu zapisa i uvedenih pokrata. Aritmetički izrazi programskih jezika generiraju se sljedećom gramatikom:

$$G = (\{E\}, \{a, *, +, (,)\}, \{E \rightarrow E+E \mid E*E \mid (E) \mid a\}, E).$$

Zadana gramatika G generira nizove završnih znakova na sljedeći način:

- 1) $E \Rightarrow a$
- 2) $E \Rightarrow E+E \stackrel{*}{\Rightarrow} a+a$
- 3) $E \Rightarrow E*E \stackrel{*}{\Rightarrow} a*a$
- 4) $E \Rightarrow E*E \Rightarrow (E)*E \Rightarrow (E+E)*E \stackrel{*}{\Rightarrow} (a+a)*a$

Neka završni znak a označava varijablu ili konstantu programskega jezika. Na temelju značenja pojedinih operatora (zagrade, zbrajanje i množenje), zaključuje se da su generirani nizovi završnih znakova aritmetički izrazi programskega jezika.

Za zadatu gramatiku $G = (V, T, P, S)$ definira se relacija $\xrightarrow[G]{}$ nad nizovima iz skupa $(V \cup T)^*$. Ako je $A \rightarrow \beta$ produkcija skupa P i ako su α i γ iz $(V \cup T)^*$, onda vrijedi relacija:

$$\alpha A \gamma \xrightarrow[G]{} \alpha \beta \gamma.$$

Niz $\alpha\beta\gamma$ generira se neposredno iz niza $\alpha A\gamma$ primjenom produkcije $A \rightarrow \beta$, dok oznaka G određuje kojoj gramatički pripada primijenjena produkcija.

Neka su $\alpha_1, \alpha_2, \dots, \alpha_m$ nizovi iz $(V \cup T)^*$, $m \geq 1$, i neka je:

$$\underset{G}{\alpha_1 \Rightarrow} \alpha_2, \underset{G}{\alpha_2 \Rightarrow} \alpha_3, \dots, \underset{G}{\alpha_{m-1} \Rightarrow} \alpha_m.$$

Gramatika G generira niz α_m iz niza α_1 , što se zapiše pomoću relacije $\underset{G}{\Rightarrow}^*$ na sljedeći način:

$$\underset{G}{\alpha_1 \Rightarrow}^* \alpha_m.$$

Relacija $\underset{G}{\Rightarrow}^*$ jest refleksivno i tranzitivno okruženje relacije \Rightarrow . Definicije refleksivnog i tranzitivnog okruženja relacije dane su u odjeljku 1.1. Zna li se na koju gramatiku se relacije odnose, umjesto $\underset{G}{\Rightarrow}$ i $\underset{G}{\Rightarrow}^*$ koriste se označke \Rightarrow i \Rightarrow^* .

Generira li gramatika niz β primjenom i produkcija iz niza α , to se zapisuje na sljedeći način:

$$\alpha \xrightarrow{i} \beta.$$

Gramatika $G = (V, T, P, S)$ generira jezik $L(G) = \{w \mid w \in T^* \text{ za koji vrijedi } \underset{G}{S \Rightarrow}^* w\}$. Niz w je u jeziku $L(G)$ koji generira gramatika G ako za taj niz vrijedi:

- 1) Niz se sastoji isključivo od završnih znakova gramatike.
- 2) Niz je moguće generirati iz početnog nezavršnog znaka S .

Gramatike G_1 i G_2 su istovjetne ako generiraju iste jezike, tj. ako je $L(G_1) = L(G_2)$.

Kontekstno neovisna gramatika generira klasu *kontekstno neovisnih jezika*. Klasa kontekstno neovisnih jezika sadrži i jezike koji nisu regularni, na primjer jezik $N = \{wcw^R\}$. Svojstva jezika N opisuju se u odjeljku 2.2.1. U primjeru 2.15 pokazuje se da postoji kontekstno neovisna gramatika G koja generira jezik $L(G) = N$, tj. pokazuje se da je N kontekstno neovisni jezik. U odjeljku 2.4.2 pokazuje se da je za bilo koji regularni jezik moguće izgraditi kontekstno neovisnu gramatiku, odnosno da su regularni jezici ujedno i kontekstno neovisni jezici. Budući da je N kontekstno neovisni jezik, a nije regularan, zaključuje se da je klasa kontekstno neovisnih jezika širi skup od klase regularnih jezika. Slika 2.44 prikazuje da je klasa regularnih jezika pravi podskup klase kontekstno neovisnih jezika.



Slika 2.44: Kontekstno neovisni jezici KNJ i regularni jezici RJ

Primjer 2.15. Zadana je sljedeća gramatika $G = (V, T, P, S)$, gdje je $V = \{S\}$, $T = \{a, b, c\}$, $P = \{S \rightarrow aSa \mid bSb \mid c\}$ i S je početno stanje. Počev od početnog nezavršnog znaka S i primjenom produkcija $S \rightarrow aSa$ i $S \rightarrow bSb$ generira se niz u kojem se neposredno ispred i iza nezavršnog znaka S nalazi isti završni znak a ili b :

$$S \Rightarrow \underline{a}S\underline{a} \Rightarrow a\underline{a}S\underline{a}a \Rightarrow aa\underline{b}S\underline{b}aa \Rightarrow aab\underline{b}S\underline{b}baa \Rightarrow \cdots.$$

Budući da se ispred S uvijek generira isti završni znak kao i iza S , zaključuje se da primjenom produkcija $S \rightarrow aSa$ i $S \rightarrow bSb$ nastaje sljedeći niz:

$S \xrightarrow{*} wSw^R$, gdje se niz w sastoji od završnih znakova a i b , dok je niz w^R niz w napisan obrnutim redoslijedom.

Primjenom produkcije $S \rightarrow c$ generira se niz koji se sastoji isključivo od završnih znakova, tj. zadana gramatika generira sljedeće nizove:

$$S \xrightarrow{*} wSw^R \Rightarrow wcw^R, \text{ odnosno } L(G) = \{wcw^R\} = N$$

Stablo je *generativno stablo* za gramatiku $G = (V, T, P, S)$ ako vrijedi:

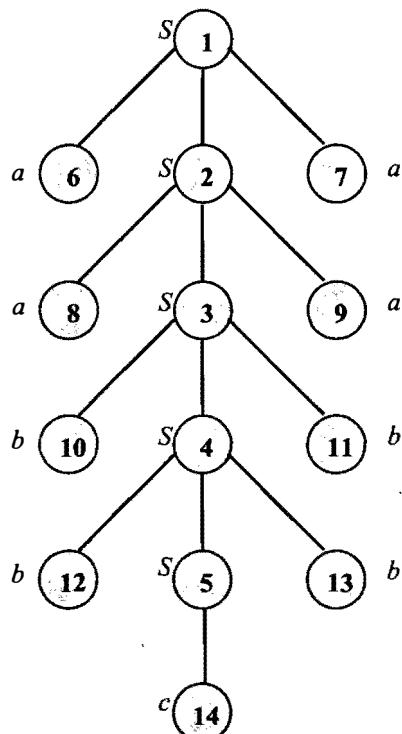
- 1) Čvorovi stabla označeni su znakovima iz $V \cup T \cup \{\epsilon\}$.
- 2) Korijen stabla označen je početnim nezavršnim znakom S .
- 3) Unutrašnji čvorovi označeni su nezavršnim znakovima $A \in V$.
- 4) Neka su čvorovi n_1, n_2, \dots, n_k svi čvorovi djeca čvora n . Ako je čvor n označen znakom A i ako su čvorovi n_1, n_2, \dots, n_k označeni znakovima X_1, X_2, \dots, X_k , onda je:

$$A \rightarrow X_1 X_2 \dots X_k$$

produkcija iz skupa P .

- 5) Znakom ϵ moguće je označiti samo list stabla. List stabla označen znakom ϵ je jedino dijete svog roditelja, odnosno dijete jednog od unutarnjih čvorova.
- 6) Listovi stabla označeni su znakovima iz skupa $T \cup \{\epsilon\}$ i čitani s lijeva na desno čine generirani niz jezika $L(G)$.

Neka gramatika G generira niz završnih znakova w . Relacija $S \xrightarrow{*} w$, gdje je w niz završnih znakova, vrijedi ako i samo ako postoji generativno stablo za gramatiku G čiji su listovi isključivo označeni znakovima niza w i znakom ϵ .



Primjer 2.16. Na slici 2.45 prikazano je generativno stablo za gramatiku G zadatu u primjeru 2.15 i niz $aabbcbbaa$. Korijen stabla jest čvor 1, čvorovi 2 do 5 su unutrašnji čvorovi, dok su čvorovi 6 do 14 listovi. Svi čvorovi stabla od 1 do 14 označeni su znakovima skupa $V \cup T$. Korijen stabla označen je početnim nezavršnim znakom S . Neposredni sljedbenici čvora 1 su čvorovi 6, 2 i 7. Čvorovi 1, 6, 2 i 7 označeni su znakovima S, a, S i a , što odgovara produkciji $S \rightarrow aSa$. Čvorovi 8, 3 i 9 su neposredni sljedbenici čvora 2. Oznake čvorova 2, 8, 3 i 9 odgovaraju produkciji $S \rightarrow aSa$. Na sličan način provjerava se da su čvorovi 3, 4 i njihovi neposredni sljedbenici označeni znakovima produkcije $S \rightarrow bSb$. Sljedbenik čvora 5 jest čvor 14. Čvor 14 označen je znakom c , što odgovara produkciji $S \rightarrow c$. Dakle, prikazano stablo je generativno stablo za niz $aabbcbbaa$ i gramatiku G definiranu u primjeru 2.15.

Slika 2.45: Generativno stablo za gramatiku G zadatu u primjeru 2.15 i niz $aabbcbbaa$

2.4.2 Regularna gramatika

U ovom odjeljku pokazuje se da su regularni jezici ujedno i kontekstno neovisni jezici. Dan je algoritam konstrukcije kontekstno neovisne gramatike za regularni jezik zadan konačnim automatom.

Konstrukcija gramatike za regularni jezik zadan pomoću DKA

Za regularni jezik zadan pomoću DKA $M=(Q, \Sigma, \delta, q_0, F)$ gradi se kontekstno neovisna gramatika $G=(V, T, P, S)$, za koju vrijedi da je $L(G)=L(M)$, na sljedeći način:

- 1) Skup završnih znakova gramatike T jednak je skupu ulaznih znakova DKA Σ , tj. $T=\Sigma$.
- 2) Skup nezavršnih znakova gramatike V jednak je skupu stanja DKA Q , tj. $V=Q$.
- 3) Početni nezavršni znak gramatike S jednak je početnom stanju DKA q_0 , tj. $S=q_0$.
- 4) Na temelju prijelaza DKA $\delta(A, a)=B$ iz stanja A u stanje B za ulazni znak a , gradi se produkcija:

$$A \rightarrow aB,$$

gdje su A i B nezavršni znakovi gramatike, a znak a jest završni znak.

- 5) Za sva prihvatljiva stanja $A \in F$ grade se produkcije:

$$A \rightarrow \epsilon,$$

gdje je A nezavršni znak gramatike, a ϵ je prazni niz.

Primjer 2.17. Zadan je DKA $M=(\{S, A, B\}, \{a, b\}, \delta, S, \{S, B\})$, gdje je δ funkcija prijelaza zadana tablicom na slici 2.46. Prema prethodno datom algoritmu gradi se gramatika $G=(\{S, A, B\}, \{a, b\}, P, S)$, gdje je skup produkcija P prikazan tablicom 2.6.

		a	b	
	S	1	1	
A	A	1	0	
B	S	1	1	

$S \rightarrow aA$	$S \rightarrow bB$	$S \rightarrow \epsilon$
$A \rightarrow aB$	$A \rightarrow bA$	
$B \rightarrow aS$	$B \rightarrow bA$	$B \rightarrow \epsilon$

Slika 2.46: Funkcija prijelaza DKA M

Tablica 2.6: Skup produkcija gramatike G izgrađene na temelju funkcija prijelaza DKA M zadanog na slici 2.46

Izrada skupa završnih znakova, skupa nezavršnih znakova i početnog nezavršnog znaka slijedi izravno iz definicija (1) do (3). Konstrukcija produkcija gramatike opisana je na primjeru nezavršnog znaka S . Na temelju definicija (4) i (5), za nezavršni znak S grade se tri produkcije. Gradi se produkcija $S \rightarrow aA$, jer DKA prelazi iz stanja S u stanje A za ulazni znak a (vidi sliku 2.46). Nadalje, gradi se produkcija $S \rightarrow bB$, jer DKA prelazi iz stanja S u stanje B za ulazni znak b . Budući da je S prihvatljivo stanje, gradi se produkcija $S \rightarrow \epsilon$.

Na primjer, za niz aba postoji sljedeći slijed prijelaza DKA:

$$S \xrightarrow{a} A \xrightarrow{b} A \xrightarrow{a} B, B \in F \text{ i niz se prihvata.}$$

Izgrađena gramatika generira niz aba na sljedeći način:

$$S \Rightarrow aA \Rightarrow abA \Rightarrow abaB \Rightarrow aba.$$

Tablica 2.7 na nizu primjera nizova uspoređuje postupke prihvatanja i generiranja niza znakova za DKA zadan na slici 2.46 i konstruiranu gramatiku prikazanu tablicom 2.6.

Niz	Slijed prijelaza DKA sa slike 2.46	Generiranje niza produkcijama prikazanim tablicom 2.6
ϵ	$S \xrightarrow{\epsilon} S, S \in F$ i niz ϵ se prihvata	$S \Rightarrow \epsilon$, gdje je ϵ prazni niz
a	$S \xrightarrow{a} A, A \notin F$ i niz a se ne prihvata	$S \Rightarrow aA$, nema produkcije $A \rightarrow \epsilon$ i niz a se ne može generirati
aa	$S \xrightarrow{a} A \xrightarrow{a} B, B \in F$ i niz aa se prihvata.	$S \Rightarrow aA \Rightarrow aaB \Rightarrow aa$
bbb	$S \xrightarrow{b} B \xrightarrow{b} A \xrightarrow{b} A, A \notin F$ i niz bbb se ne prihvata	$S \Rightarrow bB \Rightarrow bbA \Rightarrow bbbA$, nema produkcije $A \rightarrow \epsilon$ i niz bbb se ne može generirati
$bbba$	$S \xrightarrow{b} B \xrightarrow{b} A \xrightarrow{b} A \xrightarrow{a} B, B \in F$ i niz $bbba$ se prihvata	$S \Rightarrow bB \Rightarrow bbA \Rightarrow bbbA \Rightarrow bbbaB \Rightarrow bbba$

Tablica 2.7: Prihvatanje i generiranje niza

Prihvata li DKA isti jezik koji generira gramatiku, DKA i gramatika su istovjetni. Na temelju pravila (1) do (5) gradnje gramatike za zadani DKA, dokazuje se da vrijedi sljedeća tvrdnja:

$$i) \quad A \xrightarrow{*} wB \text{ ako i samo ako je } \delta(A, w)=B.$$

Neka je $C=\delta(q_0, v)$ prihvatljivo stanje, odnosno neka DKA prihvata niz v . Na temelju (i) vrijedi da je $q_0 \xrightarrow{*} vC$. Budući da je C prihvatljivo stanje, postoji produkcija $C \rightarrow \epsilon$ i vrijedi relacija $q_0 \xrightarrow{*} vC \Rightarrow v$. Ako DKA prihvata niz v , onda ga izgrađena gramatika generira.

Neka gramatika generira niz v i neka je $q_0 \xrightarrow{*} vC \Rightarrow v$ za neki nezavršni znak, odnosno stanje C . U generiranju niza v primjenjuje se produkcija $C \rightarrow \epsilon$. Na temelju (i) vrijedi da je $\delta(q_0, v)=C$. Producija $C \rightarrow \epsilon$ gradi se ako i samo ako je stanje C prihvatljivo. Budući da je C prihvatljivo stanje, DKA prihvata niz v . Ako izgrađena gramatika generira niz v , onda ga DKA prihvata. Time je pokazana istovjetnost zadanoog DKA i konstruirane gramatike.

Konstrukcija NKA za regularni jezik zadan jednostavnom gramatikom

Za jezik zadan kontekstno neovisnom gramatikom, koja ima odgovarajući oblik produkcija, moguće je izgraditi konačni automat koji prihvata isti jezik. Za oblik produkcija uzimaju se upravo oblici koji nastaju konstrukcijom gramatike na temelju zadanoog DKA, a to su oblici $A \rightarrow aB$ i $A \rightarrow \epsilon$.

Neka su produkcije gramatike $G=(V, T, P, S)$ oblika $A \rightarrow aB$ ili $A \rightarrow \epsilon$, gdje su A i B nezavršni znakovi gramatike, a znak a jest završni znak. Za zadanu gramatiku moguće je izgraditi NKA $M=(Q, \Sigma, \delta, q_0, F)$ za koji vrijedi $L(M)=L(G)$. NKA M konstruira se na sljedeći način:

- 1) Skup ulaznih znakova NKA Σ jednak je skupu završnih znakova gramatike T , tj. $\Sigma=T$;
- 2) Skup stanja NKA Q jednak je skupu nezavršnih znakova gramatike V , tj. $Q=V$;
- 3) Početno stanje NKA q_0 jednako je početnom nezavršnom znaku gramatike S , tj. $q_0=S$;
- 4) Na temelju produkcije $A \rightarrow aB$ gradi se sljedeći prijelaz:

$$\delta(A, a) = \delta(A, a) \cup \{B\}.$$

Skup prijelaza $\delta(A, a)$ proširuje se novim stanjem B . Početno su svi skupovi $\delta(A, a)$ prazni, tj. $\delta(A, a)=\emptyset$. Više produkcija može imati isti nezavršni znak na lijevoj strani i isti završni znak na desnoj strani, ali različite nezavršne znakove na desnoj strani. Dvije produkcije mogu biti oblika $A \rightarrow aB$ i $A \rightarrow aC$. Skup prijelaza iz stanja A za ulazni znak a jest $\delta(A, a) = \{B, C\}$. Izgrađeni konačni automat nije nužno DKA, već može biti i NKA.

- 5) Ako gramatika ima produkciju $A \rightarrow \epsilon$, onda je A prihvatljivo stanje NKA, tj. $A \in F$.

Primjer 2.18. Neka su programske varijable definirane sljedećom gramatikom:

$$G = (\{V, B\}, \{s, b\}, \{V \rightarrow sB, B \rightarrow bB \mid sB \mid \epsilon\}, V),$$

gdje nezavršni znak V definira *varijablu*, a B definira *brojke ili slova*. Završni znak s označava jedno slovo, a završni znak b označava jednu brojku. Producija $V \rightarrow sB$ definira da programska *varijabla* (V) započinje slovom (s) iza kojeg slijede *brojke ili slova* (B). Producije $B \rightarrow bB \mid sB$ rekursivno definiraju da se *brojke ili slova* (B) sastoje od proizvoljnog broja slova (s) i brojaka (b). I na kraju, producija $B \rightarrow \epsilon$ definira da su to svi znakovi koji čine programsku varijablu.

	s	b	
V	\emptyset	\emptyset	\emptyset
B	\emptyset	\emptyset	\emptyset
			\emptyset
			\emptyset

Budući da su produkcije gramatike oblika $A \rightarrow aB$ i $A \rightarrow \epsilon$, moguće je izgraditi sljedeći NKA $M = (\{V, B\}, \{s, b\}, \{\delta(V, s) = \{B\}, \delta(V, b) = \emptyset, \delta(B, s) = \{B\}, \delta(B, b) = \{B\}\}, V, \{B\})$. Tablica prijelaza NKA prikazana je na slici 2.47.

Slika 2.47: Funkcija prijelaza NKA M

Desno-linearna i lijevo-linearna gramatika

Postoje općenitiji oblici produkcija gramatike koji se mogu svesti na oblik $A \rightarrow aB$ i $A \rightarrow \epsilon$. Gramatika se naziva *desno-linearna gramatika* ako bilo koja producija gramatike ima najviše jedan nezavršni znak na desnoj strani, i to na krajnjem desnom mjestu:

$$A \rightarrow wB \text{ ili } A \rightarrow w.$$

A i B su nezavršni znakovi ($A, B \in V$). Niz w jest niz završnih znakova ($w \in T^*$) proizvoljne duljine, uključujući i prazni niz.

Gramatika se naziva *lijevo-linearna gramatika* ako bilo koja producija gramatike ima najviše jedan nezavršni znak na desnoj strani, i to na krajnjem lijevom mjestu:

$A \rightarrow Bw$ ili $A \rightarrow w$

A i B su nezavršni znakovi ($A, B \in V$). Niz w jest niz završnih znakova ($w \in T^*$) proizvoljne duljine, uključujući i prazni niz.

Lijevo-linearna gramatika i desno-linearna gramatika su *regularne gramatike*.

Primjer 2.19. Jezik zadan regularnim izrazom $0(10)^*$ generira desno-linearna gramatika $G_D = (\{S, A\}, \{0, 1\}, P, S)$ koja ima produkcije:

$$\begin{aligned} S &\rightarrow 0A \\ A &\rightarrow 10A \mid \epsilon \end{aligned}$$

Isti jezik generira lijevo-linearna gramatika $G_L = (\{S\}, \{0, 1\}, P, S)$ koja ima produkcije:

$$S \rightarrow S 10 \mid 0$$

Konstrukcija NKA iz desno-linearne gramatike

Jezik L jest regularan ako i samo ako postoji desno-linearna gramatika G_D koja generira jezik L , tj. $L=L(G_D)$, odnosno jezik L jest regularan ako i samo ako postoji lijevo-linearna gramatika G_L koja generira jezik L , tj. $L=L(G_L)$. Za bilo koju desno-linearnu ili lijevo-linearnu gramatiku G moguće je izgraditi konačni automat M koji prihvaca jezik koji generira zadana gramatika, tj. $L(M)=L(G)$. Lijevo-linearna ili desno-linearna gramatika preurede se tako da su sve produkcije oblika $A \rightarrow aB$ i $A \rightarrow \epsilon$, a nakon toga primjeni se algoritam za gradnju NKA koji je opisan u ovom odjeljku.

Primjer 2.20. Neka je zadana desno-linearna gramatika $G = (\{S, A\}, \{a, b, c\}, P, S)$, gdje je skup produkcija:

- | | |
|-----------------------|-----------------------------|
| 1) $S \rightarrow aA$ | 4) $A \rightarrow abbS$ |
| 2) $S \rightarrow bc$ | 5) $A \rightarrow cA$ |
| 3) $S \rightarrow A$ | 6) $A \rightarrow \epsilon$ |

Producije (1), (3), (4) i (5) su oblika $A \rightarrow wB$, a produkcije (2) i (6) su oblika $A \rightarrow w$. U produkcijama (3) i (6) niz w jest prazni niz, a u produkcijama (1) i (5) niz w sastoji se od jednog završnog znaka.

Producije se preuređuju tako da imaju oblik $A \rightarrow aB$ ili $A \rightarrow \epsilon$. Producije (1), (5) i (6) već su zadane u jednom od ova dva oblika i nije ih potrebno preuređivati. Potrebno je preuređiti produkcije (2), (3) i (4). Producija (4) $A \rightarrow abbS$ sastoji se od tri završna znaka ispred jednog nezavršnog znaka. Produciju $A \rightarrow abbS$ zamijene tri produkcije:

$$\begin{aligned} A &\rightarrow a[bbS] \\ [bbS] &\rightarrow b[bS] \\ [bS] &\rightarrow bS \end{aligned}$$

gdje su $[bbS]$ i $[bS]$ novi nezavršni znakovi.

Nove tri produkcije $A \rightarrow a[bbS]$, $[bbS] \rightarrow b[bS]$, $[bS] \rightarrow bS$ generiraju isti međuniz $A \Rightarrow a[bbS] \Rightarrow ab[bS] \Rightarrow abbS$ kao i jedna izvorna produkcija $A \rightarrow abbS$.

Produciju $S \rightarrow bc[\epsilon]$ zamijene dvije produkcije:

$$\begin{aligned} S &\rightarrow bc[\epsilon] \\ [\epsilon] &\rightarrow \epsilon \end{aligned}$$

gdje je $[\epsilon]$ novi nezavršni znak. Producija $S \rightarrow bc[\epsilon]$ preuredi se prethodno opisanim postupkom:

$$\begin{aligned} S &\rightarrow b[c\epsilon] \\ [c\epsilon] &\rightarrow c[\epsilon] \end{aligned}$$

Nove tri produkcije $S \rightarrow b[c\epsilon]$, $[c\epsilon] \rightarrow c[\epsilon]$ i $[\epsilon] \rightarrow \epsilon$ generiraju isti međuniz $S \Rightarrow b[c\epsilon] \Rightarrow bc[\epsilon] \Rightarrow bc$ kao i izvorna producija $S \rightarrow bc$.

Umjesto producije (3) $S \rightarrow A$ grade se nove produkcije:

$S \rightarrow$ desne strane producija u kojima se nezavršni znak A nalazi na lijevoj strani.

Znak A nalazi na lijevim stranama producija (4), (5) i (6). Producija (4) jest preuređena, te se za tu produciju gradi producija $S \rightarrow a[bbS]$. Na temelju producija (5) i (6) grade se produkcije:

$$\begin{aligned} S &\rightarrow cA \\ S &\rightarrow \epsilon \end{aligned}$$

Nakon preuređivanja gramatika ima sljedeće produkcije:

$$\begin{array}{lll} 1. \quad S \rightarrow aA & 3a. \quad S \rightarrow a[bbS] & 4b. \quad [bbS] \rightarrow b[bS] \\ 2a. \quad S \rightarrow b[c\epsilon] & 3b. \quad S \rightarrow cA & 4c. \quad [bS] \rightarrow bS \\ 2b. \quad [c\epsilon] \rightarrow c[\epsilon] & 3c. \quad S \rightarrow \epsilon & 5. \quad A \rightarrow cA \\ 2c. \quad [\epsilon] \rightarrow \epsilon & 4a. \quad A \rightarrow a[bbS] & 6. \quad A \rightarrow \epsilon \end{array}$$

Budući da su sve produkcije oblika $A \rightarrow aB$ ili $A \rightarrow \epsilon$, za preuređenu gramatiku moguće je konstruirati NKA koristeći prethodno opisani algoritam. Tablica prijelaza NKA prikazana je na slici 2.48.

	a	b	c	1
S	$A, [bbS]$	$[c\epsilon]$	A	1
$[c\epsilon]$			$[\epsilon]$	0
$[\epsilon]$				1
A	$[bbS]$		A	1
$[bbS]$		$[bS]$		0
$[bS]$				0

Slika 2.48: Tablica prijelaza NKA

Prethodni postupci preuređivanja producija desno-linearne gramatike u producije oblika $A \rightarrow aB$ i $A \rightarrow \epsilon$ opisuju se sljedećim algoritmom:

- 1) Za sve produkcije oblika:

$A \rightarrow w$, gdje je w neprazni niz završnih znakova,

doda se jedan novi nezavršni znak, na primjer $[\epsilon]$, i gradi se nova producija:

$$[\epsilon] \rightarrow \epsilon.$$

Sve produkcije oblika $A \rightarrow w$ zamijene se novim produkcijama oblika:

$$A \rightarrow w[\epsilon]$$

- 2) Sve produkcije oblika:

$$A \rightarrow a_1 \dots a_n B, \text{ za } n > 1,$$

zamijene se produkcijama oblika:

$$\begin{aligned} A &\rightarrow a_1 [a_2 \dots a_n B] \\ [a_2 \dots a_n B] &\rightarrow a_2 [a_3 \dots a_n B] \\ [a_3 \dots a_n B] &\rightarrow a_3 [a_4 \dots a_n B] \\ &\vdots \\ [a_i \dots a_n B] &\rightarrow a_i [a_{i+1} \dots a_n B], \text{ za } 1 < i < n \\ &\vdots \\ [a_{n-1} a_n B] &\rightarrow a_{n-1} [a_n B] \\ [a_n B] &\rightarrow a_n B \end{aligned}$$

gdje su $[a_i \dots a_n B]$ novi nezavršni znakovi, $1 < i \leq n$.

- 3) Ako je nezavršni znak B jedini znak desne strane produkcije:

$$A \rightarrow B,$$

onda se izbace sve produkcije koje imaju istu lijevu i desnu stranu:

$$B \rightarrow B.$$

Ostanu li produkcije koje imaju različite lijeve i desne strane $A \rightarrow B$, one se zamijene produkcijama:

$$A \rightarrow y$$

za sve kombinacije nezavršnih znakova A i desnih strana produkcija y za koje vrijedi:

$$A \rightarrow B \text{ i } B \rightarrow y.$$

Konstrukcija ϵ -NKA iz lijevo-linearne gramatike

Neka je $G = (V, T, P, S)$ lijevo-linearna gramatika. ϵ -NKA M' koji prihvata jezik $L(M') = L(G)$ konstruira se na sljedeći način:

- 1) Izgradi se desno-linearna gramatika $G' = (V, T, P', S)$. Skup produkcija P gramatike G preredi se tako da se desne strane produkcija napišu obrnutim redoslijedom:

$$P' = \{ A \rightarrow \alpha^R \mid A \rightarrow \alpha \text{ jest u skupu } P \}.$$

Gramatika G' generira nizove završnih znakova koji su obrnuto napisani nizovi znakova koje generira gramatika G , tj. $L(G') = L(G)^R$.

- 2) Na temelju desno-linearne gramatike G' konstruira se NKA M koji prihvata jezik $L(M) = L(G') = L(G)^R$.

- 3) Na temelju NKA M izgradi se ϵ -NKA M' koji prihvata jezik $L(M') = L(M)^R = L(G')^R = L(G)$, tj. ϵ -NKA M' prihvata jezik $L(G)$ koji generira lijevo-linearna gramatika G .

ϵ -NKA M' gradi se na sljedeći način:

NKA M preuredi se tako da ima samo jedno prihvatljivo stanje. Ima li NKA više prihvatljivih stanja, definira se novo jedinstveno prihvatljivo stanje i definiraju se ϵ -prijelazi iz svih prijašnjih prihvatljivih stanja u novo prihvatljivo stanje. U skupu prihvatljivih stanja ostaje samo novo prihvatljivo stanje. Za početno stanje ϵ -NKA M' uzima se prihvatljivo stanje NKA M , a za prihvatljivo stanje ϵ -NKA M' uzima se početno stanje NKA M . Funkcije prijelaza NKA M' grade se zamjenom smjera usmjerenih grana u dijagramu stanja. Izgrađeni ϵ -NKA M' prihvaća jezik $L(M')$. U jeziku $L(M')$ su nizovi koji su napisani obrnutim redoslijedom od nizova jezika $L(M)$ koji prihvaća NKA M .

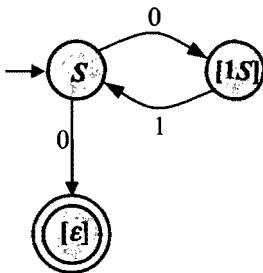
Primjer 2.21. Neka je zadana lijevo-linearna gramatika $G_L = (\{S\}, \{0, 1\}, P, S)$, gdje je skup produkcija:

$$P = \{ S \rightarrow S 10 \mid 0 \}.$$

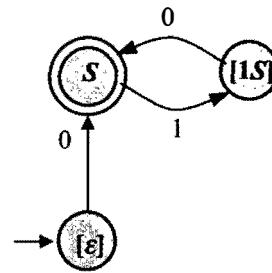
Gramatika generira jezik $0(10)^*$. Zamjenom redoslijeda znakova desnih strana produkcija izgradi se desno-linearna gramatika sa skupom produkcija:

$$P' = \{ S \rightarrow 01 S \mid 0 \}.$$

Gramatika G' generira jezik $(01)^*0$. Za izgrađenu desno-linearnu gramatiku konstruira se NKA. Izgrađeni NKA prikazan je na slici 2.49. NKA M na slici 2.49 prihvaća jezik $(01)^*0$. Zamjenom smjera usmjerenih grana na dijagramu stanja, te zamjenom uloga početnog i prihvatljivog stanja, dobije se NKA M' koji prihvaća jezik $0(10)^*$. NKA M' prikazan je na slici 2.50.



Slika 2.49: Dijagram stanja NKA M koji prihvaća jezik $(01)^*0$



Slika 2.50: Dijagram stanja NKA M' koji prihvaća jezik $0(10)^*$

Konstrukcija lijevo-linearne gramatike iz NKA

Konstrukcija lijevo-linearne gramatike za jezik L zasniva se na sljedećem postupku:

- 1) Konstruira se ϵ -NKA M koji prihvaća jezik $L(M)=L^R$.
- 2) Na temelju izgrađenog ϵ -NKA M konstruira se desno-linearna gramatika G za koju vrijedi $L(G)=L(M)=L^R$.
- 3) Desne strane produkcija napišu se obrnutim redoslijedom. Izgrađena gramatika G' jest lijevo-linearna gramatika za koju vrijedi $L(G')=L(G)^R=L(M)^R=L$.

3 KONTEKSTNO NEOVISNI JEZICI

Definicija kontekstno neovisnog jezika temelji se na kontekstno neovisnoj gramatici: neki jezik jest kontekstno neovisan ako i samo ako postoji kontekstno neovisna gramatika koja ga generira. Time je definirana istovjetnost kontekstno neovisne gramatike i kontekstno neovisnih jezika: za bilo koji kontekstno neovisni jezik moguće je izgraditi kontekstno neovisnu gramatiku koja ga generira, i obrnuto, bilo koja kontekstno neovisna gramatika generira jedan od kontekstno neovisnih jezika.

Klasa kontekstno neovisnih jezika sadrži kao podskup klasu regularnih jezika. U odjelu 2.4.1 dan je primjer jezika $N=\{wcw^R\}$ koji nije regularan. Međutim, jezik N moguće je generirati kontekstno neovisnom gramatikom, što potvrđuje da je kontekstno neovisan.

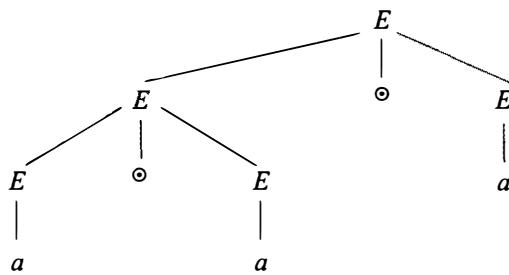
U odjelu 3.1 opisuje se kontekstno neovisna gramatika, a u odjelu 3.2 potisni automat. Svojstva kontekstno neovisnih jezika objašnjena su u odjelu 3.3.

3.1 Kontekstno neovisna gramatika

Definicija nejednoznačnosti kontekstno neovisne gramatike i način razrješavanja nejednoznačnosti dati su u odjelu 3.1.1. Postupci pojednostavljenja gramatike opisani su u odjelu 3.1.2. Različiti postupci parsiranja prikazani su u odjelu 3.1.3.

3.1.1 Nejednoznačnost gramatike, jezika i niza

Interpretacija niza zasniva se na generativnom stablu koje se gradi tijekom generiranja niza. Mogućnost gradnje više različitih stabala uzrokuje nejednoznačnost u interpretaciji niza. Na primjer, zadana je gramatika $G=(\{E\}, \{a, \odot\}, \{E \rightarrow E \odot E \mid a\}, E)$. Za niz $a \odot a \odot a$ moguće je izgraditi dva generativna stabla.

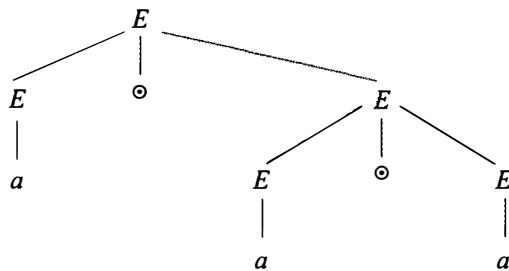


Slika 3.1: Generativno stablo za niz $a@a@a$ i gramatiku G

Niz $a@a@a$ generira se na sljedeći način:

- 1) $\underline{E} \Rightarrow \underline{E} \odot E \Rightarrow \underline{E} \odot E \odot E \Rightarrow a \odot \underline{E}$
 $\odot E \Rightarrow$
 $a \odot a \odot \underline{E} \Rightarrow a \odot a \odot a.$

Na podcrtane nezavršne znakove E primjeni se dva puta produkcija $E \rightarrow E \odot E$, a zatim tri puta produkcija $E \rightarrow a$. Na slici 3.1 prikazano je generativno stablo izrađeno postupkom generiranja niza (1).



Slika 3.2: Drugo generativno stablo za niz $a@a@a$ i gramatiku G

Promjenom redoslijeda primjena produkcija, isti niz $a@a@a$ generira se na sljedeći način:

- 2) $\underline{E} \Rightarrow \underline{E} \odot E \Rightarrow a \odot \underline{E} \Rightarrow a \odot \underline{E} \odot E$
 \Rightarrow
 $a \odot a \odot \underline{E} \Rightarrow a \odot a \odot a.$

Redoslijed primjene produkcija je sljedeći: $E \rightarrow E \odot E$, $E \rightarrow a$, $E \rightarrow E \odot E$ i dva puta produkcija $E \rightarrow a$. Na slici 3.2 prikazano je generativno stablo izgrađeno postupkom generiranja niza (2). Generativno stablo na slici 3.2 razlikuje se od generativnog stabla na slici 3.1, iako su oba generativna stabla izgrađena za istu gramatiku i za isti niz $a@a@a$.

Na gradnju stabla ima također utjecaja promjena redoslijeda nezavršnih znakova na koje se primjenjuju produkcije. U postupku generiranja niza:

- 3) $\underline{E} \Rightarrow E \odot \underline{E} \Rightarrow E \odot E \odot \underline{E} \Rightarrow E \odot \underline{E} \odot a \Rightarrow \underline{E} \odot a \odot a \Rightarrow a \odot a \odot a$

produkcije se primjenjuju istim redoslijedom kao i u postupku (1), ali se produkcije ne primjenjuju na iste nezavršne znakove. Producije se primjenjuju na nezavršne znakove koji su potcrtni. Postupak generiranja niza (3) gradi stablo prikazano na slici 3.2, dok postupak (1) gradi stablo prikazano na slici 3.1.

Primjenom produkcija istim redoslijedom kao i u postupku (2), ali ne na iste nezavršne znakove, gradi se stablo koje je jednako stablu prikazanom na slici 3.1, a ne stablu na slici 3.2:

- 4) $\underline{E} \Rightarrow E \odot \underline{E} \Rightarrow \underline{E} \odot a \Rightarrow E \odot \underline{E} \odot a \Rightarrow \underline{E} \odot a \odot a \Rightarrow a \odot a \odot a$

Generativno stablo određuje na koji način se interpretira generirani niz. Na primjer, neka zadana gramatika generira aritmetičke izraze, gdje je završni znak a konstanta ili varijabla, a završni znak \odot je operator. Generativno stablo definira prednost operatora, i to tako da se izrazi u zajedničkom podstablu grupiraju zajedno. Neka se grupiranje označi zagradama. Stablo na slici 3.1 grupira aritmetički izraz $a@a@a$ na sljedeći način:

- a) $(a \odot a) \odot a,$

a stablo na slici 3.2 grupira izraz $a@a@a$ na sljedeći način:

- b) $a \odot (a \odot a).$

Grupiranje završnih znakova određuje redoslijed izvođenja operacija. Grupiranje (a) zahtijeva primjenu operatora \odot na prve dvije varijable ili konstante, a zatim se isti operator primijeni na dobiveni rezultat i na treću varijablu ili konstantu. Za razliku od grupiranja (a), grupiranje (b) zahtijeva primjenu operatora \odot na drugu i treću varijablu ili konstantu, a zatim se isti operator primijeni na prvu varijablu ili konstantu i na prethodno dobiveni rezultat. Neka je \odot operator zbrajanja označen znakom $+$. Proces grupiranja nema utjecaja na zadani operator $+$ i krajnji rezultat, jer je $a_1 + (a_2 + a_3) = (a_1 + a_2) + a_3$. Međutim, ako je \odot operator oduzimanja označen znakom $-$, onda je $a_1 - (a_2 - a_3) \neq (a_1 - a_2) - a_3$, odnosno proces grupiranja ima utjecaja na krajnji rezultat.

Opisani primjer pokazuje važnu ulogu generativnog stabla u interpretaciji niza. Nastoji se konstruirati gramatika koja za zadani niz gradi samo jedno generativno stablo.

Na temelju opisanih postupaka generiranja niza, zaključuje se sljedeće:

- i) *Za bilo koji niz kontekstno neovisne gramatike moguće je izgraditi jedno ili više različitih generativnih stabala.* U prethodnom primjeru za niz $a\odot a\odot a$ izgrađena su dva različita stabla koja su prikazana na slikama 3.1 i 3.2.
- ii) *Bilo koje generativno stablo moguće je izgraditi primjenom jednog ili više različitih postupaka generiranja niza.* Na primjer, stablo prikazano na slici 3.1 gradi se postupkom (1) ili postupkom (4), a stablo prikazano na slici 3.2 gradi se postupkom (2) ili postupkom (3).

Generiranje niza zamjenom krajnje lijevog ili krajnje desnog nezavršnog znaka

Prethodno je pokazano da je redoslijed nezavršnih znakova na koje se primjenjuju produkcije značajan za postupak generiranja niza. Praktično se koriste dva postupka: *generiranje niza zamjenom krajnje lijevog nezavršnog znaka* i *generiranje niza zamjenom krajnje desnog nezavršnog znaka*.

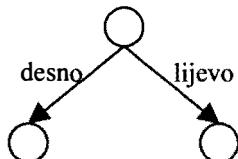
Postupak *zamjene krajnje lijevog nezavršnog znaka* jest postupak generiranja niza u kojemu se produkcije primjenjuju isključivo na krajnje lijeve nezavršne znakove u međunizu. Postupci (1) i (2) su primjeri generiranja niza zamjenom krajnje lijevog nezavršnog znaka. Produkcije u postupcima (1) i (2) primjenjuju se isključivo na nezavršne znakove koji su krajnje lijevi nezavršni znakovi u danom međunizu završnih i nezavršnih znakova:

Postupak (1)	Postupak (2)
E	E
$\underline{E} \odot E$	$E \odot \underline{E}$
$\underline{E} \odot E \odot E$	$a \odot \underline{E}$
$a \odot \underline{E} \odot E$	$a \odot \underline{E} \odot E$
$a \odot a \odot \underline{E}$	$a \odot a \odot \underline{E}$
$a \odot a \odot a$	$a \odot a \odot a$

Podcrtani su oni nezavršni znakovi na koje se primjenjuju produkcije.

U postupku *zamjene krajnje desnog nezavršnog znaka*, produkcije se primjenjuju isključivo na krajnje desne nezavršne znakove. Postupci (3) i (4) su primjeri postupaka generiranja niza zamjenom krajnjeg desnog nezavršnog znaka:

Postupak (3)	Postupak (4)
\underline{E}	\underline{E}
$E \odot \underline{E}$	$E \odot \underline{E}$
$E \odot E \odot \underline{E}$	$\underline{E} \odot a$
$E \odot \underline{E} \odot a$	$E \odot \underline{E} \odot a$
$\underline{E} \odot a \odot a$	$\underline{E} \odot a \odot a$
$a \odot a \odot a$	$a \odot a \odot a$



Slika 3.3: Određivanje smjera obilaska stabla

Postupak obilaska generativnog stabla određuje redoslijed kojim se obilaze grane i čvorovi. Uobičajeno je da se stablo obilazi primjenom *desnog obilaska stabla* ili primjenom *lijevog obilaska stabla*. Obilazak stabla započinje od korijena stabla. Desni obilazak stabla rekurzivno obilazi sve neobiđene desne grane i čvorove. Lijevi obilazak stabla rekurzivno obilazi lijeve neobiđene grane i čvorove. Smjer desno i lijevo određuje se tako da se stablo promatra od korijena stabla prema listovima, kao što je to prikazano na slici 3.3.

Postupci obilaska stabla jednoznačno definiraju redoslijed primjene produkcija i redoslijed nezavršnih znakova na koje se te produkcije primjenjuju. Na primjer, desni obilazak stabla sa slike 3.1 definira postupak generiranja niza (1), odnosno postupak generiranja niza zamjenom krajnje lijevog nezavršnog znaka. Lijevi obilazak stabla definira postupak (3), odnosno postupak generiranja niza zamjenom krajnje desnog nezavršnog znaka. Postupak (1) je jedini postupak generiranja niza zamjenom krajnje lijevog nezavršnog znaka koji gradi stablo prikazano na slici 3.1, dok je postupak (3) jedini postupak generiranja niza zamjenom krajnje desnog nezavršnog znaka koji gradi to isto stablo. Budući da zadano stablo nije moguće izgraditi primjenom više različitih postupaka generiranja niza zamjenom krajnje desnog ili zamjenom krajnje lijevog nezavršnog znaka, prethodno dani zaključci proširuju se sljedećom tvrdnjom:

- iii) Bilo koje generativno stablo moguće je izgraditi primjenom jednog i samo jednog postupka generiranja niza zamjenom krajnjeg lijevog nezavršnog znaka. Bilo koje generativno stablo moguće je izgraditi primjenom jednog i samo jednog postupka generiranja niza zamjenom krajnjeg desnog nezavršnog znaka.

Generativno stablo na slici 3.2 jednoznačno određuje postupke (2) i (4) kao postupke generiranja niza zamjenom krajnje lijevog i desnog nezavršnog znaka.

Nejednoznačnost i razrješavanje nejednoznačnosti

Nejednoznačnost kontekstno neovisne gramatike G definira se na sljedeći način:

Ako je moguće za neki niz $w \in L(G)$ izgraditi više različitih generativnih stabala, kontekstno neovisna gramatika G je nejednoznačna.

Uzme li se u obzir tvrdnja (iii), definiciju nejednoznačnosti moguće je zadati na sljedeći način:

Ako je moguće neki niz $w \in L(G)$ generirati primjenom više različitih postupaka generiranja niza zamjenom krajnje lijevog nezavršnog znaka ili primjenom više različitih postupaka generiranja niza zamjenom krajnje desnog nezavršnog znaka, onda je gramatika G nejednoznačna.

Gramatika $G = (\{E\}, \{a, \odot\}, \{E \rightarrow E \odot E \mid a\}, E)$ je nejednoznačna gramatika, jer je moguće izgraditi dva stabla za niz $a \odot a \odot a$ koja su prikazana na slikama 3.1 i 3.2, odnosno niza $a \odot a \odot a$ moguće je generirati primjenom dvaju postupaka generiranja niza zamjenom krajnje lijevog nezavršnog znaka (postupci (1) i (2))

i primjenom dvaju postupka generiranja niza zamjenom krajnjeg desnog nezavršnog znaka (postupci (3) i (4)).

Nejednoznačnost niza w definira se na sljedeći način:

Ako je moguće za niz $w \in L(G)$ izgraditi više različitih generativnih stabala, niz je nejednoznačan za zadanu gramatiku G .

Niz $a \odot a \odot a$ je nejednoznačni niz za gramatiku $G = (\{E\}, \{a, \odot\}, \{E \rightarrow E \odot E \mid a\}, E)$, jer je za taj niz moguće izgraditi dva generativna stabla.

Nejednoznačnost jezika L definira se na sljedeći način:

Ako nije moguće jezik L generirati niti jednom jednoznačnom gramatikom G , onda je jezik inheretno nejednoznačan.

Jezik $L_n = L_1 \cup L_2 = \{a^n b^m c^m d^m \mid n \geq 1, m \geq 1\} \cup \{a^n b^m c^m d^n \mid n \geq 1, m \geq 1\}$ je primjer inheretno nejednoznačnog jezika¹. Nejednoznačnost nekog jezika L razriješava se na dva načina. Umjesto gramatike G izgradi se nova jednoznačna gramatika G' koja generira jezik L . Drugi način razriješavanja nejednoznačnosti jest promjena jezika L u novi jezik L' koji je moguće generirati jednoznačnom gramatikom G' .

Promjena gramatike

Jezik L , koji generira nejednoznačna gramatika $G = (\{E\}, \{a, \odot\}, \{E \rightarrow E \odot E \mid a\}, E)$, moguće je generirati primjenom više različitih jednoznačnih gramatika.

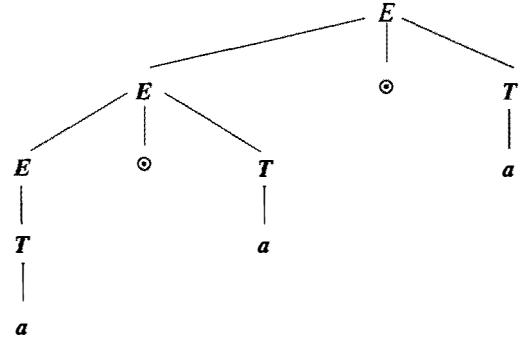
Za lijevo asocijativni operator \odot gradi se sljedeća jednoznačna gramatika: $G_1 = (\{E, T\}, \{a, \odot\}, \{E \rightarrow E \odot T \mid T, T \rightarrow a\}, E)$. Niz $a \odot a \odot a$ je jednoznačan za gramatiku G_1 . Niz $a \odot a \odot a$ moguće je generirati samo jednim postupkom generiranja niza zamjenom krajnjem lijevog nezavršnog znaka:

$$(5) \quad \begin{aligned} E &\Rightarrow \underline{E} \odot T \Rightarrow \underline{E} \odot \underline{T} \odot T \Rightarrow \underline{\underline{T}} \odot T \odot T \\ &\Rightarrow a \odot T \odot T \Rightarrow a \odot a \odot T \Rightarrow a \odot a \odot a, \end{aligned}$$

i samo jednim postupkom generiranja niza zamjenom krajnjem desnom nezavršnog znaka:

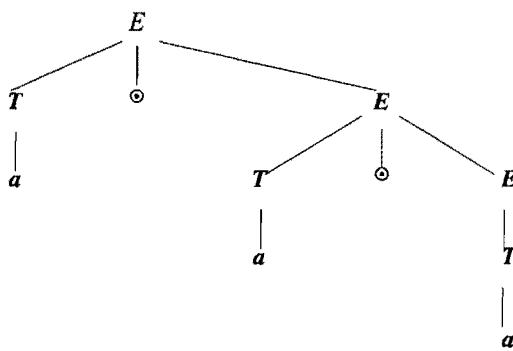
$$(6) \quad \begin{aligned} E &\Rightarrow E \odot \underline{T} \Rightarrow \underline{E} \odot a \Rightarrow E \odot \underline{\underline{T}} \odot a \Rightarrow \\ &E \odot a \odot a \Rightarrow T \odot a \odot a \Rightarrow a \odot a \odot a, \end{aligned}$$

Postupci (5) i (6) grade generativno stablo prikazano na slici 3.4.



Slika 3.4: Generativno stablo za niz $a \odot a \odot a$ i gramatiku G_1

¹ Jezici L_1 i L_2 su kontekstno neovisni jezici, jer se mogu generirati kontekstno neovisnim gramatikama G_1 i G_2 . Gramatike su slične gramatici koja je opisana u primjeru 2.15. U odjeljku 3.2.4 pokazano je da unija dvaju kontekstno neovisnih jezika jest kontekstno neovisni jezik. Opisan je postupak konstrukcije gramatike za uniju dvaju jezika zadanih kontekstno neovisnom gramatikom. Kontekstno neovisna gramatika G_n , koja generira jezik L_n , sadrži produkcije gramatike G_1 i G_2 . Nadalje, obje gramatike G_1 i G_2 generiraju niz oblika $a^n b^m c^m d^m$ (niz $a^n b^m c^m d^m \in L$ za $n=m, n \geq 1$). Dokaz da je jezik L_n inheretno nejednoznačan je složen, a zasniva se na činjenici da ne postoji gramatika G_n za koju je niz oblika $a^n b^m c^m d^m$ jednoznačni niz.



Slika 3.5: Generativno stablo za niz $a@a@a$ i gramatiku G_2

Za desno asocijativni operator \circ gradi se sljedeća jednoznačna gramatika: $G_2 = (\{E, T\}, \{a, \circ\}, \{E \rightarrow T \circ E \mid T, T \rightarrow a\}, E)$. Niz $a@a@a$ moguće je generirati samo jednim postupkom generiranja niza zamjenom krajnje lijevog nezavršnog znaka:

$$(7) \quad \begin{aligned} \underline{E} &\Rightarrow \underline{T} \circ E \Rightarrow a \circ \underline{E} \Rightarrow a \circ a \circ \underline{T} \circ E \\ &\Rightarrow a \circ a \circ \underline{E} \Rightarrow a \circ a \circ \underline{T} \Rightarrow a \circ a \end{aligned}$$

i samo jednim postupkom generiranja niza zamjenom krajnje desnog nezavršnog znaka:

$$(8) \quad \begin{aligned} \underline{E} &\Rightarrow T \circ \underline{E} \Rightarrow T \circ T \circ \underline{E} \Rightarrow T \circ T \\ &\circ \underline{T} \Rightarrow T \circ \underline{T} \circ a \Rightarrow \underline{T} \circ a \circ a \Rightarrow a \circ a \end{aligned}$$

Oba postupka grade generativno stablo prikazano na slici 3.5. Prethodni primjeri pokazuju da izbor jednoznačne gramatike određuje način gradnje generativnog stabla.

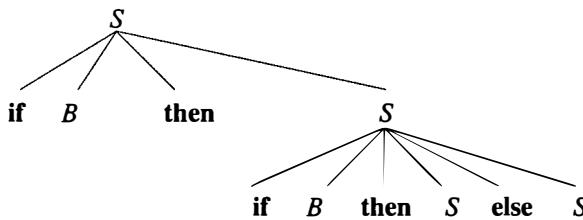
Primjer 2.1. Gramatika $G = (\{S, B\}, \{\text{if, then, else}\}, \{S \rightarrow \text{if } B \text{ then } S \text{ else } S \mid \text{if } B \text{ then } S, B \rightarrow \text{true} \mid \text{false}\}, S)$ generira složene naredbe programskog jezika. Nezavršni znak S definira naredbu programskog jezika, a nezavršni znak B logički izraz. Gramatika nije jednoznačna, jer je moguće za međuniz $\text{if } B \text{ then if } B \text{ then } S \text{ else } S$ izgraditi dva različita generativna stabla. Stabla i postupci generiranja međuniza $\text{if } B \text{ then if } B \text{ then } S \text{ else } S$ prikazani su na slikama 3.6 i 3.7.

Generativno stablo definira kojem se **if** pridružuje **else**. Budući da je moguće za međuniz $\text{if } B \text{ then if } B \text{ then } S \text{ else } S$ izgraditi dva generativna stabla, nije moguće jednoznačno interpretirati zadani međuniz. Interpretacija složene naredbe **if true then if false then PRINT("X") else PRINT("Y")** prema generativnom stablu sa slike 3.6 daje ispis "Y". Znak **else** pridružuje se onom znaku **if** koji je bliži znaku **else**, što se interpretira na sljedeći način:

```
if true then
  if false then
    PRINT("X")
  else
    PRINT("Y").
```

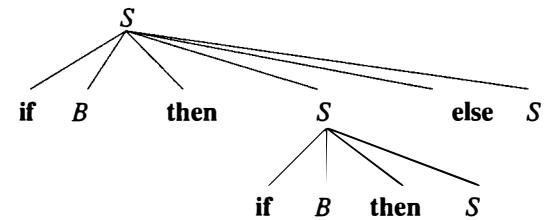
Interpretacija složene naredbe **if true then if false then PRINT("X") else PRINT("Y")** prema stablu sa slike 3.7 ne daje nikakav ispis. Znak **else** pridružuje se prvom znaku **if** u nizu, što ima za posljedicu sljedeću interpretaciju:

```
if true then
  if false then
    PRINT("X")
  else
    PRINT("Y").
```



Slika 3.6: Generativno stablo za postupak generiranja niza:

$S \Rightarrow \text{if } B \text{ then } \underline{S} \Rightarrow \text{if } B \text{ then if } B \text{ then } S \text{ else } S$



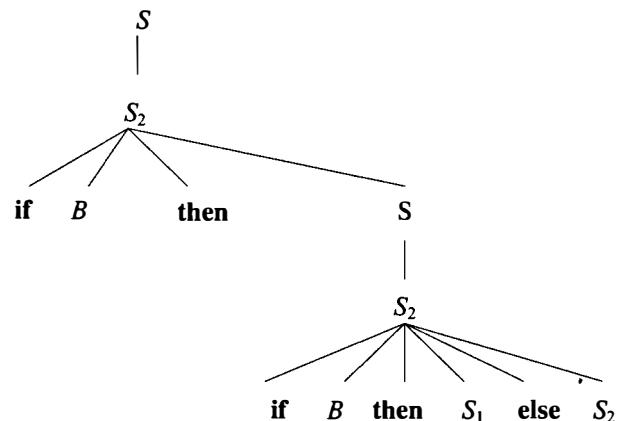
Slika 3.7: Generativno stablo za postupak generiranja niza:

$S \Rightarrow \text{if } B \text{ then } \underline{S} \Rightarrow \text{if } B \text{ then if } B \text{ then } S \text{ else } S$

Prva definicija programskog jezika ALGOL 60 bila je nejednoznačna upravo zbog naredbe grananja *if-then-else*. Jednoznačna gramatika gradi se tako da se izabere jedna od gornje dvije interpretacije. Uobičajeno je da se *else* pridružuje bližem *if*. Jednoznačna gramatika $G_1 = (\{S, S_1, S_2, B\}, \{\text{if, then, else}\}, P, S)$ ima sljedeće produkcije:

$$\begin{aligned} S &\rightarrow S_1 \mid S_2 \\ S_1 &\rightarrow \text{if } B \text{ then } S_1 \text{ else } S_2 \\ S_2 &\rightarrow \text{if } B \text{ then } S \mid \text{if } B \text{ then } S_1 \text{ else } S_2 \end{aligned}$$

Slika 3.7 prikazuje jedino moguće generativno stablo i postupak generiranja za međuniz $\text{if } B \text{ then if } B \text{ then } S_1 \text{ else } S_2$. U definiciji nezavršnog znaka S_1 ne dozvoljava se produkcija $S_1 \rightarrow \text{if } B \text{ then } S$. Budući da se nezavršni znak S_1 nalazi ispred znaka S_2 u međunizu $\text{if } B \text{ then } S_1 \text{ else } S_2$, nije moguće izgraditi stablo prikazano na slici 3.7.



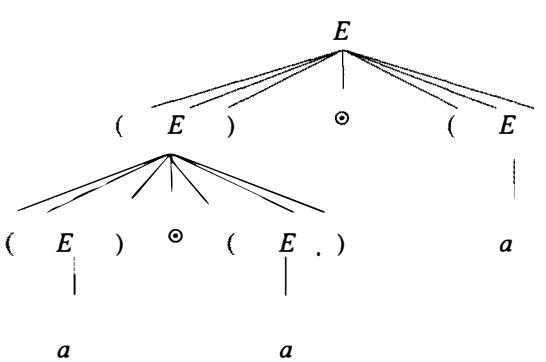
Slika 3.8: Generativno stablo za postupak generiranja niza:

$S \Rightarrow \underline{S_2} \Rightarrow \text{if } B \text{ then } \underline{S} \Rightarrow \text{if } B \text{ then } \underline{S_2} \Rightarrow \text{if } B \text{ then if } B \text{ then } S_1 \text{ else } S_2$

Promjena jezika

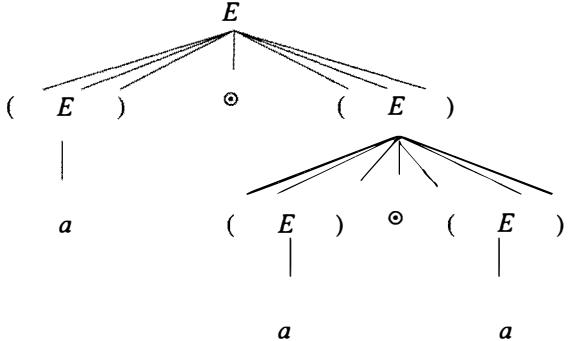
Tri su osnovna razloga razrješavanja nejednoznačnosti promjenom jezika: jezik je inheretno nejednoznačan, jednoznačna gramatika je previše složena, ili se žele sačuvati sve moguće interpretacije pojedinih nizova. Primjer promjene jezika su zagrade. Zagrade su završni znakovi gramatike i one su dio niza koji generira gramatika.

Prethodno je pokazano da je gramatika $G = (\{\dot{E}\}, \{a, \circ\}, \{E \rightarrow E \circ E \mid a\}, E)$ nejednoznačna. Ovisno o generativnom stablu, moguće su dvije interpretacije niza $a \circ a \circ a$. Operator \circ može biti lijevo asocijativan ili desno asocijativan. Dodavanjem zagrada u niz $a \circ a \circ a$ moguće je definirati redoslijed primjena operatora \circ i izbjegći nejednoznačnost. U novom jeziku umjesto jednog niza $a \circ a \circ a$ su dva različita niza: $((a) \circ (a)) \circ (a)$ i $(a) \circ ((a) \circ (a))$. Neka gramatika $G_3 = (\{E\}, \{a, \circ, (\), \}, \{E \rightarrow (E) \circ (E) \mid a\}, E)$ generira novi jezik. Oba niza $((a) \circ (a)) \circ (a)$ i $(a) \circ ((a) \circ (a))$ su jednoznačna za zadatu gramatiku G_3 . Stabla i postupci generiranja prikazani su na slikama 3.9 i 3.10.



Slika 3.9: Generativno stablo za niz $((a) \odot (a)) \odot (a)$ i gramatiku G_3

$$E \xrightarrow{*} (E) \odot (E) \xrightarrow{*} ((E) \odot (E)) \odot (E) \xrightarrow{*} ((a) \odot (a)) \odot (a)$$



Slika 3.10: Generativno stablo za niz $(a) \odot ((a) \odot (a))$ i gramatiku G_3 ,
 $E \Rightarrow (E) \odot (E) \Rightarrow (a) \odot (E) \Rightarrow$
 $((a) \odot ((E) \odot (E))) \xrightarrow{*} (a) \odot ((a) \odot (a))$

Važno je uočiti razliku između postupka razrješavanja nejednoznačnosti promjenom gramatike i postupka razrješavanja nejednoznačnosti promjenom jezika. Promjenom gramatike ne mijenja se jezik, ali se odbacuje višestruko značenje jednog niza. Promjenom jezika zadržava se višestruko značenje uvođenjem zasebnog niza za svaku interpretaciju. Gramatika $G_1 = (\{E, T\}, \{a, \odot\}, \{E \rightarrow E \odot T \mid T, T \rightarrow a\}, E)$ prethodno definirana generira isti jezik kao i gramatika G , ali se niz $a \odot a \odot a$ interpretira na jedinstven način: operacije se računaju od lijeva na desno. Želi li se izraz računati sa desna na lijevo, potrebno je niz generirati drugom gramatikom $G_2 = (\{E, T\}, \{a, \odot\}, \{E \rightarrow T \odot E \mid T, T \rightarrow a\}, E)$. Gramatika $G_3 = (\{E\}, \{a, \odot\}, \{E \rightarrow (E) \odot (E) \mid a, E\})$ generira novi jezik u kojem su dva različita niza $((a) \odot (a)) \odot (a)$ i $((a) \odot ((a) \odot (a)))$ koji se na različite načine interpretiraju.

U programskim jezicima koriste se razni znakovi kao što su vitičaste zagrade {}, uglate zagrade [] i parovi **begin** i **end**. Par **begin-end** ili vitičaste zagrade {} koriste se za definiranje složenih naredaba programskega jezika i dijelova programa, dok se zagrade () koriste za aritmetičke i logičke izraze.

3.1.2 Pojednostavljenje gramatike

Postupci pojednostavljenja gramatike odbacuju beskorisne znakove i produkcije. Postoji više postupaka pojednostavljenja produkcija kontekstno neovisne gramatike. Za bilo koji neprazni kontekstno neovisni jezik L moguće je izgraditi kontekstno neovisnu gramatiku G sa svojstvima (i) do (iii):

- i) Bilo koji znak gramatike G pojavi se u postupku generiranja barem jednog niza jezika L .

Postoji li postupak generiranja $S \xrightarrow{*} \alpha X \beta \xrightarrow{*} w$, gdje su $\alpha, \beta \in (V \cup T)^*$, a $w \in T^*$, znak X gramatike $G = (V, T, P, S)$ je *koristan*. U suprotnom, znak je *beskoristan*. Postoje dva vida beskorisnosti: znak je *mrtav* ili *nedohvatljiv*. Ako nije moguće iz znaka X generirati niz završnih znakova, tj. ne postoji postupak generiranja $X \xrightarrow{*} w_X$, gdje je $w_X \in T^*$, onda je znak X mrtav. Nije li znak mrtav, kaže se da je živ. Ako znak X nije niti u jednom nizu koji se generira iz početnog nezavršnog znaka, tj. ne postoji postupak generiranja $S \xrightarrow{*} \alpha X \beta$, onda je znak X nedohvatljiv.

Neka je znak X dohvatljiv i živ, tj. neka vrijedi $S \xrightarrow{*} \alpha X \beta$ i $X \xrightarrow{*} w_X$, gdje je $w_X \in T^*$. Nije nužno istinito da je znak X koristan. Može se dogoditi da u nizu $\alpha X \beta$, koji se dobije kao $S \xrightarrow{*} \alpha X \beta$, jedan od

podnizova α ili β sadrži mrtvi znak. Ako u bilo kojem postupku generiranja $S \xrightarrow{*} \alpha X \beta$ barem jedan podniz α ili β sadrži mrtvi znak, znak X nije koristan, jer ne postoji postupak generiranja $S \xrightarrow{*} \alpha X \beta \xrightarrow{*} w, w \in T^*$.

- ii) Gramatika G nema produkcija oblika $A \rightarrow B$, gdje su A i B nezavršni znakovi.

Producije oblika $A \rightarrow B$ su *jedinične produkcije*. Sve ostale produkcije, uključujući i produkcije oblika $A \rightarrow a$ i $A \rightarrow \epsilon$, nazivaju se *nejedinične produkcije*.

- iii) Ako prazni niz ϵ nije element jezika L , onda je moguće izbjegći korištenje produkcija oblika $A \rightarrow \epsilon$. Producije oblika $A \rightarrow \epsilon$ nazivaju se ϵ -produkcije.

Nema li pravnog niza ϵ u jeziku, produkcije gramatike G mogu se preuređiti tako da su sve oblika $A \rightarrow BC$ i $A \rightarrow a$, gdje su A, B i C nezavršni znakovi, a a je završni znak. Producije $A \rightarrow BC$ i $A \rightarrow a$ su u Chomskyjevom normalnom obliku. Nadalje, produkcije gramatike G mogu se preuređiti tako da su sve oblika $A \rightarrow a\alpha$, gdje je a završni znak, a α je niz nezavršnih znakova koji može biti i prazan. Producije $A \rightarrow a\alpha$ su u Greibachovom normalnom obliku.

Algoritmi odbacivanja beskorisnih znakova, jediničnih produkcija, ϵ -produkcija, pretvorbe produkcija u Chomskyjev normalni oblik i pretvorbe produkcija u Greibachov normalni oblik detaljno su opisani u nastavku odjeljka.

Odbacivanje mrtvih znakova

Neka kontekstno neovisna gramatika $G=(V, T, P, S)$ generira neprazni jezik, tj. $L(G) \neq \emptyset$. Moguće je izgraditi istovjetnu gramatiku² $G'=(V', T, P', S)$ koja nema mrtvih znakova, odnosno za bilo koji $A \in V'$ vrijedi $A \xrightarrow{*} w, w \in T^*$.

Dvije gramatike su istovjetne ako i samo ako generiraju isti jezik, odnosno ako vrijedi $L(G)=L(G')$.

Primjer 2.2. Zadana je gramatika $G=(\{S, A, B\}, \{a, b, c, d, f\}, P, S)$ sa sljedećim produkcijama:

- | | |
|--------------------------|--------------------------|
| 1) $S \rightarrow a S a$ | 4) $A \rightarrow c B d$ |
| 2) $S \rightarrow b A d$ | 5) $A \rightarrow a A d$ |
| 3) $S \rightarrow c$ | 6) $B \rightarrow d A f$ |

Nezavršni znakovi A i B su mrtvi znakovi. Znak A zamijeni se desnim stranama produkcija (4) ili (5) u kojima se nalazi znak B ili znak A . Znak B zamijeni se desnom stranom produkcije (6) u kojoj je ponovno znak A . Budući da su znakovi A i B na lijevim stranama samo u produkcijama (4), (5) i (6), nemoguće je zamijeniti te nezavršne znakove isključivo završnim znakovima. Odbacivanjem svih produkcija u kojima se nalaze mrtvi znakovi A i B , odnosno odbacivanjem produkcija (2), (4), (5) i (6), dobije se istovjetna gramatika $G'=(\{S\}, \{a, b, c, d, f\}, \{S \rightarrow aSa, S \rightarrow c\}, S)$.

Algoritam traženja živih znakova zasniva se na sljedećem svojstvu:

- a) Ako su živi svi znakovi X_1, X_2, \dots, X_n desne strane produkcije:

$$A \rightarrow X_1 X_2 \dots X_n,$$

onda je živ i nezavršni znak A lijeve strane produkcije.

² Napomena, dvije gramatike su istovjetne ako i samo ako generiraju isti jezik, odnosno ako vrijedi $L(G)=L(G')$.

Budući da nema mrtvih znakova, za bilo koji X_i vrijedi $X_i \Rightarrow w_i$, $w_i \in T^*$. Nadalje, vrijedi da je $A \Rightarrow X_1 X_2 \dots X_n \stackrel{*}{\Rightarrow} w_1 w_2 \dots w_n$, gdje je $w_1 w_2 \dots w_n = w$ niz završnih znakova.

Algoritam traženja živih znakova izvodi se u tri koraka:

- 1) U listu živih znakova stave se lijeve strane produkcija koje na desnoj strani nemaju nezavršnih znakova.
- 2) Postoji li produkcija koja na desnoj strani ima isključivo žive znakove, nezavršni znak lijeve strane produkcije dodaje se u listu živih znakova.
- 3) Ako nije moguće proširiti listu živih znakova, onda se algoritam zaustavlja. Svi znakovi koji nisu u listi živih znakova su mrtvi znakovi.

Detaljni opis algoritma traženja živih znakova dat je na slici 3.11.

```

Staralistaživih = Ø;
NovaListaživih = {A | A → w i w ∈ T*};

dok ( Staralistaživih != NovaListaživih )
{
    Staralistaživih = NovaListaživih;
    NovaListaživih = Staralistaživih ∪ {A | A → α i α ∈ (T ∪ Staralistaživih)*};
}

Listaživih = NovaListaživih;

```

Slika 3.11: Algoritam traženja živih znakova

Primjer 2.3. Zadana je gramatika $G = (\{S, A, B, C\}, \{a, b, c, d\}, P, S)$ sa sljedećim produkcijama:

- | | | |
|------------------------------|--------------------------|--------------------------|
| 1) $S \rightarrow a A B S$ | 4) $A \rightarrow c S A$ | 7) $B \rightarrow c S B$ |
| 2) $S \rightarrow b C A C d$ | 5) $A \rightarrow c C C$ | 8) $C \rightarrow c S$ |
| 3) $A \rightarrow b A B$ | 6) $B \rightarrow b A B$ | 9) $C \rightarrow c$ |

Budući da produkcija (9) na desnoj strani ima završni znak, algoritam stavlja znak C u listu živih znakova. Na temelju produkcije (5), koja na desnoj strani ima samo znakove koji su živi (C je stavljen u listu živih znakova u prethodnom koraku), u listu živih znakova stavlja se znak A . Na temelju produkcije (2), koja na desnoj strani ima nezavršne znakove A i C koji su prethodno stavljeni u listu živih znakova, u listu se stavlja znak S . Algoritam se zaustavlja, jer nije moguće više proširiti listu živih znakova. Živi znakovi su S , A i C , dok je znak B mrtav. Odbacivanjem znaka B i odbacivanjem produkcija u kojima se nalazi znak B , nastaje istovjetna gramatika $G' = (\{S, A, C\}, \{a, b, c, d\}, P, S)$ sa sljedećim produkcijama:

- | | | |
|------------------------------|--------------------------|------------------------|
| 2) $S \rightarrow b C A C d$ | 4) $A \rightarrow c S A$ | 8) $C \rightarrow c S$ |
| | 5) $A \rightarrow c C C$ | 9) $C \rightarrow c$ |

Odbacivanje nedohvatljivih znakova

Moguće je izgraditi gramatiku $G' = (V', T', P', S)$ koja je istovjetna kontekstno neovisnoj gramatici $G = (V, T, P, S)$ i koja nema nedohvatljivih znakova, odnosno za bilo koji $X \in V' \cup T'$ vrijedi $S \stackrel{*}{\Rightarrow} \alpha X \beta$, gdje je $\alpha, \beta \in (V' \cup T')^*$.

Primjer 2.4. Zadana je gramatika $G=(\{S, A\}, \{a, b, c\}, P, S)$ sa sljedećim produkcijama:

- | | |
|--------------------------|------------------------|
| 1) $S \rightarrow a S b$ | 3) $A \rightarrow b S$ |
| 2) $S \rightarrow c$ | 4) $A \rightarrow a$ |

Producije (1) i (2), koje imaju početni nezavršni znak S na lijevoj strani, nemaju nezavršni znak A na desnoj strani. Nije moguće generirati međuniz znakova, počev od znaka S , koji ima barem jedan znak A . Budući da je znak A nedohvatljiv, taj se znak odbacuje zajedno s produkcijama u kojima se nalazi. Pojednostavljena gramatika $G'=(\{S, A\}, \{a, b, c\}, \{S \rightarrow aSb, S \rightarrow c\}, S)$ istovjetna je zadanoj gramatici G .

Algoritam traženja dohvatljivih znakova zasniva se na sljedećem svojstvu:

- b) Ako je dohvatljiv nezavršni znak A lijeve strane produkcije:

$$A \rightarrow \alpha_1 | \alpha_2 | \dots | \alpha_n,$$

onda su dohvatljivi svi završni i nezavršni znakovi u nizovima $\alpha_1, \alpha_2, \dots$ i α_n desne strane produkcije.

Neka je S početni nezavršni znak. Budući da je A dohvatljiv znak, vrijedi $S \xrightarrow{*} \beta A \gamma$. Na temelju produkcija $A \rightarrow \alpha_1 | \alpha_2 | \dots | \alpha_n$ vrijedi $S \xrightarrow{*} \beta A \gamma \Rightarrow \beta \alpha_i \gamma$, za bilo koji α_i , gdje je $1 \leq i \leq n$. Svi znakovi niza α_i su dohvatljivi, jer za njih vrijedi $S \xrightarrow{*} \beta \alpha_i \gamma$.

Algoritam traženja dohvatljivih znakova izvodi se u tri koraka:

- 1) U listu dohvatljivih znakova stavi se početni nezavršni znak gramatike.
- 2) Nalazi li se lijeva strana produkcije u listi dohvatljivih znakova, svi znakovi desne strane produkcije dodaju se u listu dohvatljivih znakova.
- 3) Ako listu dohvatljivih znakova nije moguće proširiti, onda se algoritam zaustavlja. Svi znakovi koji nisu u listi dohvatljivih znakova su nedohvatljivi.

Detaljni opis algoritma traženja dohvatljivih znakova dat je na slici 3.12.

```

StaraListaDohvatljivih = Ø;
NovaListaDohvatljivih = {S | S je početni nezavršni znak};

dok ( StaraListaDohvatljivih != NovaListaDohvatljivih )
{
    StaraListaDohvatljivih = NovaListaDohvatljivih;
    NovaListaDohvatljivih = StaraListaDohvatljivih ∪ {X | X je u nizu αi, A → αi
    i A ∈ StaraListaDohvatljivih};
}
ListaDohvatljivih = NovaListaDohvatljivih;

```

Slika 3.12: Algoritam traženja dohvatljivih znakova

Primjer 2.5. Zadana je gramatika $G=(\{S, A, B, C, D, E\}, \{a, b, c, d, e, f, g\}, P, S)$ sa sljedećim produkcijama:

- | | | | |
|--------------------------|------------------------|--------------------------|-------------------------|
| 1) $S \rightarrow a A B$ | 4) $A \rightarrow e$ | 7) $C \rightarrow c A B$ | 10) $D \rightarrow e A$ |
| 2) $S \rightarrow E$ | 5) $B \rightarrow b E$ | 8) $C \rightarrow d S D$ | 11) $E \rightarrow f A$ |
| 3) $A \rightarrow d D A$ | 6) $B \rightarrow f$ | 9) $C \rightarrow a$ | 12) $E \rightarrow g$ |

U listu dohvatljivih znakova upiše se početni nezavršni znak S . Budući da je lijeva strana produkcija (1) i (2) znak S , lista se nadopuni nezavršnim znakovima A, B i E , te završnim znakom a , jer su ti znakovi na desnim stranama produkcija (1) i (2). Na temelju produkcija koje imaju znakove A, B , ili E na lijevoj strani, lista se nadopuni nezavršnim znakom D i završnim znakovima d, e, b, f i g . Producija (10), koja ima znak D na lijevoj strani, ne proširuje listu dohvatljivih znakova, jer su znakovi desne strane prethodno upisani u listu. Na kraju rada algoritma, u listi dohvatljivih znakova su nezavršni znakovi S, A, B, E i D , te svi završni znakovi. Nezavršni znak C je nedohvatljiv i zato se taj znak odbacuje kao i produkcije (7), (8) i (9). Pojednostavljena istovjetna gramatika $G' = (\{S, A, B, D, E\}, \{a, b, d, e, f, g\}, P, S)$ ima produkcije:

- | | | |
|--------------------------|------------------------|-------------------------|
| 1) $S \rightarrow a A B$ | 4) $A \rightarrow e$ | 10) $D \rightarrow e A$ |
| 2) $S \rightarrow E$ | 5) $B \rightarrow b E$ | 11) $E \rightarrow f A$ |
| 3) $A \rightarrow d D A$ | 6) $B \rightarrow f$ | 12) $E \rightarrow g$ |

Odbacivanje beskorisnih znakova

Primjenom algoritma odbacivanja mrtvih znakova, a zatim algoritma odbacivanja nedohvatljivih znakova, iz gramatike se odbacuju svi beskorisni znakovi. Primjena algoritama obrnutim redoslijedom neće nužno odbaciti sve beskorisne znakove, što se pokazuje sljedećim primjerom.

Primjer 2.6. Zadana je gramatika $G = (\{S, A, B\}, \{a\}, P, S)$ sa sljedećim produkcijama:

$$\begin{aligned} S &\rightarrow A B \mid a \\ A &\rightarrow a \end{aligned}$$

Prvo se primjeni alogitam traženja živih znakova. Znakovi S i A su živi, a znak B je mrtav. Odbacivanjem znaka B i produkcije $S \rightarrow A B$ nastaje gramatika $G_1 = (\{S, A\}, \{a\}, P, S)$ sa sljedećim produkcijama:

$$\begin{aligned} S &\rightarrow a \\ A &\rightarrow a \end{aligned}$$

Algoritam traženja dohvatljivih znakova određuje da su nezavršni znakovi S i završni znak a dohvatljivi. Znak A je nedohvatljiv. Odbacivanjem produkcije $A \rightarrow a$ nastaje gramatika $G_2 = (\{S\}, \{a\}, P, S)$ sa sljedećom produkcijom:

$$S \rightarrow a$$

Primjeni li se alogitam traženja dohvatljivih znakova na gramatiku G , svi znakovi su dohvatljivi i alogitam odbacivanja mrtvih znakova daje gramatiku G_1 kao konačni rezultat. U gramatici G znak A je dohvatljiv samo uz pomoć mrvog znaka B .

Za bilo koji neprazni kontekstno neovisni jezik L moguće je izgraditi kontekstnu neovisnu gramatiku koja nema beskorisnih znakova.

Zadana gramatika G generira neprazni jezik $L=L(G)$. Neka je gramatika G_1 nastala primjenom alogitma odbacivanja mrtvih znakova na gramatiku G i neka je gramatika G_2 nastala primjenom alogitma odbacivanja nedohvatljivih znakova na gramatiku G_1 . Želi se dokazati da gramatika G_2 nema beskorisnih znakova. Gramatika G_2 nema nedohvatljivih znakova i za bilo koji znak X vrijedi $\underset{G_2}{X} \Rightarrow^*$. Budući da su svi znakovi gramatike G_2 ujedno i znakovi gramatike G_1 , i budući da gramatika G_1 nema mrvih znakova, svi

znakovi u nizu $\alpha X \beta$ su živi i za njih vrijedi $\alpha X \beta \xrightarrow[G_2]{*} w, w \in T^*$. Time je pokazano da je bilo koji znak X gramatike G_2 koristan, jer se pojavljuje u postupku generiranja niza $S \xrightarrow[G_2]{*} \alpha X \beta \xrightarrow[G_2]{*} w, w \in T^*$.

Primjer 2.7. Zadana je gramatika $G = (\{S, A, B, C\}, \{a, b, c, d\}, P, S)$ sa sljedećim produkcijama:

- | | |
|--------------------------|--------------------------|
| 1) $S \rightarrow a c$ | 4) $B \rightarrow a S A$ |
| 2) $S \rightarrow b A$ | 5) $C \rightarrow b C$ |
| 3) $A \rightarrow c B C$ | 6) $C \rightarrow d$ |

Algoritam odbacivanja mrtvih znakova odbacuje znakove A i B , te nastaje gramatika $G_1 = (\{S, C\}, \{a, b, c, d\}, P_1, S)$ sa produkcijama:

- | | |
|------------------------|------------------------|
| 1) $S \rightarrow a c$ | 5) $C \rightarrow b C$ |
| 6) $C \rightarrow d$ | |

Algoritam odbacivanja nedohvatljivih znakova odbacuje nezavršni znak C i završne znakove b i d , te nastaje pojednostavljena gramatika $G_2 = (\{S\}, \{a, c\}, P_2, S)$ bez beskorisnih znakova sa samo jednom produkcijom:

- 1) $S \rightarrow a c$
-

Odbacivanje ϵ -produkcija

Neka gramatika $G = (V, T, P, S)$ generira kontekstno neovisni jezik $L(G) - \{\epsilon\}$. Moguće je izgraditi istovjetnu gramatiku $G' = (V', T, P', S)$ koja nema ϵ -produkciju.

Primjer 2.8. Zadana je gramatika $G = (\{S, A\}, \{a, b, c\}, P, S)$ sa sljedećim produkcijama:

- | | | |
|----------------------------|----------------------|-----------------------------|
| 1) $S \rightarrow a A S A$ | 2) $S \rightarrow b$ | 3) $A \rightarrow c S$ |
| | | 4) $A \rightarrow \epsilon$ |

Iz gramatike G žele se odbaciti sve ϵ -produkcije. Producija (4) je ϵ -produkcija. Umjesto jednog nezavršnog znaka A , definiraju se dva nezavršna znaka. Neka se znakom A_{NE} označi nezavršni znak A u produkciji (3) koja nije ϵ -produkcija i neka se znakom A_{DA} označi nezavršni znak A u produkciji (4) koja jest ϵ -produkcija. Budući da umjesto jednog nezavršnog znaka A postoje dva znaka A_{NE} i A_{DA} , umjesto jedne produkcije (1) grade se četiri produkcije:

- | | | |
|---------------------------------------|----------------------|----------------------------------|
| 1a) $S \rightarrow a A_{NE} S A_{NE}$ | 2) $S \rightarrow b$ | 3) $A_{NE} \rightarrow c S$ |
| 1b) $S \rightarrow a A_{NE} S A_{DA}$ | | 4) $A_{DA} \rightarrow \epsilon$ |
| 1c) $S \rightarrow a A_{DA} S A_{NE}$ | | |
| 1d) $S \rightarrow a A_{DA} S A_{DA}$ | | |

Nezavršni znakovi A_{DA} na desnoj strani produkcije (1) zamijene se desnom stronom produkcije (4). Nakon što se svi znakovi A_{DA} zamijene praznim nizovima, produkcija (4) se odbacuje, jer nezavršni znak A_{DA} postaje nedohvatljiv:

- | | | |
|---------------------------------------|----------------------|-----------------------------|
| 1a) $S \rightarrow a A_{NE} S A_{NE}$ | 2) $S \rightarrow b$ | 3) $A_{NE} \rightarrow c S$ |
| 1b) $S \rightarrow a A_{NE} S$ | | |
| 1c) $S \rightarrow a S A_{NE}$ | | |
| 1d) $S \rightarrow a S$ | | |

Odbacivanjem produkcije (4) i nezavršnog znaka A_{DA} , u produkcijama ostaje samo znak A_{NE} . Nezavršni znak A_{NE} ponovno se označi znakom A . Nastala gramatika $G' = (\{S, A\}, \{a, b, c\}, P', S)$ ima sljedeće produkcije:

- | | | |
|-----------------------------|----------------------|------------------------|
| 1a) $S \rightarrow a A S A$ | 2) $S \rightarrow b$ | 3) $A \rightarrow c S$ |
| 1b) $S \rightarrow a A S$ | | |
| 1c) $S \rightarrow a S A$ | | |
| 1d) $S \rightarrow a S$ | | |

Konstruirana gramatika G' generira isti jezik kao i gramatika G , jedino što nezavršni znak A ne generira više prazni niz.

Zadana je gramatika G . Želi se izgraditi istovjetna gramatika G' koja nema ϵ -produkcija. Algoritam odbacivanja ϵ -produkcija izvodi se u dva osnovna koraka:

- 1) Pronađu se svi nezavršni znakovi koji generiraju prazni niz. *Prazni znakovi* su oni znakovi za koje vrijedi $A \xrightarrow{*} \epsilon$. Prazni znakovi pronalaze se sljedećim iterativnim postupkom:

U listu praznih znakova stave se lijeve strane svih ϵ -produkcija. Postoji li produkcija u kojoj su svi znakovi desne strane upisani u listu praznih znakova, lista praznih znakova nadopuni se lijevom stranom te produkcije. Algoritam se nastavlja sve dok je moguće proširiti listu praznih znakova novim nezavršnim znakom.

- 2) Skup produkcija gramatike G' gradi se na sljedeći način. Ako je:

$$A \rightarrow X_1 X_2 \dots X_n$$

produkcija gramatike G , onda se u skup produkcija gramatike G' dodaju produkcije oblika:

$$A \rightarrow \xi_1 \xi_2 \dots \xi_n.$$

Oznake ξ_i mogu poprimiti sljedeće vrijednosti:

- a) Ako znak X_i nije prazni znak, onda je oznaka ξ_i jednaka X_i ;
- b) Ako je znak X_i prazni znak, onda je oznaka ξ_i jednaka ϵ ili X_i .

Produkcije se grade na temelju svih mogućih kombinacija oznaka $\xi_1 \xi_2 \dots \xi_n$. Ako sve oznake ξ_i poprime vrijednost ϵ , onda nastaje ϵ -produkcija i ona se ne dodaje u skup produkcija gramatike G' . Time se odbace sve ϵ -produkcije iz zadane gramatike G .

U prethodnom primjeru produkcija (1) ima na desnoj strani prazni znak A . Svi ostali znakovi su neprazni i prema pravilu (a) ti znakovi poprimaju samo jednu vrijednost. Budući da se prazni znak A nalazi na dva mesta, moguće su četiri kombinacije u kojima znak A poprima vrijednosti A ili ϵ . Ima li produkcija na desnoj strani k praznih znakova, potrebno je izgraditi najviše 2^k novih produkcija. Ako je na desnoj strani produkcije barem jedan znak koji nije parazan, tada je potrebno dodati 2^k novih produkcija. Ako su na desnoj strani svi znakovi prazni, potrebno je dodati $2^k - 1$ novih produkcija (produkcija $A \rightarrow \epsilon \epsilon \dots \epsilon$ se ne dodaje u novi skup produkcija).

Odbacivanje jediničnih produkacija

Neka gramatika $G=(V, T, P, S)$ generira kontekstno neovisni jezik $L(G)-\{\varepsilon\}$. Moguće je izgraditi istovjetnu gramatiku $G'=(V', T, P', S)$ koja nema jediničnih produkacija oblika $A \rightarrow B$.

Istovjetna gramatika G' koja nema jediničnih produkacija gradi se na sljedeći način:

- 1) U skup produkacija gramatike G' stave se sve produkije gramatike G koje nisu jedinične.
- 2) Neka se postupkom generiranja iz nezavršnog znaka A dobije nezavršni znak B , tj.

$$A \xrightarrow{*} B$$
, gdje su A i B nezavršni znakovi gramatike G . Za sve produkije $B \rightarrow \alpha$ koje nisu jedinične grade se nove produkije $A \rightarrow \alpha$.

Chomskyjev normalni oblik produkacija

Neka gramatika $G=(V, T, P, S)$ generira kontekstno neovisni jezik $L(G)-\{\varepsilon\}$. Moguće je izgraditi istovjetnu gramatiku $G'=(V', T, P', S)$ koja ima sve produkije oblika $A \rightarrow BC$ ili $A \rightarrow a$. Znakovi A , B i C su nezavršni znakovi gramatike, $A, B, C \in V$. Znak a jest završni znak gramatike, $a \in T$.

Prepostavi se da gramatika G nema beskorisnih znakova, ε -produkacija i jediničnih produkacija. Algoritam pretvorbe produkacija u Chomskyjev normalni oblik izvodi se u tri koraka:

- 1) U skup produkacija P' stave se sve produkije koje su u Chomskyjevom normalnom obliku, odnosno produkije oblika $A \rightarrow BC$ ili $A \rightarrow a$. U skup nezavršnih znakova V' upišu se svi nezavršni znakovi.
- 2) Neka je produkacija gramatike G oblika:

$$A \rightarrow X_1 X_2 \dots X_m, \quad A \in V, X_i \in T \text{ ili } X_i \in V, m \geq 2, 1 \leq i \leq m,$$

Ako je X_i završni znak a , $a \in T$, onda se skup nezavršnih znakova proširi novim nezavršnim znakom C_a , $C_a \in V'$. Skup produkacija P' proširuje se produkijom:

$$C_a \rightarrow a,$$

koja je u Chomskyjevom normalnom obliku. Svi završni znakovi a u produkiji $A \rightarrow X_1 X_2 \dots X_m$ zamijene se nezavršnim znakom C_a . Proces definiranja novih nezavršnih znakova i novih produkacija nastavlja se za sve završne znakove na desnoj strani produkije $A \rightarrow X_1 X_2 \dots X_m$. Zamjenom svih završnih znakova nezavršnim znakovima, produkacija $A \rightarrow X_1 X_2 \dots X_m$ ima na desnoj strani isključivo nezavršne znakove gramatike G' .

Proces pretvorbe nastavlja se za sve produkije oblika $A \rightarrow X_1 X_2 \dots X_m$.

- 3) Nakon što završi korak (2), sve produkije su oblika $A \rightarrow a$ ili $A \rightarrow B_1 B_2 \dots B_m$, gdje je $m \geq 2$. Znakovi A i B_i su nezavršni znakovi skupa V' , a znak a je završni znak. Produkije oblika $A \rightarrow a$ i $A \rightarrow B_1 B_2 \dots B_m$ su u Chomskyjevom normalnom obliku. Produkije koje imaju tri i više znakova na desnoj strani:

$$A \rightarrow B_1 B_2 \dots B_m, \text{ gdje je } m \geq 3,$$

zamijene se novim produkcjama. Definiraju se novi nazavršni znakovi D_1, D_2, \dots, D_{m-2} , a zatim se produkacija $A \rightarrow B_1 B_2 \dots B_m$ zamijeni skupom produkacija:

$$\{ A \rightarrow B_1 D_1, D_1 \rightarrow B_2 D_2, D_2 \rightarrow B_3 D_3, \dots, D_{m-3} \rightarrow B_{m-2} D_{m-2}, D_{m-2} \rightarrow B_{m-1} B_m \}$$

Primjer 2.9. Zadana je gramatika $G=(\{S, A, B\}, \{a, b\}, P, S)$ sa sljedećim produkcijama:

- | | | |
|------------------------|--------------------------|--------------------------|
| 1) $S \rightarrow b A$ | 3) $A \rightarrow b A A$ | 6) $B \rightarrow a B B$ |
| 2) $S \rightarrow a B$ | 4) $A \rightarrow a S$ | 7) $B \rightarrow b S$ |
| | 5) $A \rightarrow a$ | 8) $B \rightarrow b$ |

Producije (5) i (8) su u Chomskyjevom normalnom obliku i one se izravno stave u skup novih produkcija bez preuređivanja. Producija $S \rightarrow b A$ zamjeni se produkcijom $S \rightarrow C_b A$ i doda se producija $C_b \rightarrow b$. Na sličan način preurede se ostale produkcije. Nakon koraka (2) algoritam daje sljedeće produkcije:

- | | | | |
|--------------------------|----------------------------|----------------------------|-------------------------|
| 1) $S \rightarrow C_b A$ | 3) $A \rightarrow C_b A A$ | 6) $B \rightarrow C_a B B$ | 9) $C_a \rightarrow a$ |
| 2) $S \rightarrow C_a B$ | 4) $A \rightarrow C_a S$ | 7) $B \rightarrow C_b S$ | 10) $C_b \rightarrow b$ |
| | 5) $A \rightarrow a$ | 8) $B \rightarrow b$ | |

Sve produkcije su u Chomskyjevom normalnom obliku osim produkcija (3) i (6) koje imaju na desnoj strani tri nezavršna znaka. U koraku (3) algoritam preuređuje produkcije (3) i (6) na sljedeći način. Producija $A \rightarrow C_b A A$ zamjeni se produkcijama $A \rightarrow C_b D_1$ i $D_1 \rightarrow A A$. Producija $B \rightarrow C_a B B$ zamjeni se produkcijama $B \rightarrow C_a E_1$ i $E_1 \rightarrow B B$. Nakon koraka (3) sve produkcije gramatika G' su u Chomskyjevom normalnom obliku:

- | | | | |
|--------------------------|------------------------------|------------------------------|-------------------------|
| 1) $S \rightarrow C_b A$ | 3.a) $A \rightarrow C_b D_1$ | 6.a) $B \rightarrow C_a E_1$ | 9) $C_a \rightarrow a$ |
| 2) $S \rightarrow C_a B$ | 3.b) $D_1 \rightarrow A A$ | 6.b) $E_1 \rightarrow B B$ | 10) $C_b \rightarrow b$ |
| | 4) $A \rightarrow C_a S$ | 7) $B \rightarrow C_b S$ | |
| | 5) $A \rightarrow a$ | 8) $B \rightarrow b$ | |
-

Greibachov normalni oblik produkcija

Neka gramatika G generira kontekstno neovisni jezik $L(G)-\{\epsilon\}$. Moguće je izgraditi istovjetnu gramatiku G' koja ima sve produkcije oblika $A \rightarrow a\alpha$, gdje je a završni znak gramatike, $a \in T$, a α je niz nezavršnih znakova gramatike koji može biti i prazan, $\alpha \in V^*$.

Tijekom pretvorbe produkcija u Greibachov normalni oblik koriste se tri algoritma: algoritam pretvorbe produkcija gramatike u Chomskyjev normalni oblik, algoritam zamjene krajnje lijevog nezavršnog znaka i algoritam razrješavanja lijeve rekurzije. Prije opisa samog postupka pretvorbe produkcija u Greibachov normalni oblik, opisani su algoritam zamjene krajnje lijevog nezavršnog znaka i algoritam razrješavanja lijeve rekurzije, dok je algoritam pretvorbe produkcija gramatike u Chomskyjev normalni oblik prethodno opisan.

Algoritam zamjene krajnje lijevog nezavršnog znaka:

Neka je u gramatici r produkcija koja imaju nezavršni znak D_j na lijevoj strani:

$$\begin{array}{ll} D_j & \rightarrow \quad \alpha_1 \\ D_j & \rightarrow \quad \alpha_2 \\ & \cdots \\ D_j & \rightarrow \quad \alpha_r, \end{array}$$

i neka je u gramatici produkcija koja ima nezavršni znak D_j na krajnje lijevom mjestu desne strane:

$$D_i \rightarrow D_j \gamma,$$

gdje su α i γ nizovi završnih i nezavršnih znakova gramatike.

Prethodno zadanih $r+1$ produkcija zamijeni se sa sljedećih r produkcija:

$$\begin{array}{ll} D_i & \rightarrow \alpha_1 \gamma \\ D_i & \rightarrow \alpha_2 \gamma \\ \cdots & \cdots \\ D_i & \rightarrow \alpha_r \gamma \end{array}$$

gdje se u produkciji $D_i \rightarrow D_j \gamma$ nezavršni znak D_j zamijeni desnim stranama svih r produkcija koje su oblika $D_j \rightarrow \alpha_i$. Lako se provjeri da preuređene produkcije gramatike generiraju isti jezik kao i izvorne produkcije gramatike.

Algoritam razrješavanja lijeve rekurzije:

Ako se nezavršni znak istodobno nalazi na lijevoj strani produkcije i na krajnjem lijevom mjestu desne strane produkcije, onda se kaže da je produkcija *lijevo rekurzivna*. Gramatika koja ima lijevo rekurzivne produkcije preuređuje se na sljedeći način:

Neka za nezavršni znak D_i postoji r lijevo rekurzivnih produkcija:

$$D_i \rightarrow D_i \alpha_k, \quad 1 \leq k \leq r,$$

i s produkcija koje nisu lijevo rekurzivne:

$$D_i \rightarrow \beta_l, \quad 1 \leq l \leq s,$$

gdje su α_k i β_l nizovi završnih i nezavršnih znakova. Dane produkcije zamijene se sljedećim produkcijama:

$$\begin{array}{ll} D_i \rightarrow \beta_l, & 1 \leq l \leq s, \\ D_i \rightarrow \beta_l C_i, & 1 \leq l \leq s, \\ C_i \rightarrow \alpha_k, & 1 \leq k \leq r, \\ C_i \rightarrow \alpha_k C_i, & 1 \leq k \leq r, \end{array}$$

gdje je C_i novi nezavršni znak. Lako se provjeri da preuređena gramatika generira isti jezik i da nema lijevo rekurzivnih produkcija.

Algoritam pretvorbe produkcija u Greibachov normalni oblik:

Algoritam pretvorbe produkcija u Greibachov normalni oblik izvodi se u četiri koraka:

- 1) Producije gramatike G preurede se u Chomskyjev normalni oblik, odnosno u produkcije oblika $A \rightarrow BC$ ili $A \rightarrow a$. Neka je m kardinalni broj skupa nezavršnih znakova. Skup nezavršnih znakova zamijeni se skupom nezavršnih znakova $\{D_1, D_2, \dots, D_m\}$, te se na odgovarajući način preurede produkcije gramatike. Nakon pretvorbe, sve produkcije gramatike su oblika $D_i \rightarrow D_j D_k$ ili $D_i \rightarrow a$, gdje indeksi i uz znakove D_i označavaju različite nezavršne znakove. Producije oblika $D_i \rightarrow a$ su u Greibachovom normalnom obliku i one se u dalnjem postupku ne preuređuju.
- 2) U drugom koraku produkcije oblika $D_i \rightarrow D_j D_k$ preurede se na oblik $D_i \rightarrow D_j \beta$, gdje je indeks j nezavršnog znaka D_j na krajnjem lijevom mjestu desne strane produkcije veći od indeksa i nezavršnog znaka D_i na lijevoj strani produkcije, odnosno $j > i$. Niz β je niz nezavršnih znakova. Postupak pretvorbe započinje nezavršnim znakom D_1 , te se nastavlja redom za sve ostale nezavršne znakove D_2, D_3, \dots, D_m .

Za nezavršni znak D_1 i produkcije oblika $D_1 \rightarrow D_j D_k$ vrijedi $j=1$ ili $j>1$. Ako je $j>1$, onda je produkcija u zahtijevanom obliku. Potrebno je preuređiti samo produkcije oblika $D_1 \rightarrow D_1 D_k$, gdje se nezavršni znak D_1 istodobno nalazi na lijevoj strani produkcije i na krajnjem lijevom mjestu desne strane produkcije. Za preuređivanje produkcija oblika $D_1 \rightarrow D_1 D_k$ koristi se prethodno opisan algoritam

razrješavanja lijeve rekurzije. Algoritam razrješavanja lijeve rekurzije proširi skup nezavršnih znakova novim nezavršnim znakom C_i .

Ostale produkcije preuređuju se sljedećim redoslijedom: najprije produkcije koje imaju nezavršni znak D_2 na lijevoj strani, zatim produkcije koje imaju nezavršni znak D_3 na lijevoj strani, te redom sve do produkcija koje imaju nezavršni znak D_m na lijevoj strani.

Tijekom postupka preuređivanja produkcija, za nezavršni znak D_i i produkciju oblika $D_i \rightarrow D_j \beta$ vrijedi $j=i, j < i$ ili $j > i$, gdje je β niz nezavršnih znakova. Ako je $j < i$, onda se na krajnje lijevi nezavršni znak D_j na desnoj strani produkcije $D_i \rightarrow D_j \beta$ primjeni prethodno opisani algoritam zamjene krajnje lijevog nezavršnog znaka. Algoritam zamjene krajnje lijevog nezavršnog znaka uzastopno se primjenjuje sve dok za nezavršni znak D_i ima produkciju oblika $D_i \rightarrow D_j \beta$ za koji vrijedi da $j < i$.

Neka se tijekom primjene algoritma zamjene krajnje lijevog nezavršnog znaka produkcija $D_i \rightarrow D_j \beta$ zamijeni produkcijom $D_i \rightarrow D_k \alpha \beta$ na temelju produkcije $D_j \rightarrow D_k \alpha$. Budući da je postupak pretvorbe završen za sve nezavršne znakove D_j za koje vrijedi $j < i$ i budući da za sve produkcije $D_j \rightarrow D_k \alpha$ koje imaju nezavršni znak D_j na lijevoj strani vrijedi da je $j > i$, onda se indeks nezavršnog znaka D_k koji se nalazi na krajnje lijevom mjestu desne strane produkcija tijekom uzastopne primjene algoritma zamjene krajnje lijevog nezavršnog znaka stalno povećava. Nakon najviše $i-1$ uzastopnih primjena algoritma, desne strane produkcija ili započinju završnim znakom, ili su produkcije oblika $D_i \rightarrow D_j \beta$ i za njih vrijedi da je $j=i$ ili $j > i$. Producije za koje vrijedi $j=i$ su lijevo rekursivne i za njihovo preuređivanje koristi se algoritam za razrješavanje lijeve rekurzije, koji proširi skup nezavršnih znakova novim nezavršnim znakom C_i .

Nakon što završi drugi korak procesa pretvorbe za sve nezavršne znakove D_1, D_2, \dots, D_m , bilo koja produkcija ima jedan od tri moguća oblika:

- $D_i \rightarrow D_j \beta$, gdje je $j > i$, β je niz nezavršnih znakova,
- $D_i \rightarrow a\beta$, gdje je $a \in T$, β je niz nezavršnih znakova,
- $C_i \rightarrow D_j \xi$, gdje je ξ niz nezavršnih znakova iz skupa $\{D_1, D_2, \dots, D_m\} \cup \{C_1, C_2, \dots, C_{i-1}\}$.

Algoritam razrješavanja lijeve rekurzije osigurava da desna strana produkcije koja ima na lijevoj strani nezavršni znak C_i obavezno započinje isključivo jednim od znakova iz skupa $\{D_1, D_2, \dots, D_m\}$. Nezavršni znak D_m ima najveći indeks i desne strane svih produkcija koje imaju na lijevoj strani nezavršni znak D_m započinju isključivo završnim znakovima, odnosno te produkcije su u Greibachovom normalnom obliku.

- 3) U trećem koraku produkcije oblika $D_i \rightarrow D_j \beta$ preurede se na oblik $D_i \rightarrow a\alpha\beta$, gdje je a završni znak gramatike, odnosno $a \in T$, dok su α i β nizovi nezavršnih znakova. Postupak pretvorbe započinje nezavršnim znakom D_{m-1} , te se nastavlja redom za sve ostale nezavršne znakove $D_{m-2}, D_{m-3}, \dots, D_1$.

U trećem koraku pretvorba produkcija kreće od nezavršnog znaka D_{m-1} . Producije koje imaju na lijevoj strani nezavršni znak D_{m-1} su oblika: $D_{m-1} \rightarrow D_m \alpha$ ili $D_{m-1} \rightarrow a\alpha$, gdje je $a \in T$. Budući da su sve produkcije koje imaju na lijevoj strani nezavršni znak D_m oblika $D_m \rightarrow a\beta$, gdje je $a \in T$, produkcija $D_{m-1} \rightarrow D_m \alpha$ zamijeni se skupom produkcija $D_{m-1} \rightarrow a\beta\alpha$ za sve produkcije $D_m \rightarrow a\beta$. Nakon zamjene, desne strane svih produkcija koje imaju na lijevoj strani nezavršni znak D_{m-1} započinju isključivo završnim znakovima, odnosno te produkcije su u Greibachovom normalnom obliku.

Ostale produkcije preuređuju se sljedećim redoslijedom: najprije produkcije koje imaju nezavršni znak D_{m-2} na lijevoj strani, zatim produkcije koje imaju nezavršni znak D_{m-3} na lijevoj strani, te redom sve do produkcija koje imaju nezavršni znak D_1 na lijevoj strani.

Tijekom primjene algoritma zamjene krajnje lijevog nezavršnog znaka produkcija $D_i \rightarrow D_j \beta$ zamijeni se produkcijom $D_i \rightarrow a\alpha\beta$ na temelju produkcije $D_j \rightarrow a\alpha$. Budući da je postupak pretvorbe završen za sve nezavršne znakove D_j za koje vrijedi $j > i$ i budući da desne strane svih produkcija koje imaju nezavršni znak D_j na lijevoj strani započinju završnim znakovima, onda nakon primjene algoritma zamjene krajnje lijevog nezavršnog znaka desne strane svih produkcija koje imaju nezavršni znak D_i na lijevoj strani započinju završnim znakom.

Nakon što završi treći korak pretvorbe sa svim nezavršnim znakovima $D_{m-1}, D_{m-2}, \dots, D_1$, sve produkcije su oblika:

$$\begin{aligned} D_i &\rightarrow a\beta, \text{ gdje je } a \in T, \beta \text{ je niz nezavršnih znakova,} \\ C_i &\rightarrow D_j\xi, \text{ gdje je } \xi \text{ niz nezavršnih znakova iz skupa } \{D_1, D_2, \dots, D_m\} \cup \{C_1, C_2, \dots, C_{i-1}\}. \end{aligned}$$

- 4) U posljednjem četvrtom koraku preurede se produkcije koje imaju nezavršne znakove C_i na lijevoj strani. Te produkcije nastaju tijekom razrješavanja lijeve rekurzije i njihove desne strane započinju isključivo jednim od nezavršnih znakova iz skupa $\{D_1, D_2, \dots, D_m\}$. Budući da su prethodno sve produkcije za nezavršne znakove D_1, D_2, \dots, D_m preuređene tako da njihove desne strane započinju isključivo završnim znakovima, produkcije koje imaju na lijevim stranama nezavršne znakove C_i primjenom algoritma zamjene krajnje lijevog nezavršnog znaka preurede se u Greibachov normalni oblik.

Nakon što završi četvrti korak pretvorbe, sve produkcije su oblika:

$$\begin{aligned} D_i &\rightarrow a\beta, \\ C_i &\rightarrow a\beta, \text{ gdje je } a \in T, \beta \text{ je niz nezavršnih znakova.} \end{aligned}$$

Primjer 2.10. Neka je zadana gramatika $G=(\{S, A, B\}, \{a, b\}, P, S)$ i neka su produkcije gramatike preuređene u Chomskyjev normalni oblik:

$$\begin{array}{lll} 1) S \rightarrow A B & 2) A \rightarrow B S & 4) B \rightarrow S A \\ & 3) A \rightarrow b & 5) B \rightarrow a \end{array}$$

Korak (1):

U prvom koraku odredi se novi skup nezavršnih znakova $\{D_1, D_2, D_3\}$. Nezavršni znak S zamijeni se nezavršnim znakom D_1 , nezavršni znak A zamijeni se nezavršnim znakom D_2 , te se nezavršni znak B zamijeni nezavršnim znakom D_3 . Na odgovarajući način promijene se produkcije gramatike:

$$\begin{array}{lll} 1) D_1 \rightarrow D_2 D_3 & 2) D_2 \rightarrow D_3 D_1 & 4) D_3 \rightarrow D_1 D_2 \\ & 3) D_2 \rightarrow b & 5) D_3 \rightarrow a \end{array}$$

Korak (2):

Producije (3) i (5) su u Greibachovom normalnom obliku i one se dalje ne preuređuju. U drugom koraku sve ostale produkcije preurede se na oblik $D_i \rightarrow D_j\beta$, gdje je indeks j veći od indeksa i , odnosno $j > i$, a β je niz nezavršnih znakova. Postupak započinje nezavršnim znakom D_1 . Producija (1) se ne preuređuje, jer je u zahtijevanom obliku (indeks krajnje lijevog nezavršnog znaka D_2 na desnoj strani produkcije jednak je 2 i veći je od indeksa nezavršnog znaka D_1 lijeve strane produkcije koji je jednak 1). Nakon nezavršnog znaka D_1 , obrađuje se nezavršni znak D_2 . Producija (2) je također u zahtijevanom obliku i postupak se nastavlja obradom znaka D_3 . Budući da je indeks krajnje lijevog nezavršnog znaka D_1 desne strane produkcije (4) jednak 1 i manji je od indeksa nezavršnog znaka D_3 na lijevoj strani produkcije koji je jednak 3, krajnje lijevi nezavršni znak D_1 desne strane produkcije (4) zamijeni se desnom stranom produkcije (1). Producija $D_3 \rightarrow D_1 D_2$ zamijeni se produkcijom $D_3 \rightarrow D_2 D_3 D_2$:

$$\begin{array}{lll} 1) D_1 \rightarrow D_2 D_3 & 2) D_2 \rightarrow D_3 D_1 & 4a) D_3 \rightarrow D_2 D_3 D_2 \\ & 3) D_2 \rightarrow b & 5) D_3 \rightarrow a \end{array}$$

Izgrađena producija (4a) ima na krajnje lijevom mjestu nezavršni znak D_2 koji još uvijek ima manji indeks od nezavršnog znaka D_3 na lijevoj strani produkcije. Krajnje lijevi nezavršni znak D_2 na desnoj strani produkcije (4a) zamijeni se desnim stranama produkcija (2) i (3). Producija $D_3 \rightarrow D_2 D_3 D_2$ zamijeni se skupom produkcija $D_3 \rightarrow D_3 D_1 D_3 D_2$ i $D_3 \rightarrow b D_3 D_2$:

$$\begin{array}{lll} 1) D_1 \rightarrow D_2 D_3 & 2) D_2 \rightarrow D_3 D_1 & 4aa) D_3 \rightarrow D_3 D_1 D_3 D_2 \\ & 3) D_2 \rightarrow b & 4ab) D_3 \rightarrow b D_3 D_2 \\ & & 5) D_3 \rightarrow a \end{array}$$

Producija (4aa) je lijevo rekurzivna jer ima isti nezavršni znak D_3 na lijevoj strani i na krajnje lijevom mjestu desne strane. Tijekom razrješavanja lijeve rekurzije definira se novi nezavršni znak C_3 i produkcija $D_3 \rightarrow D_3 D_1 D_3 D_2$ zamijeni se skupom produkcija $D_3 \rightarrow b D_3 D_2 C_3$, $D_3 \rightarrow b D_3 D_2 C_3$, $C_3 \rightarrow D_1 D_3 D_2$ i $C_3 \rightarrow D_1 D_3 D_2 C_3$:

- | | | | |
|------------------------------|------------------------------|--|--|
| 1) $D_1 \rightarrow D_2 D_3$ | 2) $D_2 \rightarrow D_3 D_1$ | 4aaa) $D_3 \rightarrow b D_3 D_2 C_3$ | 4aac) $C_3 \rightarrow D_1 D_3 D_2$ |
| | 3) $D_2 \rightarrow b$ | 4aab) $D_3 \rightarrow a C_3$ | 4aad) $C_3 \rightarrow D_1 D_3 D_2 C_3$ |
| | | 4ab) $D_3 \rightarrow b D_3 D_2$ | |
| | | 5) $D_3 \rightarrow a$ | |

Sve produkcije koje imaju nezavršni znak na krajnje lijevom mjestu desne strane produkcije su oblika $D_i \rightarrow D_j \beta$, gdje je $j > i$. Nadalje, desne strane svih produkcija koje imaju nezavršni znak D_3 na lijevoj strani započinju završnim znakom.

Korak (3):

Treći korak započinje nezavršnim znakom D_2 . U produkciji (2) nezavršni znak D_3 zamijeni se desnim stranama produkcija (4aaa), (4aab), (4ab) i (5):

- | | | | |
|------------------------------|--|--|--|
| 1) $D_1 \rightarrow D_2 D_3$ | 2a) $D_2 \rightarrow b D_3 D_2 C_3 D_1$ | 4aaa) $D_3 \rightarrow b D_3 D_2 C_3$ | 4aac) $C_3 \rightarrow D_1 D_3 D_2$ |
| | 2b) $D_2 \rightarrow a C_3 D_1$ | 4aab) $D_3 \rightarrow a C_3$ | 4aad) $C_3 \rightarrow D_1 D_3 D_2 C_3$ |
| | 2c) $D_2 \rightarrow b D_3 D_2 D_1$ | 4ab) $D_3 \rightarrow b D_3 D_2$ | |
| | 2d) $D_2 \rightarrow a D_1$ | 5) $D_3 \rightarrow a$ | |
| | 3) $D_2 \rightarrow b$ | | |

Postupak se nastavlja obradom znaka D_1 . U produkciji (1) nezavršni znak D_2 zamijeni se desnim stranama produkcija (2a), (2b), (2c), (2d) i (3):

- | | | | |
|--|--|--|--|
| 1a) $D_1 \rightarrow b D_3 D_2 C_3 D_1 D_3$ | 2a) $D_2 \rightarrow b D_3 D_2 C_3 D_1$ | 4aaa) $D_3 \rightarrow b D_3 D_2 C_3$ | 4aac) $C_3 \rightarrow D_1 D_3 D_2$ |
| 1b) $D_1 \rightarrow a C_3 D_1 D_3$ | 2b) $D_2 \rightarrow a C_3 D_1$ | 4aab) $D_3 \rightarrow a C_3$ | 4aad) $C_3 \rightarrow D_1 D_3 D_2 C_3$ |
| 1c) $D_1 \rightarrow b D_3 D_2 D_1 D_3$ | 2c) $D_2 \rightarrow b D_3 D_2 D_1$ | 4ab) $D_3 \rightarrow b D_3 D_2$ | |
| 1d) $D_1 \rightarrow a D_1 D_3$ | 2d) $D_2 \rightarrow a D_1$ | 5) $D_3 \rightarrow a$ | |
| 1e) $D_1 \rightarrow b D_3$ | 3) $D_2 \rightarrow b$ | | |

Korak (4):

Producije koje na lijevim stranama imaju nezavršne znakove D_1 , D_2 i D_3 su u Greibachovom normalnom obliku. U posljednjem četvrtom koraku pretvorbe potrebno je u produkcijama (4aac) i (4aad), koje imaju na lijevoj strani nezavršni znak C_3 , zamijeniti nezavršni znak D_1 desnim stranama produkcija (1a), (1b), (1c), (1d) i (1e). Sve izgrađene produkcije su u Greibachovom normalnom obliku:

- | | | | |
|--|--|--|---|
| 1a) $D_1 \rightarrow b D_3 D_2 C_3 D_1 D_3$ | 2a) $D_2 \rightarrow b D_3 D_2 C_3 D_1$ | 4aaa) $D_3 \rightarrow b D_3 D_2$ | 4aac) $C_3 \rightarrow b D_3 D_2 C_3 D_1 D_3 D_2 D_2$ |
| 1b) $D_1 \rightarrow a C_3 D_1 D_3$ | 2b) $D_2 \rightarrow a C_3 D_1$ | 4ab) $D_3 \rightarrow a C_3$ | 4aacb) $C_3 \rightarrow a C_3 D_1 D_3 D_2 D_2$ |
| 1c) $D_1 \rightarrow b D_3 D_2 D_1 D_3$ | 2c) $D_2 \rightarrow b D_3 D_2 D_1$ | 5) $D_3 \rightarrow a$ | 4aacc) $C_3 \rightarrow b D_3 D_2 D_1 D_3 D_2 D_2$ |
| 1d) $D_1 \rightarrow a D_1 D_3$ | 2d) $D_2 \rightarrow a D_1$ | | 4aacd) $C_3 \rightarrow a D_1 D_3 D_2 D_2$ |
| 1e) $D_1 \rightarrow b D_3$ | 3) $D_2 \rightarrow b$ | | 4aace) $C_3 \rightarrow b D_3 D_2 D_2$ |
| | | 4ab) $D_3 \rightarrow b D_3 D_2$ | 4aada) $C_3 \rightarrow b D_3 D_2 C_3 D_1 D_3 D_2 D_2 C_3$ |
| | | 5) $D_3 \rightarrow a$ | 4aadb) $C_3 \rightarrow a C_3 D_1 D_3 D_2 C_3$ |
| | | | 4aadc) $C_3 \rightarrow b D_3 D_2 D_1 D_3 D_2 D_2 C_3$ |
| | | | 4aadd) $C_3 \rightarrow a D_1 D_3 D_2 C_3$ |
| | | | 4aaade) $C_3 \rightarrow b D_3 D_2 C_3$ |

3.1.3 Parsiranje niza

U odjeljku 3.1.1 pokazana je primjena gramatike za generiranje jezika $L(G)$. Druga važna primjena gramatike je određivanje pripadnosti proizvoljnog niza završnih znakova w jeziku $L(G)$, odnosno određivanje da li gramatika G generira niz w . Proces određivanja pripadnosti niza w jeziku $L(G)$ naziva se *prepoznavanje niza*. Nakon što se ustanovi da niz w pripada jeziku $L(G)$, potrebno je izgraditi generativno stablo. Generativno stablo koristi se za interpretaciju niza. Sveukupni proces prepoznavanja niza i gradnje generativnog stabla naziva se *parsiranje niza*.

U procesu parsiranja nastoji se izgraditi generativno stablo za zadani niz završnih znakova w i za zadanu gramatiku G . Uspije li se izgraditi generativno stablo kojem su listovi označeni isključivo završnim znakovima niza w , niz w pripada jeziku $L(G)$. Zbog toga što se primjenjuje u procesu parsiranja, pored naziva generativno stablo, u literaturi se često koristi naziv *stablo parsiranja*.

Metode parsiranja razlikuju se po načinu gradnje generativnog stabla: *metoda parsiranja od vrha prema dnu* i *metoda parsiranja od dna prema vrhu*. Metoda parsiranja od vrha prema dnu gradi generativno stablo od korijena stabla, odnosno od početnog nezavršnog znaka gramatike, prema listovima stabla, odnosno prema završnim znakovima gramatike. Uobičajeno je da se stablo crta tako da je korijen stabla na vrhu stranice i zato se ovaj postupak naziva parsiranje od vrha prema dnu. Parsiranje od dna prema vrhu gradi stablo od listova stabla prema korijenu stabla.

Parsiranje od vrha prema dnu

Zadana je gramatika $G=(V, T, P, S)$. Parsiranje od vrha prema dnu započinje gradnju stabla od početnog nezavršnog znaka S . Ostali čvorovi grade se primjenom produkcija iz skupa P gramatike G . Producije gramatike primjenjuju se sve dok se listovi stabla ne označe isključivo završnim znakovima zadanog niza w ($w \in T^*$). Tijekom gradnje stabla završni znakovi niza w određuju koja se produkcija gramatike primjenjuje.

Parsiranje od vrha prema dnu našlo je široku primjenu zbog svoje učinkovitosti i jednostavne programske ostvarivosti. Sljedeći primjer pokazuje način parsiranja od vrha prema dnu.

Primjer 3.11. Gramatika G generira **begin-end** blokove programskog jezika. Postoje dvije vrste naredbi: naredba pridruživanja i naredba **while**-petlje. Skup nezavršnih znakova V jednak je:

$$V=\{C, S, S_1, S_2\},$$

gdje je C početni nezavršni znak. Skup završnih znakova T sadrži sljedeće znakove:

$$T=\{\text{begin, end, while, do, ;, :=, ≠, var}\}$$

gdje znakovi **begin**, **end**, **while** i **do** označavaju ključne riječi programskog jezika, a znak **var** varijablu programskog jezika (napomena: jedan završni znak označen je kao niz slova). Operator pridruživanja označen je znakom **:=** (operator **:=** je jedan završni znak koji je također označen kao niz znakova), operator nejednakosti označen je znakom **≠**, a oznaka ulančavanja naredbi programskog jezika jest znak **;**.

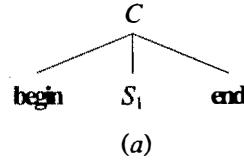
Skup produkcija P jednak je:

$$\begin{aligned} P = & \{ C \rightarrow \text{begin } S_1 \text{ end} \\ & S_1 \rightarrow S \ S_2 \\ & S_2 \rightarrow ; \ | \ \epsilon \\ & S \rightarrow \text{var} \ := \ \text{var} \ | \ \text{while } \text{var} \neq \text{var } \text{do } S \ | \ \text{begin } S_1 \text{ end } \} \end{aligned}$$

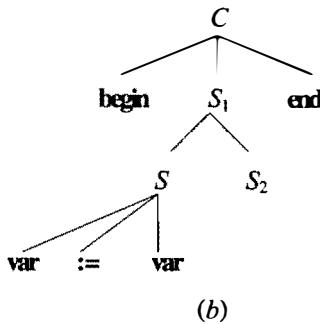
Parsiranjem od vrha prema dnu potrebno je odrediti da li sljedeći niz pripada jeziku $L(G)$:

```
begin var := var ;
    while var ≠ var do
        var := var
end
```

Parsiranje od vrha prema dnu započinje od korijena stabla, odnosno od početnog nezavršnog znaka C . Na početku moguće je primijeniti samo produkciju $C \rightarrow \text{begin } S_1 \text{ end}$. To je jedina produkcija koja ima početni nezavršni znak C na lijevoj strani. Početak gradnje stabla prikazan je na slici 3.13a. Budući da je $C \rightarrow \text{begin } S_1 \text{ end}$ jedina produkcija koja na lijevoj strani ima početni nezavršni znak C , bilo koji niz jezika $L(G)$ mora započeti završnim znakom **begin**. Zadani niz započinje završnim znakom **begin**, te se parsiranje niza nastavlja.



Slika 3.13: Početak parsiranja od vrha prema dnu



Slika 3.13: Nastavak parsiranja od vrha prema dnu

Slijedi primjena produkcije $S_1 \rightarrow SS_2$:

```
begin S  $S_2$  end
```

Parsiranje se nastavlja zamjenom lijevog nezavršnog znaka, a to je nezavršni znak S . Za znak S postoje tri produkcije. Budući da se iza znaka **begin** u zadanim nizu nalazi znak **var**, primjeni se produkcija $S \rightarrow \text{var} := \text{var}$:

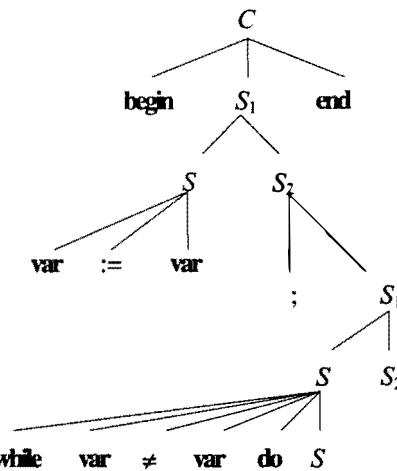
```
begin var := var  $S_2$ 
end
```

Ostale dvije produkcije generiraju na drugom mjestu međuniza završni znak **while** ili završni znak **begin**. Niti jedan od tih znakova nije jednak drugom znaku u zadanim nizu i zato se te produkcije ne primjenjuju. Generativno stablo nastalo nakon primjene produkcija $S_1 \rightarrow SS_2$ i $S \rightarrow \text{var} := \text{var}$ prikazano je na slici 3.13b. Prefiks generiranog međuniza sve do nezavršnog znaka S_2 jednak je prefiksu zadanih niza i gradnja stabla se nastavlja.

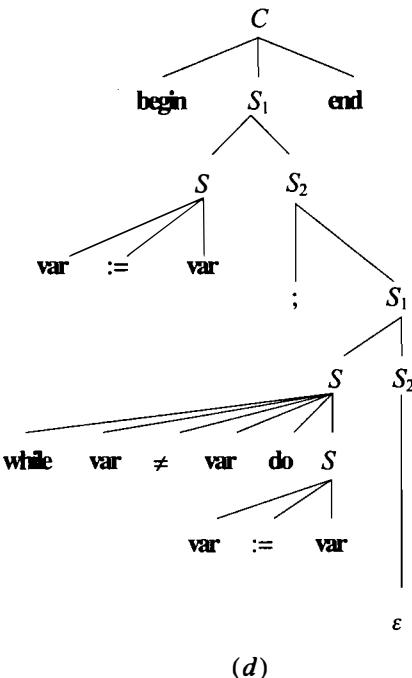
Budući da je sljedeći znak u nizu **;**, a mijenja se nezavršni znak S_2 , primjeni se produkcija $S_2 \rightarrow ;S_1$. Jedino moguća produkcija za znak S_1 jest produkcija $S_1 \rightarrow S S_2$. Na temelju znaka **while**, parser zamjeni nezavršni znak S nizom **while** var ≠ var **do** S :

```
begin var := var ;  $S_1$  end ⇒
begin var := var ;  $S S_2$  end ⇒
begin var := var ; while var ≠ var do  $S S_2$ 
end
```

Gradnja stabla prikazana je na slici 3.13c. Budući da je prefiks generiranog međuniza sve do nezavršnog znaka S jednak prefiksu zadanih niza, proces parsiranja se nastavlja.



Slika 3.13: Nastavak parsiranja od vrha prema dnu



Slika 3.13: Nastavak parsiranja od vrha prema dnu

Sljedeći znak u nizu jest znak **var**. Znak **S** zamijeni se nizom **var := var**:

begin var := var ;
while var ≠ var do var := var S₂ end

Budući da je sljedeći znak u zadanom nizu **end**, a ne znak **;**, parsiranja završava zamjenom znaka **S₂** praznim nizom. Cjelokupno stablo koje ima listove označene završnim znakovima zadanog niza, prikazano je na slici 3.13d. Na temelju izgrađenog generativnog stabla zaključuje se da zadani niz pripada jeziku $L(G)$.

Opisano je parsiranje jednostavno i učinkovito. Prilikom izbora jedne od više produkcija koje imaju istu lijevu stranu, bilo je moguće donijeti jednoznačnu odluku na temelju samo jednog pročitanog znaka zadanog niza. Niti jednom nije postojala mogućnost izbora više od jedne produkcije. Postoji li mogućnost izbora više produkcija, potrebno je ispitati sve moguće kombinacije primjena produkcija kako bi se utvrdilo da li gramatika generira zadani niz.

Opisana gramatika i proces parsiranja od vrha prema dnu nazivaju se *LL(1)* gramatika i *LL(1)* parsiranje. Prvi "L" u nazivu *LL* označava da se ulazni niz čita sa lijeva na desno (*Left-to-right scanning*), a drugi "L" označava da se produkcije primjenjuju na krajnje lijevi nezavršni znak u generiranom međunizu (*Leftmost derivation*). Tijekom procesa parsiranja čita se jedan po jedan znak ulaznog niza. Na temelju pročitanog znaka ulaznog niza i na temelju krajnje lijevog nezavršnog znaka u nizu, primijeni se produkcija gramatike. Primjenjuje se ona produkcija koja ima na lijevoj strani nezavršni znak koji je jednak nezavršnom znaku u generiranom međunizu i koja na desnoj strani na krajnje lijevom mjestu ima završni znak koji je jednak pročitanom znaku u nizu. Nakon primjene produkcije, generirani završni znakovi koji se nalaze lijevo od krajnje lijevog nezavršnog znaka usporede se sa završnim znakovima u ulaznom nizu. Ako su uspoređeni znakovi isti, onda se nastavlja s čitanjem sljedećeg znaka u ulaznom nizu koji slijedi neposredno iza do tada uspoređenih znakova i odlučuje se o primjeni sljedeće produkcije. Nastavi li se proces parsiranja čitanjem samo jednog narednog znaka u nizu, kaže se da je parsiranje *LL(1)*. Brojka 1 u nazivu *LL(1)* označava da se odluka o primjeni produkcije donosi na temelju samo jednog pročitanog znaka. Metoda *LL(1)* parsiranja posebice se koriste u sintaksnoj analizi programskih jezika. Čita li se narednih k znakova u nizu, proces parsiranja se naziva *LL(k)*. *LL(k)* parsiranje donosi odluku o primjeni produkcije na temelju k pročitanih znakova ulaznog niza ($k \geq 1$).

Tehnika rekurzivnog spusta

Parsiranje od vrha prema dnu moguće je jednostavno programski ostvariti tehnikom rekurzivnog spusta. Tehnika rekurzivnog spusta ostvaruje se programskim jezikom koji ima svojstvo rekurzivnog poziva potprograma. Nezavršnim znakovima gramatike pridružuju se potprogrami. Potprogram ispituje da li podniz zadanog niza zadovoljava strukturu zadalu desnom stranom one produkcije u kojoj se nezavršni znak pridružen tom potprogramu nalazi na lijevoj strani. Završni znakovi na desnoj strani produkcije uspoređuju se sa znakovima u nizu. Za nezavršne znakove na desnoj strani produkcije pozivaju se potprogrami koji provjeravaju strukturu za te nezavršne znakove. Nade li se jedan te isti nezavršni znak na lijevoj i desnoj strani produkcije, potprogram pridružen tom nezavršnom znaku poziva se rekurzivno. Potprogrami se mogu rekurzivno pozivati i preko drugih potprograma.

Primjer 3.12. Gramatika $G=(\{C, S, S_1, S_2\}, \{\text{begin, end, while, do, ;, :=, }, \neq, \text{var}\}, P, C)$ s produkcijama:

- | | | |
|--|-------------------------------|--|
| 1) $C \rightarrow \text{begin } S_1 \text{ end}$ | 3) $S_2 \rightarrow ; S_1$ | 5) $S \rightarrow \text{var} := \text{var}$ |
| 2) $S_1 \rightarrow S S_2$ | 4) $S_2 \rightarrow \epsilon$ | 6) $S \rightarrow \text{while var} \neq \text{var} \text{do } S$ |
| | | 7) $S \rightarrow \text{begin } S_1 \text{ end}$ |

želi se programski ostvariti tehnikom rekurzivnog spusta. Budući da gramatika ima četiri nezavršna znaka, tehnika rekurzivnog spusta zasnovana je na glavnom programu i četiri potprograma. Neka je \perp oznaka kraja ulaznog niza w .

```

GlavniProgram()
{
    ulaz = prvi znak iz niza w;
    c();
    ako( ulaz !=  $\perp$  )
        ispiši( " w  $\notin L(G)$  " );
    inače
        ispiši( " w  $\in L(G)$  " );
}

c()                                // potprogram pridružen nezavršnom znaku C
{
    ako( ulaz != begin )           // programsko ostvarenje produkcije (1)
        ispiši( " w  $\notin L(G)$  " );
    ulaz = sljedeći znak iz niza w;
    S1();
    ako( ulaz != end )
        ispiši( " w  $\notin L(G)$  " );
    ulaz = sljedeći znak iz niza w;
}

S1()                                // potprogram pridružen nezavršnom znaku S1
{
    S();
    S2();
}

S2()                                // potprogram pridružen nezavršnom znaku S2
{
    ako( ulaz == ; )              // programsko ostvarenje produkcije (3)
    {
        ulaz = sljedeći znak iz niza w;
        S1();
    }                               // programsko ostvarenje produkcije (4)
}

S()                                // potprogram pridružen nezavršnom znaku S
{
    slučaj ( ulaz )
    {
        var:                      // programsko ostvarenje produkcije (5)
        ulaz = sljedeći znak iz niza w;
        ako( ulaz != := )
            ispiši( " w  $\notin L(G)$  " );

        ulaz = sljedeći znak iz niza w;
        ako( ulaz != var )
            ispiši( " w  $\notin L(G)$  " );
        ulaz = sljedeći znak iz niza w;
    }
}

```

```

while:           // programsko ostvarenje produkcije (6)

    ulaz = sljedeći znak iz niza w;
    ako( ulaz != var )
        ispiši( " w ∉ L(G) " );

    ulaz = sljedeći znak iz niza w;
    ako( ulaz != * )
        ispiši( " w ∉ L(G) " );

    ulaz = sljedeći znak iz niza w;
    ako( ulaz != var )
        ispiši( " w ∉ L(G) " );

    ulaz = sljedeći znak iz niza w;
    ako( ulaz != do )
        ispiši( " w ∉ L(G) " );

    ulaz = sljedeći znak iz niza w;
    S();

begin:           // programsko ostvarenje produkcije (7)

    ulaz = sljedeći znak iz niza w;
    S1();

    ako( ulaz != end )
        ispiši( " w ∉ L(G) " );

    ulaz = sljedeći znak iz niza w;

    svi ostali znakovi:
        ispiši( " w ∉ L(G) " );
}

}

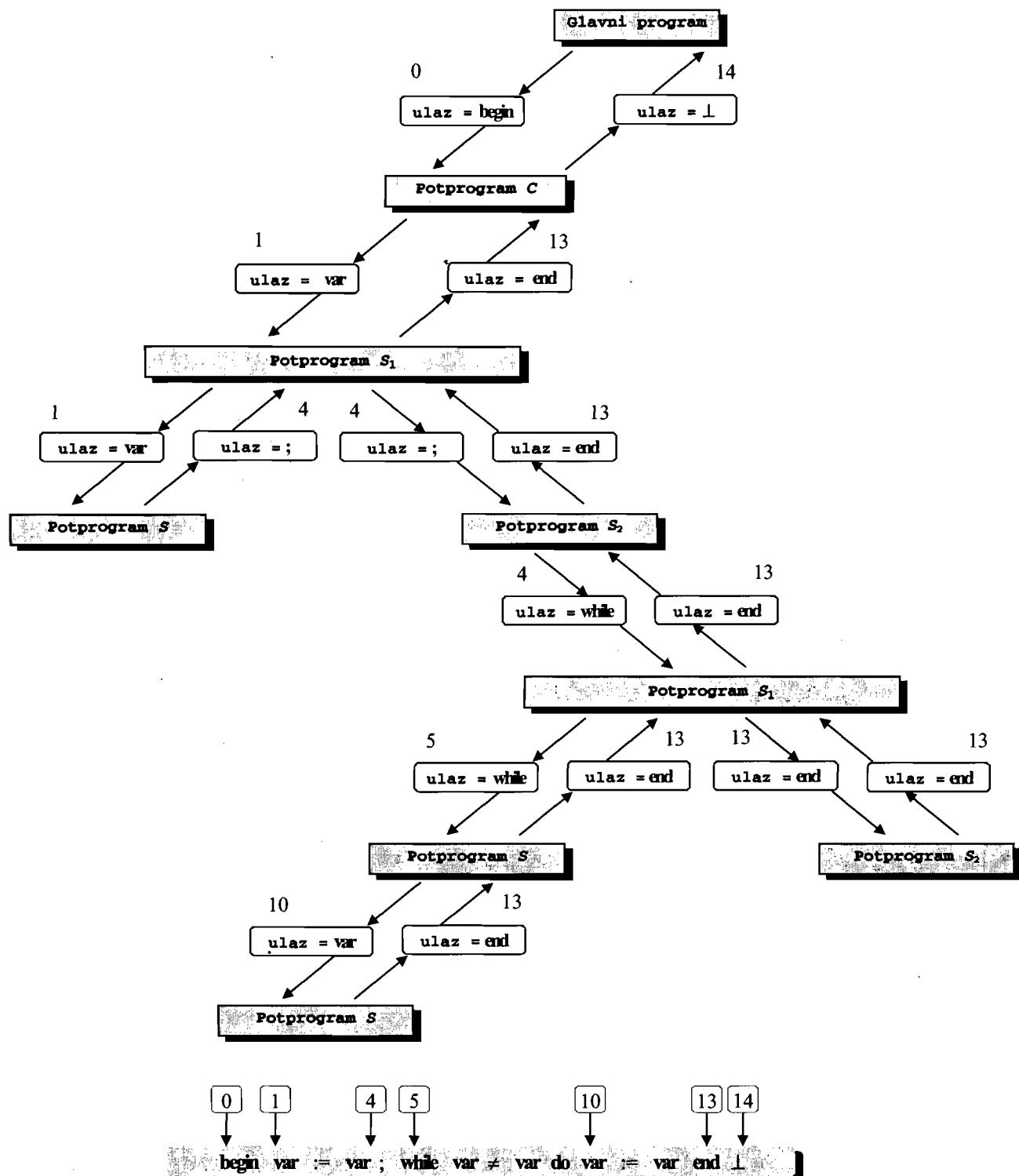
```

U glavnom programu pročita se prvi znak niza w i pozove se potprogram C . Nakon što se ispita zadovoljavaju li pročitani znakovi strukturu zadani nezavršnim znakom C , provjerava se jesu li pročitani svi znakovi ulaznog niza. Ako je sljedeći znak oznaka kraja niza \perp , niz w je element jezika koji generira gramatika G . Ako sljedeći znak nije oznaka kraja niza \perp , onda nisu pročitani svi znakovi niza. U tom slučaju niz w nije element jezika $L(G)$, iako jedan od njegovih pravih prefiksa pripada zadanim jeziku. Ispiše se poruka da se niz ne prihvata i zaustavlja se daljnji rad parsera.

Potprogram C provjerava strukturu zadani desnom stranom produkcije (1). Ako niz ne započinje znakom **begin**, onda niz w nije element jezika $L(G)$. Započinje li niz znakom **begin**, pročita se sljedeći znak i pozove se potprogram S_1 . Nakon uspješne provjere da ostatak niza zadovoljava strukturu zadani nezavršnim znakom S_1 , provjerava se da li je sljedeći znak niza **end**. Ovisno o rezultatu ispitivanja, prekida se rad parsera i ispiše se da niz w nije element jezika $L(G)$, ili se nadzor izvođenja programa vraća glavnom programu.

Nezavršni znak S_1 zadan je produkcijom (2). Desna strana produkcije (2) određuje da se u potprogramu S_1 najprije pozove potprogram S , a nakon toga potprogram S_2 .

Nezavršni znak S_2 jest lijeva strana produkcija (3) i (4). Producije grade ulančane naredbe. Producija (3) određuje da se naredbe ulančavaju znakom $;$. Nakon posljednje generirane naredbe, ϵ -producija (4) obustavlja daljnje generiranje ulančanih naredbi, a nezavršni znak S_2 zamijeni se praznim nizom. Potprogram S_2 ispituje da li je pročitan znak $;$. Pročita li se znak $;$, nastavi se s čitanjem sljedećeg znaka niza w i pozove se potprogram S_1 . Ako sljedeći znak nije $;$, onda se primjeni ϵ -producija (4), odnosno završi se s izvođenjem potprograma S_2 .



Slika 3.14: Dijagram poziva potprograma prilikom parsiranja niza
begin var := var ; while var ≠ var do var := var end

Producije (5), (6) i (7) imaju nezavršni znak S na lijevoj strani. Potprogram S ispituje koji je znak pročitan prilikom njegova poziva. Pročita li se znak **var**, ispituje se jesu li sljedeća dva znaka u nizu $\mathbf{:=}$ i **var**. Ako sljedeća dva znaka nisu jednaka znakovima $\mathbf{:=}$ i **var**, onda niz w nije u jeziku $L(G)$. Ako su sljedeća dva znaka $\mathbf{:=}$ i **var**, onda se pročita sljedeći znak, zaustavlja se izvođenje potprograma i nadzor izvođenja programa vraća se na mjesto poziva potprograma S .

Pročita li se prilikom poziva potprograma S znak **while**, za ispitivanje sljedećih znakova koristi se produkcija (6). Ako se pročita znak **begin**, onda se koristi produkcija (7).

Neka se tehnikom rekurzivnog spusta parsira niz:

```
begin var  $\mathbf{:=}$  var ;
      while var  $\neq$  var do
          var  $\mathbf{:=}$  var
end
```

Na slici 3.14 opisan je način izvođenja potprograma prilikom parsiranja zadanog niza. Vrijednost varijable **ulaz** prikazana je prije poziva potprograma i nakon završetka izvođenja potprograma. Broj uz varijablu označava vrijednost kazaljke koja pokazuje na završni znak u ulaznom spremniku koji se u danom trenutku ispituje.

Osnova načela programskog ostvarenja parsera tehnikom rekurzivnog spusta su:

- 1) U glavnom programu pročita se prvi znak niza w i pozove se potprogram pridružen početnom nezavršnom znaku gramatike. Nakon što završi izvođenje potprograma, provjerava se je li pročitana oznaka kraja niza. Pročita li se oznaka kraja niza, niz w se prihvaca. U suprotnom, niz w nije u jeziku koji generira zadana gramatika i niz se ne prihvaca.

```
GlavniProgram()
{
    ulaz = prvi znak iz niza w;
    pozovi potprogram pridružen početnom nezavršnom znaku;

    ako( ulaz  $\neq$   $\perp$  )
        ispiši( " w  $\notin L(G)$  " );
    inače
        ispiši( " w  $\in L(G)$  " );
}
```

- 2) Za svaki nezavršni znak gramatike izgradi se jedan potprogram. Postoji li više produkcija istih lijevih strana, za svaku produkciju gradi se zasebni dio potprograma. Na primjer, neka su u gramatici zadane produkcije:

$$A \rightarrow a_1\beta_1 \mid a_2\beta_2 \mid \dots \mid a_n\beta_n,$$

gdje su znakovi a_i različiti završni znakovi gramatike. Za zadane produkcije istih lijevih strana gradi se sljedeći potprogram A :

```
A()
{
    slučaj( ulaz )
    {
        a1:
            ispitaj sljedeće znakove prema nizu  $\beta_1$ ;
        a2:
            ispitaj sljedeće znakove prema nizu  $\beta_2$ ;
        ...
    }
}
```

```

...
...
 $a_n:$ 
    ispitaj sljedeće znakove prema nizu  $\beta_n$ ;
    svi ostali znakovi:
        ispiši( "  $w \notin L(G)$  " );
}
}

```

Ako znak pročitan prilikom poziva potprograma nije jednak niti jednom završnom znaku desne strane produkcijske rečenice na krajnje lijevom mjestu, onda se niz w ne prihvaca.

- 3) Dijelovi potprograma koji ispituju znakove prema desnoj strani produkcijske rečenice β_i grade se na sljedeći način:

- a) Za bilo koji završni znak b na desnoj strani produkcijske rečenice $A \rightarrow a\alpha b\gamma$, a koji se ne nalazi na krajnje lijevom mjestu, izgradi se sljedeći dio potprograma:

```

A()
{
    ...
    ulaz = sljedeći znak iz ulaznog niza w;
    ako( ulaz != b )
        ispiši( "  $w \notin L(G)$  " );
    ...
}

```

Ako pročitani znak nije b , onda se niz w ne prihvaca.

- b) Za bilo koji nezavršni znak B na desnoj strani produkcijske rečenice $A \rightarrow a\alpha B\gamma$, a koji se ne nalazi na krajnje lijevom mjestu, pročita se sljedeći znak niza w i pozove se potprogram pridružen tom nezavršnom znaku:

```

A()
{
    ...
    ulaz = sljedeći znak iz ulaznog niza w;
    B();
    ...
}

```

- c) Ako se nezavršni znak B nalazi na krajnje lijevom mjestu desne strane produkcijske rečenice, onda se samo pozove potprogram pridružen tom nezavršnom znaku:

```

A()
{
    B();
    ...
}

```

Parsiranje od dna prema vrhu

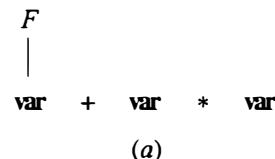
Parsiranje od dna prema vrhu započinje gradnjom generativnog stabla od listova, odnosno od završnih znakova gramatike. U nizu završnih znakova w , odnosno u dobivenim međunizovima završnih i nezavršnih znakova, nastoji se prepoznati jedna od desnih strana produkcija. Ako je dio međuniza jednak desnoj strani produkcije, onda se taj dio međuniza zamjeni lijevom stranom produkcije. Zamjenama desnih strana produkcija lijevim stranama nastaje se izgraditi svi čvorovi stabla uključujući i korijen stabla, odnosno nastoji se niz završnih znakova w zamjeniti početnim nezavršnim znakom gramatike. Budući da se tijekom parsiranja smanjuje (reducira) ili ostaje jednak broj znakova u međunizu, produkcije se u postupku parsiranja od dna prema vrhu nazivaju *redukcije*. Metode parsiranja od dna prema vrhu posebice se koriste u raznim generatorima parsera.

Primjer 3.13. Zadana je gramatika $G = (\{E, T, F\}, \{\text{var}, +, *, (,)\}, P, E)$ s produkcijama:

- | | |
|--------------------------|-------------------------------|
| 1) $E \rightarrow E + T$ | 4) $T \rightarrow F$ |
| 2) $E \rightarrow T$ | 5) $F \rightarrow (E)$ |
| 3) $T \rightarrow T * F$ | 6) $F \rightarrow \text{var}$ |

i niz $w = \text{var} + \text{var} * \text{var}$. Gramatika G generira aritmetičke izraze sa zagradama i operacijama zbrajanja i množenja. Završni znak **var** označava programsku varijablu.

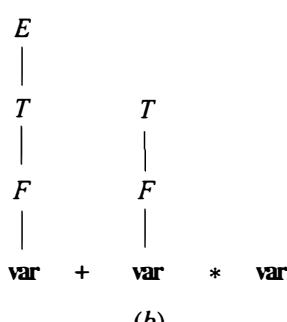
Parsiranjem od dna prema vrhu želi se provjeriti da li niz **var+var*var** pripada jeziku $L(G)$. Niz w čita se sa lijeva na desno. Desne strane produkcija nastoje se pronaći što je moguće više u lijevo u danom međunizu, odnosno nastoji se izgraditi postupak generiranja niza w zamjenom krajnje desnog nezavršnog znaka.



Slika 3.15: Početak parsiranja od dna prema vrhu

Najprije se primjeni produkcija (6), odnosno redukcija (6). Prvi lijni završni znak **var** u nizu w jest desna strana produkcije $F \rightarrow \text{var}$, te se taj znak zamjeni nezavršnim znakom F . Primjena redukcije (6) i početak gradnje stabla prikazan je na slici 3.15a. Primjenom redukcije (6), iz niza završnih znakova **var+var*var** nastaje međuniz **F+var*var**:

$$\text{var+var*var} \Leftarrow F+\text{var*var} \quad - \text{primjena redukcije (6)}.$$



Slika 3.15: Nastavak parsiranja od dna prema vrhu

Daljnja gradnja stabla je jednoznačna. Traži se podniz na lijevoj strani međuniza završnih i nezavršnih znakova koji je jednak jednoj od desnih strana produkcija, a onda se pronađeni podniz zamjeni lijevom stranom produkcije:

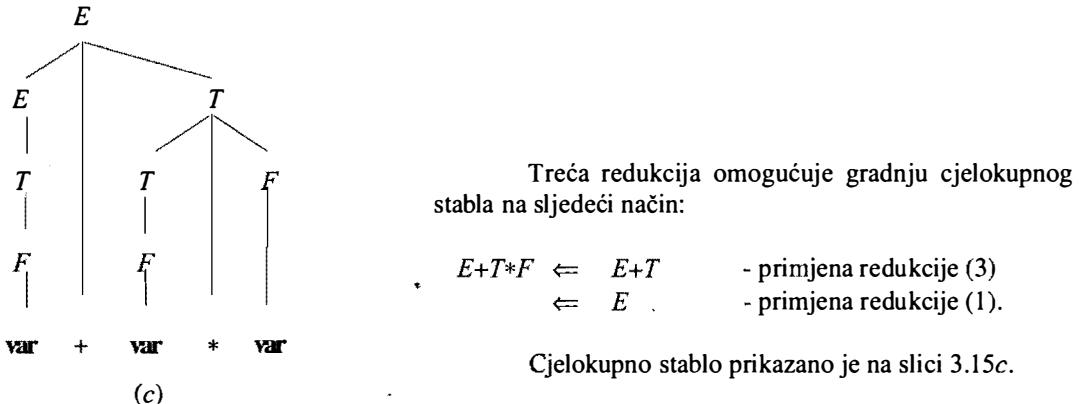
$$\begin{aligned} F + \text{var*var} &\Leftarrow T + \text{var*var} \quad - \text{primjena redukcije (4)} \\ &\Leftarrow E + \text{var*var} \quad - \text{primjena redukcije (2)} \\ &\Leftarrow E + F * \text{var} \quad - \text{primjena redukcije (6)} \\ &\Leftarrow E + T * \text{var} \quad - \text{primjena redukcije (4)}. \end{aligned}$$

Slika 3.14b prikazuje gradnju stabla primjenom prethodno opisanih redukcija.

Mogućnost primjene triju različitih produkcija čini nastavak procesa parsiranja nejednoznačnim:

- 1) primjena redukcije (1), odnosno zamjena niza $E + T$ znakom E : $E + T * \text{var} \Leftarrow E * \text{var}$,
- 2) primjena redukcije (2), odnosno zamjena znaka T znakom E : $E + T * \text{var} \Leftarrow E + E * \text{var}$,
- 3) primjena redukcije (6), odnosno zamjena znaka **var** znakom F : $E + T * \text{var} \Leftarrow E + T * F$.

Prve dvije redukcije ne omogućavaju završetak gradnje generativnog stabla. U oba međuniza nalazi se podniz E^*var . Podniz E^*var nije moguće reducirati primjenom niti jedne redukcije na međuniz T^*F koji omogućuje primjenu redukcije $T \rightarrow T^*F$. Desna strana produkcije $T \rightarrow T^*F$ jedina sadrži operator $*$.



Slika 3.15: Nastavak parsiranja od dna prema vrhu

Za razliku od primjera parsiranja od vrha prema dnu, dani primjer parsiranja od dna prema vrhu nije jednoznačan. Nakon prvi pet primjenjenih redukcija postojala je mogućnost primjene tri redukcije. Neka se te redukcije nazovu višezačne redukcije za niz w i za dani postupak parsiranja. U slučaju nejednoznačnosti, izabere se jedna od višezačnih redukcija i nastoji se izgraditi ostatak generativnog stabla. Ne uspije li se izgraditi generativno stablo, proces parsiranja se postupkom unazadnog pretraživanja vraća na mjesto nejednoznačnosti i pokuša se izabrati sljedeća višezačna redukcija. Postupak unazadnog pretraživanja ponavlja se sve dok se ne ispitaju sve moguće višezačne redukcije. Uspije li se primjenom barem jedne višezačne redukcije izgraditi generativno stablo, niz w jest element jezika $L(G)$. Ako nije moguće izgraditi stablo primjenom niti jedne višezačne redukcije, onda niz nije element jezika $L(G)$.

Učinkovitost procesa parsiranja metodom od vrha prema dnu i metodom od dna prema vrhu zavisi o nejednoznačnosti. Unazadno pretraživanje značajno usporava rad obje metode parsiranja, naročito u slučaju kada se nejednoznačnost javlja na više mesta.

LR parser

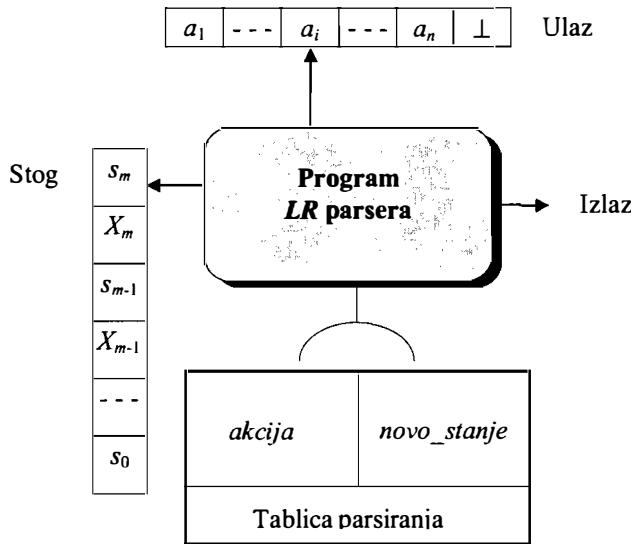
$LR(k)$ parser koristi metodu parsiranja od dna prema vrhu. Znak "L" označava da se niz w čita sa lijeva na desno (Left-to-right scanning). Znak "R" označava da se stablo gradi obrnutim postupkom generiranja niza zamjenom krajnje desnog nezavršnog znaka (Rightmost derivation). Obrnuti postupak generiranja niza zamjenom krajnje desnog nezavršnog znaka započinje od niza završnih znakova i nastoji primjenom redukcija zamijeniti niz završnih znakova početnim nezavršnim znakom. U prethodnom primjeru koristi se upravo obrnuti postupak generiranja niza zamjenom krajnje desnog nezavršnog znaka. Broj k određuje da je potrebno pročitati najviše k znakova unaprijed za potrebe donošenja odluke o primjeni redukcije.

Iako postoje i drugi postupci parsiranja od dna prema vrhu, LR postupak parsiranja jest najopćenitiji poznati postupak parsiranja bez unazadnog pretraživanja koji je moguće učinkovito primjeniti na širok skup kontekstno neovisnih gramatika. Postoje kontekstno neovisne gramatike za koje nije moguće izgraditi LR parser, odnosno za njih kažemo da nisu LR gramatike. Ostale kontekstno neovisne gramatike, za koje je moguće izgraditi LR parser, su LR gramatike.

Konstrukcija LR parsera je znatno složenija od konstrukcije parsera tehnikom rekurzivnog spusta. LR parser gradi se posebnim generatorom. Generator LR parsera gradi parser izravno na temelju zadanih produkcija kontekstno neovisne gramatike. Postoje različite klase generatora, od jednostavnih do složenih.

LR parseri različitih gramatika razlikuju se samo u tablici parsiranja. Tablicu parsiranja gradi generator *LR* parsera na temelju zadane kontekstno neovisne gramatike, slično kao što generator konačnog automata gradi tablicu prijelaza na temelju regularnih izraza. Program parsera koji se koristi tablicom parsiranja jest isti za sve *LR* parsere.

U nastavku odjeljka opisan je model *LR* parsera, algoritam *LR* parsiranja i primjer rada *LR* parsera. Zbog svoje složenosti, u nastavku odjeljka se ne razmatra postupak gradnje tablice *LR* parsera. Postupak gradnje tablice *LR* parsera istražuje se u okviru generatora sintaksnih analizatora programskih jezika.



Slika 3.16: Model *LR* parsera

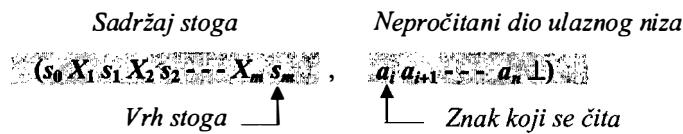
Model *LR* parsera. Model *LR* parsera prikazan je na slici 3.16. Dijelovi *LR* parsera su: ulazni spremnik, potisni stog (*LIFO* stog), program *LR* parsera, tablica parsiranja i izlaz.

Program *LR* parsera čita znak po znak iz ulaznog spremnika. *Ulazni spremnik* sadrži niz w koji se parsira. Program parsiranja koristi se *potisnim stogom* za spremanje niza oblika:

$$s_0 X_1 s_1 X_2 s_2 \dots X_m s_m,$$

gdje je s_i na vrhu stoga. Znakovi X_i su završni ili nezavršni znakovi gramatike. Znakovi s_i su stanja. U stanju na vrhu stoga preslikava se sadržaj stoga koji se nalazi ispod tog stanja. Tijekom rada *LR* parser koristi samo stanja s_i . Znakove gramatike X_i nije potrebno stavljati na stog. Oni su dodani na stog zbog lakšeg praćenja rada *LR* parsera.

Tablica parsiranja sastoji se od dva dijela: *tablica akcija* i *tablica novo_stanje*. Program *LR* parsera čita stanje na vrhu stoga s_m i znak a_i niza w u ulaznom spremniku. Na temelju indeksa $[s_m, a_i]$ tablice akcija odredi se *akcija* koja se izvodi nad ulaznim nizom i stogom. Postoje četiri vrste akcija: *pomak s* (s jest oznaka stanja), *reduciraj A → β* ($A \rightarrow \beta$ jest produkcija zadane gramatike) *prihvati* i *odbaci*. Akcije *prihvati* i *odbaci* određuju *izlaz* *LR* parsera, odnosno određuju da li zadani niz w pripada jeziku $L(G)$. Neka se niz znakova stoga, koji se u danom trenutku nalazi na stogu, i niz znakova u ulaznom spremniku, koji se nalaze desno od znaka a_i , spoje u jedinstvenu listu i međusobno odvoje zarezom:



Dobivena lista naziva se *konfiguracija* *LR* parsera. Izuzmu li se iz konfiguracije oznake stanja, dobije se niz:

$$X_1 X_2 \dots X_m a_i a_{i+1} \dots a_n,$$

koji je jedan od međunizova dobivenih postupkom generiranja zamjenom krajnje desnog nezavršnog znaka.

Neka je *LR* parser u konfiguraciji:

$$(s_0 X_1 s_1 X_2 s_2 \dots X_m s_m, a_i a_{i+1} \dots a_n \perp)$$

LR parser čita znak a_i , a na vrhu stoga nalazi se stanje s_m . Na temelju tablice $akcija[s_m, a_i]$ odredi se sljedeća akcija. Akcija mijenja konfiguraciju *LR* parsera na sljedeći način:

1) $akcija[s_m, a_i] = pomak\ s$

Parser stavlja na stog pročitani znak a_i , a zatim stanje s koje se dobije iz tablice $akcija[s_m, a_i]$. Nastavlja se s čitanjem znaka a_{i+1} . Nova konfiguracija LR parsera je:

$$(s_0 X_1 s_1 X_2 s_2 \dots X_m s_m a_i s \dots, a_{i+1} \dots a_n \perp).$$

2) $akcija[s_m, a_i] = reduciraj\ A \rightarrow \beta$

S vrha stoga uzme se $2r$ znakova, gdje je r broj znakova u nizu β . Niz β jest desna strana produkcije $A \rightarrow \beta$. Uzima se ukupno $2r$ znakova: r oznaka stanja i r znakova gramatike. Nakon što se s vrha stoga uzme $2r$ znakova, na vrhu stoga je stanje s_{m-r} . Na vrh stoga stavlja se nezavršni znak A , koji je lijeva strana produkcije $A \rightarrow \beta$, i stanje $s=novo_stanje[s_{m-r}, A]$, koje se dobije iz tablice $novo_stanje$. Parser ne čita sljedeći znak iz ulaznog spremnika, već glava za čitanje ostaje na istom znaku a_i . LR parser prelazi u konfiguraciju:

$$(s_0 X_1 s_1 X_2 s_2 \dots X_{m-r} s_{m-r} A s \dots, a_{i+1} \dots a_n \perp).$$

3) $akcija[s_m, a_i] = prihvati$

Ovom akcijom uspješno završava proces parsiranja i niz w se prihvata.

4) $akcija[s_m, a_i] = odbaci$

Niz w se ne prihvata i proces parsiranja se zaustavlja.

Algoritam LR parsera. Algoritam LR parsera opisuje se na sljedeći način:

Ulaz

Ulaz algoritma je zadani niz w i tablica LR parsera. Tablica LR parsera generira se na temelju zadane kontekstno neovisne gramatike G .

Izlaz

Ako je niz u jeziku $L(G)$, onda LR parser ispiše da se niz prihvata. LR parser gradi generativno stablo počev od niza završnih znakova w , i to obrnutim postupkom generiranja niza zamjenom krajnjeg desnog nezavršnog znaka. Nije li niz w u jeziku $L(G)$, LR parser ispiše da se niz odbaci.

Postupak parsiranja

Na početku rada na stogu se nalazi početno stanje s_0 , a u ulaznom spremniku niz $w\perp$, gdje je \perp oznaka kraja niza. Parser izvodi program LR parsera sve dok se ne izvede akcija *prihvati* ili akcija *odbaci*.

Program LR parsera. LR parser izvodi sljedeći program:

```
ProgramLrparsera()
{
    postavi kazaljku da pokazuje na prvi znak u nizu w\perp;
    dok( 1 )                                // ponavljač zauvijek
    {
        // neka je s stanje na vrhu stoga
        // neka KAZALJKA pokazuje na znak a u nizu w\perp
```

```

    slučaj( akcija[ s, a ] )
    {
        pomak s':
            stavi a na stog;
            stavi s' na stog;
            pomakni KAZALJKU na sljedeći znak u nizu wL;

        reduciraj A→β:
            skinji sa stoga  $2^*|\beta|$  znakova;
            // neka je s' stanje na vrhu stoga
            // nakon što se skine  $2^*|\beta|$  znakova

            stavi A na stog;
            stavi NovoStanje[s', A] na stog;
            ispiši( "primjena produkcije A→β" );

        prihvati:
            ispiši( "niz w je iz jezika L(G)" );
            završi izvođenje programa;

        svi ostali slučajevi:
            ispiši( "niz w nije iz jezika L(G)" );
            završi izvođenje programa;
    }
}

```

Primjer 3.14. Zadana je gramatika $G = (\{E, T, F\}, \{\text{var}, +, *, (,)\}, P, E)$ s produkcijama:

1) $E \rightarrow E + T$	4) $T \rightarrow F$
2) $E \rightarrow T$	5) $F \rightarrow (E)$
3) $T \rightarrow T * F$	6) $F \rightarrow \text{var},$

i niz **var+var*var**. Gramatika G generira aritmetičke izraze sa zagradama i operacijama zbrajanja i množenja. Završni znak **var** označava programsku varijablu. Slika 3.17 prikazuje tablicu parsera za zadanu gramatiku.

Stanje	akcija						novo_stanje		
	var	+	*	()	\perp	E	T	F
0	s5			s4			1	2	3
1		s6				prihvati			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4			9	3	
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

- si* - stavi ulazni znak na stog i stavi stanje i na vrh stoga;
- rij* - reduciraj primjenom produkcijske pravila broj j ;
- prihvati* - niz w jest u jeziku $L(G)$;
- praznine označavaju *odbaci*, odnosno da niz w nije u jeziku $L(G)$;
- početno stanje* jest 0.

Slika 3.17: Tablica LR parsera za gramatiku G

Parsiranje niza **var+var*var** opisano je tablicom 3.1. Prvi stupac tablice 3.1 prikazuje sadržaj stoga. Drugi stupac prikazuje znakove u ulaznom spremniku. Znakovi prvog stupca i znakovi drugog stupca čine konfiguraciju LR parsera. Treći stupac definira akciju koja se primjenjuje.

	Stog	Ulaz	Akcija
(1)	0	var+var*var \perp	pomak 5
(2)	0 var 5	+ var*var \perp	redukcija $F \rightarrow \text{var}$
(3)	0 F 3	+ var*var \perp	redukcija $T \rightarrow F$
(4)	0 T 2	+ var*var \perp	redukcija $E \rightarrow T$
(5)	0 E 1	+ var*var \perp	pomak 6
(6)	0 E 1 + 6	var*var \perp	pomak 5
(7)	0 E 1 + 6 var 5	* var \perp	redukcija $F \rightarrow \text{var}$
(8)	0 E 1 + 6 F 3	* var \perp	redukcija $T \rightarrow F$
(9)	0 E 1 + 6 T 9	* var \perp	pomak 7
(10)	0 E 1 + 6 T 9 * 7	var \perp	pomak 5
(11)	0 E 1 + 6 T 9 * 7 var 5	\perp	redukcija $F \rightarrow \text{var}$
(12)	0 E 1 + 6 T 9 * 7 F 10	\perp	redukcija $T \rightarrow T*F$
(13)	0 E 1 + 6 T 9	\perp	redukcija $E \rightarrow E+T$
(14)	0 E 1	\perp	prihvati

Tablica 3.1: LR parsiranje niza **var+var*var**

U retku (1) na vrhu stoga jest početno stanje 0, a u ulazom spremniku čita se znak **var**. U retku 0 i stupcu **var** tablice *akcija* na slici 3.17 označena je akcija s5, odnosno *pomak 5*. Na stog se stavlja znak **var**, a zatim stanje 5.

Na vrhu stoga je stanje 5, a u ulazom spremniku je znak +. Redak 5 i stupac + određuju akciju *reduciraj 6*, odnosno primjeni se redukcija $F \rightarrow \text{var}$. Desna strana produkcije ima jedan znak, tj. $|\text{var}|=r=1$. S vrha stoga uzima se $2r=2$ znakova, odnosno uzima se stanje 5 i znak **var**. Na vrh stoga stavlja se lijeva strana produkcije, odnosno znak F . U retku 0 i stupcu F tablice *novo_stanje* upisano je stanje 3. Stanje 3 stavlja se na vrh stoga.

U recima (3) i (4) primjenjuju se daljnje dvije redukcije, sve dok se u retku (5) ne pojavi stanje 1 na vrhu stoga. Stanje 1 i ulazni znak + definiraju akciju s6, odnosno na vrh stoga stavlja se znak + i stanje 6.

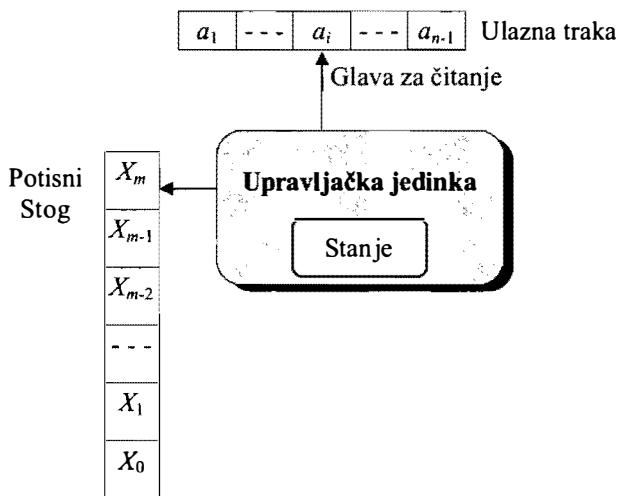
Daljnje parsiranje niza nastavlja se sve dok se ne izvede akcija *prihvati*, koja označava da je niz **var+var*var** element jezika $L(G)$.

3.2 Potisni automat (PA)

Za potrebe prihvaćanja kontekstno neovisnih jezika gradi se potisni automat. Model potisnog automata opisan je u poglavju 3.2.1, dok je formalna definicija dana u odjelu 3.2.2. Istovjetnost kontekstno neovisne gramatike i potisnog automata pokazana je u odjelu 3.2.3.

3.2.1 Model potisnog automata

U odjelu 2.4.1 pokazano je postojanje kontekstno neovisnih jezika koji nisu regularni. Za potrebe prihvaćanja kontekstno neovisnih jezika proširuje se model konačnog automata. Upravljačkoj jedinki, glavi za čitanje i ulaznoj traci dodaje se *potisni stog* (*LIFO* stog). Automat se naziva *potisni automat* (PA).



Model potisnog automata prikazan je na slici 3.18. Rad *glave za čitanje* ostaje isti kao i kod modela NKA. Pored čitanja znakova ulazne trake, upravljačka jedinka čita i jedan znak vrha *potisnog stoga*. Nakon čitanja znaka, s vrha stoga uzima se pročitani znak, a na vrh stoga stavlja se niz znakova. Znakovi koji se nalaze na ulaznoj traci nazivaju se *ulazni znakovi*, dok se znakovi na stogu nazivaju *znakovi stoga*. Upravljačka jedinka je u jednom od konačnog broja *stanja*. Stanja su podijeljena u dva skupa: prihvatljiva stanja i neprihvatljiva stanja.

Slika 3.18: Model potisnog automata

Upravljačka jedinka donosi odluku o promjeni sadržaja vrha stoga, o pomaku glave za čitanje i o promjeni stanja. Odluka se donosi na temelju tri podatka:

- 1) stanja upravljačke jedinke,
- 2) znaka koji se nalazi na vrhu stoga,
- 3) i znaka na ulaznoj traci.

Moguće je da upravljačka jedinka doneše odluku na temelju samo dva podatka: stanja upravljačke jedinke i znaka na vrhu stoga. Budući da se odluka donosi bez čitanja znaka ulazne trake, glava za čitanje se ne miče za jedno mjesto u desno.

Upravljačka jedinka odlučuje koji se niz stavlja na vrh stoga. Na vrh stoga moguće je staviti:

- 1) *Prazni niz* ε .

Budući da se čitanjem uzima pročitani znak s vrha stoga, stavljanje praznog niza jednako je uzimanju jednog znaka s vrha stoga.

- 2) *Niz od jednog znaka.*

Na vrh stoga moguće je staviti isti znak koji se pročitao, ili neki drugi znak. Stavljanje istog znaka ne mijenja sadržaj stoga, dok se stavljanjem drugog znaka postiže zamjena znakova na vrhu stoga.

- 3) *Niz od više znakova.*

U procesu simulacije postupka generiranja niza, stog potisnog automata koristi se za čuvanje dijela generiranog međuniza. Zamjena vrha stoga nizom znakova simulira primjenu produkcijske pravila, gdje se nezavršni znak lijeve strane produkcijske pravile zamjeni nizom znakova desne strane produkcijske pravile.

Označimo stanje upravljačke jedinke znakom q . Upravljačka jedinka čita znak a ulazne trake, a znak Z je na vrhu stoga. Tijekom rada upravljačka jedinka primjeni jedan od dva prijelaza:

- i) Na temelju trojke (q, a, Z) upravljačka jedinka mijenja stanje u novo stanje p , pomakne glavu za čitanje za jedno mjesto u desno i zamjeni znak na vrhu stoga nizom znakova γ .
- ii) Na temelju trojke (q, ϵ, Z) upravljačka jedinka mijenja stanje u novo stanje p , ostavi glavu za čitanje na istom mjestu i zamjeni znak na vrhu stoga nizom znakova γ . Oznaka ϵ u (q, ϵ, Z) ima značenje da se odluka donosi bez pročitanog znaka ulazne trake.

Odluka o prihvaćanju niza donosi se isključivo na jedan od dva moguća načina:

- a) PA M koji prihvaca prihvatljim stanjem: uđe li upravljačka jedinka u prihvatljivo stanje nakon što pročita sve znakove na ulaznoj traci, niz se prihvaca. Jezik koji prihvaca PA prihvatljivim stanjem označava se $L(M)$.
- b) PA M koji prihvaca praznim stogom: isprazni li se stog čitanjem svih znakova niza, upravljačka jedinka prihvaca niz. Jezik koji prihvaca PA praznim stogom označava se $N(M)$.

Budući da nije nužno da jedan te isti PA M prihvaca isti jezike prihvatljivim stanjem i praznim stogom, odnosno u većini slučajeva $L(M) \neq N(M)$, uvedene su različite oznake kako bi se ti jezici razlikovali.

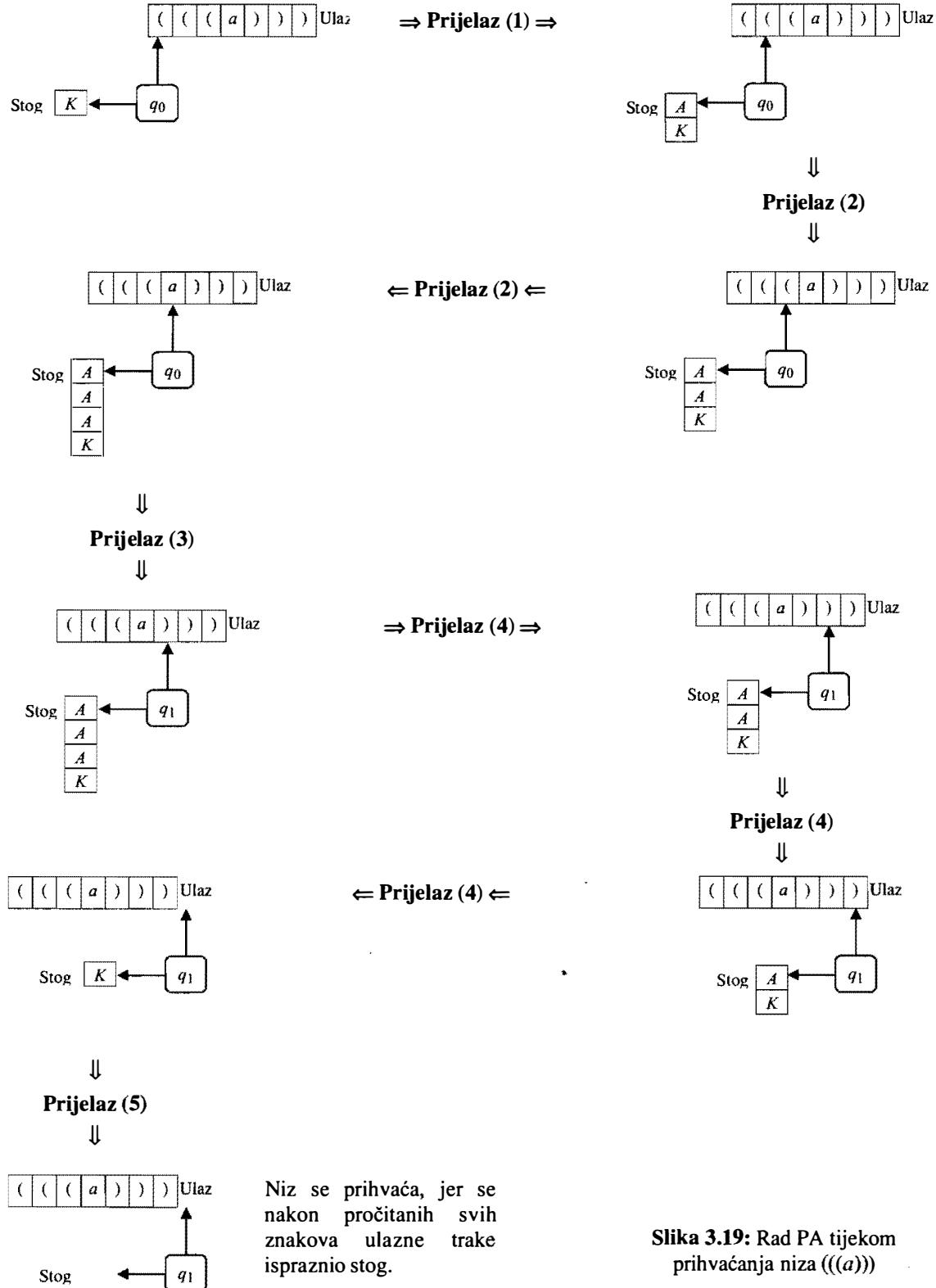
Primjer 3.15. Gradi se PA M koji prihvaca praznim stogom jezik $N(M)=\{(\overset{n}{a})^n \mid n \geq 1\}$, gdje je znak a okružen jednakim brojem otvorenih i zatvorenih zagrada. Ulazni znakovi PA su $($, $)$, i a .

PA ima dva stanja: q_0 i q_1 . Na početku rada PA jest u stanju q_0 , a na vrhu stoga je znak K . PA u stanju q_0 čita redom ulazne znakove sve do znaka a . PA stavlja jedan znak A na vrh stoga za svaku pročitanu otvorenu zagradu. Nakon što pročita znak a , PA prelazi u stanje q_1 . PA u stanju q_1 uzima jedan znak A s vrha stoga za svaku pročitanu zatvorenu zagradu. Broj znakova A na stogu mora biti jednak broju pročitanih zatvorenih zagrada. Nema li znakova A na stogu nakon pročitanih svih znakova ulazne trake, niz se prihvaca. Skup znakova stoga jednak je $\{A, K\}$. Prijelazi PA određuju se na sljedeći način:

Prijelaz	Stanje	Ulaz	Stog	Novo stanje	Novi vrh stoga	Glava za čitanje
1)	q_0	(K	q_0	AK	Pomak u desno
2)	q_0	(A	q_0	AA	Pomak u desno
3)	q_0	a	A	q_1	A	Pomak u desno
4)	q_1)	A	q_1	ϵ	Pomak u desno
5)	q_1	ϵ	K	q_1	ϵ	Zadrži na istom mjestu

Prijelaz (1) ima sljedeće značenje: ako je stanje PA q_0 , na ulazu se čita znak $($, a na vrhu stoga je znak K , onda PA ostaje u istom stanju q_0 , na vrh stoga stavlja niz AK i glavu za čitanje miče za jedno mjesto

u desno. Zamjena znaka K na vrhu stoga nizom AK jednaka je stavljanju znaka A na stog iznad znaka K . Prijelaz (3) zamijeni znak A na vrhu stoga nizom A , odnosno sadržaj stoga ostaje nepromijenjen. Tim prijelazom PA mijenja stanje u q_1 . Prijelaz (4) zamijeni znak A na vrhu stoga praznim nizom ϵ , što ima za posljedicu uzimanje znaka A sa stoga. Ako je PA u stanju q_1 , a na vrhu stoga je K , onda PA bez čitanja znaka ulazne trake uzima znak K sa stoga. Tim prijelazom stog ostaje prazan i ne sadrži niti jedan znak.

Slika 3.19 prati rad PA M za niz $((a))$.

Niz (((a))) se prihvata, jer se čitanjem znakova niza ispraznio stog.

Ima li u nizu veći broj otvorenih zagrada od zatvorenih zagrada, stog PA se ne isprazni nakon pročitanih svih znakova niza. Broj znakova A na stogu jednak je razlici između broja otvorenih zagrada i broja zatvorenih zagrada. Budući da se stog ne isprazni, niz se ne prihvata.

Ima li veći broj zatvorenih zagrada od otvorenih zagrada, stog ostaje prazan prije nego što su pročitani svi znakovi ulazne trake. Daljnji prijelazi PA nisu definirani i nije moguće nastaviti s čitanjem ulaznih znakova. Budući da nisu pročitani svi znakovi, niz se ne prihvata.

Nema li u nizu znaka a , niz se ne prihvata. PA ne može promijeniti stanje u q_1 i ne može pokrenuti proces uzimanja znakova A sa vrha stoga za svaku pročitanu zatvorenu zgradu. Prijelaz za ulazni znak a , znak na stogu A i stanje q_0 nije definiran. Budući da za trojku (q_0, a, A) nije definiran prijelaz, niz se ne prihvata.

Neka se PA M' definira na sličan način kao PA M , osim što se stanje q_1 definira kao prihvatljivo, a početno stanje q_0 kao neprihvatljivo. PA M' prihvata jezik $L(M') = \{("a")^m | n \geq 1, m \geq 0 \text{ i } m \leq n\}$. Jezik $L(M')$ nije isti kao i jezik $N(M)$. Na primjer, niz (((a se prihvata, jer PA nakon pročitanog znaka a prelazi u stanje q_1 koje je prihvatljivo. Budući da je PA u jednom od prihvatljivih stanja i budući da je pročitan čitav niz, niz se prihvata bez obzira što stog nije prazan. Prihvata se niz oblika ("a)^m" koji ima jednak ili manji broj zatvorenih zagrada od otvorenih zagrada. Ima li veći broj zatvorenih zagrada, PA M' neće prihvati niz. Veći broj zatvorenih zagrada isprazni stog. Prazni stog ne omogućava čitanje svih zatvorenih zagrada i niz se ne prihvata. Nadalje, ne prihvata se niz koji nema znaka a , jer jedino znak a omogućava prijelaz PA u prihvatljivo stanje.

3.2.2 Definicija potisnog automata

Potisni automat (PA) formalno se zadaje kao uređena sedmorka:

$$pa = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

gdje je:

- Q - konačan skup stanja;
- Σ - konačan skup ulaznih znakova (abeceda ulaznih znakova);
- Γ - konačan skup znakova stoga (abeceda znakova stoga);
- δ - funkcija prijelaza δ pridružuje trojki $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma$ konačan podskup skupa svih mogućih parova $Q \times \Gamma^*$;
- $q_0 \in Q$ - početno stanje;
- $Z_0 \in \Gamma$ - početni znak stoga;
- $F \subseteq Q$ - skup prihvatljivih stanja.

Mala slova na početku abecede označavaju ulazne znakove (a, b, c, d, \dots), a mala slova na kraju abecede označavaju nizove ulaznih znakova (v, w, x, y, z). Velikim slovima označavaju se znakovi stoga, dok se grčkim slovima označavaju nizovi znakova stoga.

Funkcija prijelaza potisnog automata δ pridružuje trojci (q, a, Z) konačni skup parova (p_i, γ_i) :

$$\delta(q, a, Z) = \{(p_1, \gamma_1), (p_2, \gamma_2), \dots, (p_m, \gamma_m)\},$$

gdje je q stanje potisnog automata ($q \in Q$), a je ulazni znak ($a \in \Sigma \cup \{\epsilon\}$) i Z je znak stoga ($Z \in \Gamma$). Stanja p_i ($p_i \in Q$) i nizovi znakova stoga γ_i ($\gamma_i \in \Gamma^*$) čine par (p_i, γ_i) , gdje je $1 \leq i \leq m$. Funkcija prijelaza ima sljedeće značenje: ako se na ulazu PA koji je u stanju q pojavi znak a , a na vrhu stoga je znak Z , onda PA prelazi u jedno od stanja p_i , vrh stoga Z zamjeni odgovarajućim nizom γ_i i glavu za čitanje pomakne na sljedeći ulazni znak. Ako je $i \neq j$, onda prijelaz u stanje p_i i zamjena znaka na vrhu stoga nizom γ_j nije moguća u jednom koraku. Niz γ_i stavljaju se na stog tako da se najprije stavi krajnje desni znak u nizu γ_i . Znakovi se stavljaju redom sa desna na lijevo sve dok se na vrh stoga ne stavi krajnje lijevi znak niza. Funkcija $\delta(q, a, Z)$ PA jest

nedeterministička u istom smislu kao i funkcija prijelaza $\delta(q, a)$ NKA. Funkcija $\delta(q, a, Z)$ ne određuje jednoznačno novo stanje PA i novi sadržaj vrha stoga, već je vrijednost funkcije $\delta(q, a, Z)$ skup parova stanja i sadržaja vrha stoga.

Postoji li funkcija prijelaza:

$$\delta(q, \varepsilon, Z) = \{(p_1, \gamma_1), (p_2, \gamma_2), \dots, (p_m, \gamma_m)\},$$

PA prelazi iz stanja q u jedno od stanja p_i i zamjeni vrh stoga Z nizom γ_i , $1 \leq i \leq m$. Prijelaz PA definiran je samo stanjem PA i znakom na vrhu stoga. Ulažni znak nema utjecaja na prijelaz i glava za čitanje PA ne miče se na sljedeći ulazni znak. Prijelaz $\delta(q, \varepsilon, Z)$ naziva se ε -prijelaz.

Primjer 3.16. Zadan je PA $M_1 = (\{q_1, q_2, q_3\}, \{0, 1, 2\}, \{N, J, K\}, \delta, q_1, K, \{q_3\})$ sa sljedećim prijelazima:

- | | |
|--|--|
| 1) $\delta(q_1, 0, K) = \{(q_1, NK)\}$ | 2) $\delta(q_1, 1, K) = \{(q_1, JK)\}$ |
| 3) $\delta(q_1, 0, N) = \{(q_1, NN)\}$ | 4) $\delta(q_1, 1, N) = \{(q_1, JN)\}$ |
| 5) $\delta(q_1, 0, J) = \{(q_1, NJ)\}$ | 6) $\delta(q_1, 1, J) = \{(q_1, JJ)\}$ |
| 7) $\delta(q_1, 2, K) = \{(q_2, K)\}$ | |
| 8) $\delta(q_1, 2, N) = \{(q_2, N)\}$ | |
| 9) $\delta(q_1, 2, J) = \{(q_2, J)\}$ | |
| 10) $\delta(q_2, 0, N) = \{(q_2, \varepsilon)\}$ | 11) $\delta(q_2, 1, J) = \{(q_2, \varepsilon)\}$ |
| 12) $\delta(q_2, \varepsilon, K) = \{(q_3, \varepsilon)\}$ | |

PA M_1 prihvata jezik $L(M_1) = \{w2w^R \mid w \text{ jest niz nula i jedinica } (0+1)^*\}$, a niz w^R je niz w napisan obrnutim redoslijedom prihvatljivim stanjme q_3 . Prijelazi (1) do (9) omogućeni su u stanju q_1 , a prijelazi (10) do (12) omogućeni su u stanju q_2 . Za svaki pročitani znak 0 prijelazi (1), (3) i (5) na stog stave jedan znak N (Nula). Za svaku pročitani znak 1 prijelazi (2), (4) i (6) na stog stave jedan znak J (Jedan). Čitanjem znaka 2 PA prelazi iz stanja q_1 u stanje q_2 , odnosno prijelazi (7), (8) i (9) mijenjaju stanje PA. Čita li se na ulazu znak 0, a na vrhu stoga je znak N , prijelaz (10) uzima jedan znak N s vrha stoga. Nadalje, čita li se na ulazu znak 1, a na vrhu stoga je znak J , prijelaz (11) uzima jedan znak J s vrha stoga. Uzimanje znakova sa stoga nastavlja se sve dok se na vrhu stoga ne pojavi znak K . ε -prijelaz (12) uzima znak K s vrha stoga i mijenja stanje u prihvatljivo stanje q_3 . Pojavi li se na ulazu niz oblika $w2w^R$, usporedbom znakova podniza w i w^R isprazni se stog, promijeni se stanje u prihvatljivo stanje q_3 i PA prihvati niz.

Tablica 3.2 opisuje slijed prijelaza za niz 0012100. Na početku rada na vrhu stoga je početni znak stoga K , PA je u početnom stanju q_1 i na ulazu je prvi znak niza 0012100, a to je znak 0. Nakon prijelaza:

$$\delta(q_1, 0, K) = \{(q_1, NK)\}$$

na stogu je niz NK , stanje ostaje q_1 , a nastavi se s čitanjem drugog znaka u nizu. Opisani prijelaz prikazan je u prvom i drugom retku tablice 3.2. U prvom stupcu tablice 3.2 navedeno je stanje PA, u drugom stupcu je nepročitani dio ulaznog niza, u trećem stupcu su znakovi na stogu, a u četvrtom stupcu je funkcija prijelaza. U tablici su prikazani svi prijelazi uključujući i prijelaz kojim PA prihvata niz. U posljednjem retku tablice stog ostaje prazan i pročitani su svi znakovi ulaznog niza. Niz se prihvata, odnosno niz jest iz jezika $L(M_1)$.

Stanje	Nepročitani dio niza	Stog (vrh stoga je krajnje lijevi znak)	Funkcija prijelaza
q_1	0012100	K	$\delta(q_1, 0, K) = \{(q_1, NK)\}$
q_1	012100	NK	$\delta(q_1, 0, N) = \{(q_1, NN)\}$
q_1	12100	NNK	$\delta(q_1, 1, N) = \{(q_1, JN)\}$
q_1	2100	$JNNK$	$\delta(q_1, 2, J) = \{(q_2, J)\}$
q_2	100	$JNNK$	$\delta(q_2, 1, J) = \{(q_2, \varepsilon)\}$
q_2	00	NNK	$\delta(q_2, 0, N) = \{(q_2, \varepsilon)\}$
q_2	0	NK	$\delta(q_2, 0, N) = \{(q_2, \varepsilon)\}$
q_2		K	$\delta(q_2, \varepsilon, K) = \{(q_3, \varepsilon)\}$
q_3			$q_3 \in F \text{ i niz se prihvata}$

Tablica 3.2: Slijed prijelaza PA za niz 0012100

Ako bi isti PA M_1 prihvaćao jezik praznim stogom, onda je moguće lako provjeriti da je jezik koji se prihvata praznim stogom $N(M_1)$ jednak je jeziku koji PA M_1 prihvata prihvatljivim stanjem $L(M_1)$.

Konfiguracija PA

Prva tri stupca tablice 3.2 čine konfiguraciju PA. Konfiguracija se zadaje stanjem, nepročitanim dijelom ulaznog niza i znakovima koji se nalaze na stogu. Konfiguracija PA jest uređena trojka:

$$(q, w, \gamma),$$

gdje je q stanje, w je nepročitani dio ulaznog niza, a γ je sadržaj stoga (vrh stoga jest krajnje lijevi znak niza γ).

Neka je zadan PA $M=(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$. Konfiguracija PA M mijenja se iz $(q, aw, Z\alpha)$ u $(p, w, \beta\alpha)$ ako i samo ako skup $\delta(q, a, Z)$ sadrži (p, β) . Znak a jest ϵ ili jedan od ulaznih znakova skupa Σ .

Promjena konfiguracije formalno se zapisuje pomoću relacije \succ_M . PA M mijenja konfiguraciju:

$$(q, aw, Z\alpha) \succ_M (p, w, \beta\alpha) \text{ ako i samo ako je u skupu } \delta(q, a, Z) \text{ par } (p, \beta).$$

Oznaka M ispušta se ako se zna na koji PA se odnosi relacija \succ .

Promjena konfiguracije za niz 0012100 i PA zadan u primjeru 3.16 zapiše se na sljedeći način:

$$\begin{aligned} (q_1, 0012100, K) &\succ (q_1, 012100, NK) \succ (q_1, 12100, NNK) \succ (q_1, 2100, JNNK) \\ &\succ (q_2, 100, JNNK) \succ (q_2, 00, NNK) \succ (q_2, 0, NK) \succ (q_2, \epsilon, K) \succ (q_2, \epsilon, \epsilon). \end{aligned}$$

*

Znakom \succ označava se refleksivno i tranzitivno okruženje relacije \succ . Promjena konfiguracije PA za niz 0012100 zapiše se na sljedeći način:

$$(q_1, 0012100, K) \succ^* (q_2, \epsilon, \epsilon).$$

i

Prelazak iz konfiguracije J u konfiguraciju K primjenom i prijelaza označava se izrazom $J \succ^i K$. Za gornji niz 0012100 funkcija prijelaza δ primjeni se točno 8 puta:

$$(q_1, 0012100, K) \succ^8 (q_2, \epsilon, \epsilon).$$

Prihvatanje jezika zadanim PA

Prihvatanje jezika potisnim automatom definira se na jedan od dva moguća načina:

PA $M=(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ prihvata *prihvatljim stanjem* jezik:

$$a) \quad L(M)=\{w \mid (q_0, w, Z_0) \succ^* (p, \epsilon, \gamma) \text{ za neko stanje } p \in F \text{ i } \gamma \in \Gamma^*\}$$

PA $M=(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ prihvata *praznim stogom* jezik:

$$b) \quad N(M)=\{w \mid (q_0, w, Z_0) \succ^* (p, \epsilon, \epsilon) \text{ za neko stanje } p \in Q\}$$

Da bi se niz w prihvatio, stanje p u definiciji (a) mora biti iz skupa prihvatljivih stanja F . U definiciji (b) ne provjerava se da li je stanje p iz skupa F . Isprazni li se u potpunosti stog nakon pročitanih svih znakova niza w , niz se prihvata.

Primjer 3.17. Zadan je PA $M_2 = (\{q_1, q_2\}, \{0, 1\}, \{N, J, K\}, \delta, q_1, K, \emptyset)$ koji prihvata jezik $N(M_2)$ praznim stogom sa sljedećim prijelazima:

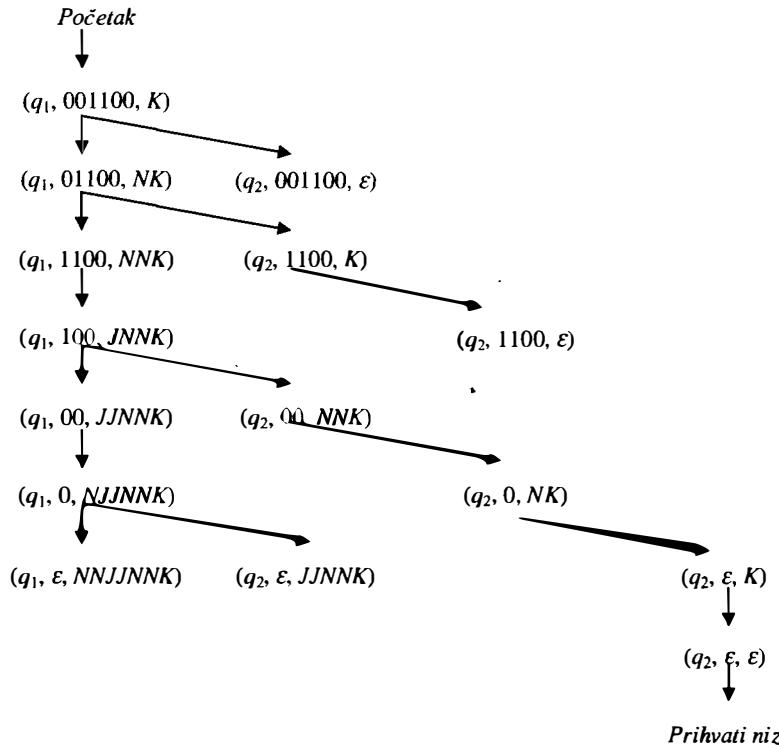
- | | |
|---|---|
| 1) $\delta(q_1, 0, K) = \{(q_1, NK)\}$ | 2) $\delta(q_1, 1, K) = \{(q_1, JK)\}$ |
| 3) $\delta(q_1, 0, N) = \{(q_1, NN), (q_2, \epsilon)\}$ | 4) $\delta(q_1, 1, N) = \{(q_1, JN)\}$ |
| 5) $\delta(q_1, 0, J) = \{(q_1, NJ)\}$ | 6) $\delta(q_1, 1, J) = \{(q_1, JJ), (q_2, \epsilon)\}$ |
| 7) $\delta(q_2, 0, N) = \{(q_2, \epsilon)\}$ | 8) $\delta(q_2, 1, J) = \{(q_2, \epsilon)\}$ |
| 9) $\delta(q_1, \epsilon, K) = \{(q_2, \epsilon)\}$ | |
| 10) $\delta(q_2, \epsilon, K) = \{(q_2, \epsilon)\}$ | |

Za razliku od PA u primjeru 3.16 koji jest deterministički, PA zadan u ovom primjeru nije deterministički. U prijelazu (3) za stanje q_1 , ulazni znak 0 i znak stoga N postoje dvije mogućnosti. PA stavlja znak N na vrh stoga i ostaje u stanju q_1 , ili prelazi u stanje q_2 i uzima znak N s vrha stoga. Sličan nedeterminizam zadan je prijelazom (6). Malo je teže uočiti nedeterminizam koji je posljedica prijelaza (1), (2) i (9). Za stanje q_1 , ulazne znakove 0 i 1, te vrh stoga K postoje dvije mogućnosti: primjena prijelaza (1) i (2), ili primjena ϵ -prijelaza (9).

PA prihvata jezik $N(M_2) = \{ww^R \mid w \text{ jest niz nula i jedinica } (0+1)^*\}$, a niz w^R je niz w napisan obrnutim redoslijedom}. Niz jezika $N(M_2)$ nema u sredini znak 2, kao što to ima niz jezika $N(M_1)$ u primjeru 3.16. Znak 2 u nizu $w2w^R$ omogućuje donošenje odluke o prekidu stavljanja znakova N i J na stog i o pokretanju usporedbe ostatka niza sa sadržajem stoga. Budući da niz jezika $N(M_2)$ nema znak 2 u sredini, PA M_2 ne može jednoznačno donijeti odluku o zaustavljanju procesa stavljanja znakova N i J na stog. Zato funkcija prijelaza PA M_2 daje više mogućnosti za prijelaz (3) i prijelaz (6). Isprazni li stog barem jedan od prijelaza čitanjem svih znakova ulaza, niz se prihvata.

Slijed prijelaza za niz 001100 i zadani PA prikazan je na slici 3.20. Niz 001100 je oblika ww^R , gdje je $w=001$, te ga PA M_2 prihvata.

Nedeterminizam PA sličan je nedeterminizmu NKA: postoji li mogućnost izbora više prijelaza, započinje istodobno s radom više determinističkih PA. Uspije li barem jedan deterministički PA isprazniti stog čitanjem svih znakova ulazog niza, niz se prihvata.



Slika 3.20: Prihvatanje niza 001100

Deterministički PA (DPA)

PA $M=(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ jest deterministički PA (DPA) ako i samo ako su ispunjena oba sljedeća uvjeta:

- i) Ako je $\delta(q, \varepsilon, Z)$ neprazni skup, onda za bilo koje stanje q iz Q , za bilo koji znak stoga Z iz Γ i za bilo koji ulazni znak a iz Σ mora vrijediti da je $\delta(q, a, Z)$ prazni skup.
- ii) Skup $\delta(q, a, Z)$ može imati najviše jedan element, i to za bilo koje stanje q iz Q , za bilo koji znak stoga Z iz Γ i za bilo koji ulazni znak a iz $\Sigma \cup \{\varepsilon\}$.

Uvjet (i) ne dozvoljava mogućnost izbora između ε -prijelaza i prijelaza koji koristi ulazni znak. U primjeru 3.17 prijelazi (1) i (9) imaju za posljedicu opisani nedeterminizam. Iz konfiguracije $(q_1, 001100, K)$ prelazi se u konfiguraciju $(q_2, 001100, \varepsilon)$ primjenom ε -prijelaza $\delta(q_1, \varepsilon, K)=\{(q_2, \varepsilon)\}$, ili u konfiguraciju $(q_1, 01100, NK)$ primjenom prijelaza $\delta(q_1, 0, K)=\{(q_1, NK)\}$. Sličan nedeterminizam vrijedi za prijelaze (2) i (9).

Uvjet (ii) ne dopušta da u skupu $\delta(q, a, Z)$ ima više od jednog elementa. Prijelazi (3) i (6) su primjeri nedeterminističkih prijelaza. Skupovi $\delta(q_1, 0, N)$ i $\delta(q_1, 1, J)$ imaju po dva elementa. Na slici 3.20 prikazano je više nedeterminističkih prijelaza. Na primjer, iz konfiguracije $(q_1, 01100, NK)$ prelazi se u konfiguraciju $(q_1, 1100, NNK)$ ili u konfiguraciju $(q_2, 1100, K)$.

Za razliku od NKA koji prihvata istu klasu jezika kao i DKA, nedeterministički PA prihvata širu klasu jezika od determinističkog PA. U primjeru 3.17 pokazano je da postoji nedeterministički PA M_2 koji prihvata jezik $N(M_2)=\{ww^R\}$. Međutim, za jezik $N(M_2)$ nije moguće izgraditi deterministički PA. U dalnjem tekstu pod pojmom PA podrazumijeva se nedeterministički PA. Govori li se o determinističkom PA, to se posebno nagrasi i označi kao DPA.

3.2.3 Potisni automat i kontekstno neovisna gramatika

Dva PA su istovjetna ako i samo ako prihvataju isti jezik. U ovom odjeljku pokazana je istovjetnost PA koji prihvata jezik praznim stogom i PA koji prihvata jezik prihvatljivim stanjem. Također je pokazano da potisni automati prihvataju klasu kontekstno neovisnih jezika.

Konstrukcija PA koji prihvata praznim stogom iz zadanog PA koji prihvata prihvatljivim stanjem

Neka PA:

$$M_2=(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

prihvata jezik $L(M_2)$ prihvatljivim stanjem. Želi se izgraditi istovjetni PA M_1 koji prihvata praznim stogom. Konstrukcija PA M_1 zasniva se na simulaciji PA M_2 . Uđe li tijekom simulacije PA M_2 u jedno od prihvatljivih stanja, PA M_1 isprazni svoj stog. Da bi se omogućilo da PA M_1 isprazni stog ako i samo ako PA M_2 uđe u jedno od prihvatljivih stanja, uvodi se dodatni znak stoga X_0 . Na početku rada PA M_1 stavi na dno stoga znak X_0 . PA M_2 svojim prijelazima ne može uzeti sa stoga znak X_0 . Isprazni li PA M_2 stog, a ne uđe u jedno od prihvatljivih stanja, na dnu stoga ostaje znak X_0 koji onemogućava da PA M_1 prihvati niz.

Konstruira se PA M_1 :

$$M_1=(Q \cup \{q'_0, q_e\}, \Sigma, \Gamma \cup \{X_0\}, \delta', q'_0, X_0, \emptyset),$$

Funkcija prijelaza PA M_1 gradi se na sljedeći način:

1) $\delta'(q_0, \varepsilon, X_0) = \{(q_0, Z_0 X_0)\}.$

Na početku rada PA M_1 prelazi iz svoje početne konfiguracije u početnu konfiguraciju PA M_2 . PA M_1 ostavlja znak X_0 na dnu stoga.

2) U skup $\delta'(q, a, Z)$ stave se svi elementi skupa $\delta(q, a, Z)$.

Skup $\delta'(q, a, Z)$ računa se za sva stanja q iz Q , za sve znakove a iz $\Sigma \cup \{\varepsilon\}$ i za sve znakove stoga Z iz Γ . Ovi prijelazi omogućuju simuliranje rada PA M_2 .

3) U skup $\delta'(q, \varepsilon, Z)$ dodaje se ε -prijelaz $(q_e, \varepsilon), q \in F$.

ε -prijelazi dodaju se za sva stanja q iz skupa prihvatljivih stanja F i za sve znakove stoga Z iz $\Gamma \cup \{X_0\}$. Uđe li PA M_2 u jedno od prihvatljivih stanja, skup prijelaza proširuje se ε -prijelazom u stanje q_e . Istodobno se s vrha stoga uzme jedan znak.

4) U skup $\delta'(q_e, \varepsilon, Z)$ dodaje se ε -prijelaz (q_e, ε) .

ε -prijelazi dodaju se za sve znakove stoga Z iz $\Gamma \cup \{X_0\}$. ε -prijelazi u stanju q_e prazne čitav stog, uključujući i znak X_0 .

Dokaz istovjetnosti PA M_1 i PA M_2 izvodi se u dva dijela. U prvom dijelu pokazuje se da PA M_1 prihvata niz x ako ga prihvata PA M_2 , a u drugom dijelu da PA M_2 prihvata niz x ako ga prihvata PA M_1 .

Prepostavlja se da PA M_2 prihvata niz x prihvatljivim stanjem:

$$(q_0, x, Z_0) \xrightarrow[M_2]{*} (q, \varepsilon, A\gamma), \text{ gdje je } q \in F, A \in \Gamma \text{ i } \gamma \in \Gamma^*.$$

Na temelju koraka (1) konstrukcije PA M_1 , za početnu konfiguraciju PA M_1 vrijedi:

$$(q_0', x, X_0) \xrightarrow[M_1]{*} (q_0, x, Z_0 X_0).$$

Na temelju koraka (2) konstrukcije PA M_1 , svi prijelazi PA M_2 su ujedno i prijelaz PA M_1 :

$$(q_0, x, Z_0 X_0) \xrightarrow[M_1]{*} (q, \varepsilon, A\gamma X_0).$$

Budući da je $q \in F$, korak (3) omogućuje prijelaz u stanje q_e , a s vrha stoga se uzima jedan znak:

$$(q, \varepsilon, A\gamma X_0) \xrightarrow[M_1]{*} (q_e, \varepsilon, \gamma X_0).$$

Korak (4) konstrukcije PA M_1 omogućuje pražnjenje stoga:

$$(q_e, \varepsilon, \gamma X_0) \xrightarrow[M_1]{*} (q_e, \varepsilon, \varepsilon).$$

Prihvata li PA M_2 niz x prihvatljivim stanjem, PA M_1 prelazi u konfiguraciju:

$$(q_0', x, X_0) \xrightarrow[M_1]{*} (q_e, \varepsilon, \varepsilon)$$

i prihvati niz x praznim stogom.

U drugom dijelu potrebno je pokazati da PA M_2 prihvata niz x prihvatljivim stanjem ako PA M_1 prihvata niz x praznim stogom. Slijed prijelaza PA M_1 prilikom prihvaćanja niza x sastoji se od prijelaza zadanog u koraku (1) konstrukcije PA M_1 , zatim slijeda prijelaza izgrađenih u koraku (2) koji simuliraju rad PA M_2 , i na kraju slijed prijelaza koji prazni stog. Na temelju koraka (3) i (4) stog se prazni ako i samo ako u

konfiguraciji $(q, \varepsilon, \gamma X_0)$ stanje q je prihvatljivo stanje. Prema tome, PA M_2 prihvata niz prihvatljivim stanjem ako i samo ako PA M_1 prihvata niz praznim stogom.

Primjer 3.18. Zadan je PA $M_2 = (\{q_1, q_2\}, \{0, 1\}, \{N, K\}, \delta, q_1, K, \{q_2\})$ sa sljedećim prijelazima:

- 1) $\delta(q_1, 0, K) = \{(q_1, NK)\}$
- 2) $\delta(q_1, 0, N) = \{(q_1, NN)\}$
- 3) $\delta(q_1, 1, N) = \{(q_2, \varepsilon)\}$
- 4) $\delta(q_2, 1, N) = \{(q_2, \varepsilon)\}$

PA M_2 prihvata jezik $L(M_2) = \{0^n 1^m \mid n \geq 1, m \geq 1, m \leq n\}$ prihvatljivim stanjem q_2 . Prijelazi (1) i (2) stavljaju po jedan znak N na vrh stoga za svaki pročitan znak 0. Pojavom prvog znaka 1, PA prijelazom (3) mijenja stanje u prihvatljivo stanja q_2 . Prijelaz (4) provjerava da znakova 1 nema više od znakova 0.

Istovjetni PA $M_1 = (\{q_1, q_2, q'_0, q_e\}, \{0, 1\}, \{N, K, X_0\}, \delta', q'_0, X_0, \emptyset)$ koji prihvata praznim stogom konstruiru se na sljedeći način. Korak konstrukcije (1) daje prijelaz u početnu konfiguraciju PA M_2 :

$$0) \quad \delta'(q'_0, \varepsilon, X_0) = \{(q_1, KX_0)\}.$$

U koraku (2) preuzimaju se svi prijelazi PA M_2 :

- 1) $\delta'(q_1, 0, K) = \{(q_1, NK)\}$
- 2) $\delta'(q_1, 0, N) = \{(q_1, NN)\}$
- 3) $\delta'(q_1, 1, N) = \{(q_2, \varepsilon)\}$
- 4) $\delta'(q_2, 1, N) = \{(q_2, \varepsilon)\}$

Korak (3) dodaje ε -prijelaze u stanje q_e :

- 5) $\delta'(q_2, \varepsilon, N) = \{(q_e, \varepsilon)\}$
- 6) $\delta'(q_2, \varepsilon, K) = \{(q_e, \varepsilon)\}$
- 7) $\delta'(q_2, \varepsilon, X_0) = \{(q_e, \varepsilon)\}$

Korak (4) dodaje ε -prijelaze koji prazne stog:

- 8) $\delta'(q_e, \varepsilon, N) = \{(q_e, \varepsilon)\}$
- 9) $\delta'(q_e, \varepsilon, K) = \{(q_e, \varepsilon)\}$
- 10) $\delta'(q_e, \varepsilon, X_0) = \{(q_e, \varepsilon)\}.$

Slijed prijelaza PA M_1 za niz 00011 je:

$$\begin{aligned} (q'_0, 00011, X_0) &\succ (q_1, 00011, KX_0) \succ (q_1, 0011, NKX_0) \succ (q_1, 011, NNKX_0) \succ \\ &(q_1, 11, NNNKX_0) \succ (q_2, 1, NNKX_0) \succ (q_2, \varepsilon, NKX_0) \succ (q_e, \varepsilon, KX_0) \succ \\ &(q_e, \varepsilon, X_0) \succ (q_e, \varepsilon, \varepsilon). \end{aligned}$$

PA M_1 prihvata niz 00011, jer su pročitani svi znakovi niza, a stog je prazan.

Slijed prijelaza PA M_2 za niz 00011 je:

$$\begin{aligned} (q_1, 00011, K) &\succ (q_1, 0011, NK) \succ (q_1, 011, NNK) \succ \\ &(q_1, 11, NNNK) \succ (q_2, 1, NNK) \succ (q_2, \varepsilon, NK). \end{aligned}$$

PA M_2 prihvata niz 00011, jer su pročitani svi znakovi niza, a stanje q_2 jest prihvatljivo stanje.

Konstrukcija PA koji prihvata prihvatljivim stanjem iz zadanog PA koji prihvata praznim stogom

Neka PA:

$$M_1 = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \emptyset)$$

prihvata jezik *praznim stogom*. Želi se izgraditi istovjetni PA M_2 koji prihvata *prihvatljivim stanjem*. Izgrađeni PA M_2 simulira rad PA M_1 , te prelazi u prihvatljivo stanje ako i samo ako PA M_1 isprazni svoj stog.

Konstruira se PA M_2 :

$$M_2 = (Q \cup \{q'_0, q_f\}, \Sigma, \Gamma \cup \{X_0\}, \delta', q'_0, X_0, \{q_f\})$$

Funkcija prijelaza PA M_2 gradi se na sljedeći način:

$$1) \quad \delta'(q'_0, \epsilon, X_0) = \{(q_0, Z_0 X_0)\}.$$

Na početku rada PA M_2 prelazi iz svoje početne konfiguracije u početnu konfiguraciju PA M_1 . Na dno stoga stavlja se znak X_0 . Pojavi li se tijekom simulacije na vrhu stoga znak X_0 , to je znak da je PA M_1 ispraznio svoj stog.

$$2) \quad \text{U skup } \delta'(q, a, Z) \text{ stave se svi elementi skupa } \delta(q, a, Z).$$

Skup $\delta'(q, a, Z)$ računa se za sva stanja q iz Q , za sve znakove a iz $\Sigma \cup \{\epsilon\}$ i za sve znakove stoga Z iz Γ . Ovi prijelazi omogućuju simuliranje rada PA M_1 .

$$3) \quad \text{U skup } \delta'(q, \epsilon, X_0) \text{ dodaje se } \epsilon\text{-prijelaz } (q_f, \epsilon).$$

ϵ -prijelazi dodaju se za sva stanja q iz Q . Pojavi li se na vrhu stoga znak X_0 , to je znak da je PA M_1 ispraznio svoj stog. PA M_2 prelazi u prihvatljivo stanje q_f . Prihvata li PA M_1 niz praznim stogom, prelazak u stanje q_f omogućuje prihvatanje niza prihvatljivim stanjem.

Na temelju koraka (1) do (3) konstrukcije PA M_2 dobije se slijed prijelaza:

$$(q'_0, x, X_0) \xrightarrow[M_2]{*} (q_0 x, Z_0 X_0) \xrightarrow[M_2]{*} (q, \epsilon, X_0) \xrightarrow[M_2]{*} (q_f, \epsilon, \epsilon),$$

gdje je prvi prijelaz definiran korakom (1) konstrukcije PA M_2 , slijed prijelaza $(q_0 x, Z_0 X_0) \xrightarrow[M_2]{*} (q, \epsilon, X_0)$ * simuliра prijelaze $(q_0 x, Z_0) \xrightarrow[M_1]{*} (q, \epsilon, \epsilon)$ PA M_1 , i posljednji prijelaz definiran je korakom (3). Prikazani slijed prijelaza pokazuje da PA M_2 prihvata niz x prihvatljivim stanjem ako i samo ako PA M_1 prihvata niz x praznim stogom.

Primjer 3.19. PA $M_1 = (\{q_1, q_2\}, \{0, 1\}, \{N, J, K\}, \delta, q_1, K, \emptyset)$ prihvata praznim stogom i ima sljedeće prijelaze:

- | | |
|---|---|
| 1) $\delta(q_1, 0, K) = \{(q_1, NK)\}$ | 2) $\delta(q_1, 1, K) = \{(q_1, JK)\}$ |
| 3) $\delta(q_1, 0, N) = \{(q_1, NN), (q_2, \epsilon)\}$ | 4) $\delta(q_1, 1, N) = \{(q_1, JN)\}$ |
| 5) $\delta(q_1, 0, J) = \{(q_1, NJ)\}$ | 6) $\delta(q_1, 1, J) = \{(q_1, JJ), (q_2, \epsilon)\}$ |
| 7) $\delta(q_2, 0, N) = \{(q_2, \epsilon)\}$ | 8) $\delta(q_2, 1, J) = \{(q_2, \epsilon)\}$ |
| 9) $\delta(q_1, \epsilon, K) = \{(q_2, \epsilon)\}$ | |
| 10) $\delta(q_2, \epsilon, K) = \{(q_2, \epsilon)\}$ | |

PA M_1 prihvata jezik $N(M_1) = \{ww^R \mid w \text{ jest niz nula i jedinica } (0+1)^*\}$, a niz w^R je niz w napisan obrnutim redoslijedom}.

Na temelju koraka (1) do (3) gradi se PA $M_2 = (\{q_1, q_2, q'_0, q_f\}, \{0, 1\}, \{N, J, K, X_0\}, \delta', q'_0, X_0, \{q_f\})$ koji prihvata jezik prihvatljivim stanjem. Korak (1) daje prijelaz:

$$0) \quad \delta'(q'_0, \epsilon, X_0) = \{(q_1, KX_0)\}.$$

U koraku (2) preuzimaju se svi prijelazi PA M_1 :

- | | |
|--|--|
| 1) $\delta'(q_1, 0, K) = \{(q_1, NK)\}$ | 2) $\delta'(q_1, 1, K) = \{(q_1, JK)\}$ |
| 3) $\delta'(q_1, 0, N) = \{(q_1, NN), (q_2, \epsilon)\}$ | 4) $\delta'(q_1, 1, N) = \{(q_1, JN)\}$ |
| 5) $\delta'(q_1, 0, J) = \{(q_1, NJ)\}$ | 6) $\delta'(q_1, 1, J) = \{(q_1, JJ), (q_2, \epsilon)\}$ |
| 7) $\delta'(q_2, 0, N) = \{(q_2, \epsilon)\}$ | 8) $\delta'(q_2, 1, J) = \{(q_2, \epsilon)\}$ |
| 9) $\delta'(q_1, \epsilon, K) = \{(q_2, \epsilon)\}$ | |
| 10) $\delta'(q_2, \epsilon, K) = \{(q_2, \epsilon)\}$ | |

Korak (3) dodaje ϵ -prijelaze u prihvatljivo stanje q_f :

- 11) $\delta'(q_1, \epsilon, X_0) = \{(q_f, \epsilon)\}$
- 12) $\delta'(q_2, \epsilon, X_0) = \{(q_f, \epsilon)\}.$

Slijed prijelaza PA M_2 za niz 001100 je:

$$(q_0', 001100, X_0) \succ (q_1, 001100, KX_0) \succ (q_1, 01100, NKX_0) \succ (q_1, 1100, NNKX_0) \succ \\ (q_1, 100, JNNKX_0) \succ (q_2, 00, NNKX_0) \succ (q_2, 0, NKX_0) \succ (q_2, \epsilon, KX_0) \succ \\ (q_2, \epsilon, X_0) \succ (q_f, \epsilon, \epsilon).$$

Niz se prihvaća, jer je PA M_2 u prihvatljivom stanju $q_f \in F$ nakon pročitanih svih znakova niza.

Slijed prijelaza PA M_1 za niz 001100 je:

$$(q_1, 001100, K) \succ (q_1, 01100, NK) \succ (q_1, 1100, NNK) \succ \\ (q_1, 100, JNNK) \succ (q_2, 00, NNK) \succ (q_2, 0, NK) \succ (q_2, \epsilon, K) \succ (q_2, \epsilon, \epsilon).$$

Niz se prihvaća, jer je PA M_1 ispraznio svoj stog nakon pročitanih svih znakova niza.

Konstrukcija PA koji prihvaća praznim stogom jezik zadan kontekstno neovisnom gramatikom

Istovjetnost potisnih automata koji prihvaćaju jezike na različite načine omogućuje da se u pokazivanju istovjetnosti PA i kontekstno neovisne gramatike koristi samo jedan od dva modela PA. U nastavku odjeljka pokazuje se istovjetnost kontekstno neovisne gramatike i PA koji prihvaća praznim stogom.

Nedeterministički PA prepoznaju klasu kontekstno neovisnih jezika. Za bilo koji kontekstno neovisni jezik L postoji nedeterministički PA M koji prihvaća jezik praznim stogom $N(M)=L$. Prepostavlja se da prazni niz ϵ nije element jezika L . Kontekstno neovisni jezik L zadani je gramatikom:

$$G=(V, T, P, S)$$

koja ima produkcije u Greibachovom normalnom obliku. Sve produkcije su oblika $A \rightarrow a\beta$, gdje je $A \in V$, $a \in T$, $\beta \in V^*$. Konstruira se PA:

$$M=((q), \Sigma, \Gamma, \delta, q, S, \emptyset)$$

na sljedeći način:

- a) PA M ima samo jedno stanje q koje je ujedno i početno stanje;
- b) skup ulaznih znakova Σ jednak je skupu završnih znakova T , $\Sigma=T$;
- c) skup znakova stoga Γ jednak je skupu nezavršnih znakova gramatike V , $\Gamma=V$;
- d) početni znak stoga jest početni nezavršni znak gramatike S ;
- e) skup prihvatljivih stanja jest prazni skup, $F=\emptyset$;
- f) PA M prihvaća praznim stogom.

Funkcije prijelaza zadaju se na sljedeći način:

$$\delta(q, a, A) \text{ sadrži } (q, \gamma) \text{ ako i samo ako postoji produkcija } A \rightarrow a\gamma.$$

PA M simulira postupak generiranja niza zamjenom krajnje lijevog nezavršnog znaka. Budući da su sve produkcije oblika $A \rightarrow a\beta$, gdje $A \in V$, $a \in T$, $\beta \in V^*$ i budući da se primjenjuje postupak generiranja niza zamjenom krajnje lijevog nezavršnog znaka, bilo koji generirani međuniz je oblika:

$$x\alpha, \text{ gdje je } x \in T^* \text{ i } \alpha \in V^*.$$

Prefiks x sastoji se isključivo od završnih znakova gramatike, dok se sufiks α sastoji isključivo od nezavršnih znakova gramatike.

Nakon što PA M pročita niz ulaznih znakova x , na stogu je niz nezavršnih znakova gramatike α . Matematičkom indukcijom, zasnovanom na broju koraka u slijedu prijelaza, moguće je dokazati sljedeće:

i) PA M prolazi slijedom prijelaza $(q, x, S) \xsucc_M^* (q, \epsilon, \alpha)$ ako i samo ako gramatika

generira niz $x\alpha$ postupkom zamjene krajnje lijevog nezavršnog znaka $S \xRightarrow[G]{*} x\alpha$.

Uvrsti li se u (i) da je α prazni niz, dobije se:

$(q, x, S) \xsucc_M^* (q, \epsilon, \epsilon)$ ako i samo ako $S \xRightarrow[G]{*} x$,

odnosno PA M prihvata niz x praznim stogom ako i samo ako gramatika G generira niz x .

Primjer 3.20. Zadana je gramatika:

$$G = (\{S, A\}, \{a, b\}, \{S \rightarrow aAA, A \rightarrow aS \mid bS \mid a\}, S).$$

Na temelju pravila (a) do (f) izgradi se PA M :

$$M = (\{q\}, \{a, b\}, \{S, A\}, \delta, q, S, \emptyset)$$

sa funkcijama prijelaza:

$\delta(q, a, S) = \{(q, AA)\}$ na temelju produkcije $S \rightarrow aAA$;

$\delta(q, a, A) = \{(q, S), (q, \epsilon)\}$ na temelju produkcija $A \rightarrow aS$ i $A \rightarrow a$;

$\delta(q, b, A) = \{(q, S)\}$ na temelju produkcija $A \rightarrow bS$.

Gramatika G generira niz $abaaaa$:

$$S \Rightarrow aAA \Rightarrow abSA \Rightarrow abaAAA \Rightarrow abaaAA \Rightarrow abaaaA \Rightarrow abaaaa.$$

Konstruirani PA M prihvata niz $abaaaa$ praznim stogom:

$$\begin{aligned} (q, abaaaa, S) &\succ (q, baadaa, AA) \succ (q, aaaa, SA) \succ (q, aaa, AAA) \succ \\ &(q, aa, AA) \succ (q, a, A) \succ (q, \epsilon, \epsilon). \end{aligned}$$

Konstrukcija kontekstno neovisne gramatike za jezik koji se prihvata praznim stogom zadanog PA

Za zadani PA:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

konstruira se kontekstno neovisna gramatika:

$$G = (V, T, P, S).$$

Nezavršni znakovi gramatike označeni su zagrada $[q, A, p] \in V$. U oznaci nezavršnog znaka, q i p su stanja skupa Q , a A je znak stoga iz skupa Γ . U skup nezavršnih znakova doda je se početni nezavršni znak S . Skup produkcija gradi se na sljedeći način:

- 1) Za početno stanje q_0 , početni znak stoga Z_0 i sva stanja q iz Q grade se produkcije:

$$S \rightarrow [q_0, Z_0, q].$$

Ako je kardinalni broj skupa stanja Q jednak n , onda je broj produkcija $S \rightarrow [q_0, Z_0, q]$ jednak također n .

- 2) Ako skup $\delta(q, a, A)$ sadrži:

$$(q_1, B_1 B_2 \dots B_m),$$

onda se grade produkcije:

$$[q, A, q_{m+1}] \rightarrow a[q_1, B_1, q_2][q_2, B_2, q_3] \dots [q_m, B_m, q_{m+1}]$$

gdje su znakovi $q, A, a, q_1, B_1, B_2, \dots$ i B_m određeni datom funkcijom prijelaza PA, a za znakove q_2, q_3, \dots i q_{m+1} uzimaju se sve moguće kombinacije svih stanja iz skupa Q . Ako je kardinalni broj skupa stanja Q jednak n , a budući da se uzimaju sve moguće kombinacije za niz q_2, q_3, \dots i q_{m+1} , za svaku funkciju prijelaza gradi se n^{m-1} produkcija.

Ako $\delta(q, a, A)$ sadrži (q_1, ε) , onda se gradi produkcija $[q, A, q_1] \rightarrow a$.

Gramatika G postupkom generiranja niza zamjenom krajnje lijevog nezavršnog znaka simulira rad PA M . Niz nezavršnih znakova u generiranom međunizu jednak je nizu znakova na stogu PA M . Gramatika generira niz x počev od nezavršnog znaka $[q, A, p]$ ako i samo ako PA M čitanjem niza x izbriše znak A sa stoga i promjeni stanje iz q u p .

Dokaz da je $L(G)=N(M)$ temelji se na činjenici da je:

$$i) \quad [q, A, p] \xrightarrow[G]{*} x \text{ ako i samo ako } (q, x, A) \succ_M^* (p, \varepsilon, \varepsilon)$$

Činjenica (i) dokazuje se matematičkom indukcijom.

Primjer 3.21. Zadan je PA $M=(\{q_1, q_2\}, \{0, 1\}, \{N, K\}, \delta, q_1, K, \{\})$ sa sljedećim prijelazima:

- | | |
|---|---|
| 1) $\delta(q_1, 0, K) = \{(q_1, NK)\}$ | 2) $\delta(q_1, 0, N) = \{(q_1, NN)\}$ |
| 3) $\delta(q_1, 1, N) = \{(q_2, \varepsilon)\}$ | 4) $\delta(q_2, 1, N) = \{(q_2, \varepsilon)\}$ |
| 5) $\delta(q_2, \varepsilon, N) = \{(q_2, \varepsilon)\}$ | 6) $\delta(q_2, \varepsilon, K) = \{(q_2, \varepsilon)\}$ |

PA M prihvata jezik $N(M)=\{0^n 1^m \mid n \geq 1, m \geq 1, m \leq n\}$ praznim stogom. Na temelju koraka (1) i (2) konstruira se gramatika $G=(V, \{0, 1\}, P, S)$. Skup nezavršnih znakova V ima sljedeće znakove:

$$V=\{S, [q_1, N, q_1], [q_1, N, q_2], [q_2, N, q_1], [q_2, N, q_2], [q_1, K, q_1], [q_1, K, q_2], [q_2, K, q_1], [q_2, K, q_2]\}$$

Gradnja produkcija započinje od početnog nezavršnog znaka S . Ostale produkcije grade se samo za dohvatljive nezavršne znakove. Redoslijed gradnje novih produkcija zasniva se na postupku traženja dohvatljivih znakova. Svaki put kada se u listu dohvatljivih znakova upiše novi nezavršni znak, onda se grade produkcije za taj znak.

Budući da je početni znak stoga K , početno stanje jest q_1 i u skupu stanja su dva stanja q_1 i q_2 , korak konstrukcije (1) daje sljedeće produkcije za početni nezavršni znak gramatike S :

$$i) \quad S \rightarrow [q_1, K, q_1] \mid [q_1, K, q_2].$$

Nastavlja se s gradnjom produkcija za nezavršni znak $[q_1, K, q_1]$. Na temelju prijelaza $\delta(q_1, 0, K)=\{(q_1, NK)\}$ i pravila (2), grade se dvije produkcije za $[q_1, K, q_1]$:

$$ii) \quad [q_1, K, q_1] \rightarrow 0 [q_1, N, q_1] [q_1, K, q_1] \mid 0 [q_1, N, q_2] [q_2, K, q_1].$$

Na temelju prijelaza $\delta(q_1, 0, K)=\{(q_1, NK)\}$, grade se dvije produkcije za $[q_1, K, q_2]$:

$$iii) \quad [q_1, K, q_2] \rightarrow 0 [q_1, N, q_1] [q_1, K, q_2] \mid 0 [q_1, N, q_2] [q_2, K, q_2].$$

Na temelju prijelaza $\delta(q_1, 0, N)=\{(q_1, NN)\}$, grade se sljedeće produkcije za znakove $[q_1, N, q_1]$ i $[q_1, N, q_2]$:

$$iv) \quad \begin{aligned} [q_1, N, q_1] &\rightarrow 0 [q_1, N, q_1] [q_1, N, q_1] \mid 0 [q_1, N, q_2] [q_2, N, q_1] \\ [q_1, N, q_2] &\rightarrow 0 [q_1, N, q_1] [q_1, N, q_2] \mid 0 [q_1, N, q_2] [q_2, N, q_2]. \end{aligned}$$

Na temelju prijelazi (3) do (6) grade se produkcije:

$$v) \quad \begin{aligned} [q_1, N, q_2] &\rightarrow 1, \quad \text{jer je } \delta(q_1, 1, N)=\{(q_2, \epsilon)\}, \\ [q_2, K, q_2] &\rightarrow \epsilon, \quad \text{jer je } \delta(q_2, \epsilon, K)=\{(q_2, \epsilon)\}, \\ [q_2, N, q_2] &\rightarrow \epsilon, \quad \text{jer je } \delta(q_2, \epsilon, N)=\{(q_2, \epsilon)\}, \\ [q_2, N, q_2] &\rightarrow 1, \quad \text{jer je } \delta(q_2, 1, N)=\{(q_2, \epsilon)\}. \end{aligned}$$

Nema prijelaza PA M iz stanja q_2 u stanje q_1 i ne grade se produkcije za nezavršne znakove $[q_2, N, q_1]$ i $[q_2, K, q_1]$. Budući da su nezavršni znakovi $[q_2, N, q_1]$ i $[q_2, K, q_1]$ mrtvi, na temelju produkcija (iv) i (ii) mrtvi su i znakovi $[q_1, N, q_1]$ i $[q_1, K, q_1]$. Odbacuju se sve produkcije u kojima su mrtvi znakovi $[q_2, N, q_1]$, $[q_2, K, q_1]$, $[q_1, N, q_1]$ i $[q_1, K, q_1]$. Naposlijetku, potrebno je provjeriti da li nakon što se odbace svi mrtvi znakovi ima nedohvatljivih znakova, kako bi se iz gramatike odbacili svi nekorisni znakovi. Gramatika koja prihvata jezik $L(G)=N(M)$ ima sljedeće produkcije:

$$\begin{aligned} S &\rightarrow [q_1, K, q_2], \\ [q_1, K, q_2] &\rightarrow 0 [q_1, N, q_2] [q_2, K, q_2], \\ [q_1, N, q_2] &\rightarrow 0 [q_1, N, q_2] [q_2, N, q_2], \\ [q_1, N, q_2] &\rightarrow 1, \\ [q_2, K, q_2] &\rightarrow \epsilon, \\ [q_2, N, q_2] &\rightarrow \epsilon, \\ [q_2, N, q_2] &\rightarrow 1. \end{aligned}$$

Gramatika G generira niz 00011 na sljedeći način:

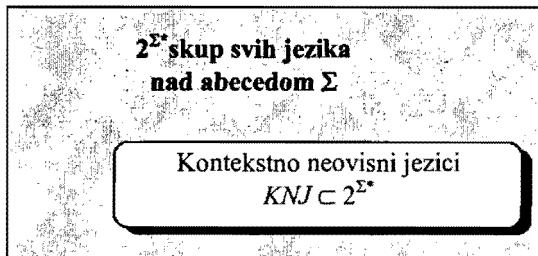
$$\begin{aligned} S \Rightarrow [q_1, K, q_2] &\Rightarrow 0 [q_1, N, q_2] [q_2, K, q_2] \Rightarrow 0' 0 [q_1, N, q_2] [q_2, N, q_2] [q_2, K, q_2] \Rightarrow \\ &\Rightarrow 0 0 0 [q_1, N, q_2] [q_2, N, q_2] [q_2, N, q_2] [q_2, K, q_2] \Rightarrow 0 0 0 1 [q_2, N, q_2] [q_2, N, q_2] [q_2, K, q_2] \Rightarrow \\ &\Rightarrow 0 0 0 1 1 [q_2, N, q_2] [q_2, K, q_2] \Rightarrow 0 0 0 1 1 \epsilon [q_2, K, q_2] \Rightarrow 0 0 0 1 1 \epsilon \epsilon \Rightarrow 00011. \end{aligned}$$

PA M prihvata niz 00011 praznim stogom:

$$\begin{aligned} (q_1, 00011, K) &\succ (q_1, 0011, NK) \succ (q_1, 011, NNK) \succ (q_1, 11, NNNK) \succ (q_2, 1, NNK) \succ \\ &\succ (q_2, \epsilon, NK) \succ (q_2, \epsilon, K) \succ (q_2, \epsilon, \epsilon). \end{aligned}$$

3.3 Svojstva kontekstno neovisnih jezika

Na temelju pokazane istovjetnosti, jezik L jest kontekstno neovisan ako i samo ako postoji PA koji ga prihvaca. Jezik $L = \{a^i b^i c^i \mid i \geq 1\}$ je primjer jezika koji nije kontekstno neovisni jezik, jer za taj jezik nije moguce izgraditi PA. Jezik L jest skup nizova znakova koji se sastoje od jednakog broja znakova a , b i c . Stavljanjem znakova a na stog, te uspoređivanjem znakova stoga sa znakovima ulaznog niza, najviše što se može provjeriti jest da ima jednak broj znakova a i b . Uspoređivanjem znakova a i b , znakovi a se uzimaju sa stoga i više nije moguce provjeriti da znakova c ima jednak broj kao i znakova a i znakova b .

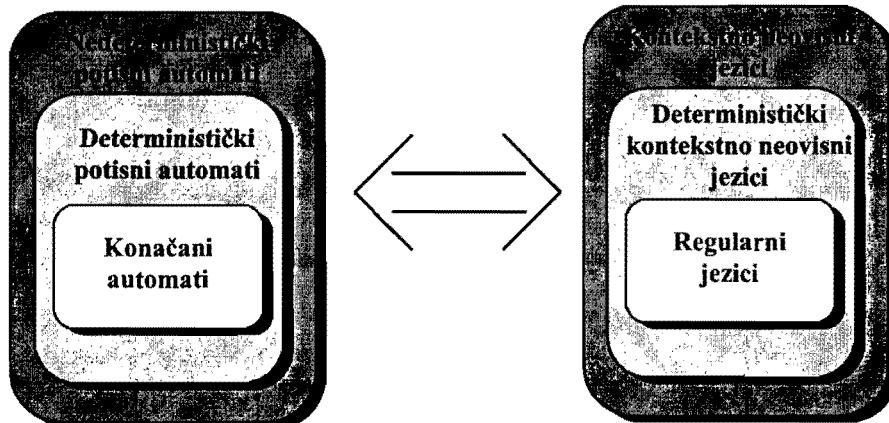


Slika 3.21 prikazuje kontekstno neovisne jezike. Klasa kontekstno neovisnih jezika jest pravi podskup skupa svih jezika.

Slika 3.21: Skup kontekstno neovisnih jezika

U primjeru 3.17 opisan je jezik $N(M_2) = \{ww^R\}$ za koji nije moguce izgraditi deterministički PA, ali je moguce izgraditi nedeterministički PA. Neka se klasa jezika koju prihvaca deterministički PA označi kao deterministički kontekstno neovisni jezici, a klasa jezika koju prihvaca nedeterministički PA neka se označi kao kontekstno neovisni jezici. Primjer 3.17 pokazuje da postoje jezici koji su kontekstno neovisni, ali nisu deterministički kontekstno neovisni. Na temelju jezika iz primjera 3.17 zaključuje se da klasa determinističkih kontekstno neovisnih jezika jest pravi podskup klase kontekstno neovisnih jezika.

Budući da je DKA poseban slučaj determinističkog PA koji ne koristi potisni stog, regularne jezike moguće je prihvatiti determinističkim PA, odnosno klasa regularnih jezika jest pravi podskup klase determinističkih kontekstno neovisnih jezika. Slika 3.22 prikazuje do sada upoznatu hijerarhiju automata i jezika.



Slika 3.22: Hijerarhija automata i jezika

3.3.1 Svojstva zatvorenosti kontekstno neovisnih jezika

Istovjetnost kontekstno neovisne gramatike, kontekstno neovisnih jezika i potisnih automata koristi se za opis svojstava kontekstno neovisnih jezika. Kontekstno neovisni jezici i kontekstno neovisna gramatika istovjetni su na temelju definicije kontekstno neovisnih jezika, dok je njihova istovjetnost s potisnim automatima pokazana u odjeljku 3.2.3.

Kontekstno neovisna gramatika koristi se za pokazavanje svojstva da su kontekstno neovisni jezici zatvoreni s obzirom na operacije unije, nadovezivanja, Kleeneovog operatora i supstitucije. Kontekstno neovisna gramatika koristi se i za pokazavanje svojstva da presjek i komplement dvaju kontekstno neovisnih jezika nisu nužno kontekstno neovisni jezici. PA i DKA koriste se za pokazivanje svojstva da presjek kontekstno neovisnog jezika i regularnog jezika jest kontekstno neovisni jezik.

Unija

Unija dvaju kontekstno neovisnih jezika jest kontekstno neovisni jezik.

Neka gramatika $G_1=(V_1, T_1, P_1, S_1)$ generira jezik $L(G_1)$, a gramatika $G_2=(V_2, T_2, P_2, S_2)$ neka generira jezik $L(G_2)$. Nezavršni znakovi gramatike G_1 i nezavršni znakovi gramatike G_2 izabiru se tako da je $V_1 \cap V_2 = \emptyset$, tj. niti jedan nezavršni znak nije istodobno nezavršni znak obje gramatike.

Gramatika $G_3=(V_3, T_3, P_3, S_3)$ koja generira jezik $L(G_1) \cup L(G_2)$ konstruira se na sljedeći način:

- 1) $V_3 = V_1 \cup V_2 \cup \{S_3\}$, gdje je $S_3 \notin V_1$ i $S_3 \notin V_2$, $V_1 \cap V_2 = \emptyset$.
- 2) $T_3 = T_1 \cup T_2$.
- 3) U skup produkcija $P_3 = P_1 \cup P_2$ dodaju se produkcije:

$$S_3 \rightarrow S_1 \mid S_2.$$

Najprije se dokazuje da je $L(G_1) \cup L(G_2) \subseteq L(G_3)$. Pretpostavimo da je w niz jezika $L(G_1)$. Budući da smo u koraku (3) konstrukcije dodali gramatici G_3 produkciju $S_3 \rightarrow S_1$ i budući da su sve produkcije gramatike G_1 ujedno i produkcije gramatike G_3 , gramatika G_3 generira niz w sljedećim postupkom:

$$S_3 \xrightarrow[G_3]{*} S_1 \xrightarrow[G_1]{*} w.$$

Ako je niz w u jeziku $L(G_2)$, onda gramatika G_3 generira niz w sljedećim postupkom:

$$S_3 \xrightarrow[G_3]{*} S_2 \xrightarrow[G_2]{*} w.$$

Zaključuje se da je bilo koji niz w iz unije jezika $L(G_1) \cup L(G_2)$ ujedno u jeziku $L(G_3)$, odnosno da je $L(G_1) \cup L(G_2) \subseteq L(G_3)$. Da se dokaže da je $L(G_1) \cup L(G_2) = L(G_3)$, potrebno je dokazati da je bilo koji niz jezika $L(G_3)$ ujedno u jeziku $L(G_1) \cup L(G_2)$, odnosno da je $L(G_3) \subseteq L(G_1) \cup L(G_2)$. Na temelju produkcija $S_3 \rightarrow S_1 \mid S_2$ postupci generiranja niza w gramatikom G_3 mogu poprimiti jedan od dva oblika:

$$S_3 \xrightarrow[G_3]{*} S_1 \xrightarrow[G_3]{*} w \text{ ili } S_3 \xrightarrow[G_3]{*} S_2 \xrightarrow[G_3]{*} w.$$

*

Neka se prvo razmatra slučaj $S_3 \xrightarrow{G_3} S_2 \xrightarrow{G_3} w$. Budući da skupovi nezavršnih znakova V_1 i V_2 nemaju zajedničkih elemenata, i budući da je $S_2 \in V_2$, onda se u postupku generiranja niza $S_2 \xrightarrow{G_2} w$ primjenjuju samo produkcije gramatike G_2 , odnosno vrijedi da je $S_2 \xrightarrow{G_2} w$ i niz $w \in L(G_2)$. U postupku generiranja $S_1 \xrightarrow{G_1} w$ mogu se primijeniti samo produkcije gramatike G_1 , te vrijedi $S_1 \xrightarrow{G_1} w$ i niz $w \in L(G_1)$. Zaključuje se da je $L(G_3) \subseteq L(G_1) \cup L(G_2)$.

*

*

Pokazane tvrdnje $L(G_1) \cup L(G_2) \subseteq L(G_3)$ i $L(G_3) \subseteq L(G_1) \cup L(G_2)$ dokazuju da unija dvaju kontekstno neovisnih jezika jest kontekstno neovisni jezik, odnosno da je $L(G_1) \cup L(G_2) = L(G_3)$.

Nadovezivanje

Nadovezivanje dvaju kontekstno neovisnih jezika jest kontekstno neovisni jezik.

Neka gramatika $G_1 = (V_1, T_1, P_1, S_1)$ generira jezik $L(G_1)$, a gramatika $G_2 = (V_2, T_2, P_2, S_2)$ neka generira jezik $L(G_2)$. Gramatika $G_4 = (V_4, T_4, P_4, S_4)$ koja generira jezik $L(G_4) = L(G_1)L(G_2)$ konstruira se na sljedeći način:

- 1) $V_4 = V_1 \cup V_2 \cup \{S_4\}$, gdje je $S_4 \notin V_1$ i $S_4 \notin V_2$, $V_1 \cap V_2 = \emptyset$.
- 2) $T_4 = T_1 \cup T_2$.
- 3) U skup produkcija $P_4 = P_1 \cup P_2$ doda se produkcija:

$$S_4 \rightarrow S_1 S_2.$$

Dokazuje se na sličan način kao i za operaciju unije. Dokaz se zasniva na sljedećem postupku generiranja niza:

$$S_4 \xrightarrow{G_4} S_1 S_2 \xrightarrow{G_1} w_1 S_2 \xrightarrow{G_2} w_1 w_2, \text{ gdje su nizovi završnih znakova } w_1 w_2 \in L(G_4), w_1 \in L(G_1) \text{ i } w_2 \in L(G_2).$$

Kleeneov operator

Kontekstno neovisni jezici zatvoreni su s obzirom na Kleeneov operator.

Neka gramatika $G_1 = (V_1, T_1, P_1, S_1)$ generira jezik $L(G_1)$. Gramatika $G_5 = (V_5, T_5, P_5, S_5)$ koja generira jezik $L(G_5) = L(G_1)^*$ konstruira se na sljedeći način:

- 1) $V_5 = V_1 \cup \{S_5\}$, gdje je $S_5 \notin V_1$.
- 2) $T_5 = T_1$.
- 3) U skup produkcija $P_5 = P_1$ dodaju se produkcije:

$$S_5 \rightarrow S_1 S_5 \mid \varepsilon.$$

Dokaz se zasniva na sljedeća dva postupka generiranja niza:

$$1) \quad S_5 \xrightarrow[G_5]{*} S_1 S_5 \xrightarrow[G_1]{*} w_1 S_5 \xrightarrow[G_5]{*} w_1 S_1 S_5 \xrightarrow[G_1]{*} w_1 w_1 S_5 \xrightarrow[G_5]{*} \dots \xrightarrow[G_1]{*} w_1^+ S_5 \xrightarrow[G_5]{*} w_1^+ S_5, \text{ gdje su nizovi završnih znakova } w_1^+ \in L(G_5) \text{ i } w_1 \in L(G_1);$$

$$2) \quad S_5 \xrightarrow[G_5]{*} \epsilon.$$

Supsticija

Kontekstno neovisni jezici zatvoreni su s obzirom na supsticiju.

Prepostavimo da gramatika $G=(V, T, P, S)$ generira kontekstno neovisni jezik $L(G)$. Neka se svi završni znakovi a_i jezika $L(G)$ zamijene nizovima kontekstno neovisnog jezika $L(G_i)$, gdje $1 \leq i \leq k$. Broj k jest kardinalni broj skupa završnih znakova T . Neka gramatika $G_i=(V_i, T_i, P_i, S_i)$ generira jezik $L(G_i)$. Nastali jezik L' jest kontekstno neovisni jezik za koji je moguće konstruirati gramatiku $G'=(V', T', P', S')$ na sljedeći način:

- 1) U skup nezavršnih znakova V' stave se svi nezavršni znakovi gramatike G i nezavršni znakovi svih gramatika G_i :

$$V' = V \cup V_1 \cup V_2 \cup \dots \cup V_k, \quad V \cap V_i = \emptyset, \quad V_i \cap V_j = \emptyset, \quad \text{za } 1 \leq i \leq k, \quad 1 \leq j \leq k, \quad i \neq j$$

- 2) U skupu završnih znakova T' su samo završni znakovi gramatika G_i :

$$T' = T_1 \cup T_2 \cup \dots \cup T_k.$$

- 3) Početni nezavršni znak S' jest početni nezavršni znak S gramatike G :

$$S' = S.$$

- 4) U skup produkcija $P' = P_1 \cup P_2 \cup \dots \cup P_k$ dodaju se produkcije gramatike G koje se prethodno preurede na sljedeći način:

U produkcijama gramatike G svaki završni znak a_i zamijeni se početnim nezavršnim znakom S_i gramatike G_i .

Primjer 3.22. Zadan je jezik L u kojem su nizovi koji imaju jednak broj znakova a i b . Jezik L generira se gramatikom $G=(\{S\}, \{a, b\}, P, S)$ koja ima produkcije:

$$S \rightarrow aSbS \mid bSaS \mid \epsilon$$

Znak a zamijeni se nizovima jezika $L_1=\{0^n1^n \mid n \geq 1\}$. Jezik L_1 generira se gramatikom $G_1=(\{S_1\}, \{0, 1\}, P, S_1)$ koja ima produkcije:

$$S_1 \rightarrow 0 \ S_1 \ 1 \mid 0 \ 1$$

Znak b zamijeni se nizovima jezika $L_2=\{ww^R \mid w \text{ jest niz iz } (0+2)^*\}$. Jezik L_2 generira se gramatikom $G_2=(\{S_2\}, \{0, 2\}, P, S_2)$ koja ima produkcije:

$$S_2 \rightarrow 0 \ S_2 \ 0 \mid 2 \ S_2 \ 2 \mid \epsilon$$

Zamjenama znakova a i b nizovima jezika L_1 i L_2 nastaje jezik L' koji generira gramatika G' :

- 1) $V' = \{S\} \cup \{S_1\} \cup \{S_2\} = \{S, S_1, S_2\}$.
- 2) $T' = \{0, 1\} \cup \{0, 2\} = \{0, 1, 2\}$.
- 3) $S' = S$.
- 4) Skupu produkcija $P' = \{S_1 \rightarrow 0S_11 \mid 01\} \cup \{S_2 \rightarrow 0S_20 \mid 2S_22 \mid \epsilon\}$ dodaju se produkcije gramatike G u kojima se završni znakovi a zamijene nezavršnim znakom S_1 , a završni znakovi b zamijene se nezavršnim znakom S_2 :

$$S \rightarrow S_1 S S_2 S \mid S_2 S S_1 S \mid \epsilon.$$

Gramatika $G' = (\{S, S_1, S_2\}, \{0, 1, 2\}, P, S)$ ima produkcije:

$$\begin{aligned} S &\rightarrow S_1 S S_2 S \mid S_2 S S_1 S \mid \epsilon \\ S_1 &\rightarrow 0 S_1 1 \mid 0 1 \\ S_2 &\rightarrow 0 S_2 0 \mid 2 S_2 2 \mid \epsilon \end{aligned}$$

Presjek i komplement

Presjek dvaju kontekstno neovisnih jezika nije nužno kontekstno neovisni jezik, odnosno kontekstno neovisni jezici nisu zatvoreni s obzirom na operaciju presjeka.

Kontekstno neovisni jezici nisu zatvoreni s obzirom na operaciju komplementa.

Prethodno je pokazano, a u primjeru 3.24 je dodatno objašnjeno, da jezik $L = \{a^i b^i c^i \mid i \geq 1\}$ nije kontekstno neovisan. Jezik L je presjek jezika $L_1 = \{a^i b^i c^j \mid i \geq 1 \text{ i } j \geq 1\}$ i jezika $L_2 = \{a^j b^i c^i \mid i \geq 1 \text{ i } j \geq 1\}$. U jeziku L_1 su nizovi koji imaju jednak broj znakova a i b , te proizvoljan broj znakova c . U jeziku L_2 su nizovi koji imaju jednak broj znakova b i c , te proizvoljan broj znakova a . Presjek jezika L_1 i L_2 jest jezik L koji ima jednak broj znakova a , b i c .

Jezik $L_1 = \{a^i b^i c^j \mid i \geq 1 \text{ i } j \geq 1\}$ prihvata PA koji za svaki pročitani ulazni znak a spremi na stog znak stoga A , a zatim uspoređuje broj znakova A spremljenih na stog sa brojem pročitanih znakova b . Na kraju PA provjera da li se u nizu nalazi barem jedan znak c . Jezik L_1 generira kontekstno neovisna gramatika $G_1 = (\{S, A, B\}, \{a, b, c\}, P, S)$ koja ima produkcije:

$$\begin{aligned} S &\rightarrow A C \\ A &\rightarrow a A b \mid a b \\ C &\rightarrow c C \mid c. \end{aligned}$$

Konstrukcijom PA i kontekstno neovisne gramatike pokazano je da je jezik L_1 kontekstno neovisni jezik.

Na sličan način moguće je izgraditi PA za jezik $L_2 = \{a^j b^i c^i \mid i \geq 1 \text{ i } j \geq 1\}$. PA provjerava ima li jednak broj znakova b i c . Gramatika $G_2 = (\{S, C, D\}, \{a, b, c\}, P, S)$ koja generira jezik L_2 ima produkcije:

$$\begin{aligned} S &\rightarrow A B \\ A &\rightarrow a A \mid a \\ B &\rightarrow b B c \mid b c. \end{aligned}$$

Budući da su oba jezika L_1 i L_2 kontekstno neovisni jezici, a jezik L koji je njihov presjek nije kontekstno neovisni jezik, pokazano je da kontekstno neovisni jezici nisu zatvoreni s obzirom na operaciju presjeka.

Dokažimo sada da komplement kontekstno neovisnog jezika nije nužno kontekstno neovisni jezik. Prepostavimo da komplement kontekstno neovisnog jezika jest kontekstno neovisni jezik. Na temelju DeMorganovih zakona $L_1 \cap L_2 = (L_1^c \cup L_2^c)^c$ zaključuje se da presjek dva kontekstno neovisna jezika jest

kontekstno neovisan, što je suprotno prethodno pokazanoj činjenici. Prema tome, kontekstno neovisni jezici nisu zatvoreni s obzirom na operaciju komplementa.

Presjek kontekstno neovisnog jezika i regularnog jezika

Presjek kontekstno neovisnog jezika i regularnog jezika jest kontekstno neovisni jezik.

Prepostavimo da kontekstno neovisni jezik L_1 prihvata PA $M_1=(Q_1, \Sigma, \Gamma, \delta_1, q_0, Z_1, F_1)$, a da regularni jezik L_2 prihvata DKA $M_2=(Q_2, \Sigma, \delta_2, p_0, F_2)$. Moguće je izgraditi PA $M'=(Q', \Sigma, \Gamma, \delta', q'_0, Z_0, F')$ koji prihvata jezik prihvatljivim stanjem $L=L_1 \cap L_2$:

$$1) \quad Q' = Q_2 \times Q_1.$$

$$2) \quad q'_0 = [p_0, q_0].$$

$$3) \quad F' = F_2 \times F_1.$$

4) Skup:

$$\delta'([p, q], a, X) \text{ sadrži } ([p', q'], \gamma)$$

ako i samo ako za funkciju prijelaza DKA M_2 vrijedi:

$$\delta_2(p, a) = p',$$

a za funkciju prijelaza PA M_1 vrijedi:

$$(q', \gamma) \in \delta_1(q, a, X).$$

Ako a jest ε , onda je $p' = p$.

PA M' simulira rad automata M_1 i M_2 . Dokaz da M' prihvata jezik $L=L_1 \cap L_2$ zasniva se na indukciji s obzirom na broj koraka i . Dokazuje se da za PA M' vrijedi:

$$([p_0, q_0], w, Z_0) \xrightarrow[M']^i ([p, q], \varepsilon, \gamma)$$

ako i samo ako je:

$$(q_0, w, Z_0) \xrightarrow[M_1]^i (q, \varepsilon, \gamma) \quad i \quad \delta_2(p_0, w) = p,$$

gdje je $[p, q] \in F'$ ako i samo ako je $p \in F_2$ i $q \in F_1$.

Primjer 3.23. PA $M_1=(\{q_1, q_2\}, \{0, 1\}, \{N, K\}, \delta_1, q_1, K, \{q_2\})$ ima sljedeće prijelaze:

- | | |
|---|---|
| 1) $\delta_1(q_1, 0, K) = \{(q_1, NK)\}$ | 2) $\delta_1(q_1, 0, N) = \{(q_1, NN)\}$ |
| 3) $\delta_1(q_1, 1, N) = \{(q_2, \varepsilon)\}$ | 4) $\delta_1(q_2, 1, N) = \{(q_2, \varepsilon)\}$ |

PA M_1 prihvata jezik $L(M_1)=\{0^n 1^m \mid n \geq 1, m \geq 1, m \leq n\}$ prihvatljivim stanjem q_2 .

DKA $M_2=(\{p_1, p_2, p_3\}, \{0, 1\}, \delta_2, p_1, \{p_3\})$ ima sljedeće prijelaze:

$$\begin{array}{ll} \delta_2(p_1, 0) = p_1 & \delta_2(p_1, 1) = p_2 \\ \delta_2(p_2, 0) = p_2 & \delta_2(p_2, 1) = p_3 \\ \delta_2(p_3, 0) = p_3 & \delta_2(p_3, 1) = p_3 \end{array}$$

Niz jezika $L(M_2)$ ima barem dva znaka 1. Presjek jezika $L(M_1)$ i $L(M_2)$ jest jezik $L_3=L(M_1) \cap L(M_2)=\{0^n1^m \mid n \geq 2, m \geq 2, m \leq n\}$ koji prihvaća PA $M'=(Q', \Sigma, \Gamma, \delta', q'_0, Z_0, F')$:

- 1) $Q'=\{[p_1, q_1], [p_1, q_2], [p_2, q_1], [p_2, q_2], [p_3, q_1], [p_3, q_2]\}.$
- 2) $q'_0=[p_1, q_1].$
- 3) $F'=\{[p_3, q_2]\}.$
- 4)
 - 1) $\delta'([p_1, q_1], 0, K)=\{([p_1, q_1], NK)\}$
 - 2) $\delta'([p_1, q_1], 0, N)=\{([p_1, q_1], NN)\}$
 - 3) $\delta'([p_1, q_1], 1, N)=\{([p_2, q_2], \epsilon)\}$
 - 4) $\delta'([p_2, q_2], 1, N)=\{([p_3, q_2], \epsilon)\}$
 - 5) $\delta'([p_3, q_2], 1, N)=\{([p_3, q_2], \epsilon)\}$

Funkcija prijelaza dana je samo za dohvatljiva stanja.

Neka je na ulazu PA M_1 niz 00011 jezika $L_3=L(M_1) \cap L(M_2)$. Niz se prihvaća, jer postoji sljedeći slijed prijelaza u prihvatljivo stanje q_2 :

$$\begin{aligned} ([p_1, q_1], 00011, K) &\succ ([p_1, q_1], 0011, NK) \succ ([p_1, q_1], 011, NNK) \succ \\ &([p_1, q_1], 11, NNNK) \succ ([p_2, q_2], 1, NNK) \succ ([p_2, q_2], \epsilon, NKX_0) \text{ i } q_2 \in F_1. \end{aligned}$$

DKA M_2 također prihvaća niz 00011 :

$$\delta(p_1, 00011) = \delta(p_1, 0011) = \delta(p_1, 011) = \delta(p_1, 11) = \delta(p_2, 1) = p_3 \text{ i } p_3 \in F_2.$$

PA M' prihvaća niz, jer je:

$$\begin{aligned} ([p_1, q_1], 00011, K) &\succ ([p_1, q_1], 0011, NK) \succ ([p_1, q_1], 011, NNK) \succ \\ &([p_1, q_1], 11, NNNK) \succ ([p_2, q_2], 1, NNK) \succ ([p_2, q_2], \epsilon, NKX_0) \text{ i } [p_2, q_2] \in F'. \end{aligned}$$

3.3.2 Svojstvo napuhavanja

Svojstvo napuhavanja kontekstno neovisnih jezika slično je svojstvu napuhavanja regularnih jezika. Svojstvo napuhavanja koristi se za dokazivanje kontekstne neovisnosti jezika. Međutim, primjena svojstva nije jednostavna kao za regularne jezike.

Svojstvo napuhavanja zasniva se na broju čvorova generativnog stabla i broju nezavršnih znakova gramatike. Za dovoljno dugački niz broj unutarnjih čvorova generativnog stabla veći je od kardinalnog broja skupa nezavršnih znakova gramatike, što znači da je više čvorova označeno istim nezavršnim znakom.

Neka gramatika $G=(V, T, P, S)$ generira stablo koje ima više čvorova od kardinalnog broja skupa nezavršnih znakova V . Pokazuje se da postoji put stabla u kojem se jedan nezavršni znak nalazi barem na dva mjesta na istom putu, i to pri dnu stabla. Za dano stablo pokazuje se da postoji sljedeći postupak generiranja niza:

$$S \xrightarrow[G]{*} u \underline{A} y \xrightarrow[G]{*} u \underline{v} \underline{A} x y \xrightarrow[G]{*} u v w x y,$$

gdje $A \xrightarrow[G]{*} vAx$, a zatim $A \xrightarrow[G]{*} w$. Nezavršni znak A pojavljuje se dva puta u postupku generiranja niza $uvwxy$,

gdje su u, v, w, x i y nizovi završnih znakova. Budući da je postupak generiranja međuniza $A \xrightarrow[G]{*} vAx$ moguće ponavljati proizvoljni broj puta, gramatika G generira niz sljedećeg oblika:

$$S \xrightarrow[G]{*} uAy \xrightarrow[G]{*} uvAxy \xrightarrow[G]{*} uvvAxxxy \xrightarrow[G]{*} uvvvAxxxxy \dots y \xrightarrow[G]{*} uv^jAx^jy \xrightarrow[G]{*} uv^iwx^iy,$$

gdje je $i \geq 0$.

U postupku dokazivanja da je jezik L kontekstno neovisan, koristi se sljedeće svojstvo:

- i) Neka je L kontekstno neovisni jezik. Postoji konstantna n koja ovisi samo o jeziku L takva da ako je niz z iz jezika L i $|z| \geq n$, onda je niz z moguće napisati kao niz $uvwxy$ za koji vrijedi:
- $|vx| \geq 1$,
 - $|vwx| \leq n$,
 - za bilo koji $i \geq 0$ vrijedi da je niz uv^iwx^iy iz jezika L .
-

Primjer 3.24. Pomoću svojstva napuhavanja moguće je pokazati da jezik $L = \{a^i b^i c^i \mid i \geq 1\}$ nije kontekstno neovisan. Na početku se pretpostavi da je jezik L kontekstno neovisan. Promatra se konstanta n i niz $z = a^n b^n c^n$. Za danu konstantu n i niz z vrijedi da je $|z| \geq n$. Niz z pokuša se napisati kao $uvwxy$ tako da zadovolji uvjete (a) do (c) svojstva napuhavanja. Potrebno je odrediti podnizove v i x u nizu $a^n b^n c^n$ koji se proizvoljno puta mogu ponoviti, a da je dobiveni niz u jeziku L . Budući da mora biti zadovoljeno $|vwx| \leq n$, nije moguće da podniz vx sadrži znakove a i c , jer krajnje desni znak a je za $n+1$ mjesta udaljen od krajnje lijevog znaka c u nizu $a^n b^n c^n$. Sastoje li se podnizovi v i x samo od znakova a , niz uw (niz uw jest niz uv^iwx^iy za $i=0$) ima n znakova b , n znakova c , ali ima manje od n znakova a , jer je $|vx| \geq 1$. Time je pokazano da niz uw nije oblika $a^i b^j c^j$ što je suprotno uvjetu (c) da niz uw jest u jeziku L . Ako se podnizovi v i x sastoje od znakova b ili c , onda ti izbori dovode do suprotnosti na sličan način kao i za slučaj kada se podnizovi v i x sastoje samo od znakova a .

Sadrži li vx znakove a i b , niz uw ima više znakova c od znakova a ili znakova b . Sadrži li vx znakove b i c , niz uw ima više znakova a od znakova b ili znakova c . Bilo koji izbor nizova v i x dovodi do suprotnosti, što znači da jezik L nije kontekstno neovisan.

Na primjeru jezika $L = \{a^i b^j c^k d^l \mid \text{gdje je } i=0 \text{ ili } j=k=l\}$ koji nije kontekstno neovisan pokazuje se da uvjeti (a) do (c) nisu dovoljni. Napiše li se niz $z = b^j c^k d^l$ kao $z = uvwxy$, za bilo koji izbor podnizova u , v , w , x i y i konstante m niz uv^mwx^my jest u jeziku L . Na primjer, neka se niz vwx sastoji samo od znakova b . Izabere li se niz $z = a^i b^j c^j d^j$, podnizovi v i x mogu se sastojati samo od znakova a , što opet pokazuje da je bilo koji niz uv^mwx^my u jeziku L za bilo koju konstantu m .

Budući da svojstava (a) do (c) ponekad ne mogu otkriti da jezik nije kontekstno neovisan, definiraju se stroži uvjeti koji preciznije određuju mesta u nizu gdje se moraju smjestiti podnizovi v i x . Na primjer, u nizu $a^i b^j c^j d^j$ podnizovi v i x moraju se smjestiti u dio niza gdje su znakovi b , c i d da se pokaže da jezik nije kontekstno neovisan. Postupak u tom slučaju sličan je postupku u primjeru 3.24. Postoji više različitih definicija uvjeta napuhavanja ovisno o njihovoj pouzdanosti i složenosti, ali se oni zbog svoje složenosti neće dalje razmatrati.

4 REKURZIVNO PREBROJIVI JEZICI

Definicija rekurzivno prebrojivih jezika zasniva se na Turingovom stroju: jezik jest rekurzivno prebrojiv ako i samo ako postoji Turingov stroj koji ga prihvaca. Time je definirana istovjetnost Turingovog stroja i rekurzivno prebrojivih jezika: za bilo koji rekurzivno prebrojiv jezik moguce je izgraditi Turingov stroj koji ga prihvaca, i obrnuto, bilo koji Turingov stroj prihvaca jedan od rekurzivno prebrojivih jezika.

U odjeljku 4.1 definira se Turingov stroj, a u odjeljku 4.2 gramatika neograničenih produkacija. Svojstva rekurzivnih i rekurzivno prebrojivih jezika objasnjena su u odjeljku 4.3.

4.1 Turingov stroj (TS)

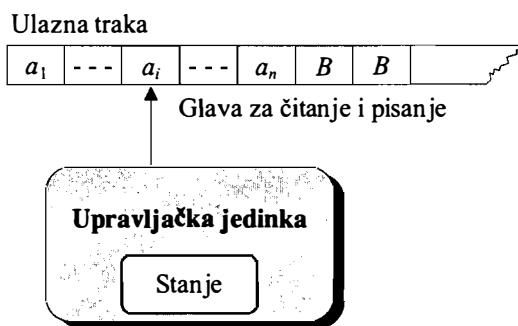
Bez obzira na svoju jednostavnost, Turingov stroj ima iste mogućnosti računanja kao bilo koje digitalno računalo, te predstavlja najopćenitiji matematički model računanja. Osnovna primjena Turingovog stroja jest prihvaćanje jezika. Budući da je omogućeno pisanje po ulaznoj traci, Turingov stroj koristi se za generiranje jezika i računanje cjelobrojnih funkcija.

Osnovni model TS definira se u odjeljku 4.1.1. U odjeljku 4.1.2 opisane su metode izrade TS. Razne inačice osnovnog modela TS prikazane su u odjeljcima 4.1.3 i 4.1.4. U odjeljku 4.1.5 opisuje se postupak generiranja jezika.

4.1.1 Osnovni model Turingovog stroja

U ovom odjeljku opisuje se rad osnovnog modela TS, dana je definicija osnovnog modela TS, te se definira način prihvaćanja jezika i računanja cjelobrojnih funkcija primjenom TS.

Definicija Turingovog stroja



Slika 4.1: Model Turingovog stroja

Na slici 4.1 prikazan je osnovni model Turingovog stroja. Upravljačka jedinka jest u jednom od konačnog broja stanja. Za razliku od konačnog automata, Turingov stroj nakon čitanja znaka ulazne trake zapiše novi znak na traku. Nadalje, glava za čitanje i pisanje miče se u lijevo i desno. Traka ima krajnje lijevu ćeliju, dok je beskonačna na desnu stranu. Na početku rada n krajnje lijevih ćelija sadrže niz w , gdje je $|w|=n$ i $n \geq 0$. Na ostatku trake nalaze se prazne ćelije koje se označavaju znakom B . Ulazni znakovi niza w i znakovi koje TS zapisuje na ulaznu traku čine skup znakova trake.

Tijekom rada upravljačka jedinka donosi odluku na temelju dva podatka:

- 1) stanje;
- 2) znak na traci.

Na temelju pročitanog znaka i stanja jedinke, Turingov stroj odlučuje:

- 1) u koje novo stanje prelazi upravljačka jedinka;
- 2) koji znak se zapiše na traku umjesto pročitanog znaka;
- 3) u koju stranu se miče glava za čitanje i pisanje.

Turingov stroj (TS) formalno se zadaje kao uređena sedmorka:

$$ts = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

gdje je:

Q	- konačan skup stanja;
Γ	- konačan skup znakova trake;
$B \in \Gamma$	- znak kojim se označava prazna ćelija;
$\Sigma \subseteq (\Gamma - \{B\})$	- konačan skup ulaznih znakova;
δ	- funkcija prijelaza $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$, gdje L i R označavaju pomak glave u lijevo i desno;
$q_0 \in Q$	- početno stanje;
$F \subseteq Q$	- skup prihvativih stanja.

Dozvoljava se da je funkcija prijelaza δ ne definirana za pojedine argumente. Funkcija prijelaza $\delta(q, V)=(p, Z, W)$ određuje da TS iz stanja q ($q \in Q$) čitanjem znaka V ($V \in \Gamma$) prelazi u stanje p , na traku zapiše znak Z ($Z \in \Gamma$) umjesto znaka V , a glava za čitanje i pisanje miče se u lijevo ili desno ovisno o W ($W \in \{L, R\}$).

Primjer 4.1. Zadan je TS $M=(\{q_0, q_1, q_2, q_3, q_4\}, \{0, 1\}, \{0, 1, X, Y, B\}, \delta, q_0, B, \{q_4\})$ sa sljedećim prijelazima:

- | | |
|---------------------------------|----------------------------------|
| 1. $\delta(q_0, 0)=(q_1, X, R)$ | 6. $\delta(q_0, Y)=(q_3, Y, R)$ |
| 2. $\delta(q_1, 0)=(q_1, 0, R)$ | 7. $\delta(q_1, Y)=(q_1, Y, R)$ |
| 3. $\delta(q_2, 0)=(q_2, 0, L)$ | 5. $\delta(q_2, X)=(q_0, X, R)$ |
| | 8. $\delta(q_2, Y)=(q_2, Y, L)$ |
| | 9. $\delta(q_3, Y)=(q_3, Y, R)$ |
| | 10. $\delta(q_3, B)=(q_4, B, R)$ |
| 4. $\delta(q_1, 1)=(q_2, Y, L)$ | |

Funkcije prijelaza za ista stanja nalaze se u istom retku, dok se funkcije prijelaza za iste znakove trake nalaze u istom stupcu. Na primjer, u trećem retku su definirane funkcije prijelaza za stanje q_2 , a u četvrtom stupcu su definirane funkcije prijelaza za znak trake Y .

TS M prihvata jezik $L(M)=\{0^n 1^n \mid n \geq 1\}$. Na početku rada u krajnje lijevim čelijama trake zapisan je niz znakova 0 i 1, a ostale čelije su prazne. TS M jest u stanju q_0 , a glava za čitanje i pisanje čita znak zapisan u krajnje lijevoj čeliji. Prijelaz (1) zamjeni krajnji lijevi znak 0 znakom X , TS M prelazi u stanje q_1 i miče glavu u desno. U stanju q_1 prijelazi (2) i (7) miču glavu u desno preko znakova 0 i Y sve do krajnje lijevog znaka 1. Pronade li se krajnje lijevi znak 1, prijelaz (4) zamjeni znak 1 znakom Y , promjeni stanje u q_2 i pomakne glavu u lijevo.

Prijelazi (3) i (8) miču glavu u lijevo preko znakova Y i 0 sve do krajnje desnog znaka X . Pronade li se krajnje desni znak X , prijelaz (5) mijenja stanje u q_0 i cijeli se proces nastavlja za sljedeći par krajnje lijevih znakova 0 i 1.

Ima li isti broj znakova 0 i 1, svi znakovi 0 i 1 zamijene se znakovima X i Y . TS M čita znak Y u stanju q_0 ako i samo ako je znak Y neposredno desno uz znak X . Čitanjem znaka Y prijelaz (6) mijenja stanje TS M u stanje q_3 . Prijelaz (9) miče glavu preko svih znakova Y do krajnje lijeve praznine. Prijelaz (10) mijenja stanje TS M u q_4 . Budući da nije definiran niti jedan prijelaz za stanje q_4 , TS M se zaustavi. Budući da se TS M zaustavlja u stanju q_4 koje je u skupu prihvatljivih stanja, niz se prihvata.

Primjer rada TS M opisan je tablicom 4.1. U prvom stupcu jest sadržaj trake koji se nalazi lijevo od glave za čitanje i pisanje, a u trećem stupcu jest sadržaj trake desno od glave za čitanje i pisanje. TS čita krajnje lijevi znak u trećoj koloni. U drugom stupcu je stanje TS. Na primjer, neka je na traci zapisan niz 0011. TS M je u početnom stanju q_0 i čita krajnje lijevi znak niza 0011, a to je znak 0. Za stanje q_0 i znak trake 0 TS M primjeni sljedeći prijelaz:

$$\delta(q_0, 0) = (q_1, X, R)$$

Prijelaz mijenja stanje q_0 u stanje q_1 , znak 0 zamjeni znakom X , a glava se miče u desno. Stanje TS M prije prijelaza (1) dano je u prvom retku tablice, a stanje nakon prijelaza (1) dano je u drugom retku. U drugom retku stanje je q_1 i čita se znak 0. TS M primjeni prijelaz (2). TS M miče glavu u desno preko znaka 0 i ostaje u stanju q_1 . Novo stanje prikazano je u trećem retku tablice. Ostali reci tablice prikazuju prijelaze sve do trenutka kad TS uđe u stanje q_4 , zaustavi daljnji rad i prihvati niz 0011.

Sadržaj trake lijevo od glave	Stanje	Sadržaj trake desno od glave	Funkcija prijelaza
ϵ	q_0	0 0 1 1 B B B ...	$\delta(q_0, 0) = (q_1, X, R)$
X	q_1	0 1 1 B B B ...	$\delta(q_1, 0) = (q_1, 0, R)$
$X 0$	q_1	1 1 B B B ...	$\delta(q_1, 1) = (q_2, Y, L)$
X	q_2	0 Y 1 B B B ...	$\delta(q_2, 0) = (q_2, 0, L)$
ϵ	q_2	$X 0 Y 1 B B B ...$	$\delta(q_2, X) = (q_0, X, R)$
X	q_0	0 Y 1 B B B ...	$\delta(q_0, 0) = (q_1, X, R)$
XX	q_1	$Y 1 B B B ...$	$\delta(q_1, Y) = (q_1, Y, R)$
XXY	q_1	$1 B B B ...$	$\delta(q_1, 1) = (q_2, Y, L)$
XX	q_2	$YY B B B ...$	$\delta(q_2, Y) = (q_2, Y, L)$
X	q_1	$XY Y B B B ...$	$\delta(q_2, X) = (q_0, X, R)$
XX	q_0	$YY B B B ...$	$\delta(q_0, Y) = (q_3, Y, R)$
XXY	q_3	$Y B B B ...$	$\delta(q_3, Y) = (q_3, Y, R)$
$XXY Y B$	q_3	$B B B ...$	$\delta(q_3, B) = (q_4, B, R)$
$XXY Y B$	q_4	$B B ...$	Budući da je $q_4 \in F$ i budući da nema daljnji prijelaza, TS se zaustavlja i niz se prihvata.

Tablica 4.1: Slijed prijelaza TS M za niz 0011

Ima li više znakova 0 od znakova 1, niz se ne prihvata. Svi znakovi 1 zamijene se znakovima Y i micanjem u desno u stanju q_1 ne pronade se niti jedan znak 1. Glava se pomakne preko svih znakova Y i pročita se praznina B . Budući da za stanje q_1 i prazninu B nije definiran niti jedan prijelaz, TS M stane. Stanje q_1 nije u skupu prihvatljivih stanja, te se niz ne prihvata. Tablica 4.2 prikazuje slijed prijelaza za niz 001.

Sadržaj trake lijevo od glave	Stanje	Sadržaj trake desno od glave	Funkcija prijelaza
ϵ	q_0	0 0 1 B B B ...	$\delta(q_0, 0) = (q_1, X, R)$
X	q_1	0 1 B B B ...	$\delta(q_1, 0) = (q_1, 0, R)$
$X 0$	q_1	1 B B B ...	$\delta(q_1, 1) = (q_2, Y, L)$
X	q_2	0 Y B B B ...	$\delta(q_2, 0) = (q_2, 0, L)$
ϵ	q_2	X 0 Y B B B ...	$\delta(q_2, X) = (q_0, X, R)$
X	q_0	0 Y B B B ...	$\delta(q_0, 0) = (q_1, X, R)$
XX	q_1	Y B B B ...	$\delta(q_1, Y) = (q_1, Y, R)$
XXY	q_1	B B B ...	Budući da $q_1 \notin F$ i budući da nema dalnjih prijelaza, TS se zaustavlja i niz se ne prihvata.

Tablica 4.2: Slijed prijelaza TS M za niz 001

Ima li više znakova 1 od znakova 0, niz se ne prihvata. Svi znakovi 0 zamijene se znakovima X i TS M mijenja stanje u q_3 . Budući da za stanje q_3 i znak 1 nije definiran niti jedan prijelaz, micanjem glave u desno u stanju q_3 TS M se zaustavi na krajnje lijevom znaku 1. Stanje q_3 nije u skupu prihvatljivih stanja i niz se ne prihvata. Tablica 4.3 prikazuje slijed prijelaza za niz 011.

Sadržaj trake lijevo od glave	Stanje	Sadržaj trake desno od glave	Funkcija prijelaza
ϵ	q_0	0 1 1 B B B ...	$\delta(q_0, 0) = (q_1, X, R)$
X	q_1	1 B B B ...	$\delta(q_1, 1) = (q_2, Y, L)$
ϵ	q_2	X Y 1 B B B ...	$\delta(q_2, X) = (q_0, X, R)$
X	q_0	Y 1 B B B ...	$\delta(q_0, Y) = (q_3, Y, R)$
XY	q_3	1 B B B ...	Budući da $q_3 \notin F$ i budući da nema dalnjih prijelaza, TS se zaustavlja i niz se ne prihvata.

Tablica 4.3: Slijed prijelaza TS M za niz 011

TS M ne prihvata ni niz oblika (01)*. Tablica 4.4 prikazuje slijed prijelaza za niz 0101.

Sadržaj trake lijevo od glave	Stanje	Sadržaj trake desno od glave	Funkcija prijelaza
ϵ	q_0	0 1 0 1 B B B ...	$\delta(q_0, 0) = (q_1, X, R)$
X	q_1	1 0 1 B B B ...	$\delta(q_1, 1) = (q_2, Y, L)$
ϵ	q_2	X Y 0 1 B B B ...	$\delta(q_2, X) = (q_0, X, R)$
X	q_0	Y 0 1 B B B ...	$\delta(q_0, Y) = (q_3, Y, R)$
XY	q_3	0 1 B B B ...	Budući da $q_3 \notin F$ i budući da nema dalnjih prijelaza, TS se zaustavlja i niz se ne prihvata.

Tablica 4.4: Slijed prijelaza TS M za niz 0101

Prihvatanje jezika Turingovim strojem

Prve tri kolone tablica 4.1 do 4.4 čine konfiguraciju TS. Konfiguracija TS zadaje se sadržajem ćelija lijevo od glave za čitanje i pisanje, stanjem upravljačke jedinice i sadržajem ćelija koje se nalaze desno od glave za čitanje i pisanje. TS čita krajnje lijevi znak koji je desno od oznake stanja. Stanje dijeli lijevi i desni dio zapisa na traci:

$$\alpha_1 \ q \ \alpha_2$$

gdje je q stanje ($q \in Q$), α_1 je zapis na traci koji se nalazi lijevo od glave za čitanje i pisanje, a α_2 je zapis na traci koji se nalazi desno od glave za čitanje i pisanje ($\alpha_1, \alpha_2 \in \Gamma^*$). TS čita krajnje lijevi znak niza α_2 . Jednoznačnost označavanja konfiguracije postiže se definiranjem zasebnih oznaka za stanja u skupu Q koja su različita od oznaka znakova trake u skupu Γ .

Prijelaz iz konfiguracije u konfiguraciju definira se na sljedeći način. Prepostavimo da je TS u konfiguraciji:

a) $X_1 X_2 \dots X_{i-1} q X_i X_{i+1} \dots X_n$

Neka je zadana funkcija prijelaza:

b) $\delta(q, X_i) = (p, Y, L)$

Ako je $i-1=n$, onda je glava za čitanje postavljena na prazninu i desno od glave za čitanje su sve prazne ćelije. Znak X_i je praznina B .

Ako je $i=1$, onda je glava za čitanje postavljena na krajnju lijevu ćeliju. Budući da glavu nije moguće pomaknuti lijevo od krajnje lijeve ćelije, nema ni sljedeće konfiguracije TS. TS zaustavlja daljnji rad.

Ako je $i>1$, onda TS iz konfiguracije (a) primjenom funkcije prijelaza (b) prelazi u novu konfiguraciju. Prijelaz iz jedne konfiguracije u drugu konfiguraciju zapiše se na sljedeći način:

c) $X_1 X_2 \dots X_{i-1} q X_i X_{i+1} \dots X_n \xrightarrow[M]{} X_1 X_2 \dots X_{i-2} p X_{i-1} Y X_{i+1} \dots X_n$

Zada li se funkcija prijelaza:

d) $\delta(q, X_i) = (p, Y, R)$,

TS iz konfiguracije (a) prelazi u konfiguraciju:

e) $X_1 X_2 \dots X_{i-1} q X_i X_{i+1} \dots X_n \xrightarrow[M]{} X_1 X_2 \dots X_{i-1} Y p X_{i+1} \dots X_n$

Ako je $i-1=n$, onda je niz $X_1 X_2 \dots X_n$ prazni niz. U tom slučaju TS nadopiše znak Y na kraju niza.

Relacija \succ jest refleksivno i tranzitivno okruženje relacije \succ . Slijed prijelaza $q_0 0 0 1 1 \succ \text{XXXXB} q_4$ TS M zadanog u primjeru 4.1 moguće je prikazati korak po korak na sljedeći način:

$$\begin{aligned} q_0 0 0 1 1 &\succ X q_1 0 1 1 \succ X 0 q_1 1 1 \succ X q_2 0 Y 1 \succ q_2 X 0 Y 1 \succ X q_0 0 Y 1 \succ X X q_1 Y 1 \succ \\ &\quad XX Y q_1 1 \succ X X q_2 Y Y \succ X q_2 X Y Y \succ X X q_0 Y Y \succ X X Y q_3 Y \succ X X Y Y q_3 \succ \\ &\quad X X Y Y B q_4 \end{aligned}$$

m

Iraz $J \xrightarrow[m]{} K$ označava da se iz konfiguracije J prelazi u konfiguraciju K primjenom m prijelaza. Za niz 0011 funkcija prijelaza δ primjeni se 13 puta:

$$\begin{array}{c} 13 \\ q_0 0 0 1 1 \succ X X Y Y B q_4. \end{array}$$

TS $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ prihvata jezik:

$$L(M) = \{w \mid w \in \Sigma^* \text{ i } q_0 w \xrightarrow{*} \alpha_1 p \alpha_2, \text{ gdje je } p \in F, \alpha_1 \text{ i } \alpha_2 \text{ su u } \Gamma^*\}$$

Niz w zapisuje se u krajnje lijevim celiama ulazne trake. TS je u početnom stanju q_0 , a glava je postavljena na krajnje lijevi znak niza w koji je zapisan u krajnju lijevu celiu ulazne trake. Uđe li TS u jedno od prihvatljivih stanja, niz w se prihvata. Prošire li se uvjeti prihvatanja niza zahtjevom da se niz prihvata ako i samo ako TS uđe u prihvatljivo stanje i istodobno zaustavi daljni rad, prihvata se ista klasa jezika i ne gubi se na općenitosti definicije prihvatanja niza Turingovim strojem. Ako za zadano prihvatljivo stanje nema definiranih prijelaza niti za jedan znak trake, onda TS zaustavi rad i prihvati niz. U dalnjem tekstu podrazumijeva se da TS prihvata niz ako i samo ako TS ulazkom u prihvatljivo stanje zaustavi svoj daljnji rad. U slučaju da se niz ne prihvata, dozvoljava se da TS nikad ne stane.

U primjeru 4.1 zadan je TS M i opisan je njegov rad za nizove 0011 i 001. Budući da je

$q_0 0 0 1 1 \succ XXYYBq_4$ i q_4 je u skupu prihvatljivih stanja F , niz 0011 se prihvata. Za prihvatljivo stanje q_4 nije definirana funkcija prijelaza niti za jedan znak trake i zato se rad TS M zaustavlja. Budući da je

$q_0 0 0 1 \succ XYq_1$ i q_1 nije u skupu prihvatljivih stanja F , niz 001 se ne prihvata. Za stanje q_1 i oznaku prazne celiye nije definirana funkcija prijelaza i zato se rad TS zaustavlja.

Turingovi strojevi prihvataju klasu *rekurzivno prebrojivih jezika*. Naziv "prebrojivi" jezici dobili su na temelju sljedećeg svojstva: za bilo koji rekurzivno prebrojivi jezik moguće je izgraditi TS koji ispisuje (nabraja) sve nizove tog jezika. Naziv "rekurzivni" uveden je prije pojave suvremenih digitalnih računala, a ima slično značenje kao i pojam rekurzije u današnjem računarstvu. Kontekstno neovisni jezici su pravi podskup skupa rekurzivno prebrojivih jezika. Postoje rekurzivno prebrojivi jezici za koje nije moguće izgraditi TS M koji uvijek stane za svaki ulazni niz. Ako je niz $w \in L(M)$, onda TS M stane i prihvati niz w . Međutim, ako niz w nije u jeziku $L(M)$, moguće je da TS nikad ne stane.

Klasa jezika za koju je moguće izgraditi TS koji uvijek stane za svaki ulazni niz, naziva se klasa *rekurzivnih jezika*. Klasa rekurzivnih jezika jest pravi podskup klase rekurzivno prebrojivih jezika.

Računanje cjelobrojnih funkcija Turingovim strojem

Turingov stroj koristi se za računanje vrijednosti cjelobrojnih funkcija. Uobičajeno je da se cijeli brojevi zapisuju kao niz znakova 0. Broj znakova 0 označava vrijednost cijelog broja. Cijeli broj $i \geq 0$ zapiše se kao niz 0^i . Ima li cjelobrojna funkcija k argumenata i_1, i_2, \dots, i_k , oni se na traci odvajaju znakom 1. Argumenti se zapišu na traku na sljedeći način: $0^{i_1} 1 0^{i_2} 1 \dots 1 0^{i_k}$.

Zaustavi li se TS, a na traci je zapisano 0^m , vrijednost funkcije $f(i_1, i_2, \dots, i_k)$ jednaka je m bez obzira u kojem je stanju TS. TS može računati različite funkcije koje mogu imati različiti broj argumenata. Nije nužno da funkcija sa k argumenata ima definiranu vrijednost za sve argumenate i_1, i_2, \dots, i_k , odnosno moguće je da TS nikad ne stane.

Funkcije koje je moguće izračunati Turingovim strojem nazivaju se *parcijalno rekurzivne funkcije*. Rekurzivno prebrojivi jezici i parcijalno rekurzivne funkcije su analogni u smislu da se prihvataju, odnosno da se računaju Turingovim strojem koji nužno ne mora uvijek stati za svaki ulazni niz.

Ako je funkcija $f(i_1, i_2, \dots, i_k)$ definirana za sve argumente i_1, i_2, \dots, i_k , onda se kaže da je funkcija f *potpuno rekurzivna funkcija*. Rekurzivni jezici i potpuno rekurzivne funkcije su analogni u smislu da se prihvataju, odnosno da se računaju Turingovim strojem koji uvijek stane za svaki ulazni niz. Množenje, $n!$ i 2^n su primjeri potpuno rekurzivnih funkcija.

Primjer 4.2. Vrijednost funkcije $m+n$ definira se na sljedeći način. Ako je $m \geq n$, onda je vrijednost funkcije $m+n$ jednaka razlici brojeva $m-n$. Ako je $m < n$, onda je vrijednost funkcije $m+n$ jednaka 0.

TS M računa vrijednost funkcije $m+n$ na sljedeći način. Na početku rada na traku se zapiše niz:

$$0^m 1 0^n.$$

TS M zamijeni krajnje lijevi znak 0 prazninom B :

$$B 0^{m-1} 1 0^n,$$

te pomakne glavu u desno preko preostalih $m-1$ znakova 0 sve do znaka 1. Traži se krajnje lijevi znak 0 koji slijedi iza znaka 1. Pronađeni znak 0 mijenja se u znak 1:

$$B 0^{m-1} 1 1 0^{n-1}.$$

TS M promijeni smjer gibanja glave u lijevo. Glava se miče u lijevo na krajnje desnu prazninu. Pročita li TS M prazninu, glava mijenja smjer gibanja ponovno u desno. Krajnje lijevi znak 0 mijenja se u prazninu B :

$$BB 0^{m-2} 1 1 0^{n-1}.$$

Glava se miče u desno do krajnje lijevog znaka 1, prelazi se preko znakova 1 do krajnje lijevog znaka 0, te se pronađeni znak 0 zamjeni znakom 1:

$$BB 0^{m-2} 1 1 1 0^{n-2}.$$

Glava ponovno mijenja smjer gibanja i miče se u lijevo do krajnje desne praznine B . TS M nastavlja opisani postupak sve dok se ne dogodi jedan od dva moguća slučaja:

- a) Micanjem u desno preko znakova 1 TS M ne pronađe znak 0, već prazninu. U tom slučaju je $m \geq n$ i svih n znakova 0 promijenjeno je u znak 1. Na početku trake $n+1$ znakova 0 promijenjeno je u praznine B . Traka ima sljedeći sadržaj:

$$B^{n+1} 0^{m-n-1} 1^{n+1}.$$

TS M miče glavu u lijevo, mijenja $n+1$ znakova 1 u praznine B , prijeđe preko $m-n-1$ znakova 0 i mijenja krajnje desnu prazninu u 0:

$$B^n 0^{m-n}.$$

Na traci ostaje zapisano $m-n$ znakova 0, što znači da je vrijednost funkcije $m+n$ jednaka $m-n$.

- b) Micanjem u lijevo preko znakova 1 TS M pročita prazninu B , pomakne glavu u desno i umjesto znaka 0 pročita znak 1. U tom slučaju je $m \leq n$ i svih m znakova 0 na početku trake zamijenjeno je prazninama:

$$B^m 1^{m+1} 0^{n-m}.$$

TS M zamijeni preostale znakove 0 i 1 prazninama B . Vrijednost funkcije $m+n$ jednaka je 0. Na traci ostaju zapisane samo praznine:

$$B^{m+n+1}.$$

TS M gradi se na sljedeći način:

$$M = (\{q_0, q_1, q_2, \dots, q_6\}, \{0, 1\}, \{0, 1, B\}, \delta, q_0, B, \emptyset)$$

Funkcija prijelaza δ je:

1) $\delta(q_0, 0)=(q_1, B, R)$

Krajnje lijevi znak 0 zamijeni se prazninom B i prelazi se iz stanja q_0 u stanje q_1 .

2) $\delta(q_1, 0)=(q_1, 0, R)$

$\delta(q_1, 1)=(q_2, 1, R)$

Prelazi se preko znakova 0 i traži se krajnje lijevi znak 1. Pronađe li se znak 1, prelazi se u stanje q_2 .

3) $\delta(q_2, 1)=(q_2, 1, R)$

$\delta(q_2, 0)=(q_3, 1, L)$

Prelazi se preko znakova 1 i traži se krajnje lijevi znak 0. Pronađe li se znak 0, on se mijenja u znak 1 i prelazi se u stanje q_3 .

4) $\delta(q_3, 0)=(q_3, 0, L)$

$\delta(q_3, 1)=(q_3, 1, L)$

$\delta(q_3, B)=(q_0, B, R)$

Glava se miče u lijevo do krajnje desne praznine B . Stanje se mijenja u q_0 i rad TS M započinje od koraka (1).

5) $\delta(q_2, B)=(q_4, B, L)$

$\delta(q_4, 1)=(q_4, B, L)$

$\delta(q_4, 0)=(q_4, 0, L)$

$\delta(q_4, B)=(q_6, 0, R)$

Zadani prijelazi odgovaraju prethodno opisanom slučaju (a). Ne pronađe li se u koraku (3) znak 0, već praznina B , stanje se promjeni u q_4 . Glava se miče u lijevo, svi znakovi 1 mijenjaju se u praznine B , dok se znakovi 0 ne mijenjaju. Nailaskom na krajnje desnu prazninu ona se mijenja u znak 0. Na traci ostaje $n-m$ znakova 0, odnosno vrijednost funkcije je $m+n$ jednaka je $m-n$.

6) $\delta(q_0, 1)=(q_5, B, R)$

$\delta(q_5, 0)=(q_5, B, R)$

$\delta(q_5, 1)=(q_5, B, R)$

$\delta(q_5, B)=(q_6, B, R)$

Zadani prijelazi odgovaraju prethodno opisanom slučaju (b). Ne pronađe li se u koraku (1) znak 0, već znak 1, svi su znakovi 0 na početku trake zamijenjeni prazninama B . Prelazi se u stanje q_5 , a preostali znakovi 0 i 1 zamijene se prazninama B . Vrijednost funkcije je $m+n$ jednaka je 0.

Želi li se izračunati $2+1$, na ulaznu traku TS M zapiše se niz 0010. Krajnja lijeva dva znaka 0 predstavljaju broj 2, dok znak 0 na desnoj strani predstavlja broj 1. Vrijednost funkcije $m+n$ jednaka je 1. Sljedeći slijed konfiguracija pokazuje na koji način TS M računa vrijednost funkcije $2+1$:

$$\begin{array}{l} q_00010 \succ Bq_1010 \succ B0q_110 \succ B01q_20 \succ B0q_311 \succ Bq_3011 \succ q_3B011 \succ Bq_0011 \\ \quad \searrow \\ BBq_111 \succ BB1q_21 \succ BB11q_2 \succ BB1q_41 \succ BBq_41 \succ Bq_4 \succ B0q_6 \end{array}$$

Na traci ostaje zapisan jedan znak 0, što znači da je vrijednost funkcije $2+1$ jednaka 1.

Sljedeći slijed konfiguracija pokazuje računanje funkcije $1+2$. Na početku rada na traku se zapiše niz 0100. Budući da je vrijednost funkcije $1+2$ jednaka 0, na traci ostaju samo praznine B :

$$\begin{array}{l} q_00100 \succ Bq_1100 \succ B1q_200 \succ Bq_3110 \succ q_3B110 \succ \\ Bq_0110 \succ BBq_510 \succ BBBq_5 \succ BBBBq_5 \succ BBBBBq_6 \end{array}$$

4.1.2 Metode izrade Turingovog stroja

Razlože li se oznake stanja i oznake znakova trake na više komponenata, moguće je znatno olakšati izradu TS. Složeni način označavanja stanja i znakova trake ne mijenja osnovnu definiciju TS. U nastavku odjeljka opisuje se primjena složenih oznaka u gradnji TS.

Višekomponentna oznaka stanja

Umjesto jedinstvene oznake stanja, koristi se složena oznaka koja se sastoji od više komponenata. Komponente složenih oznaka stanja pišu se u uglatim zagradama $[q_1, q_2, \dots, q_n]$, gdje je q_i komponenta složene oznake stanja, $1 \leq i \leq n$. Ako je konačan broj komponenata q_i i ako je konačan skup njihovih vrijednosti, onda je i konačan kardinalni broj skupa složenih stanja što zadovoljava definiciju osnovnog modela TS.

Komponente oznake stanja koriste se za različite primjene. Na primjer, u komponente oznake stanja moguće je spremiti podatak, više komponenata moguće je iskoristiti za pomak znakova na traci, itd.

Komponente stanja dijele se u dvije grupe: upravljačke komponente i radne komponente. Upravljačke komponente upravljaju radom TS, dok se radne komponente koriste za *pohranu podataka*. Na primjer, u radnu komponentu moguće je spremati znak ulazne trake. Budući da je skup znakova trake konačan skup, radna komponenta poprima konačni skup vrijednosti. Nakon što se pročitani znak ulazne trake spremi u radnu komponentu stanja, glava se miče na odgovarajuću ćeliju na traci. Spremljeni znak uspoređuje se sa znakom na traci, ili se zapiše u novu ćeliju, itd.

Primjer 4.3. U jeziku L su nizovi u kojima se krajne lijevi znak niza ne nalazi niti na jednom drugom mjestu u nizu. Skup ulaznih znakova sastoји se od znakova 0 i 1. Potrebno je izgraditi TS M koji prihvaca zadani jezik L .

Složena stanja TS M $[q, a]$ imaju dvije komponente: upravljačku komponentu q i radnu komponentu a . U radnu komponentu a spremi se pročitani znak ulazne trake. Upravljačka komponenta stanja poprima dvije vrijednosti: q_0 i q_1 . U stanju q_0 TS M čita krajne lijevi znak niza i spremi ga u radnu komponentu stanja. U stanju q_1 TS M čita ostatak niza i uspoređuje pročitane znakove sa znakom spremšnjim u radnoj komponenti stanja. Budući da upravljačka komponenta poprima dvije vrijednosti q_0 i q_1 , a radna komponenta poprima tri vrijednosti 0, 1 i B , skup složenih stanja Q ima šest elemenata $Q = \{[q_0, B], [q_0, 0], [q_0, 1], [q_1, B], [q_1, 0], [q_1, 1]\}$.

Stanje $[q_0, B]$ jest početno stanje TS M . Pročita li TS znak 0, stanje se promjeni u $[q_1, 0]$. Upravljačka komponenta stanja q_1 označava da je pročitan krajnje lijevi znak niza i da se nastavlja s uspoređbom spremšnjeg znaka u radnoj komponenti sa znakovima na traci. Budući da je 0 krajnje lijevi znak niza, radna komponenta poprima vrijednost 0. Ako je krajnje lijevi znak niza 1, onda TS mijenja stanje u $[q_1, 1]$. Ne nalazi li se u ostatku niza niti jedan znak koji je jednak krajnje lijevom znaku niza, TS promjeni stanje u prihvatljivo stanje $[q_1, B]$.

Jezik L prihvaca sljedeći TS M :

$$M = (Q, \{0, 1\}, \{0, 1, B\}, \delta, [q_0, B], B, \{[q_1, B]\})$$

gdje je Q prethodno opisan skup složenih stanja, a funkcija prijelaza δ zadaje se na sljedeći način:

- 1) $\delta([q_0, B], 0) = ([q_1, 0], 0, R)$
 $\delta([q_0, B], 1) = ([q_1, 1], 1, R)$

U početnom stanju $[q_0, B]$ TS M mijenja upravljačku komponentu stanja u q_1 , dok radna komponenta poprima vrijednost pročitanog znaka. Znak na traci se ne mijenja, a glava se miče u desno.

- 2) $\delta([q_1, 0], 1) = ([q_1, 0], 1, R)$
 $\delta([q_1, 1], 0) = ([q_1, 1], 0, R)$

Pročitani znakovi s ulazne trake uspoređuju se sa znakom koji je spremjen u radnoj komponenti stanja. Nije li spremjeni znak jednak znaku koji je zapisan na traci, TS ostaje u istom stanju, miče glavu u desno i nastavlja s usporednjom znakova. Ako je pročitani znak jednak znaku koji je spremjen u radnoj komponenti stanja, onda se rad TS zaustavlja. Prijelaz za stanje $[q_1, 0]$ i znak 0, te prijelaz za stanje $[q_1, 1]$ i znak 1, nisu definirani. Budući da stanja $[q_1, 0]$ i $[q_1, 1]$ nisu u skupu prihvatljivih stanja F , niz se ne prihvata.

- 3) $\delta([q_1, 0], B) = ([q_1, B], B, L)$
 $\delta([q_1, 1], B) = ([q_1, B], B, L)$

Ne nalazi li se na niti jednom drugom mjesu u nizu krajnje lijevi znak, pomakom glave u desno pročita se praznina B . TS M prelazi u prihvatljivo stanje $[q_1, B]$. Budući da za prihvatljivo stanje $[q_1, B]$ nema definiran niti jedan prijelaz, rad TS se zaustavlja i niz se prihvata.

Radne komponente oznake stanja moguće je iskoristiti za *pomak znakova* ulazne trake u desno ili u lijevo. Pomak znakova ulazne trake izvodi se u više koraka. U pojedinim koracima prenosi se više znakova ovisno o broju radnih komponenata stanja. Svaka radna komponenta prenosi jedan znak. Nakon što se popune sve radne komponente pročitanim znakovima trake, TS M miče glavu u odgovarajuću stranu prema odredišnom mjestu. Spremljeni znakovi zapišu se na odredišno mjesto, glava TS M vraća se na izvorišno mjesto niza i započinje prijenos sljedeće grupe znakova.

Primjer 4.4 opisuje TS koji omogućuje učinkoviti pomak znakova za n čelija primjenom n radnih komponenata. U danom primjeru TS s dvije radne komponente miče niz za dvije čelije u desno.

Primjer 4.4. Gradi se TS $M=(Q, \Sigma, \Gamma, \delta, q_0, B, F)$ koji pomiče znakove za dva mesta u desno. Kraj niza označen je praznom čelijom. Složena stanja imaju tri komponente $[q, R_1, R_2]$, jednu upravljačku i dvije radne. Upravljačka komponenta q poprima vrijednosti q_1 i q_2 , dok radne komponente R_1 i R_2 poprimaju vrijednosti znakova trake Γ . TS M započinje pomak znakova u stanju $[q, B, B]$. TS M koristi sljedeće funkcije prijelaza za potrebe pomaka znakova trake:

- 1) $\delta([q_1, B, B], A_1) = ([q_1, B, A_1], X, R)$ za znakove A_1 iz skupa $\Gamma - \{B, X\}$.

TS M čita znak trake A_1 , spremi znak A_1 u drugu radnu komponentu stanja, zapiše znak X na traku i pomakne glavu u desno. Znak X izabere se tako da nije jednak niti jednom znaku trake koji je potrebno pomaknuti.

- 2) $\delta([q_1, B, A_1], A_2) = ([q_1, A_1, A_2], X, R)$ za znakove A_1 i A_2 iz skupa $\Gamma - \{B, X\}$.

TS M spremi vrijednost druge radne komponente A_1 u prvu radnu komponentu, pročitani znak A_2 spremi u drugu radnu komponentu, zapiše znak X na traku i pomakne glavu u desno.

- 3) $\delta([q_1, A_1, A_2], A_3) = ([q_1, A_2, A_3], A_1, R)$ za znakove A_1 , A_2 i A_3 iz skupa $\Gamma - \{B, X\}$.

TS M čita znak trake A_3 , spremi znak A_3 u drugu radnu komponentu, vrijednost druge radne komponente A_2 spremi u prvu radnu komponentu, znak A_1 spremjen u prvoj radnoj komponenti zapiše na traku i pomakne glavu u desno. Znak A_1 zapiše se dvije čelije desno od izvorišnog mesta.

- 4) $\delta([q_1, A_1, A_2], B) = ([q_1, A_2, B], A_1, R)$ za znakove A_1 i A_2 iz skupa $\Gamma - \{B, X\}$.

Čitanjem krajnje lijeve praznine B , druga radna komponenta poprimi vrijednost B , prva radna komponenta poprimi vrijednost treće komponente A_2 , a na traku se zapiše znak A_1 spremjen u prvoj radnoj komponenti.

- 5) $\delta([q_1, A_1, B], B) = ([q_2, B, B], A_1, L)$ za znakove A_1 iz skupa $\Gamma - \{B, X\}$.

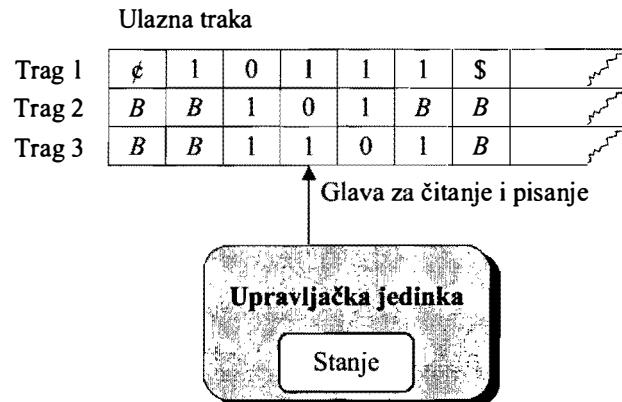
Znak A_1 spremljen u prvoj radnoj komponenti zapiše se na traku. Ovim prijelazom završava pomak znakova. Obje radne komponente poprime vrijednosti praznine B , upravljačka komponenta poprimi vrijednost q_2 i glava mijenja smjer gibanja u lijevo.

- 6) $\delta([q_2, B, B], A) = ([q_2, B, B], A, L)$ za znakove A iz skupa $\Gamma - \{B, X\}$.

Glava se miče u lijevo sve do znaka X . Čitanjem znaka X , TS M prelazi u stanje koje omogućuje daljnji rad TS.

Višekomponentni znakovi trake

Slično oznakama složenih stanja, znakovi trake i ulazni znakovi mogu imati više komponenata: $A_j = [a_1, a_2, \dots, a_n]$, gdje je A_j složeni znak s komponentama a_i , $1 \leq i \leq n$. Ako je konačan broj komponenata a_i i ako je konačan skup njihovih vrijednosti, onda su konačni kardinalni broj skupa složenih znakova trake i kardinalni broj skupa složenih ulaznih znakova što zadovoljava definiciju osnovnog modela TS.



Zapišu li se pojedine komponente složenih znakova trake u zasebne tragove ulazne trake, lakše je pratiti rad TS. Broj tragova ulazne trake jednak je broju komponenata složenih znakova trake. Na slici 4.2 dan je primjer TS koji ima složene znakove trake. Budući da složeni znakovi imaju tri komponente, ulazna traka sastoji se od tri traga.

Slika 4.2: Model Turingovog stroja sa tri traga ulazne trake

Primjer 4.5. Model sa tri traga ulazne trake prikazan na slici 4.2 osnovica je za izradu TS koji prihvaca jezik u kojem su prim brojevi. Prvi trag je ulazni trag i on sadrži binarni broj. Binarni broj ograđen je graničnicima ¢ i \$. Druga dva traga su pomoćna. Pomoćni tragovi koriste se za ispitivanje binarnog broja. TS ispituje da li je binarni broj zapisan na ulaznom tragu prim broj. Ulagni znakovi ¢, 0, 1 i \$ zapisani su na traci primjenom složenih označki s tri komponente: $[\$, B, B]$, $[0, B, B]$, $[1, B, B]$ i $[$, B, B]$. Praznina se označava složenom označkom $[B, B, B]$.

Na početku rada TS zapiše binarni broj 2 na prvi pomoćni trag i prepiše binarni broj sa ulaznog traga na drugi pomoćni tragu. TS dijeli broj zapisan na drugom pomoćnom tragu s binarnim brojem zapisanim na prvom pomoćnom tragu. Rezultat dijeljenja zapiše se na drugi pomoćni tragu. Dijeljenje se ostvaruje primjenom uzastopnog oduzimanja. Binarni broj zapisan na prvom pomoćnom tragu oduzima se od broja zapisanog na drugom pomoćnom tragu. Razlika brojeva zapisuje se na drugi pomoćni tragu. Moguća su tri različita ishoda dijeljenja brojeva zapisanih na pomoćnim tragovima i usporedbe brojeva zapisanih na prvom pomoćnom tragu i na ulaznom tragu:

- 1) Ako ostatak dijeljenja nije jednak 0 i ako broj na prvom pomoćnom tragu nije jednak broju na ulaznom tragu, broj na prvom pomoćnom tragu poveća se za 1, prepiše se broj s ulaznog traga na drugi pomoćni trag i postupak dijeljenja se ponavlja s djeliteljem uvećanim za jedan.

- 2) Ako je broj na prvom pomoćnom tragu jednak broju na ulaznom tragu, onda se rad TS zaustavlja. Budući da ostatak dijeljenja nije jednak 0 niti za jedan broj koji je veći od 1 i manji od zadanog broja, broj zapisan na ulaznom tragu je prim broj.,
- 3) Ako je ostatak dijeljenja jednak 0, onda se rad TS zaustavlja. Budući da je broj na ulaznom tragu djeljiv s brojem zapisanim na prvom pomoćnom tragu, broj zapisan na ulaznom tragu nije prim broj.

Slika 4.2 prikazuje ispitivanje broja 23 (binarno 10111). Broj 23 dijeli se s brojem 5 (binarno 101) koji je zapisan na prvom pomoćnom tragu. Budući da se broj 5 dva puta oduzeo od broja 23, na drugom pomoćnom tragu zapisan je broj 13 (binarno 1101).

Pomoćni tragovi mogu imati posebnu namjenu. Na primjer, za potrebe označavanja znakova na ulaznoj traci koristi se *označni trag*. U označnom tragu ćelija je prazna, ili je u ćeliji zapisan znak \checkmark . Nakon što se završi s ispitivanjem ili uspoređivanjem određenog znaka, u ćeliju označnog traga zapiše se znak \checkmark . Označavanje znakova na traci posebice je korisno pomagalo za izradu TS koji prihvataju jezike u kojima se dijelovi niza ponavljaju:

$$\{ww \mid w \in \Sigma^*\}, \quad \{wcy \mid w, y \in \Sigma^*, w \neq y\}, \quad \{ww^R \mid w \in \Sigma^*\}.$$

i za izradu TS koji prihvataju jezike u kojima se duljine dijelova niza međusobno uspoređuju:

$$\{a^i b^i \mid i \geq 1\} \quad i \quad \{a^i b^j c^k \mid i \neq j \text{ ili } j \neq k\}.$$

Primjer 4.6. Zadan je TS $M=(Q, \Sigma, \Gamma, \delta, q_0, B, F)$ koji prihvata jezik $L(M)=\{wcw \mid w \in (\mathbf{a+b})^+\}$. Složena stanja $[q, d]$ imaju upravljačku komponentu q i radnu komponentu d . Skup složenih stanja Q jednak je:

$$Q = \{[q, d] \mid \begin{array}{l} q \text{ poprima vrijednost } q_1, q_2, \dots \text{ ili } q_9, \\ d \text{ poprima vrijednost ulaznih znakova } a, b \text{ ili praznine } B \end{array}\}.$$

Ulazna traka ima dva traga: ulazni trag i označni trag. Složeni znakovi trake $[X, d]$ imaju ulaznu komponentu d i označnu komponentu X . Skup složenih znakova trake Γ jednak je:

$$\Gamma = \{[X, d] \mid \begin{array}{l} X \text{ poprima vrijednost } B \text{ ili } \checkmark, \\ d \text{ poprima vrijednost } a, b, c \text{ ili praznine } B \end{array}\}.$$

Budući da ulazna traka ima dva traga, za zapis ulaznih znakova a, b i c koriste se dvije komponente:

$$\Sigma = \{[B, d] \mid d \text{ poprima vrijednost } a, b \text{ ili } c\}.$$

Početno stanje TS jest $q_0=[q_1, B]$. Skup prihvatljivih stanja jest $F=\{[q_9, B]\}$. Praznina se označava složenom oznakom $[B, B]$. U funkciji prijelaza δ znakovi d i e predstavljaju znak a ili b :

1) $\delta([q_1, B], [B, d]) = ([q_2, d], [\checkmark, d], R)$

TS M pročita znak d sa trake, spremi pročitani znak d u radnu komponentu stanja, promijeni upravljačku komponentu stanja u q_2 , zapiše oznaku \checkmark u označni trag i pomakne glavu u desno.

2) $\delta([q_2, d], [B, e]) = ([q_2, d], [B, e], R)$

TS M miče glavu u desno sve do znaka c .

3) $\delta([q_2, d], [B, c]) = ([q_3, d], [B, c], R)$

Čitanjem znaka c mijenja se upravljačka komponenta stanja u q_3 .

4) $\delta([q_3, d], [\vee, e]) = ([q_3, d], [\vee, e], R)$

TS M miče glavu u desno preko svih označenih znakova koji su na desnoj strani od znaka c . Traži se krajnje lijevi znak desno od znaka c koji nije označen.

5) $\delta([q_3, d], [B, d]) = ([q_4, B], [\vee, d], L)$

TS M uspoređuje krajnje lijevi neoznačeni znak na desnoj strani od znaka c sa znakom spremlijenim u radnoj komponenti stanja. Ako su pročitani i spremlijeni znak jednaki, onda se znak na traci označava, upravljačka komponenta stanja mijenja se u q_4 i glava mijenja smjer kretanja u lijevo. Nisu li znakovi jednaki, funkcija prijelaza nije definirana, TS M zaustavlja rad i niz se ne prihvata.

6) $\delta([q_4, B], [\vee, d]) = ([q_4, B], [\vee, d], L)$

TS M miče glavu u lijevo preko svih označenih znakova na desnoj strani znaka c .

7) $\delta([q_4, B], [B, c]) = ([q_5, B], [B, c], L)$

TS M čita znak c , mijenja upravljačku komponentu stanja u q_5 i nastavlja s micanjem glave u lijevo.

8) $\delta([q_5, B], [B, d]) = ([q_6, B], [B, d], L)$

Nije li označen krajnje desni znak na lijevoj strani od znaka c , TS M mijenja upravljačku komponentu stanja u q_6 i nastavlja s micanjem glave u lijevo do krajnje desnog označenog znaka na lijevoj strani od znaka c .

9) $\delta([q_6, B], [B, d]) = ([q_6, B], [B, d], L)$

TS M miče glavu u lijevo preko neoznačenih znakova.

10) $\delta([q_6, B], [\vee, d]) = ([q_1, B], [\vee, d], R)$

TS M čita krajnje desni označeni znak na lijevoj strani od znaka c , mijenja smjer gibanja glave u desno i mijenja upravljačku komponentu stanja u q_1 . Nastavlja se od koraka (1).

11) $\delta([q_5, B], [\vee, d]) = ([q_7, B], [\vee, d], R)$

U koraku (7) TS M čita znak c i mijenja stanje u $[q_5, B]$. Ako je označen znak koji je neposredno lijevo do znaka c , upravljačka komponenta stanja mijenja se u q_7 i glava mijenja smjer gibanja u desno. Budući da su svi znakovi na lijevoj strani od znaka c označeni, TS M provjerava da li su označeni svi znakovi na desnoj strani.

12) $\delta([q_7, B], [B, c]) = ([q_8, B], [B, c], R)$

TS M čita znak c , mijenja upravljačku komponentu stanja u q_8 i nastavlja s micanjem glave u desno.

13) $\delta([q_8, B], [\vee, d]) = ([q_8, B], [\vee, d], R)$

TS M miče glavu u desno preko označenih znakova. Pročita li TS M neoznačen znak, za stanje $[q_8, B]$ i neoznačeni znak ne postoji funkcija prijelaza i TS M se zaustavlja. Budući da $[q_8, B]$ nije u skupu prihvatljivih stanja F , niz se ne prihvata.

14) $\delta([q_8, B], [B, B]) = ([q_9, B], [B, B], L)$

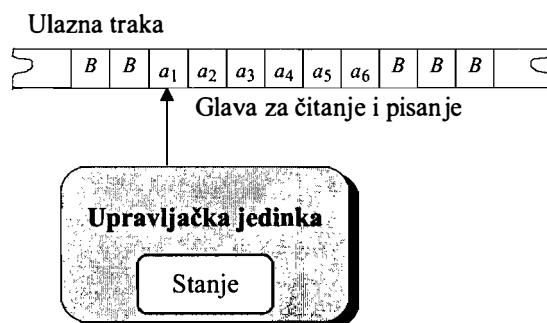
Pročita li TS M prazninu $[B, B]$, nema niti jedne neoznačene ćelije desno od znaka c . TS M mijenja stanje u prihvatljivo stanje $[q_9, B]$, rad TS se zaustavlja i niz se prihvata.

4.1.3 Prošireni modeli Turingovog stroja

Osnovni model TS moguće je proširiti na više načina. Šest osnovnih načina proširenja osnovnog modela TS su: TS s dvostranom beskonačnom trakom, TS s višestrukim trakama, nedeterministički TS, TS s višedimenzionalnim ulaznim poljem, TS s više glave za čitanje i pisanje, te neizravni TS. Dva TS su istovjetna ako i samo ako prihvataju isti jezik. U odjeljcima koji slijede detaljno su opisane razne inačice TS i pokazana je njihova istovjetnost s osnovnim modelom TS. Za konstrukciju istovjetnog osnovnog modela TS koriste se metode izrade TS opisane u odjeljku 4.1.2.

Za rješavanje mnogih problema jednostavnije je izgraditi jedan od proširenih modela TS. Iako je rješenje proširenim modelom TS jednostavno i učinkovito, istovjetni osnovni model TS može biti složen. Na primjer, za jedan prijelaz proširenog modela TS, moguće je da istovjetni osnovni model TS izvodi daleko veći broj prijelaza.

TS s dvostranom beskonačnom trakom



Slika 4.3: Model Turingovog stroja s dvostranom beskonačnom trakom

Za razliku od osnovnog modela TS, ulazna traka jest beskonačna na lijevu stranu i na desnu stranu. Nema krajnje lijeve ni krajnje desne ćelije. Model TS s dvostranom beskonačnom trakom prikazan je na slici 4.3. Osim u dva slučaja, relacija \succ_M jednaka je relaciji osnovnog modela.

Osnovni model TS nema pomaka lijevo od krajnje lijeve ćelije ulazne trake. Budući da prošireni model TS ima beskonačnu traku na lijevu stranu, u slučaju pomaka lijevo od krajnje lijevog znaka X , glava za čitanje i pisanje postavlja se na praznu ćeliju:

- 1) Ako je $\delta(q, X) = (p, Y, L)$, onda je $q \ X \alpha \succ_M p \ B \ Y \alpha.$

Za razliku od osnovnog modela TS koji funkcijom $\delta(q, X) = (p, B, R)$ prelazi iz konfiguracije $qX\alpha$ u konfiguraciju $Bp\alpha$, gdje je praznina B lijevo od stanja p , za TS s dvostranom beskonačnom trakom vrijedi:

- 2) Ako je $\delta(q, X) = (p, B, R)$, onda je $q \ X \alpha \succ_M p \ \alpha.$

U novoj konfiguraciji nema oznake praznine B lijevo od oznake stanja p .

TS s dvostranom beskonačnom trakom istovjetan je osnovnom modelu TS.

Dokaz istovjetnosti sastoji se od dva dijela. U prvom dijelu dokaza potrebno je za zadani osnovni model TS M_1 izgraditi TS M_2 s dvostranom beskonačnom trakom. Ćelija TS M_2 koja se nalazi lijevo od početnog položaja glave za čitanje označi se posebnim znakom. TS M_2 simulira rad TS M_1 . U trenutku kad glava za čitanje dođe do ćelije označene posebnim znakom, TS M_2 se zaustavlja i niz se ne prihvata. U tom trenutku TS M_1 pokušava pomaknuti glavu za čitanje lijevo od krajnje lijeve ćelije.

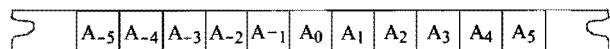
U drugom dijelu dokaza potrebno je za zadani TS M_2 s dvostranom beskonačnom trakom izgraditi osnovni model TS M_1 . Ulazna traka osnovnog modela TS M_1 podijeli se na dva traga. Ćelije TS M_2 s dvostranom beskonačnom trakom prikazane su na slici 4.4a, a ćelije onovnog modela TS M_1 prikazane su na slici 4.4b. Početni položaj glave za čitanje TS M_2 s dvostranom beskonačnom trakom je ćelija A_0 . Krajnje lijeva ćelija TS M_1 označena je znakom \notin koji je zapisan u drugom tragu. U gornji trag ulazne trake TS M_1 spreme se sadržaji ćelija TS M_2 koje se nalaze desno od početnog položaja glave A_0 , a u donji trag ulazne trake TS M_1 spreme se sadržaji ćelija TS M_2 koje su lijevo od početnog položaja glave A_0 .

Sadržaji ćelija TS M_1 u donjem tragu spreme se obrnutim redoslijedom od redoslijeda pohrane na traci TS M_2 . Na primjer, na slici 4.4b ćelija A_{-2} je desno od ćelije A_{-1} , a ne lijevo kao što je to na slici 4.4a. Ovisno o položaju glave TS M_2 u odnosu na početni položaj A_0 , TS M_1 koristi gornji ili donji trag svoje trake. Ako TS M_1 koristi gornji trag, onda se glava TS M_1 miče u istom smjeru kao i glava TS M_2 . Koristi li TS M_1 donji trag, glava TS M_1 miče se u suprotnom smjeru od glave TS M_2 .

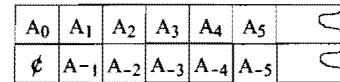
TS $M_1 = (Q_1, \Sigma_1, \Gamma_1, \delta_1, q_1, B_1, F_1)$ simulira rad TS $M_2 = (Q_2, \Sigma_2, \Gamma_2, \delta_2, q_2, B, F_2)$ na sljedeći način:

- 1) Ulazni znakovi $[a, B]$ iz Σ_1 sadrže ulazni znak a u gornjem tragu i prazninu B u donjem tragu, gdje je $a \in \Sigma_2$.
- 2) Praznina se označava znakom $B_1 = [B, B]$.
- 3) Sva stanja u skupu Q_1 osim početnog stanja q_1 imaju dvije komponente. U prvu komponentu spreme se stanje TS M_2 , a druga komponenta jest oznaka U ili D ovisno o tome da li TS M_1 koristi gornji ili donji trag. Ako TS M_1 koristi gornji trag, onda je stanje TS M_1 jednako $[q, U]$, gdje je $q \in Q_2$. Koristi li TS M_1 donji trag, stanje TS M_1 jest $[q, D]$, gdje je $q \in Q_2$.
- 4) Znakovi trake u skupu Γ_1 su oblika $[X, Y]$, gdje su X i Y iz skupa Γ_2 . Znak Y može biti \notin .
- 5) $F_1 = \{[q, U], [q, D] \mid q \text{ jest u skupu } F_2\}$.
- 6) Funkcija δ_1 definira se na sljedeći način:
 - a) Ako je $\delta_2(q_2, a) = (q, X, R)$, onda se za bilo koji ulazni znak $a \in (\Sigma_2 \cup \{B\})$ definira prijelaz:

$$\delta_1(q_1, [a, B]) = ([q, U], [X, \notin], R), \text{ gdje su } q_1 \text{ i } q_2 \text{ početna stanja TS } M_1 \text{ i } TS M_2.$$



(a) Ulazna traka TS M_2



(b) Ulazna traka TS M_1

Slika 4.4: Ulazne trake TS M_1 i M_2

Izvodi li TS M_2 prvi pomak glave u desno, TS M_1 zapiše znak \notin u donji dio traga da označi krajnju lijevu ćeliju. U gornji trag TS M_1 zapiše isti znak X kao i TS M_2 . Vrijednost prve komponente novog stanja TS M_1 jest stanje q u koje prelazi TS M_2 . Budući da TS M_2 miče glavu u desno, druga komponenta U označava da TS M_1 koristi gornji trag.

- b) Ako je $\delta_2(q_2, a)=(q, X, L)$, onda se za bilo koji ulazni znak $a \in (\Sigma_2 \cup \{B\})$ definira prijelaz:

$$\delta_1(q_1, [a, B]) = ([q, D], [X, \emptyset], R), \text{ gdje su } q_1 \text{ i } q_2 \text{ početna stanja TS } M_1 \text{ i TS } M_2.$$

Izvodi li TS M_2 prvi pomak glave u lijevo, TS M_1 zapiše znak \emptyset u donji dio traga da označi krajnju lijevu ćeliju. U gornji trag TS M_1 zapiše isti znak X kao i TS M_2 . Vrijednost prve komponente novog stanja TS M_1 jest stanje q u koje prelazi TS M_2 . Budući da TS M_2 miče glavu u lijevo, druga komponenta D označava da TS M_1 koristi donji trag. Iako zadani TS M_2 miče glavu u lijevo, konstruirani TS M_1 miče glavu u desno.

- c) Ako je $\delta_2(q, X)=(p, Z, A)$, onda se za bilo koji znak trake $[X, Y] \in \Gamma_1$, gdje je $Y \neq \emptyset$ i A je L ili R , definira prijelaz:

$$\delta_1([q, U], [X, Y]) = ([p, U], [Z, Y], A).$$

TS M_1 simulira rad TS M_2 na gornjem tragu. Mijenja se samo gornji trag trake, dok donji trag ostaje nepromijenjen. Glava TS M_1 miče se u istom smjeru kao i glava TS M_2 . Promjena gornjeg traga i promjena prve komponente stanja TS M_1 jednaka je promjeni znaka trake i promjeni stanja TS M_2 .

- d) Ako je $\delta_2(q, Y)=(p, Z, \bar{A})$, onda se za bilo koji znak trake $[X, Y] \in \Gamma_1$, gdje je $Y \neq \emptyset$, definira prijelaz:

$$\delta_1([q, D], [X, Y]) = ([p, D], [X, Z], A)$$

Ako je \bar{A} jednako R , onda je A jednako L , odnosno ako je \bar{A} jednako L , onda je A jednako R . TS M_1 simulira rad TS M_2 na donjem tragu. Mijenja se samo donji trag trake, dok gornji trag ostaje nepromijenjen. Glava TS M_1 miče se u suprotnom smjeru od smjera kretanja glave TS M_2 . Promjena donjeg traga i promjena prve komponente stanja TS M_1 jednaka je promjeni znaka trake i promjeni stanja TS M_2 .

- e) Ako je $\delta_2(q, X)=(p, Y, A)$, onda se definira prijelaz:

$$\delta_1([q, U], [X, \emptyset]) = \delta_1([q, D], [X, \emptyset]) = ([p, C], [Y, \emptyset], R).$$

Ako je A jednako R , onda je C jednako U , odnosno ako je A jednako L , onda je C jednako D . Čita li znak \emptyset , TS M_1 je na krajnje lijevoj ćeliji i simulira rad TS M_2 na početnoj ćeliji. TS M_1 nastavlja koristiti gornji ili donji trag (oznaka C) ovisno o smjeru gibanja glave TS M_2 (oznaka A). TS M_1 miče glavu uvijek u desno bez obzira na smjer kretanja glave TS M_2 .

TS s višestrukim trakama

Model TS s višestrukim dvostranim beskonačnim trakama prikazan je na slici 4.5. TS ima k glava za čitanje i pisanje i k dvostrano beskonačnih traka. Upravljačka jedinka TS donosi odluku na temelju dviju grupa parametara:

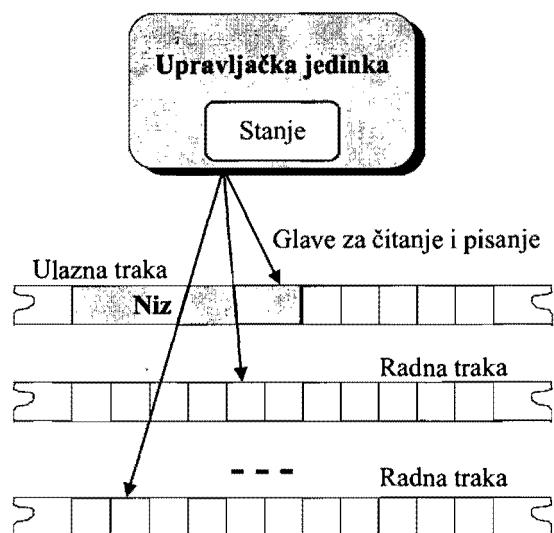
- stanje upravljačke jedinke;
- k pročitanih znakova sa k traka.

Jednim prijelazom TS:

- promijeni stanje;
- zapiše k znakova na k traka;
- pomakne bilo koju od k glava nezavisno u desno ili lijevo.

Na jednu traku, koja se naziva ulazna traka, zapiše se niz koji se ispituje. Sve ostale trake nazivaju se radnim trakama.

TS s višestrukim trakama istovjetan je osnovnom modelu TS.



Slika 4.5: Model TS s višestrukim trakama

Položaj glave 1		X				
Sadržaj trake 1	A ₁	A ₂	- - -	- - -	- - -	A _m
Položaj glave 2				X		
Sadržaj trake 2	B ₁	B ₂	- - -	- - -	- - -	B _m
Položaj glave 3	X					
Sadržaj trake 3	C ₁	C ₂	- - -	- - -	- - -	C _m

Slika 4.6: Simulacija tri trake primjenom šest tragova jedne trake

Zadan je TS M_1 sa k traka koji prihvata jezik L . Istovjetni osnovni model TS M_2 gradi se na sljedeći način. Ulazna traka TS M_2 podijeli se u $2k$ tragova. Za svaku traku TS M_1 odrede se dva traga na traci TS M_2 . Prvi trag koristi se za označavanje položaja glave TS M_1 , a drugi trag koristi se za spremanje sadržaja trake TS M_1 . Sve ćelije u prvom tragu su prazne, osim jedne ćelije u kojoj je zapisan znak X . Znak X označava položaj glave TS M_1 . Raspored i uloga tragova trake TS M_2 prikazani su na slici 4.6.

Stanje TS M_2 ima $k+2$ komponenti. U jednu komponentu stanja spremi se stanje TS M_1 . Druga komponenta ima ulogu brojila. Brojilo broji oznake položaja glava TS M_1 . Ostalih k komponenti koristi se za spremanje znakova koji su pročitani sa k traka.

Simulacija jednog prijelaza TS M_1 izvodi se u dva koraka. U prvom koraku simulacije glava TS M_2 miče se od krajnje lijeve oznake položaja glave do krajnje desne oznake položaja glave. Nailaskom na oznaku položaja glave, u jednu od k komponenata stanja pročita se znak koji je zapisan u toj ćeliji. Znak se čita sa traga na kojem je zapisan sadržaj trake TS M_1 . Komponenta stanja koja ima ulogu brojila provjerava da li su obidene sve oznake položaja glava. Na početku obilaska vrijednost brojila jednak je k , odnosno jednaka je broju glava TS M_1 . Naiđe li se na oznaku glave, vrijednost brojila smanji se za jedan. Obilaskom svih glava vrijednost brojila postane 0.

Nakon što vrijednost brojila postane 0, u k komponenti stanja spremljeni su sadržaji svih k ćelija TS M_1 . TS M_2 ima sve potrebne podatke da doneće odluku: stanje TS M_1 spremljeno u jednoj komponenti stanja TS M_2 i k znakova sa k traka TS M_1 spremljenih u k komponenti stanja TS M_2 . Nema li definiranih daljnih

prijelaza za TS M_1 , rad TS M_2 se zaustavlja. Niz se prihvata ovisno o prihvativosti stanja. Ima li daljnjih prijelaza za TS M_1 , nastavlja se simulacija TS M_1 .

U drugom koraku simulacije glava TS M_2 miće se od krajnje desne oznake položaja glave do krajnje lijeve oznake položaja glave. Nailaskom na oznaku položaja glave, promijeni se sadržaj ćelije na tragu na kojem je spremjeni sadržaj trake TS M_1 . Oznaka položaja glave pomakne se na novi položaj. Novi sadržaj ćelije i novi položaj glave određen je funkcijom prijelaza TS M_1 . Komponenta stanja koja ima ulogu brojila provjerava da li su obidene sve oznake položaja glava. Nakon što se obidu sve oznake položaja glava, komponenta stanja u kojoj je spremljeno stanje TS M_1 promijeni se ovisno o promjeni stanja TS M_1 . Simulacija TS M_1 nastavlja se od prvog koraka.

Prethodno se opisuje simulacija TS s dvostranom beskonačnom trakom. Jedan pomak glave TS s dvostranom beskonačnom trakom simulira se jednim pomakom glave osnovnog modela TS. Međutim, za simulaciju jednog pomaka glave TS sa k traka potrebno je mnogo više pomaka glave osnovnog modela TS.

Prepostavimo da TS M_1 sa k traka pomakne glavu m puta. Budući da TS M_1 miće glavu m puta, dvije glave mogu se nezavisno mičati u suprotne strane. Jedna glava miće se u desnu stranu za m ćelija, a druga glava miće se u lijevu stranu za m ćelija. U tom slučaju, krajnje lijeva i krajnje desna glava TS M_1 udaljene su $2m$ ćelija. Prepostavimo da je za simulaciju i -tog pomaka glave TS M_1 ($1 \leq i \leq m$), kad su krajnje glave udaljenje $2i$ ćelija, potrebno najmanje $2i$ pomaka glave TS M_2 . Na temelju opisanog algoritma simulacije, potrebno je najmanje i pomaka glave u desno da se pročitaju sadržaji k ćelija, te još dodatnih i pomaka glave u lijevo da se upišu odgovarajuće vrijednosti u k ćelija. Pomakne li TS M_1 glavu m puta, TS M_2 izvodi najmanje:

$$\sum_{i=1}^m 2i \approx 2m^2$$

pomaka glave. Broj pomaka glave TS s jednom trakom je kvadratna funkcija broja pomaka glave TS s više traka. Prepostavi li se da svaki pomak glave troši jedinicu vremena, smanjivanje više traka TS na jednu traku može značajno usporiti vrijeme prihvatanja jezika.

Primjer 4.7. Zadan je jezik:

$$L = \{ww^R \mid w \in (\mathbf{a+b})^*\}.$$

Za zadani jezik L gradi se osnovni model TS koji prihvata jezik $L(M)=L$. TS M ima jednu traku s dva traga: u prvi trag zapiše se ulazni niz ww^R , a drugi trag je označni trag. Rad TS M započinje označavanjem i usporedbom krajnjih znakova niza ww^R . Usporedba i označavanje znakova izvodi se na sljedeći način. Glava se postavlja na krajnje lijevu neoznačenu ćeliju. Sadržaj ćelije spremi se u jednu komponentu stanja i zapiše se oznaka \checkmark u označni trag. Glava se miče u desno na krajnju desnu neoznačenu ćeliju. Sadržaj krajnje desne neoznačene ćelije usporedi se sa znakom spremljenim u komponenti stanja. Ako su uspoređeni znakovi jednaki, u označni trag zapiše se oznaka \checkmark . Glava se pomakne u lijevo na krajnje lijevu neoznačenu ćeliju i postupak se nastavlja za sljedeći par neoznačenih znakova.

Ima li TS na raspolaganju dvije trake umjesto jedne, funkcije prijelaza i rad TS značajno se pojednostavljuju. Rad TS s dvije trake izvodi se u tri koraka, što značajno smanjuje broj pomaka glave. U prvom koraku niz ww^R prepise se s jedne trake na drugu traku. U drugom koraku jedna od glava vrati se na početak niza. U trećem koraku uspoređuju se znakovi tako da se jedna glava miće od krajnje lijevog znaka u nizu u desno, dok se druga glava miće od krajnje desnog znaka u nizu u lijevo. U tri koraka TS miće glave tri puta preko čitavog niza, odnosno broj pomaka glave i vrijeme jest proporcionalno duljini niza. Iako TS s dvije trake miće dvije glave u različitim smjerovima, to se računa kao jedan pomak glave koji troši jedinično vrijeme. Ima li TS samo jednu traku, broj pomaka glave i vrijeme prihvatanja niza jest proporcionalno kvadratu duljine niza.

Nedeterministički TS

Za razliku od funkcije prijelaza δ determinističkog TS koja je jednoznačna, funkcija δ nedeterminističkog TS nije jednoznačna:

$$\delta(q, X) = \{(p_1, Z_1, D_1), (p_2, Z_2, D_2), \dots, (p_k, Z_k, D_k)\}.$$

Ako TS čita znak X , a upravljačka jedinka je u stanju q , onda TS prelazi u jedno od stanja p_i , zapiše na traku znak Z_i i miče glavu u smjeru D_i . Promijeni li se stanje u p_i , na traku nije moguće zapisati znak Z_j ili pomaknuti glavu u smjeru D_j , gdje je $i \neq j$. Završi li barem jedan slijed prijelaza prihvativim stanjem, niz se prihvata. Ne završi li niti jedan slijed prijelaza prihvativim stanjem, niz se ne prihvata. Nedeterminizam ne omogućava da TS prihvati širi skup jezika:

Nedeterministički TS istovjetan je determinističkom TS.

Neka TS M_2 ima više nedeterminističkih prijelaza i neka je r kardinalni broj skupa $\delta(q_i, X_j)$ koji ima najviše elemenata, odnosno $r = \max(\text{kardinalini_broj}(\delta(q_i, X_j)))$, za sve $q_i \in Q$ i za sve $X_j \in \Gamma$. Budući da bilo koji skup $\delta(q_i, X_j)$ ima najviše r različitih prijelaza, bilo koji konačni slijed od m prijelaza moguće je prikazati nizom brojeva n_1, n_2, \dots, n_m , gdje brojevi n_i poprimaju vrijednost $1 \leq n_i \leq k$ i $1 \leq k \leq r$. Neka je nakon $i-1$ prijelaza stanje q_{i-1} i neka glava čita znak X_{i-1} . U konačnom slijedu od m prijelaza, vrijednost broja n_i ima sljedeće značenje: u i -tom prijelazu primjenjuje se n_i -ta trojka iz skupa $\delta(q_{i-1}, X_{i-1})$.

Gradi se deterministički TS M_1 koji ima tri trake i simulira rad nedeterminističkog TS M_2 . Prva traka jest ulazna i na nju se zapiše ulazni niz. Na drugu traku TS M_1 generira niz brojeva n_1, n_2, \dots, n_m sljedećim redoslijedom. TS M_1 prvo generira kraće nizove brojeva. Redoslijed generiranja dvaju nizova brojeva jednake duljine određen je njihovom numeričkom vrijednošću. Treća traka koristi se za simulaciju trake TS M_2 .

Svaki puta kada TS M_1 generira novi niz brojeva n_1, n_2, \dots, n_m , započinje novi proces simulacije rada TS M_2 . TS M_1 prepiše ulazni niz sa ulazne trake na treću traku. Treća traka koristi se kao ulazna traka tijekom simulacije rada TS M_2 . Niz n_1, n_2, \dots, n_m generiran na drugoj traci određuje koje se funkcije prijelaza TS M_2 primjenjuju. Na primjer, neka je nakon simulacije $i-1$ prijelaza stanje q_{i-1} i neka glava na trećoj traci čita znak X_{i-1} , gdje je $1 \leq i \leq m$. Neka je vrijednost i -tog broja u generiranom slijedu n_i i neka je funkcija prijelaza $\delta(q_{i-1}, X_{i-1}) = \{(p_1, Z_1, W_1), (p_2, Z_2, W_2), \dots, (p_{n_i}, Z_{n_i}, W_{n_i}), \dots, (p_k, Z_k, W_k)\}$, gdje je $1 \leq n_i \leq k$ i $1 \leq k \leq r$. Budući da je i -ti broj u nizu n_i , primjeni se funkcija prijelaza $\delta(q_{i-1}, X_{i-1}) = (p_{n_i}, Z_{n_i}, W_{n_i})$.

Prihvata li niz TS M_2 , isti niz prihvata i TS M_1 . Ako postoji slijed prijelaza TS M_2 koji završava prihvativim stanjem, onda se na drugu traku generira niz brojeva n_1, n_2, \dots, n_m koji predstavlja slijed funkcija prijelaza koji završava prihvativim stanjem. Ne postoji li niti jedan slijed prijelaza TS M_2 koji završava prihvativim stanjem, TS M_1 ne prihvata niz. Budući da niti jedan generirani niz brojeva n_1, n_2, \dots, n_m ne predstavlja slijed funkcija prijelaza koji završava prihvativim stanjem, niz se ne prihvata.

TS s višedimenzionalnim ulaznim poljem

Umjesto jednodimenzionalne trake, moguće je koristiti k -dimenzionalno polje ćelija koje je beskonačno u svim smjerovima, gdje je k konačni broj. Na temelju stanja i procitanog znaka, TS odlučuje o promjeni stanja, zapiše novi znak u ćeliju i odlučuje o pomaku glave. Glavu je moguće pomaknuti uzduž svake od k osi u pozitivnom ili negativnom smjeru. Sveukupno ima $2k$ različitih načina pomaka glave. Na početku rada ulazni niz je zapisan uzduž jedne od k osi i glava je postavljena na krajnje lijevi znak ulaznog niza.

Budući da ulazni niz ima konačni broj znakova i budući da TS u bilo kojem trenutku rada pomakne glavu konačni broj puta, samo konačni broj ćelija bilo koje osi je neprazan. Neprazne ćelije u k -dimenzionalnom polju moguće je smjestiti u k -dimenzionalni prostor kvadar, a sadržaj ćelija k -dimenzionalnog kvadra moguće je spremiti na jednodimenzionalnu traku. Način pohrane sadržaja k -dimenzionalne trake na jednodimenzionalnu traku pokazuje se na primjeru dvodimenzionalnog polja.

\mathbf{B}	\mathbf{B}	\mathbf{B}	a_1	\mathbf{B}	\mathbf{B}	\mathbf{B}
\mathbf{B}	\mathbf{B}	a_2	a_3	a_4	a_5	\mathbf{B}
a_6	a_7	a_8	a_9	\mathbf{B}	a_{10}	\mathbf{B}
\mathbf{B}	a_{11}	a_{12}	a_{13}	\mathbf{B}	a_{14}	a_{15}
\mathbf{B}	\mathbf{B}	a_{16}	a_{17}	\mathbf{B}	\mathbf{B}	\mathbf{B}

(a) Dvodimenzionalno polje

$$**BBBa_1BBB^*BBa_2a_3a_4a_5B^*a_6a_7a_8a_9Ba_{10}B^*Ba_{11}a_{12}a_{13}Ba_{14}a_{15}^*BBa_{16}a_{17}BBB^{**}$$

(b) Sadržaj jednodimenzionalne trake

Slika 4.7: Simulacija dvodimenzionalnog polja
primjenom jednodimenzionalne trake

Na slici 4.7a neprazne čelije zaokružene su pravokutnikom. Sadržaji čelija pravokutnika spreme se na jednodimenzionalnu traku red po red, tako da pojedini redovi čine blokove odvojene znakom *. Početak prvog bloka i kraj posljednjeg bloka označen je znakovima **. Slika 4.7b prikazuje pohranu sadržaja čelija pravokutnika na jednodimenzionalnu traku.

TS s dvodimenzionalnim poljem čelija istovjetan je TS s jednodimenzionalnom trakom.

Gradi se TS M_1 s jednodimenzionalnom trakom koja ima dva traga. Jedan trag koristi za označavanje položaja glave. Stanje TS M_2 s dvodimenzionalnim poljem spremi se u jednu komponentu stanja TS M_1 . Novi sadržaj čelije i novo stanje određuju se na temelju funkcija prijelaza TS M_2 . Tijekom rada TS M_1 simulira četiri pomaka glave TS M_2 : horizontalni pomak unutar pravokutnika, vertikalni pomak unutar pravokutnika, vertikalni pomak izvan pravokutnika i horizontalni pomak izvan pravokutnika. Pomaci oznake položaja glave određuju se na sljedeći način:

Horizontalni pomak unutar pravokutnika: TS M_1 miče oznaku položaja glave za jednu čeliju lijevo ili desno unutar bloka ovisno o pomaku glave TS M_2 .

Vertikalni pomak unutar pravokutnika: Vertikalni pomak miče glavu u lijevi ili desni susjedni blok, dok se položaj glave unutar bloka ne mijenja. TS M_1 koristi posebnu traku za brojilo kojom se određuje položaj oznake glave. Neka je u bloku k čelija. Pomakne li TS M_2 glavu dolje, TS M_1 miče oznaku položaja glave za $k+1$ čelija u desno. Pomakne li TS M_2 glavu gore, TS M_1 miče oznaku položaja glave za $k+1$ čelija u lijevo. U oba slučaja glava se miče preko oznake kraja bloka *.

Vertikalni pomak izvan pravokutnika: TS M_1 dodaje jedan blok praznina na krajnju desnu ili lijevu stranu. Broj čelija u bloku određuje se posebnom trakom koja se koristi za brojilo. Nakon što se doda novi blok, TS M_1 simulira vertikalni pomak unutar pravokutnika.

Horizontalni pomak izvan pravokutnika: TS M_1 proširuje sve blokove jednom prazninom na lijevoj ili desnoj strani bloka, a zatim simulira horizontalni pomak unutar pravokutnika.

Na sličan način moguće je simulirati rad TS s k -dimenzionalnim poljem.

TS s više glava za čitanje i pisanje

Model TS s više glava za čitanje i pisanje ima jednu traku i k glave za čitanje i pisanje. Glave se nezavisno miču u lijevo ili desno. TS donosi odluku na temelju stanja i na temelju k pročitanih znakova.

TS s k glava za čitanje i pisanje istovjetan je TS s jednom glavom za čitanje i pisanje.

Gradi se istovjetni TS M_1 s jednom glavom za čitanje i pisanje, te s jednom trakom koja ima $k+1$ tragova. Jedan trag koristi se za pohranu sadržaja trake TS M_2 s k glava za čitanje i pisanje, dok se ostalih k tragova koristi za označavanje položaja k glava za čitanje i pisanje TS M_2 . Za simulaciju rada TS M_2 koristi se algoritam koji je sličan algoritmu koji simulira rad TS s višestrukim trakama.

Neizravni TS

Neizravni TS jest model automata koji se koristi u istraživanju prostorne složenosti prihvaćanja jezika i prostorne složenosti računanja cjelobrojnih funkcija. Neizravni TS ima više radnih traka i jednu ulaznu traku. Ulagnu traku moguće je samo čitati. Niz na ulaznoj traci ograden je graničnicima ϵ i $\$$. Glavu za čitanje ulazne trake nije moguće pomaknuti izvan područja omeđenog graničnicima.

Neizravni TS istovjetan je TS s višestrukim trakama. Rad glava za čitanje i pisanje TS s višestrukim trakama nije ograničen, što omogućuje da TS s višestrukim trakama simulira rad bilo kojeg neizravnog TS. Nadalje, dodavanjem jedne trake neizravnom TS i prijepisom niza s ulazne trake na dodatnu traku, te uporabom dodatne trake kao ulazne, omogućuje da neizravni TS simulira rad bilo kojeg TS s višestrukim trakama.

4.1.4 Pojednostavljeni modeli Turingovog stroja

U ovom odjeljku opisuju se pojednostavljeni modeli TS koji su istovjetni osnovnom modelu TS.

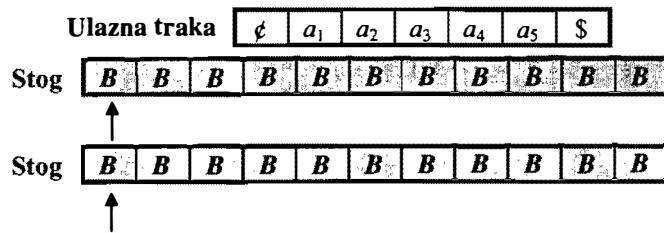
Stogovni stroj

Stogovni stroj jest deterministički TS s jednom ulaznom trakom i više stogova. Ulaznu traku moguće je samo čitati. Stog jest posebna radna traka koja ima pojednostavljene funkcije prijelaza: pomakne li se glava u lijevo, u ćeliju se obavezno zapiše praznina. Sve ćelije stoga desno od glave za čitanje i pisanje su prazne.

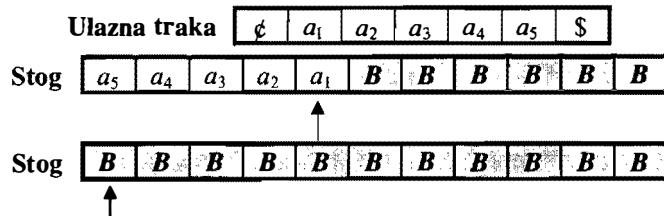
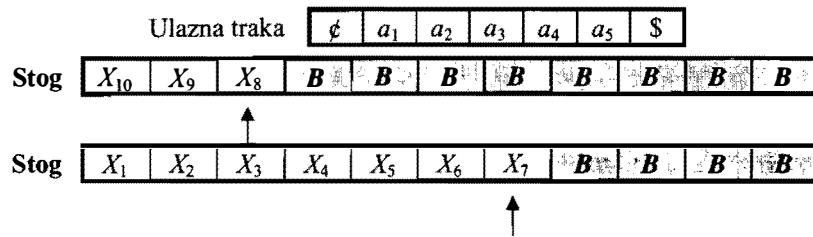
TS s jednom trakom moguće je simulirati primjenom stogovnog stroja sa dva stoga.

Jedan stog koristi se za pohranu znakova koji su lijevo od glave za čitanje i pisanje, a drugi stog koristi se za pohranu znakova koji su desno od glave za čitanje i pisanje. Na vrhu stoga nalaze se znakovni trake koji su bliže glavi za čitanje i pisanje. Nisu li stogovi prazni, na dnu jednog stoga je sadržaj krajnje lijeve ćelije, a na dnu drugog stoga je sadržaj krajnje desne ćelije do koje se pomaknula glava TS.

Na početku rada na ulaznu traku zapiše se ulazni niz. Oba stoga su prazna. Početna konfiguracija stogovnog stroja prikazana je na slici 4.8a. Ulazni niz prepiše se na stog na koji se spremaju znakovi koji su desno od glave za čitanje i pisanje. Drugi stog ostaje prazan. Simulacija početne konfiguracije TS prikazana je na slici 4.8b.



(a) Početna konfiguracija stogovnog stroja s dva stoga

(b) Simulacija početne konfiguracije TS: $q \ a_1 \ a_2 \ a_3 \ a_4 \ a_5$ (c) Simulacija konfiguracije TS: $X_1 \ X_2 \ X_3 \ X_4 \ X_5 \ X_6 \ X_7 \ p \ X_8 \ X_9 \ X_{10}$ **Slika 4.8:** Simulacija TS s jednom trakom primjenom stogovnog stroja s dva stoga

Primjer simulacije konfiguracije $X_1 \ X_2 \ X_3 \ X_4 \ X_5 \ X_6 \ X_7 \ p \ X_8 \ X_9 \ X_{10}$ prikazan je na slici 4.8c. Pomakne li se glava TS u desno, smanji se broj znakova na gornjem stogu, a poveća se broj znakova na donjem stogu. Dode li glava TS do krajnje desnog znaka različitog od praznine, gornji stog ostaje prazan, a u donjem stogu su znakovi trake TS koji se nalaze spremljeni od krajnje lijeve ćelije do ćelije gdje je postavljena glava TS. Miće li se glava TS dalje u desno, donji stog se povećava, a gornji stog ostaje prazan. Desno od glave TS nema niti jednog znaka različitog od praznine.

Pomakne li se glava TS u lijevo, smanji se broj znakova na donjem stogu, a poveća se broj znakova na gornjem stogu. Dode li glava TS do krajnje lijeve ćelije, donji stog ostaje prazan, a u gornjem stogu su znakovi TS koji se nalaze spremljeni od krajnje lijeve ćelije do krajnje desne ćelije do koje se prethodno pomaknula glava TS.

Stroj s brojilima

Model stogovnog stroja s dva stoga moguće je još više pojednostaviti. Umjesto dva stoga, stroj koristi četiri brojila. Skup znakova stoga ograniči se na dva znaka: oznaku dna stoga X i prazninu B . Oznaka dna stoga X nalazi se u krajnje lijevoj ćeliji. Glavu nije moguće pomaknuti lijevo od oznake dna stoga X . Pomakne li se glava stoga u desno za i ćelija od znaka X , vrijednost brojila jednaka je i . Pomakne li se glava u desno ili lijevo za jednu ćeliju, povećava se ili smanji vrijednost brojila za jedan. Čita li glava oznaku dna

stoga X , vrijednost brojila jednaka je 0. Dodavanje vrijednosti k postiže se dodatnom komponentom stanja koja omogući k pomaka glave u desno, a oduzimanje vrijednosti k postiže se dodatnom komponentom stanja koja omogući k pomaka glave u lijevo.

TS s jednom trakom moguće je simulirati primjenom stroja sa četiri brojila.

Budući da je u prethodnom odjeljku pokazano da je TS s jednom trakom moguće simulirati primjenom stogovnog stroja sa dva stoga, dovoljno je pokazati da dva brojila mogu simulirati rad jednog stoga. Prepostavi se da ima k različitih znakova stoga $Z_0, Z_1, Z_2, \dots, Z_{k-1}$. Indeksi znakova stoga njihove su cjelobrojne vrijednosti, odnosno i jest vrijednost znaka Z_i , gdje je $0 \leq i \leq k-1$. Neka su na stogu znakovi $Z_{i_1}, Z_{i_2}, \dots, Z_{i_m}$ i neka je na vrhu stoga znak Z_{i_m} koji ima vrijednost i_m . Vrijednost znakova stoga $Z_{i_1}, Z_{i_2}, \dots, Z_{i_m}$ jest cijeli broj po bazi k koji se računa na sljedeći način:

$$1) \quad j = i_m + k i_{m-1} + k^2 i_{m-2} + k^3 i_{m-3} + \dots + k^{m-1} i_1$$

Dva brojila A i B izvode osnove operacije stoga na sljedeći način:

Stavljanje znaka Z_r na vrh stoga. Na stogu su znakovi $Z_{i_1}, Z_{i_2}, \dots, Z_{i_m}$. Na vrhu stoga jest znak Z_{i_m} . Vrijednost znakova na stogu određena je izrazom (1) i iznosi j . Stavi li se znak Z_r na vrh stoga, vrijednost znakova stoga $Z_{i_1}, Z_{i_2}, \dots, Z_{i_m}, Z_r$ postaje $jk+r$.

Dva brojila računaju vrijednost izraza $jk+r$ na sljedeći način. Neka je vrijednost brojila A jednaka j , dok je vrijednost brojila B jednaka 0. Vrijednost jk računa se tako da se istodobno vrijednost brojila A smanji za 1, a vrijednost brojila B povećava za k . Nakon što vrijednost brojila A postane 0, u brojilu B jest vrijednost jk . Brojilu B doda se r i izračuna se vrijednost $r+jk$.

Uzimanje znaka s vrha stoga. Znak Z_{i_m} uzima se s vrha stoga $Z_{i_1}, Z_{i_2}, \dots, Z_{i_m}$. Vrijednost znakova stoga prije uzimanja znaka Z_{i_m} s vrha stoga jednaka je j , dok je vrijednost znakova stoga nakon uzimanja znaka Z_{i_m} s vrha stoga jednaka $\lfloor j/k \rfloor$, gdje je $\lfloor j/k \rfloor$ cjelobrojna vrijednost količnika j/k zakokružena na niže.

Dva brojila računaju cjelobrojnu vrijednost količnika j/k na sljedeći način. Neka je vrijednost brojila A jednaka j , dok je vrijednost brojila B jednaka 0. Vrijednost $\lfloor j/k \rfloor$ računa se na sljedeći način. Nako što se vrijednost brojila A smanji za k , vrijednost brojila B povećava za 1. Kada vrijednost brojila A postane 0, u brojilu B je vrijednost $\lfloor j/k \rfloor$.

Određivanje znaka na vrhu stoga. Primjenom funkcije $j \bmod k$ moguće je odrediti koji znak je na vrhu stoga. Ako je vrijednost znakova stoga j jednaka $i_m + ki_{m-1} + k^2 i_{m-2} + \dots + k^{m-1} i_1 = i_m + k(i_{m-1} + ki_{m-2} + \dots + k^{m-2} i_1)$, onda je na vrhu stoga znak Z_{i_m} koji ima vrijednost i_m . Vrijednost funkcije $j \bmod k$ jednaka je vrijednosti znaka na vrhu stoga.

Dva brojila računaju vrijednost funkcije $j \bmod k$ na sljedeći način. Neka je vrijednost brojila A jednaka j , dok je vrijednost brojila B jednaka 0. Složena oznaka stanja proširi se dodatnom komponentom stanja koja se ciklički mijenja na sljedeći način: 0, 1, 2, ..., $k-1$, 0, 1, 2, Vrijednost brojila A prepisuje se u brojilo B . Tijekom prijepisa, vrijednost brojila A smanji se za 1, vrijednost brojila B povećava se za 1, a vrijednost komponente stanja ciklički se mijenja 0, 1, 2, ..., $k-1$, 0, 1, 2, Nakon što se vrijednost j prenese iz brojila A u brojilo B , vrijednost komponente stanja jednaka je $j \bmod k$.

Količinu brojila moguće je smanjiti sa četiri brojila na dva brojila:

TS s jednom trakom moguće je simulirati primjenom stroja sa dva brojila.

Budući da je prethodno pokazano da je primjenom četiri brojila moguće simulirati rad TS s jednom trakom, dovoljno je pokazati da je primjenom dva brojila moguće simulirati rad četiri brojila. Ako su vrijednosti četiri brojila jednakih i, j, k i l , onda se u jedno brojilo spremi vrijednost $n=2^i3^j5^k7^l$. Budući da su $2, 3, 5$ i 7 prim brojevi, vrijednosti i, j, k i l moguće je jednoznačno odrediti na temelju vrijednosti n .

Osnovne funkcije brojila su povećavanje i smanjivanje za 1, te određivanje da li je vrijednost brojila jednak 0. Povećavanje ili smanjivanje vrijednosti i, j, k i l za 1 postiže se množenjem ili dijeljenjem vrijednosti n s odgovarajućim brojem 2, 3, 5 ili 7. Prethodno su opisani postupci množenja i dijeljenja primjenom dva brojila. Određivanje da li je neki od brojeva i, j, k i l jednak 0 postiže se dijeljenjem broja n s brojevima 2, 3, 5 ili 7. Na primjer, nije li broj n dijeljiv sa 5, vrijednost k jednak je 0. Nije li n dijeljiv sa 3 i 7, vrijednosti j i l jednakе su 0. Na temelju osnovnih funkcija brojila moguće je ostvariti prethodno opisane operacije stoga.

Na temelju pokazanih činjenica:

- 1) da je TS s jednom trakom moguće simulirati primjenom stogovnog stroja s dva stoga,
- 2) da je stogovni stroj sa dva stoga moguće simulirati primjenom stroja s četiri brojila,
- 3) da je stroj sa četiri brojila moguće simulirati primjenom stroja s dva brojila,

zaključuje se da su svi opisani pojednostavljeni modeli strojeva istovjetni osnovnom modelu TS.

TS s ograničenim brojem stanja i znakova trake

Ako se istodobno ograniči broj znakova trake, broj traka i broj stanja, onda je ograničen broj različitih Turingovih strojeva koje je moguće izgraditi. Zato TS s ograničenim brojem znakova trake, s ograničenim brojem traka i s ograničenim brojem stanja ne prihvata isti skup jezika kao i osnovni model TS. Međutim, ne ograniči li se broj znakova trake, dovoljno je za prihvatanje bilo kojeg rekurzivno prebrojivog jezika imati jednu traku i tri stanja, od toga dva stanja su neprihvatljiva i jedno stanje je prihvatljivo.

Nadalje, ako se ne ograniči broj stanja, onda je moguće ograničiti broj znakova trake na $\{0, 1, B\}$, što je dovoljno za prihvatanje bilo kojeg rekurzivno prebrojivog jezika. Neka je vrijednost kardinalnog broja skupa znakova trake TS M_1 veća od $2^{k-1}+1$, ali neka je manja od 2^k . Svi znakovi trake TS M_1 kodiraju se binarnim nizovima duljine k , uključujući i znak praznine TS M_1 . Gradi se TS M_2 sa skupom znakova trake jednakim $\Gamma=\{0, 1, B\}$. U k ćelija trake TS M_2 nalazi se k -bitovni binarni kod za jedan znak trake TS M_1 . U jednu komponentu stanja TS M_2 spremi se stanje TS M_1 , dok se druga komponenta koristi za postavljanje glave na početak binarnog koda. Tijekom simulacije funkcije prijelaza TS M_1 , TS M_2 čita k ćelija u kojima je zapisan binarni kod znaka trake TS M_1 i donosi odluku o primjeni funkcije prijelaza. Pročita li TS M_2 prazninu B , simulira se prvi pomak TS M_1 na datu ćeliju. TS M_2 zamjeni k praznina B binarnim kodom za prazninu, a zatim nastavlja simulaciju rada TS M_1 .

Univerzalni Turingov stroj M_u

Primjenom univerzalnog TS M_u moguće je simulirati rad bilo kojeg TS M s jednom trakom. Univerzalni TS ima tri trake. Na prvu traku zapišu se funkcije prijelaza TS M i niz w . Zapis funkcije prijelaza TS M i niza w označava se na sljedeći način: $\langle M, w \rangle$. U odjeljku 4.3.2 opisani su način kodiranja funkcija prijelaza i njihov zapis na traku univerzalnog TS M_u . Na treću traku zapisuje se stanje TS M . Sadržaj druge trake univerzalnog TS M_u simulira sadržaj trake proizvoljnog TS M . Univerzalni TS M_u prepiše niz w s prve trake na drugu traku, simulira rad TS M primjenom prijelaza zapisanih na prvoj traci i zapisuje stanje TS M na treću traku. Nema li za stanje zapisano na trećoj traci i za znak zapisan na drugoj traci daljnjih prijelaza zapisanih na prvoj traci, zaustavlja se rad univerzalnog TS M_u . Ako je stanje zapisano na trećoj traci prihvatljivo, univerzalni TS M_u prelazi u prihvatljivo stanje. Univerzalni TS M_u prihvata niz $\langle M, w \rangle$ ako i samo ako TS M prihvata niz w .

Moguće je izgraditi univerzalni TS M_w sa samo jednom trakom, pet stanja i sedam znakova trake koji prihvata niz $\langle M, w \rangle$ ako i samo ako TS M prihvata niz w . Primjenom TS M_w s ograničenim brojem znakova trake, s ograničenim brojem traka i s ograničenim brojem stanja moguće je odrediti da li proizvoljni niz w pripada proizvoljnom rekurzivno prebrojivom jeziku zadanim TS M .

4.1.5 Generiranje jezika Turingovim strojem

Osim prihvatanja jezika i računanja cijelobrojnih funkcija, TS se koristi za generiranje jezika. Za generiranje jezika koristi se TS s višestrukim trakama. Jedna traka TS označi se kao *izlazna traka*. Znak se zapiše na izlaznu traku trajno i nije moguće jednom zapisani znak promijeniti. Glava za čitanje i pisanje izlazne trake miče se samo u desno. Izlazna traka koristi se za generiranje jezika. Nizovi nad abecedom Σ ispisuju se na izlaznu traku i odvajaju se graničnicima $\#$. Jezik koji generira TS M označava se $G(M)$ kao bi se razlikovao od jezika $L(M)$ koji prihvata TS M . Jezik $G(M)$ je skup nizova $w \in \Sigma^*$ koji se ispisuju na izlaznu traku između graničnika $\#$. Stane li TS M , jezik $G(M)$ je konačan. Ne stane li TS M , jezik $G(M)$ je beskonačan. Nizove nije potrebno generirati nekim unaprijed definiranim redoslijedom i isti niz moguće je generirati više puta. TS generira klasu rekurzivno prebrojivih jezika:

Jezik L jest rekurzivno prebrojiv ako i samo ako postoji TS M_1 koji generira jezik $G(M_1)=L$.

Prihvatanje jezika koji je generiran Turingovim strojem

Za bilo koji TS M_1 koji generira jezik $G(M_1)$ moguće je izgraditi TS M_2 koji prihvata jezik $L(M_2)=G(M_1)$.

Za zadani TS M_1 koji generira jezik $G(M_1)$ moguće je izgraditi TS M_2 koji prihvata jezik $L(M_2)=G(M_1)$ na sljedeći način. Gradi se TS M_2 koji ima jednu traku više od TS M_1 . Dodana traka označi se kao ulazna traka. Na ulaznu traku zapiše se niz za koji se ispituje da li pripada jeziku $G(M_1)$ koji generira TS M_1 . TS M_2 simulira rad TS M_1 tako da koristi sve trake osim ulazne trake. Ispiše li tijekom simulacije TS M_1 graničnik $\#$, TS M_2 uspoređuje generirani niz s nizom na ulaznoj traci. Ako su dva niza jednaka, onda TS M_2 stane i prihvati niz. Nisu li dva niza jednaka, TS M_2 nastavlja simulirati rad TS M_1 i generira sljedeći niz. TS M_2 prihvata niz zapisan na ulaznoj traci ako i samo ako TS M_1 generira isti niz na svojoj izlaznoj traci.

Generiranje jezika koji se prihvata Turingovim strojem

Za bilo koji TS M_2 koji prihvata jezik $L(M_2)$ moguće je izgraditi TS M_1 koji generira jezik $G(M_1)=L(M_2)$.

Za zadani TS M_2 koji prihvata jezik $L(M_2)$ moguće je izgraditi jednostavni TS M_1 koji generira rekurzivni jezik $G(M_1)=L(M_2)$ ili složeni TS M_1 koji generira rekurzivno prebrojiv jezik $G(M_1)=L(M_2)$.

Za generiranje rekurzivnih jezika koristi se jednostavni TS M_1 . Jednostavni TS M_1 određenim redoslijedom ispisuje sve nizove w_1, w_2, w_3, \dots iz skupa Σ^* . Nizovi se ispisuju na radnu traku, a ne na izlaznu traku. Nakon ispisa niza w_i , TS M_1 simulira rad TS M_2 . Prihvati li TS M_2 tijekom simulacije niz w_i , TS M_1 generira niz w_i na izlaznu traku. Budući da je jezik $L(M_2)$ rekurzivan, za sve nizove w_1, w_2, w_3, \dots TS M_2 uvijek stane, te se na izlaznu traku generiraju svi nizovi iz jezika $L(M_2)$.

Međutim, ako je jezik $L(M_2)$ rekurzivno prebrojiv, onda je moguće da postoji niz w_j koji nije u jeziku $L(M_2)$ i za koji TS M_2 nikada ne stane. Budući da TS M_2 nikada ne stane za niz w_j , nije moguće pokrenuti ispitivanje niti za jedan niz w_{j+1}, w_{j+2}, \dots koji slijede niz w_j . Ima li među nizovima w_{j+1}, w_{j+2}, \dots niz koji je u jeziku $L(M_2)$, taj se niz ne generira na izlaznu traku. Ako je jezik rekurzivno prebrojiv, ne smije se niti za jedan niz dozvoliti neograničen broj prijelaza prilikom simulacije TS M_2 .

Za generiranje rekurzivno prebrojivih jezika koristi se složeni TS M_1 . Složeni TS M_1 generira par cijelih brojeva (i, j) , a zatim TS M_1 simulira rad TS M_2 uzimajući i -ti niz w_i i primjenjujući najviše j prijelaza. Redoslijed generiranja parova (i, j) određen je rastućom vrijednošću sume $i+j$. Imaju li parovi jednaku sumu, njihov redoslijed određen je rastućim vrijednostima broja i . Parovi se generiraju sljedećim redoslijedom: $(1, 1), (1, 2), (2, 1), (1, 3), (2, 2), (3, 1), (1, 4), (2, 3), (3, 2), (4, 1), \dots$.

Složeni TS M_1 generira jezik $L(M_2)$ na sljedeći način. Nakon što TS M_1 generira par (i, j) , generira se niz w_i na jednu radnu traku i simulira se rad TS M_2 primjenom j prijelaza. Prihvata li TS M_2 niz w_i primjenom j prijelaza, TS M_1 generira niz w_i na izlaznu traku. Budući da je broj prijelaza koji se primijene tijekom simulacije ograničen, ispituju se svi nizovi w_1, w_2, w_3, \dots i generiraju se svi nizovi jezika $L(M_2)$.

Istovjetnost rekurzivnih jezika i jezika koje je moguće generirati kanonskim slijedom

U kanonskom slijedu kraći nizovi su ispred dužih nizova. Redoslijed dvaju nizova jednakih duljina određuje se prema njihovoj numeričkoj vrijednosti. Baza za računanje numeričke vrijednosti određuje se na temelju kardinalnog broja skupa ulaznih znakova Σ . Na primjer, za binarnu abecedu $\Sigma=\{0, 1\}$ baza je 2 i kanonski slijed je: $\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111, 0000, 0001, \dots$.

Rekurzivne jezike moguće je generirati kanonskim slijedom.

Ako je jezik $L(M_2)$ rekurzivan, onda je za generiranje jezika moguće koristiti jednostavni TS. Jednostavni TS generira nizove jezika $L(M_2)$ na izlaznu traku onim redoslijedom kojim se ti nizovi generiraju na radnu traku. Generiraju li se nizovi na radnu traku kanonskim slijedom, jednostavni TS M_1 generira nizove jezika $L(M_2)$ kanonskim slijedom na izlaznu traku.

Jezik $G(M_1)$ koji je moguće generirati kanonskim slijedom jest rekurzivni jezik.

Neka TS M_1 generira jezik $G(M_1)$ kanonskim slijedom. TS M_2 koji prihvata jezik $L(M_2)=G(M_1)$ i koji uvijek stane za svaki ulazni niz w , konstruira se na sljedeći način. TS M_2 simulira rad TS M_1 . Pojavi li se niz w_i na ulazu, TS M_2 simulira rad TS M_1 sve do trenutka generiranja niza w_i ili nekog drugog niza w_j koji je u kanonskom slijedu iza niza w_i . Generira li niz w_i , TS M_2 prihvati niz w_i i zaustavi daljnji rad. Generira li niz w_j koji je u kanonskom slijedu iza niza w_i , TS M_2 ne prihvati niz w_i i zaustavi daljnji rad. Budući da je niz w_j iza niza w_i , a nizovi se generiraju isključivo kanonskim slijedom, onda TS M_1 sigurno ne geneira niz w_i , pa ga zato ni TS M_2 ne prihvata.

Ako je jezik $G(M_1)$ je beskonačan, onda TS M_2 uvijek stane. Ako je jezik $G(M_1)$ konačan i ako se ispituju nizovi koji su u kanonskom slijedu iza svih nizova jezika $L(M_2)$, TS M_2 nikad ne stane. Iako nije moguće izgraditi TS M_2 za opći slučaj konačnog jezika, za bilo koji točno određeni konačni jezik moguće je izgraditi zasebni TS koji ga prihvata i koji uvijek stane za svaki ulazni niz.

4.2 Gramatika neograničenih produkacija

Dozvoljeni oblici produkacija kontekstno neovisne gramatike i regularne gramatike su ograničeni: produkcijske kontekstno neovisne gramatike imaju na lijevoj strani samo jedan nezavršni znak, dok su produkcijske regularne gramatike lijevo linearne ili desno linearne. U ovom odjeljku definira se gramatika neograničenih produkacija. Neograničene produkcijske su oblike:

$$\alpha \rightarrow \beta$$

gdje su α i β nizovi završnih i nezavršnih znakova gramatike. Niz α ne smije biti prazni niz, odnosno $\alpha \neq \epsilon$. U literaturi se za gramatiku neograničenih produkacija koristi i drugi naziv: *gramatika tipa 0*.

Gramatika neograničenih produkacija jest uređena četvorka $G=(V, T, P, S)$. Za produkcijsku $\alpha \rightarrow \beta$ gramatike G definira se relacija \Rightarrow na sljedeći način:

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

Relacija \Rightarrow^* jest refleksivno i tranzitivno okruženje relacije \Rightarrow .

Gramatika $G=(V, T, P, S)$ generira jezik:

$$L(G) = \{w \mid w \in T^* \text{ i } S \xrightarrow{*} w\}.$$

Gramatika neograničenih produkacija generira klasu rekurzivno prebrojivih jezika. U odjeljku 4.2.1 opisuje se gradnja Turingovog stroja za jezik zadan gramatikom neograničenih produkacija, a u odjeljku 4.2.2 opisuje se gradnja gramatike za rekurzivno prebrojiv jezik zadan Turingovim strojem.

Primjer 4.8. Zadana je gramatika $G=(\{A, B, C, D, E\}, \{a\}, P, S)$ neograničenih produkacija:

- | | | |
|-------------------------|------------------------|------------------------------|
| 1) $S \rightarrow ACaB$ | 4) $CB \rightarrow E$ | 7) $aE \rightarrow Ea$ |
| 2) $Ca \rightarrow aaC$ | 5) $aD \rightarrow Da$ | 8) $AE \rightarrow \epsilon$ |
| 3) $CB \rightarrow DB$ | 6) $AD \rightarrow AC$ | |

Gramatika generira jezik $L(G)=\{a^i \mid i=2^k, k \text{ jest cijeli broj i } k>0\}$. U jeziku su nizovi koji imaju broj znakova a jednak $2, 4, 8, 16, 32, \dots$, itd.

Gramatika G generira niz aa na sljedeći način:

$$\begin{aligned}
 S &\Rightarrow ACaB && - \text{ primjenom produkcijske } S \rightarrow ACaB \\
 &\Rightarrow AaaCB && - \text{ primjenom produkcijske } Ca \rightarrow aaC \\
 &\Rightarrow AaaE && - \text{ primjenom produkcijske } CB \rightarrow E \\
 &\Rightarrow AaEa && - \text{ primjenom produkcijske } aE \rightarrow Ea \\
 &\Rightarrow AEaa && - \text{ primjenom produkcijske } aE \rightarrow Ea \\
 &\Rightarrow aa && - \text{ primjenom produkcijske } AE \rightarrow \epsilon.
 \end{aligned}$$

Na sličan način započinje generiranje niza $aaaa$:

$$\begin{aligned} S &\Rightarrow ACaB \\ &\Rightarrow AaaCB \end{aligned}$$

- primjenom produkcije $S \rightarrow ACaB$
- primjenom produkcije $Ca \rightarrow aaC$

Producija $CB \rightarrow DB$ zamijeni nezavršni znak C nezavršnim znakom D :
 $\Rightarrow AaaDB$

Producija $aD \rightarrow Da$ miče znak D u lijevo do znaka A :

$$\Rightarrow AaDaB \Rightarrow ADaaB$$

Producija $AD \rightarrow AC$ zamijeni znak D znakom C , a zatim producija $Ca \rightarrow aaC$ miče znak C u desno do znaka B . Istodobno s pomakom znaka C , jedan znaka a zamijeni se nizom aa :

$$\Rightarrow ACaaB \Rightarrow AaaCaB \Rightarrow AaaaaCB$$

Producija $CB \rightarrow E$ zamijeni niz znakova CB znakom E , producija $aE \rightarrow Ea$ miče znak E u desno do znaka A , a producija $AE \rightarrow \epsilon$ zamijeni niz znakova AE praznim nizom:

$$\Rightarrow AaaaaE \Rightarrow AaaaEa \Rightarrow AaaEaa \Rightarrow AaEaaa \Rightarrow AEaaaa \Rightarrow aaaa.$$

Nezavršni znakovi A i B imaju ulogu lijevog i desnog graničnika, nezavršni znak C pomakom u desno udvostručuje broj završnih znakova a , nezavršni znak D pomakom u lijevo omogućuje novi proces udvostručavanja završnih znakova a i nezavršni znak E pomakom u lijevo zaustavlja daljnji proces udvostručavanja znakova a . Za niz znakova CB postoje dvije mogućnosti: znak C zamijeni se znakom D primjenom produkcije (3) ili se niz znakova CB zamijeni znakom E primjenom produkcije (4). Zamijeni li se znak C znakom D , producija (5) miče znak D u lijevo do znaka A . Producija (6) zamijeni znak D znakom C i započinje novi proces udvostručavanja završnih znakova a primjenom produkcije (2). Zamijeni li se niz znakova CB znakom E , producija (7) miče znak E u lijevo do znaka A . Producija (8) zamijeni niz znakova AE praznim nizom i generira niz koji se sastoji isključivo od 2^k završnih znakova a .

Ne primjeni li se producija (4), za bilo koji generirani međuniz vrijedi jedna od tvrdnji:

- Generirani međuniz jednak je S .
- Generirani međuniz jednak je $Aa^i Ca^j B$, gdje je $i+2j = 2^k$, za neki $k > 0$.
- Generirani međuniz jednak je $Aa^i Da^j B$, gdje je $i+j = 2^k$, za neki $k > 0$.

Primjeni li se producija (4), nastaje međuniz $Aa^i E$, gdje je $i=2^k$ za neki $k > 0$. Primjenom i producija (7) nastaje međuniz AEa^i . Producija (8) zamijeni niz AE praznim nizom i generira niz a^i , gdje je $i=2^k$ za neki $k > 0$.

4.2.1 Konstrukcija TS za jezik za dan gramatikom neograničenih produkcija

Ako gramatika $G=(V, T, P, S)$ neograničenih produkcija generira jezik $L(G)$, onda jezik $L(G)$ jest rekurzivno prebrojiv jezik.

Budući da je po definiciji jezik rekurzivno prebrojiv ako i samo ako postoji TS koji ga prihaća, jezik $L(G)$ koji generira gramatika G jest rekurzivno prebrojiv ako i samo ako postoji TS koji prihvata jezik $L(M)=L(G)$.

Gradi se nedeterministički TS M s dvije trake koji simulira rad gramatike G na sljedeći način. Na prvu traku zapiše se niz završnih znakova w , a na drugu traku zapiše se početni nezavršni znak gramatike S . Tijekom simulacije TS M ispisuje na drugu traku međunizove α koje generira gramatika G . TS M uspoređuje nizove znakova α generirane na drugoj traci s nizom završnih znakova w zapisanim na prvoj traci. Ako je generirani niz α jednak nizu završnih znakova w , TS promijeni stanje u prihvatljivo stanje i prihvati niz završnih znakova w . Simulacija rada gramatike G izvodi se na sljedeći način:

- 1) TS nedeterministički izabere mjesto i u nizu α koji je zapisan na drugoj traci, gdje je $1 \leq i \leq |\alpha|$. Nedeterministički izbor ima za posljedicu istodobno izvođenje procesa simulacije za sve vrijednosti mjesta i za koja vrijedi $1 \leq i \leq |\alpha|$. Ostala tri koraka opisuju proces simulacije za jedan od $|\alpha|$ izbora.
- 2) TS nedeterministički izabere produkciju $\beta \rightarrow \gamma$ gramatike G . Nedeterministički izbor ima za posljedicu istodobno izvođenje procesa simulacije za sve produkcije gramatike G . Ostala dva koraka opisuju proces simulacije za izbor produkcije $\beta \rightarrow \gamma$.
- 3) Nalazi li se niz β na mjestu i , niz β zamjenjuje se nizom γ .
- 4) Niz generiran na drugoj traci uspoređuje se s nizom w koji je zapisan na prvoj traci. Ako su nizovi jednaki, onda TS M zaustavlja daljnji rad i prihvaca niz w . Nisu li nizovi jednaki, rad TS nastavlja se od koraka (1).

Budući da se u koraku (3) simulacije generiraju nizovi primjenom produkcija gramatike G i budući da nedeterministički izbori u koracima (1) i (2) generiraju sve nizove jezika $L(G)$, TS M prihvaca niz završnih znakova w ako i samo ako gramatika G generira niz w .

4.2.2 Konstrukcija gramatike za jezik zadan TS

Ako TS M prihvaca rekurzivno prebrojiv jezik $L(M)$, onda postoji gramatika G neograničenih produkcija koja generira jezik $L(G)=L(M)$.

Gradi se gramatika $G=(V, T, P, S)$ koja simulira rad TS $M=(Q, \Sigma, \Gamma, \delta, q_0, B, F)$ na sljedeći način. Gramatika G generira redom međunizove znakova koji predstavljaju konfiguracije TS M . Nezavršni znak q u međunizu generiranom gramatikom G predstavlja oznaku stanja TS M . Početna konfiguracija TS M simulira se međunizom u kojem su nezavršni znakovi oblika $[a_i, a_i]$, gdje su a_i ulazni znakovi iz skupa Σ . Prva komponenta nezavršnog znaka čuva znakove niza $a_1 a_2 \dots a_n$ tijekom simulacije, dok druga komponenta predstavlja znakove trake TS M . Na temelju sačuvanih znakova u prvoj komponenti nezavršnog znaka, gramatika G generira niz završnih znakova $a_1 a_2 \dots a_n$ ako i samo ako TS prihvaca taj isti niz. Početna konfiguracija $q_0 a_1 a_2 \dots a_n$ predstavlja se međunizom:

$$q_0[a_1, a_1] [a_2, a_2] \dots [a_n, a_n].$$

Početna konfiguracija TS određena je nezavršnim znakom q_0 i drugim komponentama nezavršnih znakova označenih uglatim zagradama. Za potrebe simulacije rada TS, gramatika G dodaje potreban broj oznaka praznih celija $[\epsilon, B]$ znakovima međuniza koji predstavljaju početnu konfiguraciju TS:

$$q_0[a_1, a_1] [a_2, a_2] \dots [a_n, a_n] [\epsilon, B]^m.$$

Početnoj konfiguraciji TS dodaje se m oznaka praznih celija $[\epsilon, B]$. Tijekom simulacije nezavršni znakovi gramatike poprimaju oblik:

$$[b, X],$$

gdje je b ulazni znak u skupu Σ ili prazni niz ϵ , a znak X jest u skupu znakova trake Γ . Konfiguracija $X_1 X_2 \dots X_{r-1} q X_r \dots X_s$ predstavlja se međunizom:

$$[a_1, X_1] [a_2, X_2] \dots [a_{r-1}, X_{r-1}] q [a_r, X_r] \dots [a_{n+m}, X_{n+m}]$$

gdje su a_1, a_2, \dots, a_n u skupu Σ , znakovi $a_{n+1}=a_{n+2}=\dots=a_{n+m}$ su prazni nizovi ϵ , znakovi X_1, X_2, \dots, X_s su u skupu Γ i znakovi $X_{s+1}=X_{s+2}=\dots=X_{n+m}$ su oznake praznine B .

Predstavlja li nezavršni znak q u generiranom međunizu $[a_1, X_1] [a_2, X_2] \dots [a_{r-1}, X_{r-1}] q [a_r, X_r] \dots [a_{n+m}, X_{n+m}]$ prihvatljivo stanje, TS M je u konfiguraciji koja prihvaca niz w . Pokaže li se simulacijom da TS prihvaca niz $a_1 a_2 \dots a_n$, gramatika G zamijeni sve nezavršne znakove oblika $[a_i, X]$ završnim znakovima a_i , sve nezavršne znakove oblika $[\epsilon, X]$ zamijeni praznim nizom i sve nezavršne znakove koji predstavljaju znakove stanja zamijeni praznim nizom. Gramatika G generira niz završnih znakova $a_1 a_2 \dots a_n$ ako i samo ako TS prihvaca niz $a_1 a_2 \dots a_n$.

Producije gramatike dijele se u četiri skupine. U prvoj skupini su produkcije koje generiraju međuniz koji predstavlja početnu konfiguraciju TS M :

- 1) $A_1 \rightarrow q_0 A_2,$
- 2) $A_2 \rightarrow [a, a] A_2, \text{ za sve } a \in \Sigma.$

A_1 i A_2 su nezavršni znakovi gramatike. Nezavršni znak A_1 jest početni nezavršni znak gramatike. Producije (1) i (2) generiraju međuniz koji predstavlja početnu konfiguraciju $q_0 a_1 a_2 \dots a_n$:

$$A_1 \xrightarrow{*} q_0 [a_1, a_1] [a_2, a_2] \dots [a_n, a_n] A_2$$

U drugoj skupni su produkcije koje dodaju potreban broj praznih čelija:

- 3) $A_2 \rightarrow A_3,$
- 4) $A_3 \rightarrow [\epsilon, B] A_3,$
- 5) $A_3 \rightarrow \epsilon.$

Neka se doda m praznih čelija prethodno generiranom nizu $[a_1, a_1] [a_2, a_2] \dots [a_n, a_n]$:

$$q_0 [a_1, a_1] [a_2, a_2] \dots [a_n, a_n] A_2 \xrightarrow{*} q_0 [a_1, a_1] [a_2, a_2] \dots [a_n, a_n] [\epsilon, B]^m$$

Treća skupina produkcija simulira rad TS M . Za sve ulazne znakove $a \in \Sigma \cup \{\epsilon\}$, za sva stanja $q \in Q$ i sve znakove trake $X \in \Gamma$ za koje postoji prijelaz $\delta(q, X) = (p, Y, R)$, gdje je $p \in Q$ i $Y \in \Gamma$, definira se produkcija:

$$6) \quad q [a, X] \rightarrow [a, Y] p$$

Producija gramatike mijenja samo drugu komponentu nezavršnog znaka, dok prva komponenta ostaje nepromijenjena.

Za prijelaz $\delta(q, X) = (p, Y, L)$, gdje su znakovi $a, b \in \Sigma \cup \{\epsilon\}$, stanje $p, q \in Q$ i znakovi $X, Y, Z \in \Gamma$, definira se produkcija:

$$7) \quad [b, Z] q [a, X] \rightarrow p [b, Z] [a, Y]$$

Dokazuje se matematičkom indukcijom s obzirom na broj primjenjenih funkcija prijelaza TS M da gramatika G primjenom produkcija (6) i (7) generira niz:

$$q_0 [a_1, a_1] [a_2, a_2] \dots [a_n, a_n] [\epsilon, B]^m \xrightarrow{*} [a_1, X_1] [a_2, X_2] \dots [a_{r-1}, X_{r-1}] q [a_r, X_r] \dots [a_{n+m}, X_{n+m}]$$

ako i samo ako TS M iz početne konfiguracije $q_0 a_1 a_2 \dots a_n$ prijeđe u konfiguraciju:

$$q_0 a_1 a_2 \dots a_n \xrightarrow{*} X_1 X_2 \dots X_{r-1} q X_r \dots X_s,$$

gdje su a_1, a_2, \dots, a_n u skupu Σ , znakovi $a_{n+1}=a_{n+2}=\dots=a_{n+m}=\epsilon$, znakovi X_1, X_2, \dots, X_s su u skupu Γ i znakovi $X_{s+1}=X_{s+2}=\dots=X_{n+m}=B$.

Za sva stanja q u skupu prihvatljivih stanja F , za sve znakove $a \in \Sigma \cup \{\epsilon\}$ i za sve znakove $X \in \Gamma$ definiraju se produkcije:

- 8) $[a, X] q \rightarrow q a q$
- 9) $q [a, X] \rightarrow q a q$
- 10) $q \rightarrow \epsilon$

Ako je stanje q prihvatljivo, gramatika G generirati niz završnih znakova $a_1 a_2 \dots a_n$:

$$[a_1, X_1] [a_2, X_2] \dots [a_{t-1}, X_{t-1}] q [a_t, X_t] \dots [a_{n+m}, X_{n+m}] \xrightarrow{*} a_1 a_2 \dots a_n.$$

4.3 Svojstva rekurzivnih i rekurzivno prebrojivih jezika

U ovom odjeljku pokazana su osnovna svojstva rekurzivnih i rekurzivno prebrojivih jezika s obzirom na operacije komplementa i unije, te su objašnjeni pojmovi izračunljivosti i odlučivosti jezika.

4.3.1 Svojstva zatvorenosti rekurzivnih i rekurzivno prebrojivih jezika

Turingov stroj koristi se za pokazivanje svojstva zatvorenosti rekurzivnih i rekurzivno prebrojivih jezika. Na temelju Turingovog stroja koji uvijek stane za svaki ulazni niz, pokazuje se da su rekurzivni jezici zatvoreni s obzirom na operacije unije i komplementa. Rekurzivno prebrojivi jezici zatvoreni su s obzirom na operaciju unije. Pokazano je sljedeće važno svojstvo komplementa rekurzivno prebrojivog jezika: ako je komplement rekurzivno prebrojivog jezika L rekurzivno prebrojiv, onda su jezik L i njegov komplement rekurzivni. Nije li komplement jezika L rekurzivno prebrojiv, jezik L nije sigurno rekurzivan, a moguće je da nije ni rekurzivno prebrojiv.

Za potrebe pokazivanja svojstva zatvorenosti rekurzivnih i rekurzivno prebrojivih jezika grade se složeni TS koji se sastoje od više prethodno izgrađenih TS. Za svaki prethodno izgrađeni TS definira se zasebna komponenta stanja i zasebne trake složenog TS.

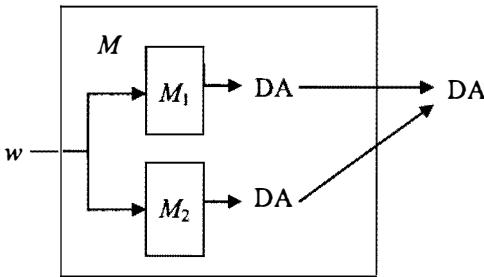
Unija

Unija dvaju rekurzivnih jezika jest rekurzivni jezik.

Neka TS M_1 prihvata rekurzivni jezik L_1 , a TS M_2 neka prihvata rekurzivni jezik L_2 . Budući da su jezici L_1 i L_2 rekurzivni i budući da TS M_1 i TS M_2 uвijek stane za svaki ulazni niz, rad TS M_1 i rad TS M_2 moguće je slijedno simulirati. Slijedni proces simulacije prikazan je shematski na slici 4.9. Ako TS M_1 stane i prihvati niz, onda stane TS M i prihvati niz. Stane li TS M_1 i ne prihvati niz, TS M pokrene simulaciju rada TS M_2 . Ako TS M_2 stane i prihvati niz, onda stane TS M i prihvati niz. Stane li TS M_2 i ne prihvati li niz, a budući da prethodno ni TS M_1 nije prihvatio niz, TS M stane i ne prihvati niz.

TS M prihvati niz w ako i samo ako niz w prihvati TS M_1 ili TS M_2 . Niz se ne prihvata ako ni TS M_1 ni TS M_2 ne prihvate niz. Budući da TS M uвijek stane za svaki ulazni niz, jezik $L(M_1) \cup L(M_2)$ jest rekurzivan.

Unija dva rekurzivno prebrojiva jezika jest rekurzivno prebrojiv jezik.

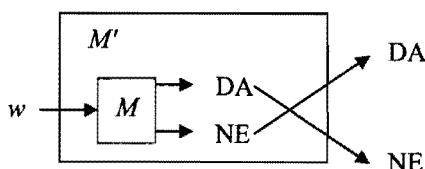


Slika 4.10: Konstrukcija TS M koji prihvata uniju dva rekurzivno prebrojiva jezika $L(M_1)$ i $L(M_2)$

Umjesto slijednog procesa simulacije, za rekurzivno prebrojive jezike koristi se paralelni proces simulacije. Gradi se TS M koji na zasebnim trakama istodobno simulira rad TS M_1 i rad TS M_2 . Ako bilo TS M_1 ili TS M_2 stane i prihvati niz, onda TS M stane i prihvati niz. Paralelni proces simulacije prikazan je shematski na slici 4.10.

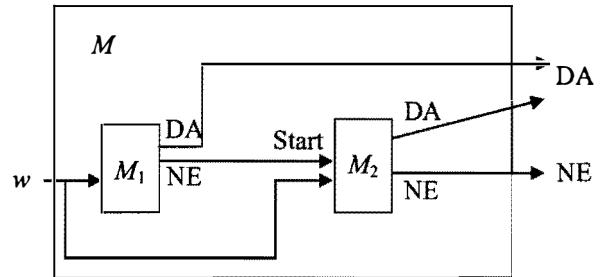
Komplement

Komplement rekurzivnog jezika jest rekurzivni jezik.



Slika 4.11: Konstrukcija TS M' koji prihvata komplement rekurzivnog jezika $L(M)$

Ako je L rekurzivni jezik, onda postoji TS M koji ga prihvata i koji uвijek stane za svaki ulazni niz w . Ako je stanje TS M prihvatljivo, niz w se prihvata. Nije li stanje TS M prihvatljivo, niz w se ne prihvata. Za prihvatanje komplementa rekurzivnog jezika $L' = L^c$ gradi se TS M' . TS M' simulira rad TS M . Ako TS M uđe u prihvatljivo stanje i stane, onda TS M' stane i ne prihvati niz. Ne uđe li TS M u prihvatljivo stanje i stane, TS M' stane i prihvati niz. Na temelju činjenice da TS M' prihvata niz ako i samo ako TS M ne prihvata niz i na temelju činjenice da TS M' ne prihvata niz ako i samo ako TS M prihvata niz, zaključuje se da TS M' prihvata



Slika 4.9: Konstrukcija TS M koji prihvata uniju rekurzivnih jezika $L(M_1)$ i $L(M_2)$

komplement jezika L , odnosno $L' = L^c$. Budući da TS M' stane za bilo koji ulazni niz, komplement rekurzivnog jezika jest rekurzivni jezik. Konstrukcija TS M' prikazana je shematski na slici 4.11.

Ako su jezik L_1 i njegov komplement $L_2 = L_1^c$ rekurzivno prebrojivi, onda su oba jezika rekurzivna.

Neka TS M_1 prihvaca rekurzivno prebrojiv jezik L_1 , a TS M_2 neka prihvaca rekurzivno prebrojiv jezik L_2 koji je komplement rekurzivno prebrojivog jezika L_1 , odnosno $L_2 = L_1^c$. Gradi se TS M koji prihvaca niz w ako i samo ako TS M_1 prihvaca niz w , odnosno TS M ne prihvaca niz w ako i samo ako TS M_2 prihvaca niz w . Shema TS M prikazana je na slici 4.12. Budući da je niz w u jeziku L_1 ili u jeziku $L_2 = L_1^c$, jedan i samo jedan TS M_1 ili TS M_2 prihvaca niz. TS M uvijek stane i jednoznačno odluči da li se niz prihvaca. Ako su jezik L_1 i njegov komplement rekurzivno prebrojivi, onda je za jezik L_1 moguće izgraditi TS M koji uvijek stane za svaki ulazni niz, odnosno jezik L_1 jest rekurzivan.

Zamijene li se uloge TS M_1 i TS M_2 , ili se primjeni svojstvo komplementa rekurzivnog jezika, moguće je pokazati da je jezik $L_2 = L_1^c$ također rekurzivan.

Na temelju svojstva komplementa, za jezik L i njegov komplement L^c vrijedi jedno od tri svojstva:

- 1) Oba jezika L i L^c su rekurzivna.
- 2) Oba jezika L i L^c nisu rekurzivno prebrojivi.
- 3) Ako je jezik L rekurzivno prebrojiv, ali ne i rekurzivan, onda jezik L^c nije rekurzivno prebrojiv.

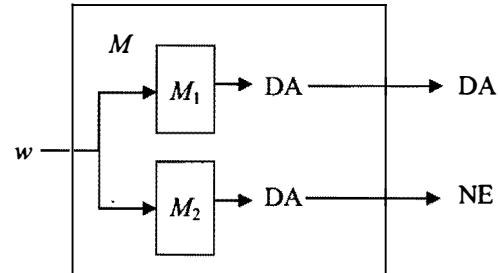
Svojstva (1) do (3) koriste se u dokazu postojanja rekurzivno prebrojivih jezika koji nisu rekurzivni.

4.3.2 Izračunljivost

Izračunljivost nije moguće odrediti općom, formalnom i preciznom definicijom. Zato se primjenjuje sljedeća intuitivna definicija: problem (na primjer prihvatanje jezika, računanje cijelobrojnih funkcija, generiranje jezika, itd.) jest izračunljiv ako postoji automat koji "mehaničkim" postupkom korak po korak rješava zadani problem. Intuitivna definicija ne nameće nikakvih ograničenja na broj koraka i na veličinu spremnika potrebnih za rješavanje problema. Nadalje, intuitivna definicija ne zahtijeva da proces računanja "mehaničkim" postupkom ikada stane. Definicija je data u najširem obliku i jedino što se zahtijeva jest mogućnost raščlambe problema na slijed "mehaničkih" koraka koji ga rješavaju.

Church-Turingova hipoteza

Na temelju intuitivne definicije izračunljivosti, Church-Turingova hipoteza ili Churchova hipoteza određuje da se izračunljive funkcije poistovjećuju s klasom parcijalno rekurzivnih funkcija. Budući da ne



Slika 4.12: Shema dokaza da je jezik $L(M)$ rekurzivni jezik ako su $L=L(M_1)$ i njegov komplement $L^c=L(M_2)$ rekurzivno prebrojivi jezici

postoji formalna definicija izračunljivosti, nije moguće dokazati točnost Church-Turingove hipoteze, već se samo pokazuje njezina prikladnost i razumnost.

Parcijalno rekurzivne funkcije su izračunljive, jer je za njihovo računanje moguće izgraditi TS. TS računa parcijalno rekurzivne funkcije "mehaničkim" putem korak po korak primjenom zadanih funkcija prijelaza. Budući da TS ne mora nužno stati za sve ulazne nizove, izračunljivost parcijalno rekurzivnih funkcija zasnovana je na najširoj definiciji.

Budući da su svi poznati formalizmi, kao što su λ -račun, Post sustavi, opće rekurzivne funkcije, itd., istovjetni klasi parcijalno rekurzivnih funkcija, sve izračunljive funkcije su ujedno parcijalno rekurzivne. Nadalje, apstraktne modele računala, kao što je na primjer model računala zasnovanog na memoriji sa slobodnim pristupom RAM (*RAM - random access machine*), moguće je simulirati primjenom TS. Primjena TS omogućuje ugradnju bilo kojeg algoritma, te simulaciju bilo kojeg programskog jezika suvremenih računala.

Simulacija računala zasnovanog na memoriji sa slobodnim pristupom Turingovim strojem

Rad apstraktnog računala RAM zasniva se na memoriji sa slobodnim pristupom i registrima. *Memorija sa slobodnim pristupom* sastoji se od beskonačnog niza memorijskih lokacija. Adrese memorijskih lokacija su brojevi 0, 1, 2, ..., a sadržaji memorijskih lokacija su proizvoljni cijeli brojevi.

Računalo RAM ima konačni broj *registara* u koje se spremaju cjelobrojne vrijednosti. Ovisno o namjeni, registri se dijele na radne registre, programsko brojilo i memorijski adresni registar. U *radnim registrima* nalaze se vrijednosti operanada aritmetičko-logičkih operacija. Nakon izvedene aritmetičko-logičke operacije, vrijednost rezultata spremi se u jedan od radnih registara. U *programskom brojilu* jest adresa naredbe koja se sljedeća izvodi. *Memorijski adresni registar* sadrži adresu memorijske lokacije iz koje se dohvata podatak ili sadrži adresu memorijske lokacije u koju se spremi podatak.

Budući da je rad bilo kojeg digitalnog računala moguće zasnovati na malom naredbenom skupu, za potpuni rad apstraktnog računala RAM nije potrebno zadati veliki broj naredaba. Računala sa smanjenim skupom naredaba RISC (*RISC - reduced instruction set computer*) dokaz su da naredbeni skup digitalnog računala ne mora biti velik.

Rad apstraktnog računala RAM i izvođenje njegovih naredbi moguće je simulirati primjenom Turingovog stroja.

Apstraktno računalo RAM simulira se primjenom TS s višestrukim trakama. Prva traka koristi se za spremanje adresa i sadržaja memorijskih lokacija. Adrese i sadržaji memorijskih lokacija spremaju se na *memorijsku traku* na sljedeći način:

$\# 0 * v_0 \# 1 * v_1 \# 10 * v_2 \# \dots \# k * v_k$

gdje je i adresa memorijske lokacije, a v_i je sadržaj memorijske lokacije. Vrijednosti adresa i sadržaja memorijskih lokacija zapisuju se binarnim brojevima. Budući da u bilo kojem trenutku rada računalo RAM koristi konačni broj memorijskih lokacija, TS na memorijskoj traci spremi adrese i vrijednosti samo onih lokacija koje se koriste. Definira se da je vrijednost svih ostalih memorijskih lokacija jednaka 0.

Druga traka koristi se za spremanje sadržaja radnih registara. Registarska traka organizira se na sličan način kao i memorijska traka. Adresni dio registrske trake adresira registre. Budući da ima konačni broj registara i njihov broj se ne mijenja tijekom rada računala RAM, adresni dio registrske trake moguće je izostaviti.

Na treću traku zapisuje se vrijednost programskog brojila, a na četvrtu traku zapisuje se sadržaj memorijskog adresnog registra.

Prvih desetak bitova koristi se za kodiranje naredaba. Na primjer, naredba LOAD dohvaća podatak sa zadane adrese u jedan od registara, naredba STORE sprema podatak iz jednog od registara na zadatu adresu, naredba ADD zbraja sadržaje zadanih registara i/ili memorijskih lokacija, naredba SUB oduzima sadržaje zadanih registara i/ili memorijskih lokacija, itd. Ostali bitovi naredbe određuju adresu memorijске lokacije i adresu registra pohrane operanada.

Postoje dva osnovna načina simulacije izvođenja naredbi računala RAM. Prvi način koristi funkcije prijelaza TS. Nakon dekodiranja naredbe, jedna od komponenti stanja TS poprimi vrijednost tipa naredbe. Za sve tipove naredbi definiraju se zasebne funkcije prijelaza koje određuju način korištenja memorijskih lokacija zapisanih na prvoj traci, radnih registara zapisanih na drugoj traci, programskog brojila zapisanog na trećoj traci i memoriskog adresnog registra zapisanog na četvrtoj traci.

Drugi način simulacije izvođenja naredbi koristi mikrokod zapisan na posebnoj traci TS. Traka ima zapisan mikrokod za sve tipove naredbi. Mikrokod naredbe određuje način korištenja raznih traka na kojima su zapisani sadržaji memorijskih lokacija, registara, programskog brojila i memoriskog adresnog registra.

Neka je u programskom brojilu cjelobrojna vrijednost i . Simulacija izvođenja naredbe računala RAM započinje simulacijom dohvata naredbe:

Dohvat naredbe. TS traži adresu i na memorijskoj traci, odnosno traži niz znakova # i . Ako se ne pronađe memorijска lokacija koja ima adresu i , onda nema ni naredbe na zadanoj adresi i proces simulacije računala RAM se zaustavlja. Pronađe li se memorijска lokacija koja ima adresu i , pristupa se dekodiranju naredbe spremljene na zadanoj adresi.

Dekodiranje naredbe. TS čita prvih desetak bitova iza oznake * koji određuju tip naredbe. Pročitani bitovi prepišu se u komponentu stanja TS koja određuje tip naredbe, ili se pročitani bitovi koriste za pretraživanje trake na kojoj su zapisani mikrokodovi naredbi. Pretpostavimo da se dekodira naredba **ADD R₂, j**. Naredba zbraja sadržaj registra **R₂** i sadržaj memorijskе lokacije na adresi **j**, te sprema rezultat zbrajanja u registar **R₂**. Iza bitova koda naredbe slijede bitovi koji određuju adresu registru **R₂** i bitovi koji određuju adresu memorijskе lokacije **j**.

Obnavljanje programskog brojila. Programsko brojilo poveća se za 1 i pokazuje na sljedeću naredbu računala RAM.

Dohvat operanada. TS ponovno pretražuje memorijsku traku. TS traži adresu **j** da dohvati vrijednost drugog operanda. Ako se adresa **j** ne pronađe, onda je vrijednost drugog operanda jednaka 0, jer na početku rada računalo RAM postavi vrijednosti svih memorijskih lokacija u 0. Ne zapiše li TS ništa u memorijsku lokaciju koja ima adresu **j**, sadržaj te lokacije jednak je 0. Budući da je vrijednost drugog operanda jednak 0, naredba **ADD R₂, j** ne mijenja vrijednost registra **R₂**, izvođenje naredbe završava i nastavlja se izvođenje sljedeće naredbe. Ako sadržaj memorijskе lokacije na adresi **j** nije jednak 0, onda se pristupa izvršenju naredbe.

Izvršenje naredbe. Sadržaj memorijskе lokacije na adresi **j** zbraja se sa sadržajem registra **R₂**. Registar **R₂** na registarskoj traci pronalazi se brojanjem graničnika koji se koriste za odvajanje sadržaja pojedinih registara. Rezultat zbrajanja sprema se na registarsku traku na mjesto pohrane sadržaja registra **R₂**. Simulacija računala RAM nastavlja se simulacijom izvođenja naredbe na koju pokazuje sadržaj programskog brojila.

Neizračunljivost dijagonalnog jezika

Ako je moguće izgraditi TS M koji prihvata jezik L , onda je na temelju definicije izračunljivosti i Church-Turingove hipoteze jezik L izračunljiv. Na primjer, rekurzivno prebrojivi jezici su izračunljivi. Nije

li moguće izgraditi TS M koji ga prihvaća, jezik L nije izračunljiv. Dijagonalni jezik L_d je primjer neizračunljivog jezika za koji nije moguće izgraditi TS M koji ga prihvaća.

Gradnja dijagonalnog jezika L_d započinje kodiranjem funkcija prijelaza proizvoljnog TS. Budući da je za bilo koji TS moguće izgraditi istovjetni TS s $n-1$ neprihvatljivih stanja, jednim prihvatljivim stanjem i skupom znakova trake $\{0, 1, B\}$, za kodiranje proizvoljnog TS koristi se sljedeći model TS: $M=(Q, \{0, 1\}, \{0, 1, B\}, \delta, q_1, B, \{q_2\})$. Skup stanja Q ima n stanja q_1, q_2, \dots, q_n . Stanje q_1 jest početno stanje, a stanje q_2 jest prihvatljivo stanje. U tablici 4.5 definirani su kodovi za sve znakove TS.

	Znak	Sinonim	Kod
Znak trake	0	X_1	0
	1	X_2	00
	B	X_3	000
Oznake pomaka glave	L	D_1	0
	R	D_2	00
Oznake stanja	q_1		0
	q_2		00
	---		---
	q_n		0^n

Tablica 4.5: Kodiranje znakova TS M

U tablici su posebnim sinonimima označeni znakovi trake i oznake pomaka glave. Sinonimi omogućuju da se proces kodiranja funkcija prijelaza lakše opiše za opći slučaj. Primjenom kodova iz tablice 4.5 funkcija prijelaza:

$$\delta(q_i, X_j) = (q_k, X_l, D_m)$$

kodira se na sljedeći način:

$$0^i 10^j 10^k 10^l 10^m.$$

Budući da skup znakova trake ima tri znaka $\{0, 1, B\}$ i budući da postoje dva pomaka glave $\{L, R\}$, za bilo koji ispravno napisan kod mora vrijediti: $1 \leq j \leq 3$, $1 \leq l \leq 3$ i $1 \leq m \leq 2$. Ukupni broj stanja n je proizvoljan i zato nema ograničenja za vrijednosti brojeva i i k .

Binarni kod za TS M je:

$$111 \text{ kod}_1 11 \text{ kod}_2 11 \dots 11 \text{ kod}_r 111,$$

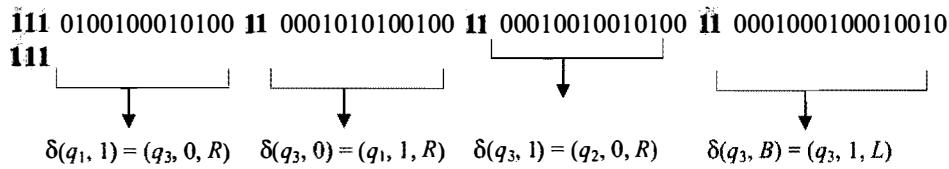
gdje su kod_p kodovi funkcije prijelaza oblika $0^i 10^j 10^k 10^l 10^m$. Budući da funkcije prijelaza nije potrebno poredati određenim redoslijedom, moguće je da jedan TS ima više različitih kodova. Međutim, bilo koji kod predstavlja najviše jedan TS.

Nadalje, bilo koji *niz binarnih znakova* 0 i 1 moguće je promatrati kao kod TS. Određeni binarni niz je kod najviše jednog TS, dok mnogi binarni nizovi nisu pravilni kodovi niti jednog TS. Ako binarni niz nije ispravno napisan kod, onda se pretpostavi da taj niz binarnih znakova predstavlja TS koji nema definiran niti jedan prijelaz. Budući da početno stanje nije prihvatljivo, jezik koji prihvaća takav TS je prazan skup.

Primjer 4.9. Zadan je TS $M=(\{q_1, q_2, q_3\}, \{0, 1\}, \{0, 1, B\}, \delta, q_1, B, \{q_2\})$ sa sljedećim funkcijama prijelaza:

$$\begin{aligned} \delta(q_1, 1) &= (q_3, 0, R) & \delta(q_3, 1) &= (q_2, 0, R) \\ \delta(q_3, 0) &= (q_1, 1, R) & \delta(q_3, B) &= (q_3, 1, L) \end{aligned}$$

Binarni kod TS M jest:



Neka je vrijednost indeksa j u označi TS M_j jednaka cijelobrojnoj vrijednosti njegovog binarnog koda. Kodovi TS M_j poredaju se u stupce dvodimenzionalne tablice prema njihovim rastućim cijelobrojnim vrijednostima. Ulazni nizovi w_i , koji se sastoje isključivo od znakova 0 i 1, poredaju se kanonskim slijedom. Niz w_i jest i -ti niz u kanonskom slijedu. Kanonski slijed definiran je u odjeljku 4.1.5. Oznaka stupaca j dvodimenzionalne tablice na slici 4.13 definira cijelobrojnu vrijednost binarnog koda TS M_j , a oznaka retka i definira i -ti niz w_i u kanonskom slijedu.

Elementi tablice (i, j) označeni su znakovima 0 ili 1. Znak 1 označava da TS M_j prihvata niz w_i , dok znak 0 označava da TS M_j ne prihvata niz w_i . Budući da binarni nizovi znakova koji nisu pravilno napisani kodovi predstavljaju TS koji prihvataju jezike koji su prazni skupovi (TS nemaju definiran niti jedan prijelaz, a početno stanje nije prihvatljivo), svi elementi stupaca na početku tablice sadrže znakove 0 (jezici koje prihvataju TS M_j s malom cijelobrojnom vrijednošću indeksa j su prazni skupovi).

		j	\rightarrow			
			x	$x+1$	$x+2$	$x+3$
i	1	1	0	0	-	-
	2	0	0	1	0	-
\downarrow	-	-	-	-	-	-
x	0	0	0	0	0	1
$x+1$	0	0	1	1	0	-
$x+2$	0	0	1	0	0	-
$x+3$	0	0	0	0	0	1
-	-	-	-	-	-	-

Slika 4.13: Tablica prihvatanja niza w_i primjenom TS M_j

Dijagonalni elementi tablice određuju *dijagonalni jezik* L_d . Niz w_i jest u jeziku L_d ako i samo ako M_i ne prihvata niz w_i , odnosno niz w_i jest u jeziku L_d ako element tablice (i, i) ima vrijednost 0.

Budući da su u stupcima tablice navedeni kodovi svih TS, prepostavimo da postoji TS M_d koji prihvata jezik $L_d = L(M_d)$. Određivanje vrijednosti elemenata tablice (d, d) dovodi do sljedećih suprotnosti.

Prvo se prepostavi da je vrijednost elementa tablice (d, d) jednaka 0. Ako je vrijednost elementa tablice (d, d) jednaka 0, onda TS M_d ne prihvata niz w_d . TS M_d prihvata dijagonalni jezik L_d u kojem su svi nizovi w_i za koje vrijedi da je vrijednost elementa tablice (i, i) jednaka 0. Budući da je vrijednost elementa tablice (d, d) jednaka 0, niz w_d jest u dijagonalnom jeziku L_d . Ako je niz w_d u dijagonalnom jeziku L_d , onda TS M_d prihvata niz w_d i vrijednost elementa tablice (d, d) mora biti 1, što je suprotno prepostavci da je vrijednost elementa tablice (d, d) jednaka 0.

Prepostavi se da je vrijednost elementa tablice (d, d) jednaka 1. Ako je vrijednost elementa tablice (d, d) jednaka 1, onda TS M_d prihvata niz w_d . Budući da je vrijednost elementa tablice (d, d) jednaka 1, niz w_d nije u dijagonalnom jeziku L_d . Nije li niz w_d u dijagonalnom jeziku L_d , TS M_d ne prihvata niz w_d i vrijednost elementa tablice (d, d) mora biti 0, što je suprotno prepostavci da je vrijednost elementa tablice (d, d) jednaka 1.

Obje prepostavke o vrijednostima elementa tablice (d, d) dovode do suprotnosti što potvrđuje da ne postoji TS M_d koji prihvata jezik L_d . Na temelju sljedećih činjenica:

- i) datim nizom kodova nabrojani su svi TS koji imaju skup znakove trake $\{0, 1, B\}$,
- ii) u datom nizu ne postoji niti jedan TS M_d koji prihvata dijagonalni jezik L_d ,
- iii) u poglavljiju 4.1.4 pokazano je da za prihvatanje bilo kojeg rekursivno prebrojivog jezika dovoljno je da TS ima skup znakove trake jednak $\{0, 1, B\}$,

zaključuje se da ne postoji niti jedan TS koji prihvata dijagonalni jezik L_d , odnosno dijagonalni jezik L_d nije rekursivno prebrojiv i nije izračunljiv.

4.3.3 Odlučivost

Kaže se da su rekurzivni jezici *odlučivi*, jer ih prihvataju Turingovi strojevi koji uvijek stane i odluče o prihvaćanju ili neprihvaćanju niza. Budući da za rekurzivno prebrojive jezike ne postoji TS koji uvijek stane, rekurzivno prebrojivi jezici nisu odlučivi. Nije li niz u jeziku, moguće je da TS nikad ne stane. Ako TS nikad ne stane, nije se moguće odlučiti prihvata li se niz. Dok su rekurzivni jezici odlučivi i izračunljivi, rekurzivno prebrojivi jezici su izračunljivi, ali nisu odlučivi. Univerzalni jezik L_u jest primjer rekurzivno prebrojivog jezika, odnosno izračunljivog jezika, koji nije rekurzivan, odnosno koji nije odlučiv.

Univerzalni Turingov stroj i univerzalni jezik

Opis osnovnog načina rada univerzalnog TS M_u dat je u odjeljku 4.1.4. Univerzalni TS M_u ima tri trake. Prva traka jest ulazna traka i na nju se zapiše niz $\langle M, w \rangle$. Niz $\langle M, w \rangle$ kodira se na sljedeći način:

$111kod_111kod_211 \dots 11kod_r111w,$

gdje su kod_p kodovi funkcije prijelaza TS M definirani u odjeljku 4.3.2, a w jest ulazni niz. Kodira se TS $M = (Q, \{0, 1\}, \{0, 1, B\}, \delta, q_1, B, \{q_2\})$ koji ima $n-1$ neprihvativih stanja, jedno prihvativivo stanje q_2 , početno stanje q_1 i skup znakova trake $\{0, 1, B\}$. Sadržaj druge trake TS M_u simulira sadržaj trake TS M . Na treću traku zapisuje se stanje q , TS M kao niz 0^l . Rad TS M_u izvodi se na sljedeći način:

- U prvom koraku provjerava se ispravnost zapisa ulaznog niza $\langle M, w \rangle$ na prvoj traci: prefiks niza $\langle M, w \rangle$ mora biti u formatu $111kod_111kod_211 \dots 11kod_r111$, funkcije prijelaza moraju biti determinističke, odnosno dva koda funkcije prijelaza ne smiju započeti istim prefiksom $0^l 10^j$ i za sve kodove $0^l 10^j 10^k 10^m$ mora vrijediti da je $1 \leq j \leq 3$, $1 \leq l \leq 3$ i $1 \leq m \leq 2$. Tijekom provjere ispravnosti zapisa ulaznog niza treća traka koristi se za potrebe uspoređivanja nizova znakova. Ako ulazni niz $\langle M, w \rangle$ nije ispravno zapisan, rad TS M_u se zaustavlja i niz se ne prihvata.
- U drugom koraku niz w prepisuje se s prve trake na drugu traku. Na treću traku zapiše se niz od jednog znaka 0 što označava početno stanje q_1 . Sve tri glave pomaknu se na krajne lijeve znakove.
- Neka je na trećoj traci zapisano stanje q_i i neka je na drugoj traci zapisan znak X_j . TS M_u pretražuje prvu traku tražeći funkciju prijelaza koja ima prefiks $110^l 10^j$.

Ne pronađe li se prefiks $110^l 10^j$, TS M nema dalnjih prijelaza i rad TS M_u se zaustavlja. Ako je na trećoj traci zapisan niz 00 , odnosno ako je zapisan kod za prihvativivo stanje q_2 , onda se niz $\langle M, w \rangle$ prihvata. Nije li na trećoj traci zapisan niz 00 , niz $\langle M, w \rangle$ se ne prihvata.

Pronađe li se tražena funkcija prijelaza $110^l 10^j 10^k 10^m$, na treću traku zapiše se novo stanje q_k kao niz 0^k , na drugu traku zapiše se znak X_l i glava druge trake pomakne se u smjeru D_m . Simulacija rada TS M nastavlja se traženjem funkcije prijelaza koja ima prefiks $110^k 10^l$.

TS M_u prihvata niz $\langle M, w \rangle$ ako i samo ako TS M prihvata niz w . Ne stane li nikad TS M za niz w , onda nikad ne stane ni TS M_u za niz $\langle M, w \rangle$. Zaustavi li se TS M bez da prihvati niz w , TS M_u stane i ne prihvata niz $\langle M, w \rangle$.

Univerzalni TS M_u prihvata *univerzalni jezik* L_u koji se definira na sljedeći način: $L_u = \{\langle M, w \rangle \mid \text{TS } M \text{ prihvata niz } w\}$, gdje je $\langle M, w \rangle$ kodirani zapis TS M i niza w . Jezik L_u naziva se univerzalan, jer je pitanje: "Da li TS M prihvata niz w ?" istovjetno pitanju: "Da li je niz $\langle M, w \rangle$ u jeziku L_u ".

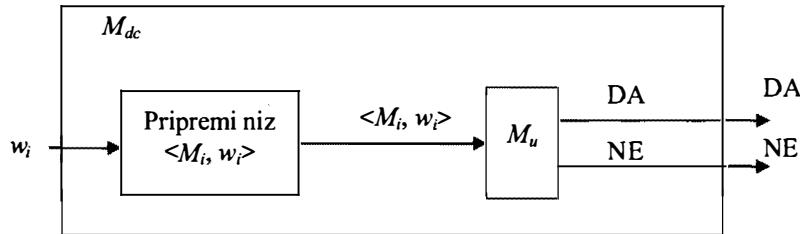
Izračunljivost i neodlučivost univerzalnog jezika

Univerzalni jezik L_u jest rekurzivno prebrojiv, odnosno izračunljiv.

Budući da je pokazano da je za univerzalni jezik L_u moguće izgraditi TS M_u , jezik L_u je rekurzivno prebrojiv, odnosno izračunljiv.

Univerzalni jezik L_u nije rekurzivan, odnosno nije odlučiv.

Svojstvo nerekurzivnosti, odnosno svojstvo neodlučivosti univerzalnog jezika L_u , pokazuje se na temelju svojstava dijagonalnog jezika L_d i njegovog komplementa L_d^c . Dijagonalni jezik $L_d = \{w_i \mid \text{TS } M_i \text{ ne prihvata niz } w_i\}$ definiran je u odjeljku 4.3.2. Komplement dijagonalnog jezika L_d jednak je $L_d^c = \{w_i \mid \text{TS } M_i \text{ prihvata niz } w_i\}$. Budući da je pokazano da dijagonalni jezik L_d nije rekurzivno prebrojiv, na temelju svojstva komplementa zaključuje se da njegov komplement L_d^c sigurno nije rekurzivan.



Slika 4.14: Konstrukcija TS M_{dc} za jezik L_d^c
primjenom univerzalnog TS M_u

Prepostavimo da je jezik L_u rekurzivan, odnosno prepostavimo da univerzalni TS M_u koji prihvata jezik L_u uvijek stane za svaki ulazni niz. Slika 4.14 pokazuje konstrukciju TS M_{dc} koji prihvata jezik L_d^c :

- 1) Pojavi li se na ulazu TS M_{dc} niz w_i , na temelju znakova niza w_i TS M_{dc} pripremi niz $\langle M_i, w_i \rangle$.
- 2) TS M_{dc} zapiše niz $\langle M_i, w_i \rangle$ na prvu traku univerzalnog TS M_u i pokrene njegovo izvođenje.

Ako je jezik L_u rekurzivan i ako TS M_u uvijek stane, onda TS M_{dc} uvijek stane za svaki ulazni niz. Na temelju činjenice da TS M_{dc} uvijek stane za svaki ulazni niz, zaključuje se da je jezik L_d^c rekurzivan, što je suprotno prethodno pokazanoj činjenici da jezik L_d^c nije rekurzivan. Budući da pretpostavka da je jezik L_u rekurzivan dovodi do suprotnosti, zaključuje se da jezik L_u nije rekurzivan.

5 KONTEKSTNO OVISNI JEZICI

Definicija kontekstno ovisnog jezika temelji se na kontekstno ovisnoj gramatici: neki jezik jest kontekstno ovisan ako i samo ako postoji kontekstno ovisna gramatika koja ga generira. Time je definirana istovjetnost kontekstno ovisne gramatike i kontekstno ovisnih jezika: za bilo koji kontekstno ovisni jezik moguće je izgraditi kontekstno ovisnu gramatiku koja ga generira, i obrnuto, bilo koja kontekstno ovisna gramatika generira jedan od kontekstno ovisnih jezika.

Klasa kontekstno ovisnih jezika jest pravi podskup klase rekurzivnih jezika. Većina jezika je u klasi kontekstno ovisnih jezika i teško je naći primjer jezika koji nije kontekstno ovisan. Na primer, univerzalni jezik L_u i dijagonalni jezik L_d nisu kontekstno ovisni jezici. Budući da su kontekstno ovisni jezici pravi podskup skupa rekurzivnih jezika i budući da univerzalni jezik L_u i dijagonalni jezik L_d nisu u klasi rekurzivnih jezika, jezici L_u i L_d nisu ni u klasi kontekstno ovisnih jezika. U odjeljku 5.3.4 pokazano je postojanje rekurzivnog jezika koji nije u klasi kontekstno ovisnih jezika.

U odjeljku 5.1 definira se kontekstno ovisna gramatika, a u odjeljku 5.2 automat koji prihvata kontekstno ovisne jezike. Automat je ograničeni model nedeterminističkog TS. Svojstva kontekstno ovisnih jezika objašnjena su u odjeljku 5.3.

5.1 Kontekstno ovisna gramatika

Neka se oblik produkcija gramatike $G=(V, T, P, S)$:

$$1) \quad \alpha \rightarrow \beta,$$

ograniči tako da je broj znakova desne strane produkcije β veći ili jednak broju znakova lijeve strane produkcije α , odnosno neka je $|\beta| \geq |\alpha|$. Nizovi α i β su proizvoljni nizovi završnih i nezavršnih znakova gramatike. Za niz α zahtijeva se da je različit od praznog niza ε . Zadovoljavaju li produkcije gramatike G dano ograničenje, kaže se da je gramatika G *kontekstno ovisna gramatika*. Naziv kontekstno ovisna gramatika dolazi od normalnog oblika produkcija:

$$2) \quad \alpha_1 A \alpha_2 \rightarrow \alpha_1 \beta \alpha_2,$$

gdje je A nezavršni znak gramatike, niz β je neprazni niz završnih i nezavršnih znakova gramatike ($\beta \neq \varepsilon$), te α_1 i α_2 su nizovi završnih i nezavršnih znakova gramatike. Za razliku od produkcija kontekstno neovisne gramatike, produkcije kontekstno ovisne gramatike dozvoljavaju zamjenu nezavršnog znaka A nizom završnih i nezavršnih znakova β ako i samo ako u generiranom međunizu ispred znaka A je niz završnih i nezavršnih znakova α_1 , a iza znaka A slijedi niz završnih i nezavršnih znakova α_2 . Budući da do zamjene dolazi samo u kontekstu α_1/α_2 , gramatika se naziva kontekstno ovisna gramatika.

Oblik produkcije (1) koristi se u dokazu istovjetnosti kontekstno ovisne gramatike i ograničenog modela nedeterminističkog TS, dok se normalni oblik produkcija (2) koristi u dokazima svojstva zatvorenosti kontekstno ovisnih jezika.

Primjer 5.1. Većina produkcija gramatike G zadane u primjeru 4.8 zadovoljava zahtjeve kontekstno ovisne gramatike, izuzev produkcija $CB \rightarrow E$ i $AE \rightarrow \varepsilon$. Desne strane tih produkcija imaju manji broj znakova od lijevih strana produkcija. Za zadani jezik $L = \{a^i \mid i > 0\}$ moguće je izgraditi kontekstno ovisnu gramatiku G' za koju vrijedi da je $L(G') = L(G)$.

Poteškoću pri gradnji produkcija kontekstno ovisne gramatike predstavlja simulacija graničnika A i B , te oznaka C , D i E . Nakon što se između graničnika A i B generira traženi niz završnih znakova a , produkcije gramatike G brišu nezavršne znakove koji imaju uloge graničnika i oznaka. Budući da desne strane produkcija kontekstno ovisne gramatike moraju biti duže ili jednako dugačke kao lijeve strane, graničnici i oznake označavaju se složenim nezavršnim znakovima umjesto jedinstvenim nezavršnim znakovima. Složeni nezavršni znakovi grupiraju graničnike i oznake zajedno sa završnim znakovima a . U skupu složenih nezavršnih znakova su sljedeći znakovi: $[ACaB]$, $[Aa]$, $[ACa]$, $[ADa]$, $[AEa]$, $[Ca]$, $[Da]$, $[Ea]$, $[aCB]$, $[CaB]$, $[aDB]$, $[aE]$, $[DaB]$ i $[aB]$. Na primjer, znak $[ACaB]$ je jedinstveni nezavršni znak, a ne niz nezavršnih i završnih znakova. Grupe produkcija kontekstno ovisne gramatike G' označene su istim brojevima kao i izvorne produkcije gramatike neograničenih produkcija G koja je zadana u primjeru 4.8 kako bi se omogućila lakša usporedba rada dviju gramatika:

1) $S \rightarrow [ACaB]$	4) $[aCB] \rightarrow [aE]$	7) $a[Ea] \rightarrow [Ea]a$
2) $[Ca]a \rightarrow aa[Ca]$	5) $a[Da] \rightarrow [Da]a$	$[aE] \rightarrow [Ea]$
$[Ca][aB] \rightarrow aa[CaB]$	$[aDB] \rightarrow [DaB]$	$[Aa][Ea] \rightarrow [AEa]a$
$[ACa]a \rightarrow [Aa]a[Ca]$	$[Aa][Da] \rightarrow [ADA]a$	8) $[AEa] \rightarrow a$
$[ACa][aB] \rightarrow [Aa]a[CaB]$	$a[DaB] \rightarrow [Da][aB]$	
$[ACaB] \rightarrow [Aa][aCB]$	$[Aa][DaB] \rightarrow [ADA][aB]$	
$[CaB] \rightarrow a[aCB]$	6) $[ADA] \rightarrow [ACa]$	
3) $[aCB] \rightarrow [aDB]$		

Izostave li se zagrade u međunizu koji generira gramatika G' , dobije se međuniz koji generira gramatika G . Gramatika G' generira niz aa na sljedeći način:

$$S \Rightarrow [ACaB] \Rightarrow [Aa] [aCB] \Rightarrow [Aa] [aE] \Rightarrow [Aa] [Ea] \Rightarrow [AEa] a \Rightarrow aa,$$

dok gramatika G generira isti niz aa na sljedeći način:

$$S \Rightarrow ACaB \Rightarrow AaaCB \Rightarrow AaaE \Rightarrow AaEa \Rightarrow AEaa \Rightarrow aa.$$

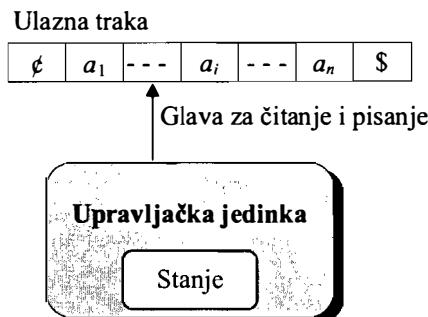
Gramatika G' generira niz $aaaa$ na sljedeći način:

$$\begin{aligned} S &\Rightarrow [ACaB] \Rightarrow [Aa] [aCB] \Rightarrow [Aa] [aDB] \Rightarrow [Aa] [DaB] \Rightarrow [ADA] [aB] \Rightarrow [ACa] \\ &\quad [aB] \Rightarrow \\ &\quad [Aa] a [CaB] \Rightarrow [Aa] aa [aCB] \Rightarrow [Aa] aa [aE] \Rightarrow [Aa] aa [Ea] \Rightarrow [Aa] a [Ea] a \\ &\quad \Rightarrow \\ &\quad [Aa] [Ea] aa \Rightarrow [AEa] aaa \Rightarrow aaaa, \end{aligned}$$

dok gramatika G generira isti niz $aaaa$ na sljedeći način:

$$\begin{aligned} S &\Rightarrow ACaB \Rightarrow AaaCB \Rightarrow AaaDB \Rightarrow AaDaB \Rightarrow ADaaB \Rightarrow ACaaB \Rightarrow \\ &AaaCaB \Rightarrow AaaaaCB \Rightarrow AaaaaE \Rightarrow AaaaEa \Rightarrow AaaEaa \Rightarrow \\ &AaEaaa \Rightarrow AEaaaa \Rightarrow aaaa. \end{aligned}$$

5.2 Linearno ograničen automat (LOA)



Slika 5.1: Model linearno-ograničenog automata (LOA)

Linearno ograničen automat (LOA) je ograničeni nedeterministički TS koji ima sljedeća svojstva:

- 1) Skup ulaznih znakova Σ sadrži dva posebna znaka: ϵ i $$$. Znak ϵ je lijevi graničnik na ulaznoj traci, a znak $$$ je desni graničnik.
- 2) Zabranjuje se pomak glave za čitanje i pisanje lijevo od znaka ϵ , te pomak glave za čitanje i pisanje desno od znaka $$$. Znakove ϵ i $$$ nije moguće prepisati drugim znakovima.

Slika 5.1 prikazuje model LOA. LOA je nedeterministički TS koji koristi samo dio trake na kojoj je zapisan ulazni niz w . Središnje čelije koriste se za spremanje ulaznog niza $w=a_1 a_2 \dots a_n$, a krajnje dvije čelije sadrže lijevi graničnik ϵ i desni graničnik $$$.

Linearno ograničeni automat (LOA) zadaje se formalno kao uređena osmorka:

$$loa = (Q, \Sigma, \Gamma, \delta, q_0, \epsilon, \$, F)$$

gdje se Q , Σ , Γ , δ , q_0 i F zadaju na isti način kao i za nedeterministički TS. Graničnici ϵ i $$$ su znakovi u skupu ulaznih znakova Σ .

LOA $M=(Q, \Sigma, \Gamma, \delta, q_0, \epsilon, \$, F)$ prihvaca jezik:

$$L(M)=\{w \mid w \in (\Sigma - \{\epsilon, \$\})^* \text{ i } q_0 \not\in w\$ \succ^* \alpha q \beta, \text{ gdje je } q \in F\}$$

Slično definiciji TS, zahtjeva se da LOA stane u slučaju prihvaćanja niza. Iako graničnici ϵ i $$$ su u skupu Σ i nalaze se zapisani na ulaznoj traci, ti znakovi nisu dio ulaznog niza w . Ako je moguće za prihvaćanje jezika L izgraditi deterministički LOA (DLOA), kaže se da je jezik L deterministički kontekstno ovisni jezik. Ako se posebno ne istakne, smatra se da je kontekstno ovisni jezik nedeterministički, odnosno da je potrebno izgraditi nedeterministički LOA. Oznaka LOA podrazumijeva nedeterministički LOA, dok oznaka DLOA označava deterministički LOA.

Naziv LOA nastao je na temelju sljedećeg svojstva. Ako je duljina radne trake TS M ograničena za bilo koji niz w linearom funkcijom f tako da TS M koristi najviše $f(|w|)$ ćelija, onda je moguće je izgraditi istovjetni TS M' koji koristi samo dio trake na koji je zapisan ulazni niz w . Svojstvo sažimanja duljine radnih traka TS za konstantni faktor opisano je u odjeljku 6.2.2.

5.2.1 Konstrukcija LOA za jezik zadan kontekstno ovisnom gramatikom

Ako kontekstno ovisna gramatika $G=(V, T, P, S)$ generira kontekstno ovisni jezik $L(G)$ (pretpostavlja se da $\epsilon \notin L(G)$), onda je za jezik $L(G)$ moguće izgraditi LOA M koji prihvata kontekstno ovisni jezik $L(M)=L(G)$.

Postupak gradnje LOA za jezik zadan kontekstno ovisnom gramatikom sličan je postupku gradnje TS za jezik zadan gramatikom neograničenih produkcija. Postupak je opisan u odjeljku 4.2.1. Umjesto dvije trake, LOA M koristi dva traga ulazne trake. U gornji trag LOA M zapiše niz $\$w\$$, a na početak donjeg traga LOA M zapiše početni nezavršni znak S gramatike G .

Za prazni niz ϵ LOA odmah stane i ne prihvati niz. Ako niz w nije prazan, LOA M primjeni postupak simulacije gramatike G opisan u odjeljku 4.2.1. Budući da se tijekom simulacije na donji trag ulazne trake generiraju nizovi primjenom produkcija gramatike G i budući da nedeterministički izbori generiraju sve nizove jezika $L(G)$, LOA M prihvata niz završnih znakova w ako i samo ako gramatika G generira niz w .

Tijekom rada LOA M ne koristi veći broj ćelija od duljine niza w . Budući da su desne strane svih produkcija kontekstno ovisne gramatike jednako dugačke ili duže od lijevih strana, u postupku generiranja niza:

$$S \xrightarrow{*} \alpha \xrightarrow{*} w$$

niti jedan međuniz α na donjem tragu ulazne trake nije duži od niza w koji je zapisan na gornjem tragu. Postane li generirani međuniz α duži od niza w , daljnja simulacija se zaustavlja bez ulaska LOA M u prihvatljivo stanje. Budući da nije moguće generirati niz w iz međuniza α u slučaju da je $|w| < |\alpha|$, nije potrebno da LOA M nastavi simulaciju generiranja međunizova niti za jedan niz α koji je duži od niza w .

5.2.2 Konstrukcija kontekstno ovisne gramatike za jezik zadan LOA

Ako LOA M prihvata jezik L , onda postoji kontekstno ovisna gramatika $G=(V, T, S)$ koja generira jezik $L(G)=L(M)-\{\epsilon\}$.

Postupak gradnje kontekstno ovisne gramatike G sličan je postupku gradnje gramatike neograničenih produkcija. Postupak je opisan u odjeljku 4.2.2. Postupci se razlikuju u načinu gradnje nezavršnih znakova gramatike. Znakovi ϵ , $\$$ i označe stanja grupiraju se zajedno sa susjednim znakovima trake u jedinstvene nezavršne znakove gramatike. Grupiranje više znakova u jedinstvene nezavršne znakove osigurava da su desne strane produkcija duže ili jednako dugačke kao lijeve strane produkcija.

Početna konfiguracija LOA jest $q_0 \notin a_1 a_2 \dots a_n \$$. Sljedeće produkcije gramatike generiraju međuniz koji predstavlja početnu konfiguraciju:

- 1) $A_1 \rightarrow [a, q_0 \notin a] A_2,$
- 2) $A_1 \rightarrow [a, q_0 \notin a \$],$

- 3) $A_2 \rightarrow [a, a] A_2$, za sve $a \in \Sigma$,
- 4) $A_2 \rightarrow [a, a \$]$, za sve $a \in (\Sigma - \{\$, \}\}$.

Produkcije koje simuliraju prijelaze LOA slične su produkcijama (6) i (7) u odjeljku 4.2.2. Oznake stanja grupiraju se zajedno sa znakovima trake:

- 5) $[b, q X] [a, Z] \rightarrow [b, Y] [a, p Z]$, 7) $[a, q \notin X \$] \rightarrow [a, \notin p X \$]$,
 $[b, q \notin X] \rightarrow [b, \notin p X]$, $[a, \notin q X \$] \rightarrow [a, \notin p Y \$]$,
 $[b, \notin q X] [a, Z \$] \rightarrow [b, Y] [a, p Z \$]$, $[a, \notin X q \$] \rightarrow [a, \notin p X \$]$,
 $[b, q X \$] \rightarrow [b, Y p \$]$, $[a, \notin q X \$] \rightarrow [a, p \notin Y \$]$.
- 6) $[b, Z] [a, q X] \rightarrow [b, p Z] [a, Y]$, [b, Z q \\$] \rightarrow [b, p Z \\$],
 $[b, Z] [a, q X \$] \rightarrow [b, p Z] [a, Y \$]$, [b, \notin Z] [a, q X] \rightarrow [b, \notin p Z] [a, Y],
 $[b, \notin q X] \rightarrow [b, p \notin Y]$, [b, \notin q X \\$] \rightarrow [b, p \notin Y \\$].

Ako je oznaka stanja q u skupu prihvatljivih stanja F , onda se na temelju nezavršnih znakova generiraju završni znakovi sljedećim produkcijama:

8) $[a, \alpha q \beta] \rightarrow a$,

za sve znakove a iz skupa $\Sigma - \{\$, \}$ i za sve nizove α i β , gdje su α i β u skupu Σ^* . Nizovi α i β su konačni i sastoje se od najviše jednog znaka trake i dva graničnika. Budući da je konačni broj znakova a i konačni broj različitih nizova $\alpha q \beta$, konačni broj je i produkcija oblika (8).

Nalazi li se do nezavršnog znaka završni znak, na temelju nezavršnog znaka generira se završni znak sljedećim produkcijama:

- 9) $[a, \alpha] b \rightarrow a b$,
10) $b [a, \alpha] \rightarrow b a$,

za sve znakove a i b iz skupa $\Sigma - \{\$, \}$ i za sve nizove α .

Budući da su desne strane produkcija duže ili jednako dugačke kao lijeve strane, izgrađene produkcije zadovoljavaju zahtjeve kontekstno ovisne gramatike.

Za razliku od LOA, za zadani TS nije moguće izgraditi kontekstno ovisnu gramatiku. Budući da konstruirana gramatika neograničenih produkcija u odjeljku 4.2.2 simulira beskonačnu traku zadanog TS, tijekom simulacije gramatika generira međunizove α duže od niza w . Nakon što TS uđe u jedno od prihvatljivih stanja, potrebno je sve nezavršne znakove zamijeniti završnim znakovima i generirati niz završnih znakova w . Budući da nije moguće unaprijed predvidjeti i ograničiti duljinu generiranih međunizova α , ne postoji niti jedna kontekstno ovisna gramatika s konačnim brojem produkcija koja bi sačuvala duljinu generiranog međuniza.

5.3 Svojstva kontekstno ovisnih jezika

U odjelu 5.3.1 koristi se normalni oblik produkcija kontekstno ovisne gramatike za pokazivanje svojstva zatvorenosti kontekstno ovisnih jezika s obzirom na operacije unije, nadovezivanja i Kleeneovog operatora L^+ . U odjelu 5.3.2 koristi se LOA za pokazivanje svojstava zatvorenosti s obzirom na operacije presjeka i komplementa.

U odjelicima 5.3.3 i 5.3.4 pokazano je da klasa kontekstno ovisnih jezika jest pravi podskup klase rekurzivnih jezika.

5.3.1 Unija, nadovezivanje i Kleeneov operator

Unija dvaju kontekstno ovisnih jezika jest kontekstno ovisni jezik.

Neka kontekstno ovisne gramatike $G_1=(V_1, T_1, P_1, S_1)$ i $G_2=(V_2, T_2, P_2, S_2)$ generiraju kontekstno ovisne jezike $L(G_1)$ i $L(G_2)$. Pretpostavlja se da su produkcije obje gramatike G_1 i G_2 u normalnom obliku i da je presjek skupova nezavršnih znakova V_1 i V_2 prazni skup. Za gradnju kontekstno ovisne gramatike $G_3=(V_3, T_3, P_3, S_3)$ koja generira jezik $L(G_3)=L(G_1)\cup L(G_2)$ koristi se postupak opisan u odjelu 3.3.1. Budući da sve produkcije sadrže barem jedan nezavršni znak na lijevoj strani i budući da je $V_1\cap V_2=\emptyset$, tijekom generiranja bilo kojeg niza w primjenom gramatike G_3 primjenjuju se produkcije jedne i samo jedne gramatike G_1 ili G_2 .

Nadovezivanje dva kontekstno ovisna jezika jest kontekstno ovisni jezik.

Neka kontekstno ovisne gramatike $G_1=(V_1, T_1, P_1, S_1)$ i $G_2=(V_2, T_2, P_2, S_2)$ generiraju kontekstno ovisne jezike $L(G_1)$ i $L(G_2)$. Pretpostavlja se da su produkcije obje gramatike G_1 i G_2 u normalnom obliku i da je presjek skupova nezavršnih znakova V_1 i V_2 prazni skup. Za gradnju kontekstno ovisne gramatike $G_4=(V_4, T_4, P_4, S_4)$ koja generira jezik $L(G_4)=L(G_1)L(G_2)$ koristi se postupak opisan u odjelu 3.3.1.

Da se izgradi gramatika G_4 , nije dovoljno osigurati da je presjek skupova nezavršnih znakova V_1 i V_2 prazni skup. Gramatika G_4 generira međunizove $\gamma\delta$ na sljedeći način: $S_4 \xrightarrow{*} S_1S_2 \xrightarrow{*} \gamma S_2 \xrightarrow{*} \gamma\delta$. Na primjer, neka je sufiks niza γ niz završnih znakova α_1 , neka je prefiks niza δ niz $A\alpha_2$ i neka je $\alpha_1A\alpha_2 \rightarrow \alpha_1\beta\alpha_2$ produkcija gramatike G_2 . Spajanjem susjednih znakova međunizova γ i δ nastaje niz $\alpha_1A\alpha_2$ koji je desna strana produkcije $\alpha_1A\alpha_2 \rightarrow \alpha_1\beta\alpha_2$. Primjenom produkcije $\alpha_1A\alpha_2 \rightarrow \alpha_1\beta\alpha_2$ na međuniz koji nastaje spajanjem znakova međunizova γ i δ moguće je generirati niz koji nije u jeziku $L(G_1)L(G_2)$.

Budući da nije moguće osigurati da je presjek skupova završnih znakova T_1 i T_2 prazni skup, definira se sljedeće ograničenje na oblik produkcija kontekstno ovisne gramatike: lijeva strana produkcija sastoji se isključivo od nezavršnih znakova gramatike. Producije bilo koje kontekstno ovisne gramatike $G=(V, T, P, S)$ moguće je preuređiti tako da zadovolje postavljeni zahtjev. Konstruira se gramatika $G'=(V', T, P', S)$ na sljedeći način. Za sve završne znakove $a \in T$ definiraju se novi nezavršni znakovi $A_a \in V'$. U skupu

nezavršnih znakova V' su svi nezavršni znakovi iz skupa V i novi nezavršni znakovi A_a . U skup produkcija P' stave se produkcije $A_a \rightarrow a$ i produkcije $\alpha' \rightarrow \beta'$. Producije $\alpha' \rightarrow \beta'$ grade se na temelju produkcija $\alpha \rightarrow \beta$ iz skupa P tako da se u nizovima α i β završni znakovi a zamjene nezavršnim znakovima A_a .

Sastoje li se lijeve strane produkcija isključivo od nezavršnih znakova i ako je presjek skupova nezavršnih znakova V_1 i V_2 jednak praznom skupu, nije moguće da se spajanjem znakova bilo koja dva

* *

međuniza γ i δ za koje vrijedi $S_1 \xrightarrow{G_1} \gamma$ i $S_2 \xrightarrow{G_2} \delta$ dobije desna strana jedne od produkcija iz skupova P_1 i P_2 .

Kontekstno ovisni jezici zatvoreni su s obzirom na Kleeneov operator L^+ .

Neka gramatika $G=(V, T, P, S)$ generira jezik $L(G)$ i neka produkcije gramatike imaju na lijevim stranama isključivo nezavršne znakove. Za gradnju gramatike $G_5=(V_5, T_5, P_5, S_5)$ koja generira jezik $L(G_5)=L(G)^+$ potrebno je promijeniti postupak opisan u odjeljku 3.3.1. Budući da se generiraju jedan do drugog međunizovi primjenom iste gramatike, potrebno je izgraditi dodatnu gramatiku G' . Gramatika $G'=(V', T, P', S')$ gradi se na temelju gramatike G tako da se nezavršni znakovi A gramatike G zamjene novim nezavršnim znakovima A' . Zamjena nezavršnih znakova mora osigurati da je $V \cap V' = \emptyset$. Gramatika $G_5=(V_5, T_5, P_5, S_5)$ konstruira se na sljedeći način:

- 1) $V_5 = V \cup V' \cup \{S_5, S_5'\}$, gdje je $S_5', S_5 \notin V$ i $S_5', S_5 \notin V'$.
- 2) $T_5 = T$.
- 3) U skup produkcija $P_5 = P \cup P'$ dodaju se produkcije:

$$\begin{aligned} S_5 &\rightarrow S S_5' \mid S \\ S_5' &\rightarrow S' S_5 \mid S'. \end{aligned}$$

5.3.2 Presjek i komplement

Presjek dvaju kontekstno ovisnih jezika jest kontekstno ovisni jezik.

Neka LOA M_1 prihvata jezik $L(M_1)$, a LOA M_2 neka prihvata jezik $L(M_2)$. LOA M_3 koji prihvata jezik $L(M_3)=L(M_1) \cap L(M_2)$ ima dva traga ulazne trake. U oba traga zapisuje se isti niz završnih znakova w , odnosno složeni znakovi trake imaju dvije komponente $[a, a]$, gdje je a završni znak gramatike. LOA M_3 simulira rad LOA M_1 na gornjem tragu ulazne trake, dok se rad LOA M_2 simulira na donjem tragu ulazne trake.

Stane li LOA M_1 i ne prihvati li niz w na gornjem tragu, LOA M_3 također stane i ne prihvati niz. Ako LOA M_1 ne stane nikada, onda ni LOA M_3 ne stane nikada. Ako LOA M_1 stane i prihvati niz w na gornjem tragu, onda LOA M_3 pokrene simulaciju rada LOA M_2 na donjem tragu ulazne trake.

Stane li LOA M_2 i ne prihvati li niz w na donjem tragu, LOA M_3 također stane i ne prihvati niz. Ako LOA M_2 ne stane nikada, onda ni LOA M_3 ne stane nikada. Stane li LOA M_2 i prihvati li niz w na donjem tragu, proces simulacije se zaustavlja i LOA M_3 prihvati niz. Budući da se niz prihvata ako i samo ako tijekom simulacije LOA M_1 i LOA M_2 prihvate niz, LOA M_3 prihvata jezik $L(M_1) \cap L(M_2)$.

Komplement determinističkog kontekstno ovisnog jezika jest deterministički kontekstno ovisni jezik.

Neka DLOA M prihvata deterministički kontekstno ovisni jezik $L(M)$. Za bilo koji DLOA moguće je izgraditi istovjetni DLOA M' koji uvijek stane za svaki ulazni niz. Budući da DLOA koristi samo ćelije trake na kojima je zapisan ulazni niz w , moguće je izračunati maksimalni broj različitih konfiguracija DLOA na sljedeći način:

$$s(n+2)t^n$$

gdje je s kardinalni broj skupa stanja Q , n je duljina niza w i t je kardinalni broj skupa znakova trake Γ . Izraz $(n+2)$ računa broj različitih položaja glave za čitanje i pisanje: n položaja za n znakova niza w i dva položaja za graničnike $\$$ i $\$$. Izraz t^n računa broj različitih sadržaja trake na kojoj je zapisan niz w .

Budući da je automat deterministički, iz konfiguracije K_x DLOA uvijek prelazi u istu konfiguraciju K_y . Pomakne li DLOA glavu za čitanje i pisanje veći broj puta od maksimalnog broja konfiguracija $s(n+2)t^n$, sigurno dolazi do ponavljanja konfiguracija. Nije li niz prihvatile niti jedna prethodna konfiguracija, ponavljanjem istih konfiguracija niz se nikad neće prihvati. Zato se nakon $s(n+2)t^n$ pomaka glave rad DLOA zaustavlja i niz se ne prihvata. Napomena: isti zaključak nije moguće primijeniti na nedeterministički LOA (NLOA). NLOA iz konfiguracije K_x prelazi u skup konfiguracija $\{K_{y1}, K_{y2}, K_{y3}, \dots\}$. Pomakne li NLOA glavu za čitanje i pisanje veći broj puta od maksimalnog broja konfiguracija $s(n+2)t^n$, to ne znači da su iscrpljene sve moguće kombinacije konfiguracija za zadani niz w . Da bi se osigurao prolaz kroz sve moguće kombinacije konfiguracija, potrebno je koristiti algoritam simulacije nedeterminističkog TS koji je opisan u odjeljku 4.1.3. Međutim, simulacija nedeterminističkog LOA zahtijeva u općem slučaju veći broj ćelija od broja znakova u nizu w . Zato se svojstvo komplementa ograničava samo na DLOA i na determinističke kontekstno ovisne jezike.

DLOA M' koji je istovjetan DLOA M i koji uvijek stane za svaki ulazni niz konstruira se na sljedeći način. Za potrebe računanja izraza $s(n+2)t^n$ ugrađuju se tri vrste brojila: brojilo koje broji do n , brojilo koje broji do s i brojilo koje broji do t . Budući da su brojevi s i t konačni i ne ovise o duljini niza, brojila koja broje do s i t ostvare se kao dodatne komponente stanja. Budući da je n duljina niza, brojilo koje broji do n ostvari se dodatnim tragom ulazne trake. Tijekom simulacije DLOA M' broj pomake glave DLOA M . Budući da je $s(n+2)t^n$ konačni broj za bilo koji ulazni niz w , osigurano je da DLOA M' uvijek stane za svaki ulazni niz:

- 1) Stane li tijekom simulacije DLOA M i prihvati niz, DLOA M' stane i prihvati niz.
- 2) Stane li tijekom simulacije DLOA M i ne prihvati niz, DLOA M' stane i ne prihvati niz.
- 3) Odbroji li tijekom simulacije DLOA M' više od $s(n+2)t^n$ pomaka glave DLOA M , DLOA M' stane i ne prihvati niz.

Zamjenom odluka o prihvaćanju i neprihvaćanju niza u prethodnim točkama (1) do (3), gradi se DLOA M_1 koji prihvata jezik $L(M_1)$ koji je komplement jezika $L(M')$, što dokazuje da komplement bilo kojeg determinističkog kontekstno ovisnog jezika jest deterministički kontekstno ovisni jezik.

Za razliku od konačnih automata i Turingovih strojeva za koje je pokazana istovjetnost determinističkih i nedeterminističkih inačica automata i za razliku od potisnog automata za koji je pokazana neistovjetnost determinističke i nedeterminističke inačice, pitanje istovjetnosti DLOA i LOA još uvijek je neriješeno. Budući da se za dokaz zatvorenosti determinističkih kontekstno ovisnih jezika s obzirom na operaciju komplementa korisiti DLOA, i budući da je neriješeno pitanje da li su svi kontekstno ovisni jezici ujedno i deterministički, nije moguće odgovoriti na pitanje da li su svi kontekstno ovisni jezici zatvoreni s obzirom na operaciju komplementa.

5.3.3 Odlučivost kontekstno ovisnih jezika

Bilo koji kontekstno ovisni jezik jest rekurzivni jezik.

TS koji uvijek stane za svaki ulazni niz izvodi sljedeći algoritam prihvaćanja kontekstno ovisnog jezika. Neka je kontekstno ovisni jezik zadan kontekstno ovisnom gramatikom $G=(V, T, P, S)$ i neka je w ulazni niz. Algoritam prihvaćanja jezika zasniva se na gradnji usmjerenog grafa. Čvorovi grafa su nizovi završnih i nezavršnih znakova gramatike α koji su jednakog dugački ili kraći od ulaznog niza w , odnosno $|\alpha| \leq |w|$. Vrijedi li relacija $\alpha \Rightarrow \beta$, čvorovi grafa α i β povezuju se usmjerrenom granom. Put usmjerenog grafa predstavlja postupak generiranja niza primjenom produkcija gramatike G . Put od čvora S do čvora w postoji ako i samo ako gramatika G generira niz w .

Budući da ima konačni broj nizova α koji su jednakog dugački ili kraći od ulaznog niza w , TS u konačnom broju koraka izgradi usmjereni graf i pretraži sve puteve usmjerenog grafa. TS se uvijek zaustavi za svaki ulazni niz i odluči o prihvaćanju niza, što dokazuje da su kontekstno ovisni jezici rekurzivni, odnosno odlučivi.

Za pronalaženje puta od čvora S do čvora w koriste se različiti algoritmi. U primjeru 5.2 opisan je jedan od algoritama pronalaženja puta u zadanom grafu.

Primjer 5.2. Iterativni algoritam pronalaženja puta u zadanom grafu gradi listu K . Neka se vrijednost liste nakon i -tog koraka iteracije označi oznakom K_i . U listi K_i su međunizovi α , gdje je $|\alpha| \leq |w|$, koje gramatika $G=(V, T, P, S)$ generira primjenom najviše i produkcija. Koristi li se nazivlje usmjerenog grafa, u listi K_i su međunizovi α za koje vrijedi da je duljina puta od početnog nezavršnog znaka S do međuniza α jednaka ili kraća od i . Iterativni algoritam izvodi se na sljedeći način:

Početni korak iteracije:

Upiši u listu K početni nezavršni znak gramatike S .

i -ti korak iteracije:

Neka je K_{i-1} lista nakon koraka $i-1$. Lista u koraku i gradi se na sljedeći način:

$$K_i = K_{i-1} \cup \{\beta \mid \alpha \Rightarrow \beta, \alpha \in K_{i-1} \text{ i } |\beta| \leq |w|\}$$

Neka se međuniz α nalazi u listi K_{i-1} . Ako je iz međuniza α moguće generirati međuniz β primjenom samo jedne produkcije gramatike i ako je niz β kraći ili jednak dugačak kao ulazni niz w , onda se u listu K_i doda međuniz β .

Budući da su desne strane svih produkcija jednakog dugačke ili duže od lijevih strana, nakon konačnog broja koraka listu K nije moguće proširiti novim međunizom β i iterativni algoritam se sigurno zaustavlja. Ako je u listi K niz w , zadani niz w je u jeziku $L(G)$. Nije li u listi, niz w nije u jeziku $L(G)$.

Neka se razmatra kontekstno ovisna gramatika G zadana u primjeru 5.1. Tijekom iterativnog postupka lista K poprima sljedeće vrijednosti za ulazni niz $w=a$, gdje je $|w|=1$:

$$i=0: \quad K_0 = \{S\}$$

$$i=1: \quad K_1 = \{S, [ACaB]\}$$

$$i=2: \quad K_2 = \{S, [ACaB]\}$$

Budući da se u drugom koraku iteracije lista K nije proširila, a u listi se ne nalazi niz a , prekida se daljnje izvođenje algoritma. Zadani niz a nije u jeziku $L(G)$.

Za ulazni niz $w=aa$, gdje je $|w|=2$, lista K poprima sljedeće vrijednosti tijekom iterativnog postupka:

$$\begin{aligned}
 i = 0: \quad & K_0 = \{S\} \\
 i = 1: \quad & K_1 = \{S, [ACaB]\} \\
 i = 2: \quad & K_2 = \{S, [ACaB], [Aa][aCB]\} \\
 i = 3: \quad & K_3 = \{S, [ACaB], [Aa][aCB], [Aa][aDB], [Aa][aE]\} \\
 i = 4: \quad & K_4 = \{S, [ACaB], [Aa][aCB], [Aa][aDB], [Aa][aE], \\
 & \quad [Aa][DaB], [Aa][Ea]\} \\
 i = 5: \quad & K_5 = \{S, [ACaB], [Aa][aCB], [Aa][aDB], [Aa][aE], \\
 & \quad [Aa][DaB], [Aa][Ea], [ADa][aB], [AEa]a, \\
 & \quad [ACa][aB]\} \\
 i = 6: \quad & K_6 = \{S, [ACaB], [Aa][aCB], [Aa][aDB], [Aa][aE], \\
 & \quad [Aa][DaB], [Aa][Ea], [ADa][aB], [AEa]a, \\
 & \quad [ACa][aB], aa\}
 \end{aligned}$$

Budući da se u listi K_6 nalazi niz aa , prekida se daljnje izvođenje algoritma. Zadani niz aa jest u jeziku $L(G)$.

5.3.4 Primjer rekurzivnog jezika koji nije kontekstno ovisni jezik

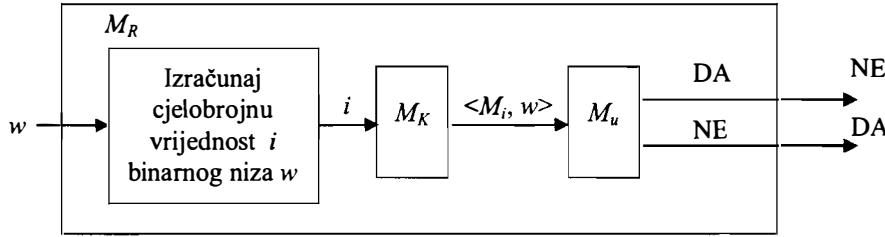
Kontekstno ovisni jezici su pravi podskup rekurzivnih jezika.

Dokaz da su kontekstno ovisni jezici pravi podskup rekurzivnih jezika zasniva se na sljedećem svojstvu:

- a) Zadan je skup jezika $K=\{L_{x1}, L_{x2}, L_{x3}, \dots\}$. Neka za bilo koji jezik L_{xi} u zadanim skupu K postoji barem jedan TS M_{xi} koji prihvata jezik L_{xi} i koji uvijek stane za svaki ulazni niz. Svi TS M_{xi} kodiraju se i poredaju određenim redoslijedom u niz M_1, M_2, M_3, \dots . Postoji li TS M_K koji redom ispisuje na izlaznu traku kodove svih TS iz skupa K , skup jezika K je pravi podskup skupa rekurzivnih jezika RJ, odnosno $K \subset RJ$.

Moguće je izgraditi rekurzivni jezik L_R koji ne prihvata niti jedan TS M_{xi} , odnosno L_R nije u skupu K . Neka je jezik L_R podskup skupa $(0+1)^*$ i neka se niz w promatra kao binarni broj. Jezik L_R definira se na sljedeći način: niz w je u jeziku L_R ako i samo ako TS M_i ne prihvata niz w , gdje je i cijelobrojna vrijednost binarnog broja w . Jezik L_R je rekurzivni jezik koji ne prihvata niti jedan TS M_i u niz M_1, M_2, M_3, \dots .

Jezik L_R prihvata TS M_R koji uvijek stane za svaki ulazni niz. Slika 5.2 prikazuje konstrukciju TS M_R . Neka je na ulazu TS M_R binarni niz w koji ima cijelobrojnu vrijednost i . TS M_R izračuna cijelobrojnu vrijednost i binarnog niza w i pokrene simulaciju rada TS M_K . Tijekom simulacije TS M_K generira redom kodove TS M_1, M_2, M_3, \dots sve dok ne generira kod TS M_i . Nakon što TS M_K generira kod TS M_i , TS M_R koristi univerzalni TS M_u za simulaciju rada TS M_i za ulazni niz w . Ako tijekom simulacije TS M_i stane i prihvati niz, TS M_R stane i ne prihvati niz. Stane li tijekom simulacije TS M_i i ne prihvati niz, TS M_R stane i prihvati niz. Budući da TS M_K u konačnom broju koraka generira kod TS M_i i budući da TS M_i uvijek stane za svaki ulazni niz, zaključuje se da TS M_u i TS M_R uvijek stanu za svaki ulazni niz. Budući da TS M_R uvijek stane za svaki ulazni niz, jezik L_R je rekurzivan.

Slika 5.2: Konstrukcija TS M_R za jezik L_R

Međutim, niti jedan TS u nizu M_1, M_2, \dots ne prihvaca jezik L_R . Na primjer, pretpostavi se da TS M_j prihvaca jezik L_R . Neka binarni niz x ima cjelobrojnu vrijednost j . Pretpostavka da TS M_j prihvaca jezik L_R dovodi do sljedećih suprotnosti. Ako je niz x u jeziku L_R , onda niz x nije u jeziku $L(M_j)$. Nije li niz x u jeziku L_R , onda je niz x u jeziku $L(M_j)$. Jezik L_R je rekurzivni jezik za koji vrijedi $L_R \neq L(M_j)$ za bilo koji TS M_j u nizu M_1, M_2, \dots . Budući da rekurzivni jezik L_R nije u skupu K , skup K je pravi podskup skupa rekurzivnih jezika.

Da se dokaže da su kontekstno ovisni jezici pravi podskup rekurzivnih jezika, dovoljno je pronaći niz TS M_1, M_2, \dots za koji vrijedi:

- 1) Svi TS M_i u nizu TS M_1, M_2, \dots uvijek stane za svaki ulazni niz.
- 2) Postoji TS M_{KOJ} koji generira redom kodove svih TS M_1, M_2, \dots .
- 3) Za bilo koji kontekstno ovisni jezik postoji barem jedan TS u nizu M_1, M_2, \dots koji ga prihvaca.

Neka je rad svih TS M_i u nizu M_1, M_2, \dots zasnovan je na algoritmu koji je opisan u odjeljku 5.3.3. Budući da opisani algoritam uvijek stane za svaki ulazni niz, bilo koji TS M_i uvijek stane za svaki ulazni niz.

Tijekom rada TS M_i koristi produkcije kontekstno ovisne gramatike koje su zapisane na jednoj od radnih traka. Budući da svi TS M_i koriste isti algoritam, a razlikuju se samo u zadanim produkcijama gramatike, kodiranje niza TS M_1, M_2, \dots poistovjećuje se s kodiranjem kontekstno ovisne gramatike.

Pretpostavi se da je gramatika zadana nad binarnom abecedom $\Sigma = \{0, 1\}$. Za sve znakove gramatike definiraju se binarni kodovi. U tablici 5.1 dati je primjer kodiranja elemenata gramatike binarnim znakovima 0 i 1.

Znak gramatike	Kod
završni znak 0	10
završni znak 1	100
,	1000
→	10000
{	100000
}	1000000
(10000000
)	100000000
nezavršni znak A_1	10^9
---	---
nezavršni znak A_i	10^{i+8}
---	---

Tablica 5.1: Kodiranje elemenata gramatike

TS M_{KOJ} koji ispisuje na izlaznu traku određenim redoslijedom kodove svih kontekstno ovisnih gramatika gradi se na sljedeći način. Najprije se na pomoćnu radnu traku kanonskim slijedom ispisuju nizovi $w \in (0+1)^*$, a zatim se provjerava da li ispisani niz w zadovoljava zadani način kodiranja gramatike i da li produkcije gramatike zadovoljavaju zahtjeve kontekstno ovisne gramatike. Ako je zadovoljen zadani način kodiranja i ako su zadovoljeni zahtjevi kontekstno ovisne gramatike, niz w je ispravno kodirana kontekstno ovisna gramatika i niz se ispisuje na izlaznu traku.

Budući da je jezik kontekstno ovisan ako i samo ako ga generira kontekstno ovisna gramatika i budući da TS M_{KOJ} ispisuje redom kodove svih kontekstno ovisnih gramatika na izlaznu traku, za bilo koji kontekstno ovisni jezik postoji barem jedan TS u nizu M_1, M_2, \dots , koji prihvata taj kontekstno ovisni jezik.

Na temelju postojanja niza TS M_1, M_2, \dots za koji su ispunjeni svi zahtjevi (1) do (3) i na temelju svojstva (a), zaključuje se da su kontekstno ovisni jezici pravi podskup rekursivnih jezika.

6 RAZREDBA JEZIKA, AUTOMATA I GRAMATIKA

Razredba jezika u klase zasniva se na strukturnoj složenosti jezika i složenosti prihvaćanja jezika. Strukturalna složenost jezika određuje se na temelju složenosti automata koji prihvaca jezik, dok složenost prihvaćanja jezika uzima u obzir vrijeme i prostor potreban da se prihvati jezik. Odjeljak 6.1 opisuje razredbu i hijerarhiju jezika temeljenu na strukturalnoj složenosti jezika, a odjeljak 6.2 opisuje razredbu i hijerarhiju jezika temeljenu na složenosti prihvaćanja jezika.

6.1 Strukturalna složenost jezika

Neka su A i B dvije različite klase jezika. Ako je klasa A pravi podskup klase B , onda za te dvije klase vrijedi:

- 1) Automat koji prihvaca jezike iz klase A ima jednostavniju strukturu od automata koji prihvaca jezike iz klase B .
- 2) Gramatika koja generira jezike iz klase A ima jednostavnije produkcije od gramatike koja generira jezike iz klase B .

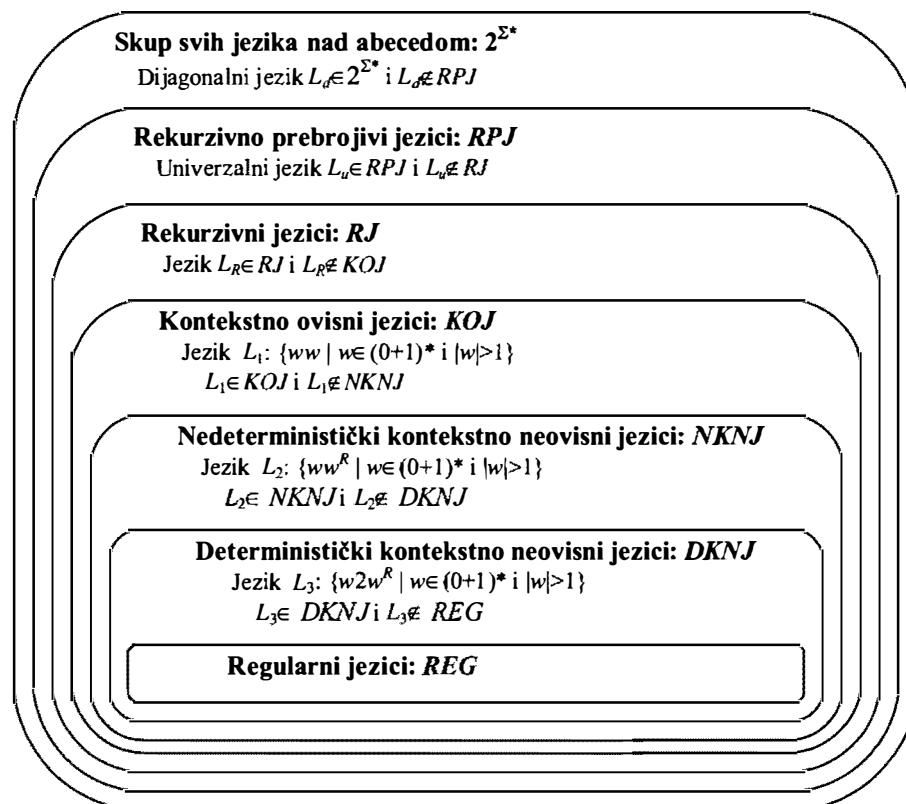
Kaže se da jezici iz klase A imaju jednostavniju strukturalnu složenost od jezika iz klase B . Na primjer, regularni jezici imaju jednostavniju strukturalnu složenost od kontekstno neovisnih jezika. Za te dvije klase jezika vrijedi: regularni jezici su pravi podskup kontekstno neovisnih jezika, konačni automat ima jednostavniju strukturu od PA i regularna gramatika ima jednostavnije produkcije od kontekstno neovisne gramatike.

Krajem 50-tih godina Noam Chomsky istraživao je moguće modele prirodnih jezika, te je definirao hijerarhiju jezika s obzirom na strukturalnu složenost jezika. Hijerarhija jezika s obzirom na strukturalnu složenost opisana je odjeljku 6.1.1, a u odjeljku 6.1.2 prikazana je odgovarajuća hijerarhija automata i gramatika.

6.1.1 Chomskyjeva hijerarhija jezika

Slika 6.1 prikazuje hijerarhiju klasa jezika nad zadanom abecedom Σ . Hijerarhija jezika zasnovana je na sljedećim relacijama:

- 1) Klasa regularnih jezika jest pravi podskup klase determinističkih kontekstno neovisnih jezika.
- 2) Klasa determinističkih kontekstno neovisnih jezika jest pravi podskup klase nedeterminističkih kontekstno neovisnih jezika.
- 3) Klasa nedeterminističkih kontekstno neovisnih jezika jest pravi podskup klase kontekstno ovisnih jezika.
- 4) Klasa kontekstno ovisnih jezika jest pravi podskup klase rekurzivnih jezika.
- 5) Klasa rekurzivnih jezika jest pravi podskup klase rekurzivno prebrojivih jezika.
- 6) Klasa rekurzivno prebrojivih jezika jest pravi podskup skupa svih jezika zadanih nad abecedom Σ .



Slika 6.1: Chomskyjeva hijerarhija jezika

Regularni jezici imaju najjednostavniju strukturu složenost: klasa regularnih jezika jest pravi podskup bilo koje druge klase jezika, konačni automat je najjednostavniji automat i regularna gramatika ima najjednostavnije produkcije.

Jezik L_3 zadan na slici 6.1 jest primjer determinističkog kontekstno neovisnog jezika koji nije regularan. Budući da je regularna gramatika ujedno i kontekstno neovisna gramatika i budući da kontekstno neovisni jezik L_3 nije regularan, klasa regularnih jezika jest pravi podskup klase determinističkih kontekstno neovisnih jezika.

Klasa determinističkih kontekstno neovisnih jezika jest pravi podskup klase nedeterminističkih kontekstno neovisnih jezika na temelju sljedećeg svojstva: za bilo koji deterministički potisni automat moguće je izgraditi istovjetni nedeterministički potisni automat; nedeterministički kontekstno neovisni jezik L_2 je primjer jezika koji nije deterministički.

Jezik L_1 jest primjer kontekstno ovisnog jezika koji nije kontekstno neovisan. Budući da je kontekstno neovisna gramatika ujedno i kontekstno ovisna, te budući da postoji kontekstno ovisni jezik L_1 koji nije kontekstno neovisan, klasa kontekstno neovisnih jezika jest pravi podskup klase kontekstno ovisnih jezika.

U odjelicima 5.3.3 i 5.3.4 pokazano je da klasa kontekstno ovisnih jezika jest pravi podskup klase rekurzivnih jezika. Jezik L_R opisan u odjeljku 5.3.4 je primjer rekurzivnog jezika koji nije kontekstno ovisan.

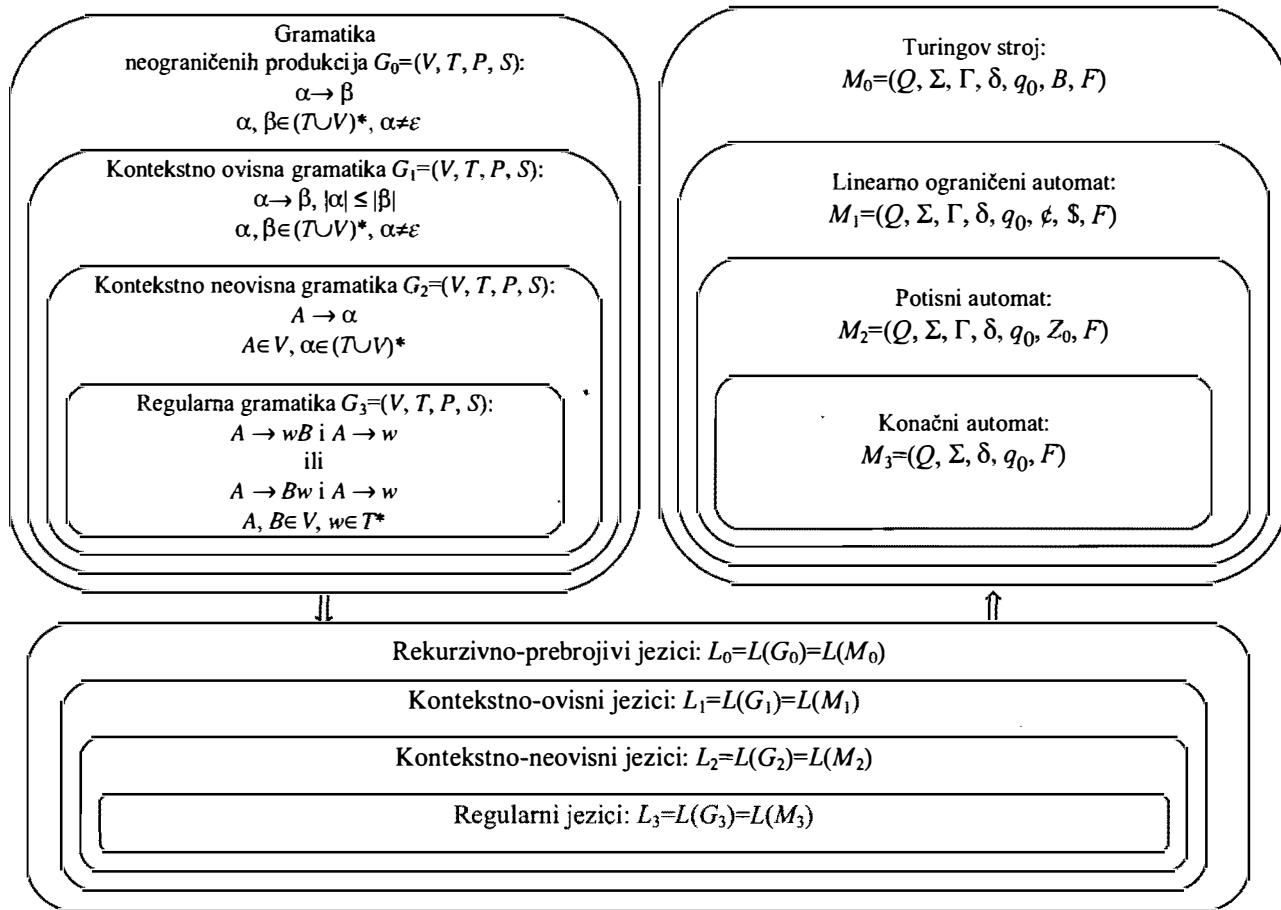
Budući da je univerzalni jezik L_u primjer rekurzivno prebrojivog jezika koji nije rekurzivan, klasa rekurzivnih jezika jest pravi podskup klase rekurzivno prebrojivih jezika.

U odjeljku 4.3.2 pokazano je da za dijagonalni jezik L_d nije moguće izgraditi Turingov stroj koji ga prihvaca, što potvrđuje da je klasa rekurzivno prebrojivih jezika pravi podskup skupa svih jezika zadanih nad abecedom Σ . Dok je strukturna složenost regularnih jezika najjednostavnija, klasa rekurzivno prebrojivih jezika ima veću strukturu složenost od ostalih klasa jezika: sve ostale klasa jezika su pravi podskup klase rekurzivno prebrojivih jezika. Turingov stroj jest najsloženiji model automata, a gramatika neograničenih produkcija ima najsloženije produkcije.

6.1.2 Hijerarhija gramatika i automata

Slika 6.2 prikazuje hijerarhiju osnovnih modela automata i gramatika. Hijerarhija automata i gramatika temelji se na prethodno opisanoj hijerarhiji jezika i prethodno pokazanim istovjetnostima:

- 1) Istovjetnost regularnih jezika, konačnih automata i regularne gramatike.
- 2) Istovjetnost kontekstno neovisnog jezika, kontekstno neovisne gramatike i potisnog automata.
- 3) Istovjetnost kontekstno ovisnog jezika, kontekstno ovisne gramatike i linearno ograničenog automata.
- 4) Istovjetnost rekurzivno prebrojivih jezika, gramatike neograničenih produkcija i Turingovog stroja.



Slika 6.2: Hijerarhija automata i gramatika

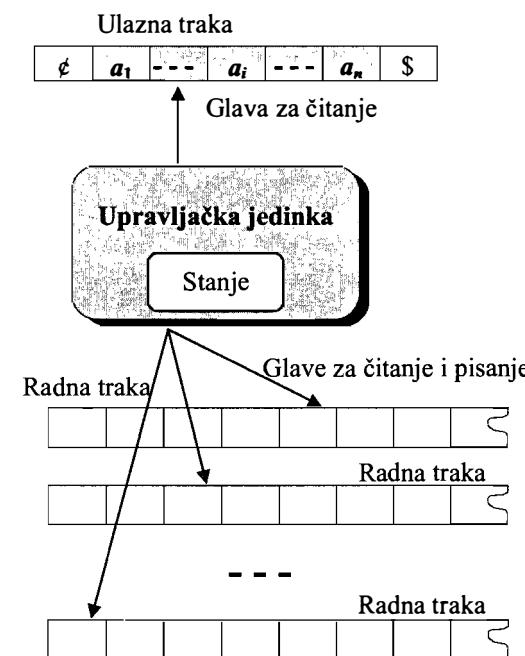
6.2 Složenost prihvatanja jezika

Složenost prihvatanja jezika uzima u obzir veličinu traka i vrijeme potrebno da automat prihvati jezik. Na temelju hijerarhije jezika, automata i gramatika koja je opisana u odjeljku 6.1 zaključuje se da Turingovi strojevi prihvataju najširu klasu jezika i sve ostale klase jezika su pravi podskup klase rekurzivno prebrojivih jezika. Budući da je u odjeljku 4.3.2 pokazano da su svi izračunljivi jezici u skupu rekurzivno prebrojivih jezika, Turingov stroj uzima se za osnovni automat za ocjenu složenosti prihvatanja jezika. Veličina traka određuje se na temelju broja ćelija koje tijekom rada koristi TS. Vrijeme se mjeri tako da se broje pomaci glave TS: jedan pomak glave TS troši jedinicu vremena. Budući da se procjena složenosti prihvatanja jezika temelji na veličini traka i vremenu, osnovne sastavnice složenosti su: *prostorna složenost* i *vremenska složenost*. Osim prostora i vremena koriste se i drugi parametri za ocjenu složenosti jezika. Na primjer, moguće je koristiti broj promjena smjera kretanja glave za čitanje i pisanje.

6.2.1 Definicija prostorne i vremenske složenosti prihvaćanja jezika

Osnovne definicije vremenske i prostorne složenosti prihvaćanja jezika zasnivaju se na determinističkom TS s višestrukim trakama.

Prostorna složenost prihvaćanja jezika



Slika 6.3: Model neizravnog TS koji se koristi za ocjenu prostorne složenosti računanja

Za ocjenu prostorne složenosti prihvaćanja jezika koristi se neizravan deterministički TS s k polubeskonačnih traka. Model TS prikazan je na slici 6.3. Uzlazna traka sadrži niz koji se ispituje. Neka je duljina uzlaznog niza n . Uzlaznu traku moguće je samo čitati. TS ima k radnih traka koje su beskonačne na desnu stranu i koje TS čita i piše. Prostorna složenost određuje se na temelju *samo jedne* radne trake, i to one radne trake na kojoj TS koristi najviše ćelija:

Koristi li TS M najviše $S(n)$ ćelija na jednoj od radnih traka tijekom prihvaćanja bilo kojeg niza duljine n , TS M jest prostorne složenosti $S(n)$. Jezik $L=L(M)$ koji prihvata TS M jest prostorne složenosti $S(n)$.

Ispravnost dane definicije temelji se na sljedećem svojstvu prostorne složenost koje je detaljno opisano u odjeljku 6.2.2: za bilo koji TS s k radnih traka moguće je izgraditi istovjetni TS koji koristi samo jednu traku i ima istu prostornu složenost.

Prostorna složenost uzima u obzir samo broj ćelija radnih traka, a ne broj ćelija uzlazne trake. Koristi li TS manje od n ćelija na radnim trakama, s povećavanjem duljine uzlaznog niza n moguće je da je porast funkcije $S(n)$ manji od linearног. Nadalje, prepostavlja se da TS koristi barem jednu ćeliju jedne od radnih traka, tj. da prostorna složenost jest $\max(1, \lceil S(n) \rceil)$, gdje je $\lceil S(n) \rceil$ cijelobrojna vrijednost funkcije $S(n)$ zaokružena na više.

Dozvoli li se da TS piše po uzlaznoj traci, prostorna složenost uzima u obzir i ćelije uzlazne trake. Budući da se zahtijeva da TS pročita sve znakove uzlaznog niza, u danom slučaju s povećavanjem duljine niza n porast funkcije $S(n)$ jest veći ili jednak linearном porastu.

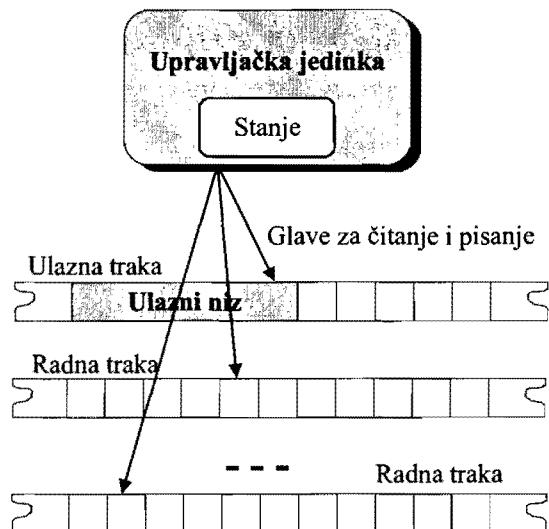
Vremenska složenost prihvaćanja jezika

Slika 6.4 prikazuje model determinističkog TS koji se koristi za ocjenu vremenske složenosti prihvaćanja jezika. TS ima k dvostrano beskonačnih traka. Jedna radna traka jest ulazna i na njoj je zapisan ulazni niz w duljine n . Sve trake, uključujući i ulaznu traku, moguće je čitati i po svim trakama moguće je pisati.

Vremenska složenost računa se na temelju broja pomaka glave za čitanje i pisanje:

Izvede li TS M najviše $T(n)$ pomaka glave tijekom prihvaćanja bilo kojeg niza duljine n , TS M jest vremenske složenosti $T(n)$. Jezik $L=L(M)$ koji prihvaca TS M jest vremenske složenosti $T(n)$.

Zahtijeva se da TS tijekom rada pročita svih n znakova niza w , što znači da funkcija $T(n)$ ima vrijednost najmanje $n+1$, odnosno $n+1 \leq T(n)$. Za potrebe čitanja znakova niza TS pomakne glavu n puta, a jedan dodatni pomak u desno na prazninu potreban je da se ustanovi da su pročitani svi znakovi niza. Budući da se čitaju svi znakovi niza, vremenska složenost jest $\max(n+1, \lceil T(n) \rceil)$, što se kraće zapisuje $T(n)$. Predloženi model ocjene vremenske složenosti ne uzima u obzir one TS koji se uvijek zaustave prije nego što pročitaju sve znakove niza.

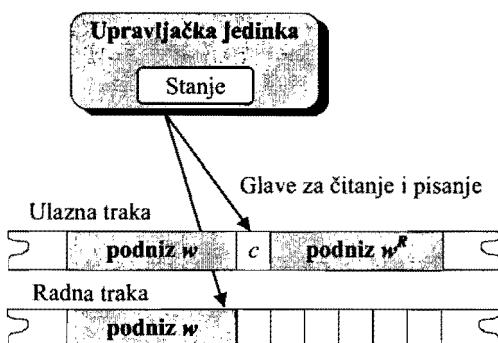


Slika 6.4: Model TS s višestrukim dvostranim beskonačnim trakama koji se koristi za ocjenu vremenske složenosti računanja

Primjer 6.1. Neka je zadan jezik:

$$L = \{ wcw^R \mid w \in (a+b)^*\}.$$

Duljina ulaznog niza wcw^R jest n . Budući da postoji TS M_1 vremenske složenosti $n+1$ koji prihvaca jezik $L(M_1)=L$, jezik L jest vremenske složenosti $n+1$. Rad TS M_1 vremenske složenosti $T(n)=n+1$ prikazan je na slici 6.5.



Slika 6.5: TS M_1 vremenske složenosti $T(n)=n+1$ koji prihvaca jezik L

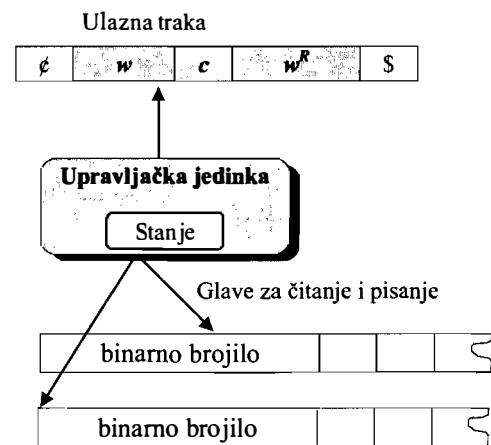
TS M_1 ispituje ulazni niz na sljedeći način. Započinje se s čitanjem podniza w lijevo od znaka c . Tijekom čitanja podniza w prepisuje se s ulazne trake na radnu traku. Slika 6.5 prikazuje TS M_1 u trenutku čitanja znaka c . Dio podniza w lijevo od znaka c prepisan je na radnu traku. Nakon što pročita znak c , glava ulazne trake nastavlja se micati u desno, a glava radne trake u lijevo. TS M_1 uspoređuje znakove koje čitaju glava ulazne trake i glava radne trake. Pročitaju li obje glave jednake znakove, nastavlja se s usporedbom sljedećeg para znakova. Nisu li pročitani znakovi jednaki, TS se zaustavlja i niz se ne prihvaca. Usporedba znakova nastavlja se sve dok obje glave istodobno ne pročitaju praznu ćeliju. Pročitaju li obje glave jednake podnizove znakova, niz se prihvaca.

Budući da je duljina ulaznog niza jednaka $n=|wcw^R|$, duljina podniza w jest $(n-1)/2$. Ne prihvaca li se niz, TS M_1 se zaustavlja prije nego što se pročitaju svi znakovi ulaznog niza. U slučaju neprihvaćanja niza broj pomaka glave TS M_1 jednak je ili manji od duljine niza n . Prihvaca li se niz, broj pomaka glave TS jest $n+1$: $(n-1)/2$ pomaka glave potrebno je za prijepis podniza w , jedan pomak glave potreban je za čitanje znaka c , $(n-1)/2$ pomaka glave potrebno je za usporedbu znakova podnizova w i w^R , te je jedan pomak glave potreban za čitanje praznih ćelija. Budući da je u slučaju prihvaćanja niza broj pomaka glave TS M_1 jednak $n+1$, a u slučaju neprihvaćanja niza broj pomaka glave TS M_1 jednak je ili manji od n , jezik L i TS M_1 su vremenske složenosti $T(n)=n+1$.

Prostorna složenost jezika L jest $\log_2 n$. Na slici 6.6 prikazan je TS M_2 prostorne složenosti $S(n)=\log_2 n$ koji prihvaca jezik L . TS M_2 je neizravan TS s jednom ulaznom trakom i s dvije radne trake. Radne trake imaju ulogu binarnog brojila. Niz wcw^R duljine n zapisan je na ulaznoj traci koja se samo čita.

TS M_2 ispituje ulazni niz u tri koraka. U prvom koraku provjeri se da li postoji jedan i samo jedan znak c u ulaznom nizu. U drugom koraku provjerava se da li ima jednak broj znakova lijevo i desno od znaka c . U trećem koraku uspoređuju se znakovi lijevo i desno od znaka c . Tijekom ispitivanja ulaznog niza binarna brojila određuju broj znakova u podnizu w i mjesto znaka u podnizu w .

Ulagna traka samo se čita i ne uzima se u obzir za ocjenu prostorne složenosti. Budući da je niz duljine n , pojedino binarno brojilo koristi najviše $\log_2 n$ ćelija. Prostorna složenost TS M_2 i jezika L određuje se na temelju radnih traka i jednaka je $S(n)=\log_2 n$. Budući da se ne uzima u obzir duljina ulazne trake, s povećanjem duljine niza n porast funkcije $S(n)$ jest manji od linearne.



Slika 6.6: TS M_2 prostorne složenosti $S(n)=\log_2 n$ koji prihvaca jezik L

6.2.2 Svojstva prostorne i vremenske složenosti prihvaćanja jezika

Broj traka nema utjecaja na prostornu složenost, ali ima zato utjecaja na vremensku složenost. Vremenska složenost povećava se sa smanjivanjem broja traka. Nadalje, konstantne faktore u obje funkcije složenosti moguće je zanemariti. Primjeri funkcija koje je potrebno uzeti u razmatranje su linearne funkcije, polinome funkcije, eksponencijalne funkcije, itd.

Broj traka i prostorna složenost

Ako TS M_1 s k radnih traka prostorne složenosti $S(n)$ prihvaca jezik $L(M_1)$, onda postoji TS M_2 s jednom radnom trakom koji je jednake prostorne složenosti $S(n)$ i koji prihvaca jezik $L(M_2)=L(M_1)$.

Neka TS M_1 koristi na jednoj od k radnih traka najviše $S(n)$ ćelija. Gradi se istovjetni TS M_2 samo s jednom trakom primjenom algoritma opisanog u odjeljku 4.1.3. TS M_2 koristi umjesto k traka jednu traku sa $2k$ tragova: k tragova TS M_2 koristi se za spremanje sadržaja k radnih traka TS M_1 i k tragova TS M_2 koristi

za oznake položaja k glava TS M_1 . Budući da izgrađeni TS M_2 ne koristi više od $S(n)$ čelija, prostorne složenosti TS M_2 jednaka je $S(n)$. Opisano svojstvo omogućava da se prostorna složenost određuje na temelju radne trake koja koristi najviše čelija.

Broj traka i vremenska složenost

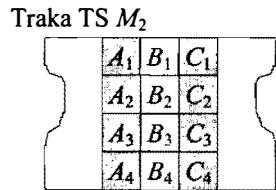
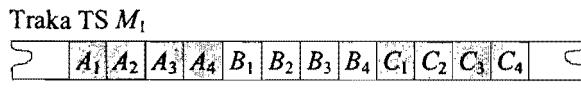
Neka se za izgradnju TS M_2 koji ima samo jednu traku i koji je istovjetan TS M_1 koji ima k traka koristi algoritam opisan u odjeljku 4.1.3. U istom odjeljku pokazano je da je povećanje broja pomaka glave TS M_2 u odnosu na broj pomaka glava TS M_1 moguće opisati kvadratnom funkcijom. Budući da je konstantne faktore moguće zanemariti i budući da je konstrukcija istovjetnog TS M_2 dana za opći slučaj bilo kojeg TS, vremenska složenosti TS M_2 nije veća od $T^2(n)$:

Ako TS M_1 s k traka vremenske složenosti $T(n)$ prihvaca jezik $L(M_1)$, onda postoji TS M_2 s jednom trakom koji je vremenske složenosti $T^2(n)$ i koji prihvaca jezik $L(M_2)=L(M_1)$.

Smanji li se broj traka sa k traka na dvije trake, a ne na jednu traku, moguće je izgraditi istovjetni TS vremenske složenosti $T(n) \log T(n)$, gdje je $T(n)$ vremenska složenost TS sa k traka.

Sažimanje prostora za konstantni faktor

Ako TS M_1 sa k radnih traka prostorne složenosti $S(n)$ prihvaca jezik $L(M_1)$, onda za bilo koju konstantu $c > 0$ postoji TS M_2 prostorne složenosti $cS(n)$ koji prihvaca jezik $L(M_2)=L(M_1)$.



Slika 6.7: Sažimanje prostora za konstantni faktor ($r=4$)

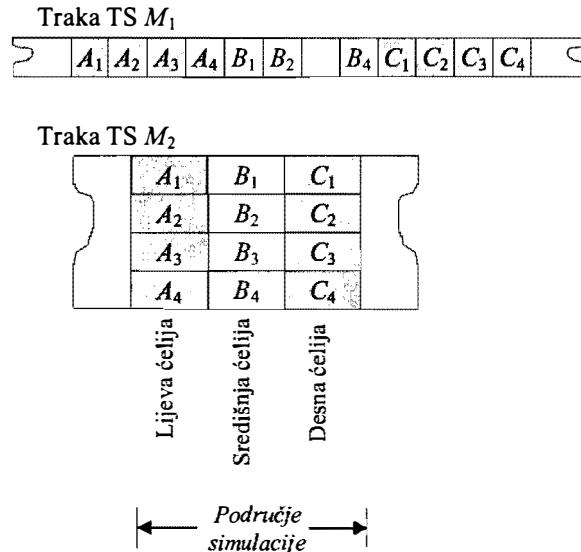
Slika 6.7 prikazuje konstrukciju TS M_2 . Znakovi u r susjednih čelija radnih traka TS M_1 kodiraju se jedinstvenim znakom TS M_2 . Jedan znak radne trake TS M_2 predstavlja sadržaj r čelija TS M_1 . Na slici 6.7 znakovi trake TS M_2 su složeni znakovi koji se sastoje od r komponenata. Svaka komponenta ima svoj zaseban trag. Tijekom simulacije TS M_2 u jednu komponentu složenog stanja spremi podatak koji trag radne trake se čita. Komponenta stanja poprima vrijednosti od 1 do r .

Želi li se sažeti broj čelija za faktor c , konstanta r odredi se tako da je $rc \geq 2$. Tijekom rada TS M_2 koristi najviše $\lceil S(n)/r \rceil$ čelija na bilo kojoj radnoj traci. Ako je $S(n) \geq r$ i ako je $rc \geq 2$, onda broj čelija $\lceil S(n)/r \rceil$ nije veći od $cS(n)$. Ako je $S(n) \leq r$, onda TS M_2 za kodiranje sadržaja bilo koje radne trake TS M_1 koristi samo jednu čeliju.

Ubrzanje za konstantni faktor

Ako TS M_1 s k traka vremenske složenosti $T(n)$ prihvaca jezik $L(M_1)$, onda za bilo koju konstantu $c > 0$ postoji TS M_2 s k traka vremenske složenosti $cT(n)$ koji prihvaca jezik $L(M_2)=L(M_1)$, gdje je $k > 1$ i $\inf_{n \rightarrow \infty} T(n)/n = \infty$. Funkcija $\inf_{n \rightarrow \infty} f(n)$ je najveća donja granica funkcije $f(n)$ kada n teži u beskonačnost.

TS M_2 ima barem dvije trake ($k>1$). Jedna traka jest ulazna, a ostale su radne. TS M_2 čita ulazni niz znak po znak s ulazne trake, kodira m susjednih znakova ulaznog niza u jedinstveni znak, te dobiveni jedinstveni znak zapiše na izabranu radnu traku. Proces prepisivanja i sažimanja se nastavlja sve dok se cijeli niz s ulazne trake ne prepiše u sažetom obliku na radnu traku. Nakon što se niz prepiše i sažme, zamijene se uloge traka: radna traka postaje ulazna traka, a ulazna traka postoje radna traka.



Slika 6.8: Područje simulacije ($m=4$)

Tijekom čitanja glava se miće jedanput u lijevo da se pročita sadržaj lijeve celijske trake, dva puta u desno da se pročita sadržaj desne celijske trake i na kraju glava se vraća na središnju celijsku traku da se pročita sadržaj središnje celijske trake.

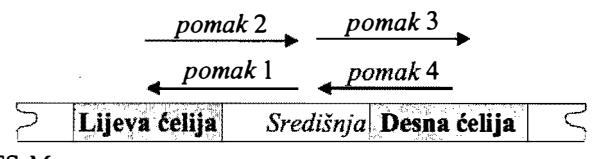
Nakon što završi s čitanjem, TS M_2 odrediti novi sadržaj celijske trake. Funkcije prijelaza TS M_2 grade se na temelju funkcija prijelaza TS M_1 tako da novi sadržaj celijske trake TS M_2 je jednak sažetom sadržaju celijske trake TS M_1 u trenutku kada TS M_1 napusti područje simulacije.

Uđe li TS M_1 u prihvatljivo stanje i stane prije nego što glava napusti područje simulacije, TS M_2 uđe u prihvatljivo stanje i stane. Ako TS M_1 stane i ne uđe u prihvatljivo stanje, onda TS M_2 stane i ne prihvati niz.

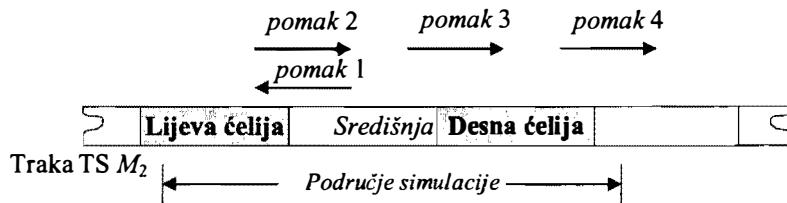
Simulacija rada TS M_1 izvodi se u ciklusima. U svakom ciklusu TS M_2 simulira rad TS M_1 na području koji obuhvaća $3m$ susjednih celijskih traka TS M_1 . Područje simulacije rada TS M_1 u jednom ciklusu prikazano je na slici 6.8, gdje su sadržaji četiri susjedne celijske trake TS M_1 sažeti u jednu celijsku traku TS M_2 , odnosno $m=4$.

U svakom ciklusu TS M_2 nastoji simulirati rad TS M_1 na području od $3m$ celijskih traka sa što manjim brojem pomaka glave. U jednom ciklusu simulira se rad TS M_1 od trenutka ulaska u područje simulacije pa sve do trenutka izlaska iz područja simulacije

Celijske trake TS M_2 označavaju se u odnosu na položaj glave. Glava TS M_2 postavljena je na središnju celijsku traku. Susjedne celijske trake središnjoj celijskoj su lijeva celijska traka i desna celijska traka. U jednoj celijskoj traci TS M_2 spremjenjen je sadržaj m susjednih celijskih traka TS M_1 . Rad TS M_1 na području od $3m$ celijskih traka moguće je simulirati s najviše 8 pomaka glave TS M_2 . Prva 4 pomaka glave potrebna su za čitanje sadržaja triju celijskih traka, a druga 4 pomaka glave potrebna su za promjenu sadržaja celijske trake i postavljanje glave na novu središnju celijsku traku. Pomaci glave TS M_2 tijekom čitanja opisani su na slici 6.9, a pomaci glave tijekom pisanja opisani su na slici 6.10.



Slika 6.9: Pomaci glave TS M_2 za vrijeme čitanja sadržaja celijske trake



Slika 6.10: Pomaci glave TS M_2 za vrijeme promjene sadržaja čelija i postavljanja glave na novu središnju čeliju

Napusti li glava TS M_1 područje simulacije, TS M_2 na odgovarajući način promijeni sve tri čelije i pomakne glavu na novu središnju čeliju. Nova središnja čelija određuje se na temelju položaja glave TS M_1 nakon što TS M_1 napusti dano područje simulacije. Potrebno je najviše 4 pomaka glave kojima TS M_2 mijenja sadržaj čelija i postavlja glavu na novu središnju čeliju. Pomaci glave TS M_2 prikazani su na slici 6.10.

Na temelju prethodnog opisanog postupka simulacije, na zadanom području simulacije TS M_2 s najviše 8 pomaka glave simulira više od m pomaka glave TS M_1 , odnosno da bi napustio područje simulacije, TS M_2 pomakne glavu najviše 8 puta, dok TS M_1 mora pomaknuti glavu više od m puta.

Pomakne li TS M_1 ukupno glavu $T(n)$ puta za niz duline n , TS M_2 pomakne glavu ne više od $8\lceil T(n)/m \rceil$ puta. Ako se broju $8\lceil T(n)/m \rceil$ doda n pomaka glave potrebnih za prepisivanje i sažimanje niza s ulazne trake na radnu traku i ako se doda $\lceil n/m \rceil$ pomaka glave potrebnih da se glava vrati na početak sažetok ulaznog niza, izračuna se najveći broj pomaka glave TS M_2 :

$$a) \quad n + \lceil n/m \rceil + 8\lceil T(n)/m \rceil$$

Želi li se ubrzati rad TS za faktor c , odnosno želi li se pokazati da je $n + \lceil n/m \rceil + 8\lceil T(n)/m \rceil < cT(n)$, konstanta m odredi se tako da je $cm \geq 16$. Budući da je $\lceil x \rceil \leq x + 1$, u (a) se umjesto izraza $\lceil x \rceil$ uvrsti izraz $x + 1$:

$$n + \lceil n/m \rceil + 8\lceil T(n)/m \rceil < n + (1 + n/m) + 8(1 + T(n)/m) = n + 9 + n/m + 8T(n)/m$$

Prepostavi li se da je $n \geq 9$, onda vrijedi $n + 9 \leq 2n$:

$$n + 9 + n/m + 8T(n)/m \leq 2n + n/m + 8T(n)/m$$

Budući da je $\inf_{n \rightarrow \infty} T(n)/n = \infty$, za bilo koju konstantu d moguće je naći n_d takav da za sve $n \geq n_d$ vrijedi da je $T(n)/n \geq d$, odnosno $n \leq T(n)/d$:

$$2n + n/m + 8T(n)/m \leq 2T(n)/d + T(n)/(md) + 8T(n)/m = (2/d + 1/md + 8/m)T(n) = ((2m+1)/dm + 8/m)T(n)$$

Vrijednost konstante d izabere se tako da je jednaka $d = (2m+1)/8$:

$$((2m+1)/dm + 8/m)T(n) = (8/m + 8/m)T(n) = (16/m)T(n)$$

Budući da je konstanta m izabrana tako da vrijedi $cm \geq 16$, umjesto m uvrsti se $16/c$:

$$(16/m)T(n) < cT(n)$$

što dokazuje da je broj pomaka glave TS M_2 $n + \lceil n/m \rceil + 8\lceil T(n)/m \rceil$ manji od $cT(n)$.

Prethodno je pokazano da je broj pomaka glave TS M_2 manji od $cT(n)$ za niz duljine $n \geq \max(9, n_d)$. Ostaje još da se pokaže kako ispitati nizove koji su kraći od $\max(9, n_d)$. Budući da nizova kraćih od $\max(9, n_d)$ imaju konačni broj, za te nizove definiraju se dve zasebne komponente složenog stanja. U jednu komponentu spremi se podatak koji niz znakova je pročitan, a u drugu komponentu spremi se podatak da li

se taj niz prihvaca. Obje komponente stanja mijenjaju se za svaki pročitani znak ulaznog niza. Nove vrijednosti komponenata određuju se na temelju vrijednosti pročitanog znaka i na temelju stare vrijednosti prve komponente stanja. Pročita li TS M_2 prazninu, a niz jest kraći od $\max(9, n_d)$, TS M_2 se zaustavi i ovisno o vrijednosti druge komponente stanja odluci se o prihvaćanju niza. Potreban broj pomaka glave jednak je $n+1$: n pomaka za čitanje ulaznog niza i jedan pomak da se ustanovi da je sljedeća ćelija prazna. Budući da broj pomaka glave za bilo koji niz duljine n nije veći od $\max(n+1, \lceil cT(n) \rceil)$, na temelju definicije dane u odjeljku 6.2.1 vremenska složenost TS M_2 jednaka je $cT(n)$.

6.2.3 Klase jezika s obzirom na složenost prihvaćanja jezika

Prethodne definicije složenosti prihvaćanja jezika zasnovane su na determinističkom TS. Jezik je *nedeterminističke* složenost ako ga prihvaca *nedeterministički* TS:

Nedeterministički TS jest prostorne složenosti $S(n)$ ako za niti jedan niz duljine n niti jedan mogući slijed prijelaza na niti jednoj od radnih traka ne koristi više od $S(n)$ ćelija. Jezik L jest nedeterminističke prostorne složenosti $S(n)$ ako i samo ako postoji nedeterministički TS M prostorne složenosti $S(n)$ koji prihvaca jezik $L(M)=L$.

Nedeterministički TS jest vremenske složenosti $T(n)$ ako za niti jedan niz duljine n niti jedan mogući slijed prijelaza ne pomakne glavu više od $T(n)$ puta. Jezik L jest nedeterminističke vremenske složenosti $T(n)$ ako i samo ako postoji nedeterministički TS M vremenske složenosti $T(n)$ koji prihvaca jezik $L(M)=L$.

Jezici su podijeljeni u četiri osnovne klase s obzirom na složenost prihvaćanja jezika: klasu $\text{DSPACE}(S(n))$ čine jezici determinističke prostorne složenosti $S(n)$, klasu $\text{NSPACE}(S(n))$ čine jezici nedeterminističke prostorne složenosti $S(n)$, $\text{DTIME}(T(n))$ čine jezici determinističke vremenske složenosti $T(n)$ i klasu $\text{NTIME}(T(n))$ čine jezici nedeterminističke vremenske složenosti $T(n)$.

Jezik L zadan u primjeru 6.1 je u klasi $\text{DTIME}(n+1)$ i klasi $\text{DSPACE}(\log_2 n)$. Jezik L je također u širim klasama: $\text{NTIME}(n)$, $\text{NTIME}(n^2)$, $\text{NSPACE}(\log_2 n)$, $\text{NSPACE}(\sqrt{n})$, itd.

Odnosi među osnovnim klasama jezika

Ako je jezik L u klasi $\text{DTIME}(f(n))$, onda je jezik L u klasi $\text{DSPACE}(f(n))$.

Budući da je jezik L u klasi $\text{DTIME}(f(n))$, postoji deterministički TS M vremenske složenosti $f(n)$. Primjenom $f(n)$ pomaka glave TS M ne može obići više od $f(n)+1$ ćelija. Sažimanjem sadržaja dviju ćelija u jednu ćeliju izgradi se TS koji ne koristi više od $\lceil (f(n)+1)/2 \rceil$ ćelija, odnosno broj ćelija koje se koriste jest manji od $f(n)$. Budući da jezik L prihvaca deterministički TS koji ne koristi više od $f(n)$ ćelija, jezik L je u klasi $\text{DSPACE}(f(n))$.

Ako je jezik L u klasi $\text{DSPACE}(f(n))$ i ako za funkciju $f(n)$ vrijedi $f(n) \geq \log_2 n$, onda je jezik L u klasi $\text{DTIME}(c^{f(n)})$. Vrijednost konstante c ovisi o jeziku L .

Budući da je jezik L u klasi $\text{DSPACE}(f(n))$, postoji neizravni deterministički TS M_1 prostorne složenosti $f(n)$. TS M_1 ima jednu ulaznu traku i jednu radnu traku. Na temelju činjenice da TS M_1 ne korisiti više od $f(n)$ ćelija, moguće je izračunati maksimalni broj različitih konfiguracija TS M_1 na sljedeći način:

$$s(n+2)f(n)t^{f(n)},$$

gdje je s kardinalni broj skupa stanja Q , n je duljina niza w , t je kardinalni broj skupa znakova trake Γ i $f(n)$ je maksimalni broj ćelija koje koristi TS M_1 . Izraz $(n+2)$ računa broj različitih položaja glave za čitanje na ulaznoj traci: n položaja za n znakova niza w i dva položaja za graničnike ϵ i $\$$. Funkcija $f(n)$ određuje broj različitih položaja glave na radnoj traci. Izraz $t^{f(n)}$ računa broj različitih sadržaja radne trake, dok se sadržaj ulazne trake ne mijenja i nema utjecaja na ukupni broj konfiguracija. Budući da je $f(n) \geq \log_2 n$, moguće je naći konstantu c takvu da za sve $n \geq 1$ vrijedi da je $c^{f(n)} \geq s(n+2)f(n)t^{f(n)}$.

Gradi se TS M_2 s više traka. Jedna traka koristi se za brojilo koje broji do $c^{f(n)}$. Ostale trake koriste se za simulaciju rada TS M_1 . Budući da je TS M_1 deterministički, nakon što TS M_1 pomakne glavu više od $c^{f(n)}$ puta, dolazi do ponavljanja konfiguracija. Nije li niz prihvatile niti jedna prethodna konfiguracija, ponavljanjem istih konfiguracija niz se nikad neće prihvati. Zato se nakon $c^{f(n)}$ pomaka glave rad TS M_1 zaustavlja i niz se ne prihvata.

Ako se TS M_1 zaustavi prije nego što pomakne glavu $c^{f(n)}$ puta i ako TS M_1 prihvati niz, onda se TS M_2 zaustavi i prihvati niz. Zaustavi li se TS M_1 prije nego što pomakne glavu $c^{f(n)}$ puta i ne prihvati li niz, TS M_2 se zaustavi i ne prihvati niz. Budući da jezik L prihvaća deterministički TS koji ne pomakne glavu više od $c^{f(n)}$ puta, jezik L je u klasi DTIME($c^{f(n)}$).

Ako je jezik L u klasi NTIME($f(n)$) i ako je funkcija $f(n) \geq \log_2 n$, onda je jezik L u klasi DTIME($c^{f(n)}$). Vrijednost konstante c ovisi o jeziku L .

Budući da je jezik L u klasi NTIME($f(n)$), postoji nedeterministički TS M_1 vremenske složenosti $f(n)$. TS M_1 ima k radnih traka. Budući da TS M_1 primjenom $f(n)$ pomaka glave ne može obići više od $f(n)+1$ ćeliju, maksimalni broj konfiguracija jednak je:

$$s(f(n)+1)^k t^{k f(n)},$$

gdje je s kardinalni broj skupa stanja Q , n je duljina niza w , t je kardinalni broj skupa znakova trake Γ , k je broj radnih traka i $f(n)$ je maksimalni broj pomaka glave TS M_1 . Izraz $f(n)+1$ računa broj različitih položaja glave na jednoj radnoj traci, a izraz $(f(n)+1)^k$ računa broj različitih položaja glave za svih k radnih traka. Izraz $t^{k f(n)}$ računa broj različitih sadržaja svih k radnih traka.

Konstruira se deterministički TS M_2 koji tijekom simulacije rada nedeterminističkog TS M_1 gradi listu svih konfiguracija TS M_1 koje su dostupne iz početne konfiguracije. Doda li se u listu konfiguracija koja prihvata niz, TS M_2 se zaustavi i prihvati niz. Ako se izgradi cijela lista i niti jedna konfiguracija ne prihvati niz, TS M_2 se zaustavi i ne prihvati niz. Za spremanje jedne konfiguracije potreban je sljedeći broj ćelija:

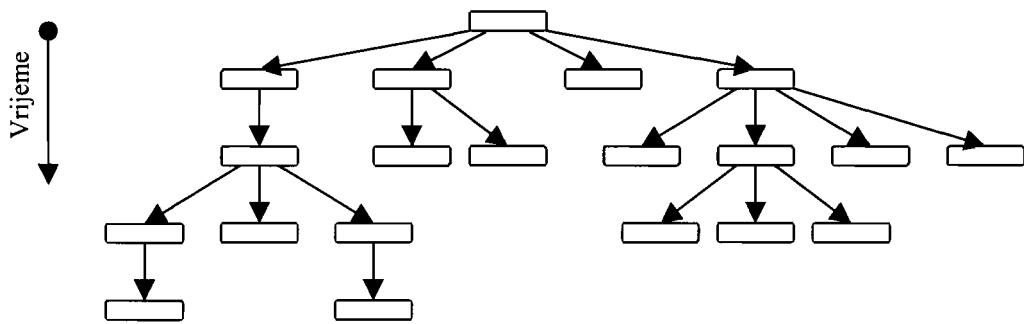
$$k(f(n)+1) + 1.$$

Za spremanje sadržaja jedne trake potrebno je najviše $f(n)+1$ ćelija, dok je za spremanje sadržaja k traka potrebno $k(f(n)+1)$ ćelija. U jednu ćeliju spremi se stanje TS M_1 .

Množenjem maksimalanog broja konfiguracija, koji je jedank $s(f(n)+1)^k t^{k f(n)}$, s brojem ćelija potrebnih za spremanje jedne konfiguracije $k(f(n)+1)+1$, izračuna se maksimalni broj ćelija potrebnih za spremanje liste konfiguracija. Neka je maksimalni broj ćelija potrebnih za spremanje liste konfiguracija jednak B . Broj pomaka glave TS M_2 potrebnih za gradnju liste konfiguracija procjeni se na sljedeći način. Prepostavi se najlošiji slučaj za dodavanje nove konfiguracije u listu: glava TS M_2 miče se s kraja liste na početak liste, čita se jedan znak konfiguracije s početka liste, pročitani znak spremi se u zasebnu komponentu složenog stanja TS M_2 , znak se prenosi s početka liste na kraj liste i na kraju se znak spremi u komponenti stanja zapiše na kraj liste. Nakon što se prenesu svi znakovi konfiguracije s početka liste na kraj liste, TS M_2 simulira prijelaz TS M_1 . Prijelaz TS M_1 simulira se tako da se na odgovarajući način promijeni prethodno zapisana konfiguracija na kraju liste. Budući da je za sve znakove konfiguracije potreban pomak glave od kraja liste do početka liste i ponovno na kraj liste, u odjeljku 4.1.3 pokazano je da

je broj pomaka glave kvadratna funkcija duljine puta između dva krajnja položaja glave. Budući da je za spremanje liste potrebno B celija, za gradnju liste nije potrebno više od B^2 pomaka glave. Budući da je $f(n) \geq \log_2 n$, moguće je naći konstantu c takvu da sve $n \geq 1$ vrijedi da je $c^{f(n)} \geq B^2$. Budući da jezik L prihvata deterministički TS koji ne pomakne glavu više od $c^{f(n)}$ puta, jezik L jest u klasi DTIME($c^{f(n)}$).

Odnos vremenske složenosti nedeterminističkog i determinističkog TS moguće je zorno opisati stablom. Slika 6.11 prikazuje primjer stabla vremenskog modela rada nedeterminističkog TS. Čvorovi grafa su konfiguracije TS. Korijen stabla jest početna konfiguracija. Grane stabla određuju prijelaze u skup konfiguracija koje su definirane nedeterminističkom funkcijom prijelaza TS. Vremenska složenost nedeterminističkog TS proporcionalna je dubini stabla. Vremenska složenost determinističkog TS proporcionalna je vremenu slijednog obilaženja stabla u cilju pronalaženja barem jedne konfiguracije u listovima stabla koja prihvata zadani niz, što za opći slučaj zahtijeva eksponencijalno vrijeme u odnosu na vrijeme nedeterminističkog TS.



Slika 6.11: Stablo vremenskog modela rada nedeterminističkog TS

Neka je TS M prostorne složenosti $S(n)$. Funkcija $S(n)$ jest *prostorno izgradiva* ako za bilo koji n postoji barem jedan niz duljine n za koji TS M koristi svih $S(n)$ celija. Sljedeće funkcije su prostorno izgradive: $\log n$, n^k , 2^n i $n!$. Ako su $S_1(n)$ i $S_2(n)$ prostorno izgradive funkcije, onda su prostorno izgradive i funkcije: $S_1(n)S_2(n)$, $2^{S_1(n)}$ i $S_1(n)^{S_2(n)}$. Funkcija $S(n)$ je *potpuno prostorno izgradiva* ako za bilo koji niz duljine n TS M koristi svih $S(n)$ celija. Za razliku od TS s prostorno izgradivom funkcijom složenosti, TS s potpuno prostorno izgradivom funkcijom složenosti koristi svih $S(n)$ celija za sve nizove duljine n . Na sličan način definiraju se *vremenski izgradive* i *potpuno vremenski izgradive* funkcije.

Ako je jezik L u klasi NSPACE($f(n)$) i ako je funkcija $f(n) \geq \log_2 n$ potpuno prostorno izgradiva, onda je jezik L u klasi DSPACE($f^2(n)$).

Budući da je jezik L u klasi NSPACE($f(n)$), postoji neizravni nedeterministički TS M_1 prostorne složenosti $f(n)$. Maksimalni broj konfiguracija TS M_1 moguće je izračunati na sljedeći način:

$$s(n+2)f(n)t^{f(n)} \leq c^{f(n)}.$$

gdje je s kardinalni broj skupa stanja Q , n je duljina niza w , t je kardinalni broj skupa znakova trake Γ i $f(n)$ je maksimalni broj celija koje koristi TS M_1 . Izraz $(n+2)$ računa broj različitih položaja glave za čitanje na ulaznoj traci: n položaja za n znakova niza w i dva položaja za graničnike ϵ i $\$$. Funkcija $f(n)$ određuje broj različitih položaja glave na radnoj traci. Izraz $t^{f(n)}$ računa broj različitih sadržaja radne trake, dok se sadržaj ulazne trake ne mijenja i nema utjecaja na ukupni broj konfiguracija.

Neka $I_1 \xrightarrow{(i)} I_2$ označava da TS M_1 iz konfiguracije I_1 prelazi u konfiguraciju I_2 primjenom ne više od 2^i prijelaza. Relacija $I_1 \xrightarrow{(i)} I_2$ računa se sljedećim rekursivnim algoritmom:

```

ispitaj( $I_1, I_2, i$ )
{
    ako (( $i == 0$ ) && (( $I_1 == I_2$ ) || ( $I_1 \succ I_2$ )))
    {
        vrati ISTINITO;
    }
    inače
    {
        vrati NEISTINITO;
    }

    ako ( $i \geq 1$ )
    {
        za bilo koju konfiguraciju  $I'$  duljine kraće od  $f(n)$ 
        {
            ako ((ispitaj( $I_1, I', i-1$ ) == ISTINITO) &&
                   (ispitaj( $I', I_2, i-1$ ) == ISTINITO))
            {
                vrati ISTINITO;
            }
            inače
            {
                vrati NEISTINITO;
            }
        }
    }
}
}

```

Gradi se deterministički TS M_2 koji simulira rad nedeterminističkog TS M_1 na sljedeći način. Neka se ispituje niz w duljine n . TS M_2 poziva algoritam $\text{ispitaj}(I_1, I_2, i)$ sa sljedećim vrijednostima parametrima:

- 1) Na temelju duljine niza n izračuna se vrijednost funkcije $f(n)$.
- 2) Konfiguracija I_1 jest početna konfiguracija TS M_1 za niz w .
- 3) Konfiguracija I_2 jest jedna od konfiguracija TS M_1 koja prihvaca niz.
- 4) Budući da maksimalni broj konfiguracija od početne konfiguracije do konfiguracije koja prihvaca niz je uvijek manji od $c^{f(n)}$, vrijednost parametar i je $\log_2(c^{f(n)}) = f(n) \log_2 c$.

Tijekom simulacije potrebno je ispitati sve konfiguracije TS M_1 koje prihvacaaju niz (parametar I_2). TS M_2 prihvaca niz ako i samo ako je rezultat izvođenja algoritama $\text{ispitaj}(I_1, I_2, i)$ potvrđan barem za jednu konfiguraciju koja prihvaca niz.

Jedna od radnih traka TS M_2 ima ulogu *LIFO* stoga. Rekurzivni poziv algoritma koristi najviše: $3f(n)$ ćelija za spremanje konfiguracija I_1, I_2 i I' , $3\log_2 n$ ćelija za binarna brojila koja pokazuju položaj glave u pojedinim konfiguracijama i $f(n) \log_2 c$ ćelija za spremanje binarnog brojila za parametar i . Sadržaj ulazne trake TS M_1 se ne mijenja i on se samo jednom spremi na početku rada TS M_2 . Budući da je $\log_2 n \leq f(n)$, a $\log_2 c$ ima konstantnu vrijednost, te budući da je moguće broj ćelija uvijek sažeti za konstantni faktor, pri svakom pozivu algoritma TS M_2 koristi naviše $f(n)$ ćelija. Maksimalni broj rekurzivnih poziva algoritma je $f(n) \log_2 c$ (početna vrijednost parametra i). Pomnoži li se broj ćelija potrebnih za jedan rekurzivni poziv algoritma i broj rekurzivnih poziva, TS M_2 ne koristi više od $f^2(n) \log_2 c$ ćelija, što se postupkom sažimanja skraćuje na $f^2(n)$ ćelija. Budući da jezik L prihvaca deterministički TS koji ne koristi više od $f^2(n)$ ćelija, jezik L jest u klasi DSPACE($f^2(n)$).

Svojstva hijerarhije jezika s obzirom na složenost prihvaćanja jezika

Jezići čine složenu hijerarhiju s obzirom na funkcije vremenske i prostorne složenost. Osnovna svojstva hijerarhije jezika s obzirom na složenost prihvaćanja jezika su: (i) hijerarhija jezika je beskonačna, (ii)-(iii) za potpuno prostorno i vremenski izgradive funkcije hijerarhija jezika je neprekinuta, (iv) hijerarhija

jezika nije neprekinuta za opći slučaj funkcija kada one nisu prostorno i vremenski izgradive, tj. postoje prazine u hijerarhiji jezika, (v) moguće je izgraditi jezik za koji ne postoji optimalni TS koji ga prihvata u minimalnom vremenu ili minimalnom prostoru, te (vi) za uniju klase jezika moguće je odrediti funkciju složenosti tako da obuhvati sve jezike iz unije. U nastavku odjeljka ukratko su opisana i objašnjena svojstva (i) do (vi).

Na temelju svojstva (i) zaključuje se da ne postoji niti jedna potpuno rekurzivna funkcija $f(n)$ takva da su svi rekurzivni jezici u klasi $\text{DTIME}(f(n))$ ili u klasi $\text{DSPACE}(f(n))$. Budući da ne postoji niti jedna potpuno rekurzivna funkcija $f(n)$ takva da su svi rekurzivni jezici u klasi $\text{DTIME}(f(n))$ ili u klasi $\text{DSPACE}(f(n))$, hijerarhija jezika s obzirom na potpuno rekurzivne funkcije složenosti je beskonačna:

- i) Neka je $f(n)$ bilo koja vremenska ili prostorna funkcija složenosti i neka je $f(n)$ potpuno rekurzivna funkcija. Dokazuje se da postoji jezik L koji nije u klasi $\text{DTIME}(f(n))$, odnosno nije u klasi $\text{DSPACE}(f(n))$.

Svojstvo (i) dokazuje se za determinističku vremensku hijerarhiju. Na sličan način moguće je dokazati da svojstvo (i) vrijedi za nedeterminističku vremensku hijerarhiju, te za determinističku i nedeterminističku prostornu hijerarhiju. Za potrebe dokaza proširuje se način kodiranja TS korišten u odjeljku 4.3.2 na TS s višestrukim trakama i na skup znakova trake $\{0, 1, B, X_4, X_5, \dots, X_m\}$. Znak trake X_k kodira se nizom nula 0^k , slično kao što se u odjeljku 4.3.2 kodiraju stanja. Relacija između kodova niza w_i i kodova TS M_i opisana je na slici 4.13 u odjeljku 4.3.2: niz w_i jest i -ti niz u kanonskom slijedu svih nizova, a vrijednost indeksa j u oznaci TS M_j jednaka je cijelobrojnoj vrijednosti njegovog binarnog koda. Definicija jezika zasniva se na dijagonalizaciji: $L = \{x_i \mid M_i \text{ ne prihvata } x_i \text{ u manje od } f(|x_i|) \text{ pomaka glave za čitanje i pisanje}\}$, gdje je funkcija $f(n)$ potpuno rekurzivna funkcija.

Potrebno je dokazati da je jezik L rekurzivan. Gradi se TS M koji prihvata jezik L i koji uvijek stane bez obzira na ulazni niz. Budući da je $f(n)$ potpuno rekurzivna funkcija, postoji TS N koji računa vrijednost funkcije $f(n)$ i koji uvijek stane bez obzira na ulazni niz. TS M izračuna duljinu ulaznog niza $n=|w|$, a zatim simulira rad TS N i izračuna vrijednost funkcije $f(n)$. Na temelju niza w_i TS M odredi indeks i u kanonskom slijedu svih nizova, odnosno odredi binarni zapis indeksa i koji je ujedno i kod TS M_i , te pokrene simulaciju rada TS M_i . Napomena: ako binarni zapis indeksa i nije valjani kod funkcije prijelaza TS, onda se pretpostavi da TS M_i nema definiran niti jedan prijelaz. Niz je u jeziku L ako tijekom simulacije TS M_i stane i ne prihvati niz. Nadalje, niz je u jeziku L ako tijekom simulacije TS M_i ne prihvati niz, a glava se pomakne više od $f(n)$ puta. TS M koristi zasebnu traku za potrebe brojila pomaka glave TS M_i . Budući da TS M uvijek stane bez obzira na ulazni niz, jezik L je rekurzivan.

Na sljedeći način dokazuje se da jezik L nije u klasi $\text{DTIME}(f(n))$. Prepostavlja se da TS M_i vremenske složenosti $f(n)$ prihvata jezik $L=L(M_i)$. Neka je duljina niza x_i jednaka n . Prvo se pretpostavi da je niz x_i u jeziku $L(M_i)$. Budući da je TS M_i vremenske složenosti $f(n)$, TS M_i prihvata niz x_i u manje od $f(n)$ pomaka glave. Međutim, na temelju definicije jezika $L = \{x_i \mid M_i \text{ ne prihvata } x_i \text{ u manje od } f(|x_i|) \text{ pomaka glave za čitanje i pisanje}\}$, niz x_i nije u jeziku L , što je suprotno pretpostavci da je $L(M_i)=L$. Druga pretpostavka je da niz x_i nije u jeziku $L(M_i)$. Budući da se niz x_i ne prihvata u manje od $f(n)$ pomaka glave, na temelju definicije jezika $L = \{x_i \mid M_i \text{ ne prihvata } x_i \text{ u manje od } f(|x_i|) \text{ pomaka glave za čitanje i pisanje}\}$, niz x_i je u jeziku L , što je suprotno pretpostavci da je $L(M_i)=L$. Zaključuje se da obje pretpostavke dovode do suprotnosti, što dokazuje da za bilo koju potpuno rekurzivnu funkciju $f(n)$ postoji jezik L koji nije u klasi $\text{DTIME}(f(n))$, odnosno hijerarhija jezika s obzirom na potpuno rekurzivne funkcije je beskonačna.

Ako je moguće postupnim povećanjem funkcije složenosti prihvatići sve šire skupove jezika, onda je hijerarhija jezika neprekinuta. U odjeljku 6.2.2 pokazano je da se konstantni faktori u funkciji složenosti mogu zanemariti, tj. množenje funkcije složenosti konstantom ne proširuje skup jezika koji se prihvata. Međutim, na temelju svojstva (ii) zaključuje se da minimalnim povećanjem funkcije složenosti, na primjer množenjem funkcije složenosti funkcijom $\log n$, moguće je prihvatići širi skup jezika:

- ii) Neka je $S_2(n)$ potpuno prostorno izgradiva funkcija za koju vrijedi da je $\inf_{n \rightarrow \infty} S_1(n)/S_2(n) = 0$. Ako su funkcije $S_1(n)$ i $S_2(n)$ najmanje $\log_2 n$, onda postoji jezik koji je u $\text{DSPACE}(S_2(n))$, a koji nije u $\text{DSPACE}(S_1(n))$.

Konstrukcija jezika L koji je u $\text{DSPACE}(S_2(n))$, a koji nije u $\text{DSPACE}(S_1(n))$, zasniva se na procesu dijagonalizacije. Gradi se TS M za koji vrijedi: TS M prihvata jezik L zadan nad binarnom abecedom $\{0,1\}$, TS M je prostorne složenosti $S_2(n)$ i TS M za barem jedan ulazni niz donosi odluku suprotnu od bilo kojeg TS koji je prostorne složenosti $S_1(n)$. Znakovi trake TS M su 0, 1 i praznina B .

Prostorna složenost $S_2(n)$ osigura se na sljedeći način. TS M simulira rad bilo kojeg TS M_{S_2} koji je prostorne složenosti $S_2(n)$. Budući da je funkcija $S_2(n)$ potpuno prostorno izgradiva, TS M_{S_2} za bilo koji niz duljine n koristi svih $S_2(n)$ ćelija. Tijekom simulacije rada TS M_{S_2} , TS M koristi zasebnu traku na kojoj označi maksimalni broj ćelija $S_2(n)$ koje koristi TS M_{S_2} za ulazni niz w duljine n . U nastavku simulacije TS M koristi isključivo označene ćelije. Pomakne li se u nastavku simulacije izvan označenih ćelija, TS M se zaustavlja i ne prihvata ulazni niz w . Time se osigurava da je prostorna složenost TS M jednaka $S_2(n)$.

U drugom dijelu simulacije postiže se da za barem jedan ulazni niz TS M doneše odluku suprotnu od bilo kojeg TS koji je prostorne složenosti $S_1(n)$. To se postiže dijagonalizacijom: za binarni ulazni niz w TS M simulira rad TS M_w . Kod TS M_w jednak je binarnom zapisu ulaznog niza w . Nije li binarni zapis niza w valjan kod funkcije prijelaza TS, onda se pretpostavi da TS M_w nema definiran niti jedan prijelaz. Ako tijekom simulacije TS M_w prihvati ulazni niz w , onda TS M ne prihvata niz w . Ne prihvati li tijekom simulacije TS M_w niz w , TS M prihvata niz w .

Zaustavljanje TS M_w moguće je osigurati na sljedeći način. Prethodno je pokazano da ako je TS M_w prostorne složenosti $S_1(n)$, onda je vremenska složenost TS M_w jednaka $c^{S_1(n)}$. Tijekom simulacije TS M broji pomake glave TS M_w . Pomakne li tijekom simulacije TS M_w glavu više od $c^{S_1(n)}$ puta i ne prihvati niz, simulacija se zaustavlja i niz se prihvata. Time se postiže zaustavljanje simulacije za sve one TS M_w koji su prostorne složenosti $S_1(n)$. Ostali TS M_w koji nisu prostorne složenosti $S_1(n)$ također se zaustavljaju, ali rezultat njihovog izvođenja nema utjecaja na zadani jezik L . Izborom odgovarajuće baze brojila moguće je osigurati da je duljina brojila manja od vrijednosti obje funkcije $S_1(n)$ i $S_2(n)$.

Neka je kardinalni broj skupa znakova trake TS M_w jednak t . Budući da je skup znakova trake TS M jednak $\{0, 1 \text{ i } B\}$, za kodiranje jedne ćelije TS M_w potrebno je $\lceil \log_2 t \rceil$ ćelija TS M . Na temelju činjenice da je prostorna složenost TS M_w jednak $S_1(n)$, zaključuje se da TS M tijekom simulacije rada TS M_w ne koristi više od $\lceil \log_2 t \rceil S_1(n)$ ćelija. Za potrebe kodiranja TS M_w proširuje se postupak opisan u odjeljku 4.3.2. Dozvoljava se da kod TS M_w ima na početku proizvoljan broj jedinica, što omogućuje da jedan te isti TS M_w ima beskonačno mnogo kodova proizvoljne duljine. Budući da je kod bilo kojeg TS M_w proizvoljne duljine, te budući da vrijedi $\inf_{n \rightarrow \infty} S_1(n)/S_2(n) = 0$, postoji dovoljno dugački niz w duljine n za koji vrijedi $\lceil \log_2 t \rceil S_1(n) < S_2(n)$. Time se omogućava da TS M simulira rad TS M_w koji je prostorne složenosti $S_1(n)$ u prethodno označenih $S_2(n)$ ćelija.

Na temelju činjenice da TS M uvijek donosi odluku suprotnu od bilo kojeg TS M_w prostorne složenosti $S_1(n)$, zaključuje se da ne postoji niti jedan TS prostorne složenosti $S_1(n)$ koji prihvata jezik $L = L(M)$. Time je dokazano da jezik L nije prostorne složenosti $S_1(n)$, dok je istodobno pokazano da je jezik L prostorne složenosti $S_2(n)$.

Moguće je pokazati da slično svojstvo vrijedi za vremensku hijerarhiju:

- iii) Ako je $T_2(n)$ potpuno vremenska izgradiva funkcija za koju vrijedi da je $\inf_{n \rightarrow \infty} T_1(n) \log T_1(n)/T_2(n) = 0$, onda postoji jezik koji je u $\text{DTIME}(T_2(n))$, a koji nije u $\text{DTIME}(T_1(n))$.

Ako je $T_2(n)$ potpuno vremenski izgradiva funkcija, onda se povećavanjem vremena prepoznaće šira klasa jezika. Funkcija $T(n)$ je *vremenski izgradiva* ako postoji TS M vremenske složenosti $T(n)$ takav da za bilo koji n postoji barem jedan niz duljine n za koji TS M izvede svih $T(n)$ pomaka glave. Funkcija $T(n)$ je *potpuno vremenski izgradiva* ako za sve nizove duljine n TS M izvede svih $T(n)$ pomaka glave. Većina poznatih funkcija je potpuno vremenski izgradiva.

Konstrukcija jezika L koji je u $\text{DTIME}(T_2(n))$, a koji nije u $\text{DTIME}(T_1(n))$, slična je prethodnom primjeru i zasniva se na dijagonalizaciji. Gradi se TS M za koji vrijedi: TS M prihvata jezik L , TS M je

vremenske složenosti $T_2(n)$ i TS M za barem jedan ulazni niz donosi odluku suprotnu od bilo kojeg TS koji je vremenske složenosti $T_1(n)$.

TS M istodobno simulira rad TS M_w za ulazni niz w i TS M_{T_2} koji je vremenske složenosti $T_2(n)$, gdje je funkcija složenosti $T_2(n)$ potpuno vremenski izgradiva. Simulacija TS M_{T_2} ima za cilj određivanje maksimalnog broja pomaka glave $T_2(n)$. Način kodiranja TS M_w je isti kao u prethodnom primjeru gradnje TS M . Budući da je funkcija složenosti $T_2(n)$ potpuno vremenski izgradiva, TS M zaustavlja simulaciju nakon što za niz w duljine n TS M_w pomakne glavu više od $T_2(n)$ puta. Zato je potrebno da je funkcija $T_2(n)$ potpuno vremenski izgradiva, kako bi se osiguralo da je TS M vremenske složenosti $T_2(n)$. TS M prihvata niz w ako i samo ako završi simulacija, a TS M_w ne prihvati niz w .

Budući da TS M_w koristi više traka od TS M , višestruke trake TS M_w smanje se na dvije trake. U odjeljku 6.2.2 pokazano je da smanjenje traka na dvije trake povećava vrijeme prihvatanja jezika, te ono iznosi $T_1(n) \log T_1(n)$. Budući da je kod bilo kojeg TS M_w proizvoljne duljine (kod TS M_w započinje s proizvoljnim brojem jedinica), te budući da vrijedi $\inf_{n \rightarrow \infty} T_1(n) \log T_1(n)/T_2(n) = 0$, postoji dovoljno dugački niz w duljine n za koji vrijedi $T_1(n) \log T_1(n) < T_2(n)$. Time se omogućava da TS M simulira rad TS M_w koji je vremenske složenosti $T_1(n)$ u vremenu $T_2(n)$.

Na temelju činjenice da TS M uvijek donosi odluku suprotnu od bilo kojeg TS M_w vremenske složenosti $T_1(n)$, zaključuje se da ne postoji niti jedan TS vremenske složenosti $T_1(n)$ koji prihvata jezik $L=L(M)$. Time je dokazano da jezik L nije vremenske složenosti $T_1(n)$, dok je istodobno pokazano da je jezik L vremenske složenosti $T_2(n)$.

Međutim, ako je funkcija složenosti proizvoljna potpuna rekurzivna funkcija, onda se u hijerarhiji nalaze praznine. Praznina podrazumijeva da za bilo koju potpunu rekurzivnu funkciju f postoji funkcija složenosti $R_f(n)$ takva da vrijedi $\text{DTIME}(R_f(n)) = \text{DTIME}(f(R_f(n)))$, odnosno ne postoji niti jedan jezik koji je u klasi $\text{DTIME}(f(R_f(n)))$, a da nije u klasi $\text{DTIME}(R_f(n))$. U hijerarhiji jezika između granica $R_f(n)$ i $f(R_f(n))$ ne postoji niti jedan jezik, *odnosno u hijerarhiji se nalaze praznine*:

- iv) Neka je $g(n)$ proizvoljna potpuna rekurzivna funkcija, i neka je $g(n) \geq n$. Postoji potpuno rekurziva funkcija $S(n)$ takva da je $\text{DSPACE}(S(n)) = \text{DSPACE}(g(S(n)))$. Isto svojstvo vrijedi i za vremensku složenost prihvatanja jezika.

Ako je funkcija $g(n)$ proizvoljna potpuna rekurzivna funkcija, onda hijerarhija jezika s obzirom na složenost prihvatanja jezika nije neprekidna kao u slučaju kad je funkcija prostorno ili vremenski potpuno izgradiva. Postoje praznine između granica $g(n)$ i $g(S(n))$ u kojima nije niti jedan jezik, odnosno svi jezici koji su složenosti $g(S(n))$ su ujedno i složenosti $S(n)$. Praznine u hijerarhiji imaju za posljedicu da postaje potpuno rekurzivne funkcije $f(n)$ za koje vrijedi:

$$\text{DTIME}(f(n)) = \text{NTIME}(f(n)) = \text{DSPACE}(f(n)) = \text{NSPACE}(f(n)).$$

U odjeljku 6.2.2 pokazana je mogućnost smanjenja vremena ili prostora prihvatanja bilo kojeg jezika za konstantni faktor. Međutim, vrijeme ili prostor za prihvatanje nekih jezika moguće je proizvoljan broj puta smanjiti. Funkciju složenosti moguće je smanjiti za više od konstantnog faktora, na primjer za funkciju drugog korjena. Za te jezike ne postoji optimalni TS koji ga prihvata u minimalnom vremenu ili prostoru:

- v) Ako je $r(n)$ bilo koja potpuna rekurzivna funkcija, onda postoji rekurzivni jezik L takav da za bilo koji TS M_i koji prihvata jezik $L(M_i)=L$ postoji TS M_j koji prihvata jezik $L(M_j)=L(M_i)=L$. Za TS M_i koji je složenosti $S_i(n)$ i TS M_j koji je složenosti $S_j(n)$ vrijedi $r(S_j(n)) \leq S_i(n)$ za skoro sve vrijednosti n . Izraz "skoro za sve vrijednosti n " znači da $r(S_j(n)) \leq S_i(n)$ ne vrijedi za konačni broj vrijednosti parametra n , dok za sve ostale vrijedi.

Za bilo koji TS M_i koji prihvata jezik L moguće je izgraditi TS M_j koji prihvata isti jezik L , ali u kraćem vremenu ili manjem prostoru. Za izgrađeni TS M_j moguće je ponovno izgraditi novi TS M_k koji prihvata isti jezik u još kraćem vremenu ili manjem prostoru. Proces izgradnje novih TS koji prihvataju

jezik L u kraćem vremenu ili manjem prostoru moguće je nastaviti neograničeni broj puta. Zaključuje se da za jezik L ne postoji optimalni TS koji ga prihvata u minimalnom vremenu ili prostoru¹.

Sljedeće svojstvo omogućava gradnju složenih klasa jezika:

- vi) Neka je $\{f_i(n) \mid i=1, 2, \dots\}$ skup rekurzivnih funkcija za koji je moguće izgraditi TS M koji ispisuje listu TS M_1, M_2, \dots koji računaju funkcije $f_1(n), f_2(n), \dots$. TS najprije ispisuje TS M_1 koji računa funkciju $f_1(n)$, zatim ispisuje TS M_2 koji računa funkciju $f_2(n)$, itd. Neka za sve parametre n i i vrijedi da je $f_i(n) < f_{i+1}(n)$. Dokazuje se da postoji rekurzivna funkcija $S(n)$ za koju vrijedi:

$$\text{DSPACE}(S(n)) = \bigcup_{i \geq 1} \text{DSPACE}(f_i(n)).$$

Moguće je dokazati da su svi jezici iz klase $\text{DSPACE}(f_i(n))$ ujedno i u klasi $\text{DSPACE}(S(n))$. Nadalje, dokazuje da su to jedini jezici koji čine klasu $\text{DSPACE}(S(n))$. Svojstvo unije omogućava gradnju različitih klasa jezika. Na primjer, najpoznatije klase jezika su one koje obuhvaćaju sve jezike polinomne funkcije složenosti. U sljedećem odjeljku opisane su najpoznatije klase jezika s obzirom na složenost prihvatanja jezika.

6.2.4 Klase jezika polinomne složenosti

Postoji mnogo jezika koji su ne samo izračunljivi, već su i odlučivi. Međutim, ne postoji učinkovit način prihvatanja tih jezika. Neke od jezika moguće je prihvati isključivo primjenom determinističkog TS eksponencijalne složenosti. Praktično prihvatanje tih jezika zahtjeva uporabu heuristika, te uvođenje različitih ograničenja koja smanjuju složenost prihvatanja jezika. Pronađe li se za dane jezike jednostavnije rješenje determinističke složenosti koja nije eksponencijalna, to bi znatno olakšalo rješavanje mnogih problema u matematici, računarstvu i ostalim znanstvenim područjima.

Značajna klasa jezika jest ona determinističke polinomne složenosti. Iako se ne može tvrditi da je prihvatanje jezika učinkovito za polinomnu funkciju složenosti stupnja n^{1958} , za većinu jezika polinomne složenosti polinom je niskog stupnja. Klasa jezika polinomne vremenske složenosti označava se znakom P . Klasa jezika P definira se na sljedeći način:

$$P = \bigcup_{i \geq 1} \text{DTIME}(n^i)$$

Na temelju svojstva unije klase jezika, u klasi jezika P su svi oni jezici za koje postoji deterministički TS vremenske složenosti n, n^2, n^3, n^4, \dots .

Mnogi značajni jezici nisu u klasi P , ali za njih postoji nedeterministički TS polinomne vremenske složenosti. Klasa jezika nedeterminističke polinomne vremenske složenosti označava se oznakom NP :

$$NP = \bigcup_{i \geq 1} \text{NTIME}(n^i)$$

¹ Konstrukcija jezika L je složena i zasniva se na procesu dijagonalizacije. Neka su nizovi M_1, M_2, M_3, \dots nabrojani kodovi svih neizravnih TS. Za kodiranje TS koristi se postupak opisan u odjeljku 4.3.2. Jezik L definira se tako da beskonačno mnogo TS M_i u nizu M_1, M_2, M_3, \dots prihvata jezik L . Nadalje, jezik L definira se tako da se s porastom indeksa i TS M_i koji prihvata jezik L u nizu M_1, M_2, M_3, \dots smanjuje vrijeme ili prostor potreban za prihvatanje jezika. Ako TS M_j koji prihvata jezik L ima veći indeks od nekog drugog TS M_i koji isto tako prihvata jezik L , odnosno $j < i$, onda TS M_i prihvata jezik u kraćem vremenu ili manjem prostoru. Budući da je niz kodova TS M_1, M_2, M_3, \dots beskonačan, vrijeme ili prostor potreban za prihvatanje jezika L moguće je beskonačno puta smanjiti.

Primjer 6.2. Sljedeća dva jezika su u klasi NP : L_{sat} i L_{vc} .

Jezik L_{sat} . Za gradnju logičkog izraza koriste se varijable, zgrade, logički operator i (\wedge), logički operator *ili* (\vee) i operator negacije (\neg). Varijable mogu poprimiti vrijednost 0 ili 1. Logički izraz promatra se kao niz znakova. Jezik L_{sat} definira se na sljedeći način: niz w , odnosno logički izraz predstavljen nizom znakova w , je u jeziku L_{sat} ako i samo ako je moguće varijablama logičkog izraza pridružiti vrijednosti 0 ili 1 tako da logički izraz poprini vrijednost 1. Na primjer, niz:

$$y_1 \wedge y_2 \text{ je u jeziku } L_{sat},$$

jer pridruživanjem $y_1=1$ i $y_2=1$ logički izraz $y_1 \wedge y_2$ poprini vrijednost 1.

Niz:

$$y_1 \wedge \neg y_1 \text{ nije u jeziku } L_{sat},$$

jer varijabli y_1 nije moguće pridružiti niti jednu vrijednost takvu da logički izraz $y_1 \wedge \neg y_1$ poprini vrijednost 1.

Dokazano je da je jezik L_{sat} u klasi NP . TS nedeterministički generira vrijednosti za sve varijable logičkog izraza, te za dobivne vrijednosti varijabli provjerava da li je vrijednost logičkog izraza 1.

Jezik L_{vc} . Neka je $G=(V, E)$ neusmjereni graf sa skupom čvorova V i skupom grana E . Definicija neusmjerenog grafa data je u odjeljku 1.1. Okrilje čvorova A je podskup skupa čvorova V ($A \subseteq V$) za koji vrijedi sljedeće: ako je (x, y) bilo koja granica skupa grana E ($(x, y) \in E$), onda je u skupu A barem jedan od čvorova x ili y . Nizovi jezika L_{vc} grade se na sljedeći način:

$$k ** 1 * 2 * 3 * 4 * 5 ** 1,3 * 3,4 * 2,2 * 2,5$$

Znakovi $**$ odjeljuju cijeli broj k , čvorove grafa G označene brojkama 1 do 5, te grane grafa x, y , gdje su x i y čvorovi grafa. Nizom je opisan primjer grafa koji je grafički prikazan na slici 1.1 u odjeljku 1.1. Niz je u jeziku L_{vc} ako i samo ako za graf G postoji okrilje čvorova u kojemu je k ili manje čvorova. Na primjer, niz:

$$2 ** 1 * 2 * 3 * 4 * 5 ** 1,3 * 3,4 * 2,2 * 2,5 \text{ je jeziku } L_{vc},$$

jer postoji okrilje čvorova $A=\{2, 3\}$ koje ima dva ili manje čvorova. Niz:

$$1 ** 1 * 2 * 3 * 4 * 5 ** 1,3 * 3,4 * 2,2 * 2,5 \text{ nije jeziku } L_{vc},$$

jer ne postoji okrilje čvorova koje ima jedan ili manje čvorova.

Dokazano je da je jezik L_{vc} u klasi NP . TS nedeterministički generira sve skupove od k čvorova, te za svaki generirani skup provjerava da li je okrilje čvorova zadano grafa.

Prethodni primjeri zorno pokazuju razliku između složenosti jezika u klasi NP i P . Odgovoriti na pitanje je li varijablama logičkog izraza moguće pridružiti vrijednosti 0 i 1 tako da vrijednost izraza postane 1 je daleko teže nego odgovoriti na pitanje da li za zadane vrijednosti varijabli logički izraz poprini vrijednost 1. Budući da je složenosti jezika L_{sat} jednaka NP , složenosti odgovor na prvo pitanje je također NP . Provjera da li izraz za zadane vrijednosti varijabli poprini vrijednost 1 koristi se u postupku odgovora na prvo pitanje i složenost tog postupka je jednostavnija i iznosi P . Slična je razlika između složenosti odgovora na pitanje da li za zadani graf i cijeli broj k postoji okrilje čvorova u kojemu je k ili manje čvorova i odgovora na pitanje da li je zadani skup čvorova okrilje čvorova zadano grafa. Da bi se odgovorilo na prvo pitanje, potrebno je odgovoriti na drugo pitanje za svaki od nedeterministički generiranih skupova od k čvorova.

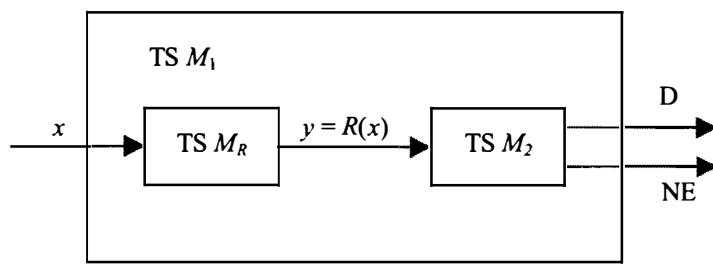
Klase jezika s obzirom na prostornu polinomnu složenost su:

$$\text{PSPACE} = \bigcup_{i \geq 1} \text{DSPACE}(n^i)$$

$$\text{NSPACE} = \bigcup_{i \geq 1} \text{NSPACE}(n^i)$$

Uzme li se u obzir svojstvo prostorne složenosti $\text{NSPACE}(f(n)) \subseteq \text{DSPACE}(f^2(n))$ koje je pokazano u odjeljku 6.2.3 i uvrsti se da je $f(n) = n^i$, zaključuje se da je $\text{NSPACE}(n^i) \subseteq \text{DSPACE}(n^{2i})$. Budući da je $\text{NSPACE}(n^i) \subseteq \text{DSPACE}(n^{2i})$, te na temelju definicija klase jezika PSPACE i NSPACE zasnovane na uniji, zaključuje se da je $\text{PSPACE} = \text{NSPACE}$. Odnosi među ostalim klasama su: $P \subseteq NP \subseteq \text{PSPACE}$. Međutim, niti za jedan podskup nije dokazano da je pravi^{*} podskup.

Svođenje jezika



Slika 6.12: Svođenje jezika $L_1 = L(M_1)$ na jezik $L_2 = L(M_2)$

Jezik L_1 svodi se na jezik L_2 ako postoji TS M_R koji generira izlazni niz $y = R(x)$ iz jezika L_2 ako i samo ako je ulazni niz x u jeziku L_1 . Prihvaćanje jezika L_1 svodi se na prihvaćanje jezika L_2 na sljedeći način: TS M_R na temelju ulaznog niza x računa izlazni niz $y = R(x)$, a onda se provjerava je li izlazni niz y u jeziku L_2 . Ulagani niz x je u jeziku L_1 ako i samo ako je izlazni niz $y = R(x)$ u jeziku L_2 . Slika 6.12 prikazuje gradnju TS M_1 koji prihvata jezik L_1 postupkom svođenja na jezik L_2 koji prihvata TS M_2 .

Ovisno o složenosti TS M_R , moguće je procijeniti složenost jezika L_2 u odnosu na složenost jezika L_1 . Ako je složenost TS M_R mala, te budući da je moguće svesti prihvaćanje jezika L_1 na prihvaćanje jezika L_2 primjenom učinkovitog TS M_R , onda je jezik L_2 jednake ili veće složenosti od jezika L_1 . Na primjer, jezik L_1 koji je polinomne vremenske složenosti moguće je u polinomnom vremenu svesti na jezik L_2 koji je eksponencijalne ili polinomne vremenske složenosti. Prema tome, vremenska složenost jezika L_2 je jednaka ili veća od vremenske složenosti jezika L_1 . Međutim, nije moguće jezik L_1 koji je eksponencijalne vremenske složenosti u polinomnom vremenu svesti na jezik L_2 koji je polinomne vremenske složenosti. Ako je moguće jezik L_1 u polinomnom vremenu svesti na jezik L_2 koji je polinomne vremenske složenosti, onda je i jezik L_1 polinomne vremenske složenosti, a ne eksponencijske, što se dokazuje u nastavku odjeljka.

Na temelju velike složenosti TS M_R nije moguće procijeniti složenost jezika L_1 i L_2 . Mogući su svi odnosi među jezicima L_1 i L_2 : jezici su jednake složenosti, jezik L_2 je veće složenosti od jezika L_1 , te jezik L_1 je veće složenosti od jezika L_2 . Na primjer, moguće je da TS M_R eksponencijalne vremenske složenosti svede jezik L_1 koji je polinomne vremenske složenosti na jezik L_2 koji je polinomne ili eksponencijalne vremenske složenosti. Nadalje, moguće je da TS M_R eksponencijalne vremenske složenosti svede jezik L_1 koji je eksponencijalne vremenske složenosti na jezik L_2 koji je polinomne ili eksponencijalne vremenske složenosti.

Učinkovito svođenje jezika zahtijeva bilo polinomnu vremensku složenost, ili logaritamsku prostornu složenost TS M_R :

Jezik L_1 se *polinomno vremenski svodi* na jezik L_2 ako postoji deterministički TS M_R polinomne vremenske složenosti koji generira izlazni niz $y=R(x)$ iz jezika L_2 ako i samo ako je ulazni niz x u jeziku L_1 .

Jezik L_1 se *logaritamski prostorno svodi* na jezik L_2 ako postoji deterministički TS M_R logaritamske prostorne složenosti koji generira izlazni niz $y=R(x)$ iz jezika L_2 ako i samo ako je ulazni niz x u jeziku L_1 .

Deterministički TS logaritamske prostorne složenosti je neizravni TS koji ima: jednu ulaznu traku, jednu radnu traku na kojoj se koristi ne više od $\log n$ čelija, gdje je n duljina ulaznog niza x , i jednu izlaznu traku na koju je moguće samo pisati, a glavu je moguće micati samo u desno.

Svodi li se jezik L_1 u polinomnom vremenu na jezik L_2 , dokazuje se sljedeće:

- a) Ako je jezik L_2 u klasi P , onda je i jezik L_1 u klasi P
- b) Ako je jezik L_2 u klasi NP , onda je i jezik L_1 u klasi NP .

Dokazi za oba slučaja su slični, pa je dan dokaz samo za slučaj (a). Neka je TS M_R koji svodi jezik L_1 na jezik L_2 polinomne vremenske složenosti $p_1(n)$, a TS M_2 koji prihvata jezik L_2 neka je polinomne vremenske složenosti $p_2(n)$. Dokazuje se da je TS M_1 koji prihvata jezik L_1 polinomne vremenske složenosti $p_1(n)+p_2(p_1(n))$. TS M_R pomakne glavu najviše $p_1(n)$ puta za generiranje izlaznog niza $y \in L_2$, gdje je $y=R(x)$ i $x \in L_1$. Budući da je moguće generirati najviše jedan znak jednim pomakom glave, duljina niza y mora biti kraća od maksimalnog broja pomaka glave $p_1(n)$, odnosno $|y| \leq p_1(n)$. Prihvatanje niza y iz jezika L_2 moguće je obaviti u vremenu $p_2(p_1(n))$, jer je vremenska složenost TS M_2 jednakna $p_2(n)$, a duljina niza je kraća od $p_1(n)$. Zaključuje se da je ukupno potrebno vrijeme da se prihvati jezik L_1 jednako $p_1(n)+p_2(p_1(n))$, odnosno TS M_1 koji prihvata jezik L_1 je vremenske složenosti $p_1(n)+p_2(p_1(n))$, što je polinomna funkcija argumenta n .

Svodi li se jezik L_1 u logaritamskom prostoru na jezik L_2 , dokazuje se sljedeće:

- c) Ako je jezik L_2 u klasi P , onda je i jezik L_1 u klasi P .
- d) Ako je jezik L_2 u klasi $\text{NSPACE}(\log^k n)$, onda je i jezik L_1 u klasi $\text{NSPACE}(\log^k n)$
- e) Ako je jezik L_2 u klasi $\text{DSPACE}(\log^k n)$, onda je i jezik L_1 u klasi $\text{DSPACE}(\log^k n)$.

Dokaz za slučaj (c) izvodi se u dva koraka. Najprije se dokazuje da determinističko logaritamsko prostorno svođenje jezika ne troši više od determinističkog polinomnog vremena. U odjeljku 6.2.3 dokazano je da jezik L koji je u klasi $\text{DSPACE}(f(n))$ je u klasi $\text{DTIME}(c^{f(n)})$. Uvrsti li se za $f(n)$ da je jednako $\log_b n$, dokazuje se da je jezik L polinome vremenske složenosti, odnosno da je jezik u klasi $\text{DTIME}(n^k)$:

$$c^{f(n)} = c^{\log_b n} \leq (b^k)^{\log_b n} = b^{k \log_b n} = (b^{\log_b n})^k = n^k$$

gdje je b baza logaritma, c je konstanta i k je cjelobrojna konstanta koja ovisi o bazi b . Budući da je pokazano da logaritamsko prostorno svođenje jezika ne troši više od polinomnog vremena, drugi dio dokaza je isti kao u slučaju (a) i (b).

Dokazi za slučajeve (d) i (e) zasnivaju se na konstrukciji TS M_1 koji simulira rad TS M_R i TS M_2 . TS M_R svodi jezik L_1 na jezik L_2 , a TS M_2 prihvata jezik L_2 . Budući da je u dokazu za slučaj (c) pokazano da je TS M_R vremenske složenosti $\text{DTIME}(n^k)$, te budući da primjenom n^k pomaka glave nije moguće generirati niz dulji od n^k , generirani niz y je kraći od n^k . Međutim, duljina niza y koja iznosi n^k je veća od $\log^k n$ i nije moguće da tijekom simulacije TS M_R odjednom generira i zapiše čitav niz $y \in L_2$ na svoju izlaznu traku. Simulacija TS M_R izvodi se korak po korak generiranjem jednog po jednog znaka niza y u samo jednu čeliju izlazne trake. TS M_1 na zasebnoj traci, koja ima ulogu brojila, zapisuje mjesto znaka u nizu y koji je u datom trenutku simulacije zapisan na ulaznoj traci TS M_2 . Budući da je duljina niza y kraća od n^k , moguće je izabrati takvu bazu brojila da je njegova duljina kraća od $\log^k n$.

TS M_1 simulira rad TS M_R i TS M_2 na sljedeći način. TS M_1 koristi zasebne trake za simulaciju sadržaja traka TS M_R i TS M_2 , a zasebne trake za spremanje stanja. Prethodno je opisano da se jedna traka koristi za brojilo mesta znaka u nizu. Neka se sadržaj brojila označi cijelobrojnom vrijednošću i . Zasebna traka koristi se za spremanje samo jednog znaka niza y , i to onog znaka koji u datom trenutku simulacije čita TS M_2 . TS M_1 pokrene simulaciju rada TS M_R . Simulacija TS M_R zaustavlja se u trenutku generiranja jednog od znakova niza y . Na temelju prethodno spremlijenog stanja, TS M_1 simulira rad TS M_2 za generirani znak niza y . Ako TS M_2 pomakne glavu u desno u nizu y , onda TS M_1 izračuna i spremi novo stanje TS M_2 , promijeni sadržaj traka TS M_2 , poveća za jedan brojilo mesta znaka u nizu ($i=i+1$) i nastavlja sa simulacijom TS M_R sve do trenutka generiranja sljedećeg znaka niza y . Miče li TS M_2 glavu u lijevo u nizu y , nije moguće više dohvatiti prethodni znak niza y , jer je on prepisan znakom koji u tom trenutku čita TS M_2 . Zato TS M_1 smanji za jedan brojilo mesta znaka u nizu y ($i=i-1$) i pokrene simulaciju rada TS M_R iz početne konfiguracije. Rad TS M_R simulira se sve do trenutka generiranja znaka koji ima mjesto i u nizu y , a koje je određeno sadržajem prethodno umanjjenog brojila. Na taj način simulira se pomak glave TS M_2 u lijevo u nizu y . Ako je sadržaj brojila $i=1$ i pomak glave je u lijevo u nizu y , TS M_1 simulira pomak glave TS M_2 na lijevi graničnik niza y . Stane li TS M_R prije nego što generira $i+1$ znak, TS M_1 simulira pomak glave TS M_2 na desni graničnik niza y . TS M_1 prihvata niz x ako i samo ako TS M_2 prihvata niz $y=R(x)$.

Na temelju sljedećih činjenica:

- i) tijekom simulacije TS M_1 na niti jednoj traci ne koristi više čelijskih skupova od TS M_R i TS M_2 ;
- ii) TS M_R je logaritamske prostorne složenosti;
- iii) TS M_2 je prostorne složenosti $\log^k n$;
- iv) za spremanje generiranog niza $y=R(x)$ koristi se samo jedna čelijska skupova;
- v) na traci koja se koristi kao brojilo ne koristi se više od $\log^k n$ čelijskih skupova,

zaključuje se da je TS M_1 koji prihvata jezik L_1 prostorne složenosti $\log^k n$.

Potpuni i teški jezici s obzirom na klasu jezika

Svojstvo tranzitivnosti svođenja jezika omogućuje razredbu jezika u klase čvrsto povezanih jezika:

Ako je jezik L u klasi K i svi jezici iz klase K su polinomno vremenski svodivi na jezik L , onda je jezik L *potpun* s obzirom na klasu K i s obzirom na polinomno vremensko svođenje.

Ako je jezik L u klasi K i svi jezici iz klase K su logaritamsko prostorno svodivi na jezik L , onda je jezik L *potpun* s obzirom na klasu K i s obzirom na logaritamsko prostorno svođenje.

Ako su svi jezici iz klase K polinomno vremenski svodivi na jezik L , a jezik L nije nužno u klasi K , onda je jezik L *težak* s obzirom na klasu K i s obzirom na polinomno vremensko svođenje.

Ako su svi jezici iz klase K logaritamsko prostorno svodivi na jezik L , a jezik L nije nužno u klasi K , onda je jezik L *težak* s obzirom na klasu K i s obzirom na logaritamsko prostorno svođenje.

Posebice su interesantne dvije klase jezika:

Jezik L je NP -potpun (NP -težak) ako i samo ako je jezik L potpun (težak) s obzirom na klasu NP i s obzirom na polinomno vremensko svođenje.

Da bi se dokazalo da je neki jezik L NP -težak, dovoljno je dokazati da se svi jezici iz klase NP mogu u polinomnom vremenu svesti na jezik L . Da bi se dokazalo da je jezik L NP -potpun, potrebno najprije dokazati da se svi jezici iz klase NP mogu u polinomnom vremenu svesti na jezik L , a zatim je potrebno dokazati da je jezik L u klasi NP . Za dokazivanje pripadnosti klasi NP koristi se prethodno

dokazano svojstvo (b). Uzme se neki jezik L_2 za koji je prethodno dokazano da je u klasi NP , a onda se dokaže da je moguće zadani jezik L svesti na jezik L_2 u polinomnom vremenu, što dokazuje da je jezik L u klasi NP .

Budući da je logaritamsko prostorno svođenje moguće obaviti u polinomnom vremenu, često se koristi sljedeća definicija za NP -potpune jezike i NP -teške jezike:

Jezik L je NP -potpun (NP -teški) ako i samo ako je jezik L potpun (težak) s obzirom na klasu NP i s obzirom na logaritamsko prostorno svođenje.

U primjeru 6.2 navedeni su primjeri jezika koji su u klasi NP -potpunih jezika.

U proučavanju složenosti jezika koriste se termini PSPACE-potpuni (PSPACE-teški):

Jezik L je PSPACE-potpun (PSPACE-teški) ako i samo ako je jezik L potpun (težak) s obzirom na klasu PSPACE i s obzirom na polinomno vremensko svođenje.

Kazalo pojmove s hrvatsko-engleskim rječnikom

ϵ -OKRUŽENJE, ϵ -CLOSURE, 35

Abeceda, alphabet, 8

Algoritam, algorithm

- ~ generiranja jezika koji se prihvata Turingovim strojem, generator of language that is accepted by Turing machine, 150
- ~ LR parsiranja, LR parsing ~, 98
- ~ odbacivanja ϵ -produkcijsa, elimination of ϵ -productions, 81
- ~ odbacivanja beskorisnih znakova, elimination of useless symbols, 80
- ~ odbacivanja jediničnih produkcijsa, elimination of unit productions, 83
- ~ odlučivanja za regularne jezike, decision ~ for regular sets, 56
- ~ pretvorbe produkcijsa u Chomskyjev normalni oblik, conversion of grammar into Chomsky normal form, 83
- ~ pretvorbe produkcijsa u Greibachov normalni oblik, conversion of grammar into Greibach normal form, 84
- ~ prihvaćanja jezika koji je generiran Turingovim strojem, recognizer of language that is generated by Turing machine, 150
- ~ prihvaćanja jezika koji je moguće generirati kanonskim slijedom, recognizer of language that is generated in canonical order, 151
- ~ pronaženja istovjetnih stanja konačnih automata, computation of equivalent states, 22
- ~ pronaženja nedohvatljivih stanja, computation of extraneous states, 27
- ~ razrješavanja lijeve rekurzije, elimination of left recursion, 85
- ~ sažimanja prostora za konstantni faktor, tape compression, 184
- ~ traženja dohvatljivih znakova, computation of reachable symbols, 79
- ~ traženja živih znakova, computation of alive symbols, 78
- ~ ubrzanja za konstantni faktor, linear speed up, 184
- ~ zamjene krajnje lijevog nezavršnog znaka, left corner substitution, 84

Analiza, analysis

leksička ~, lexical ~, 5, 15

sintaksna ~, syntax ~, 5, 69

semantička ~, semantic ~, 6

Apstraktno računalo RAM, random access machine (RAM), 159

Automat, automaton, 12

funkcija prijelaza ~a, transition function of ~, 16

način zapisa funkcije prijelaza primjenom vektora, transition-vector method, 20

našin zapisa funkcije prijelaza primjenom liste, transition-list method, 20

glava za čitanje i pisanje, head, 127

glava za čitanje, read-only head, 17

istovjetni ~, equivalent ~, 22

izlazna funkcija, output function, 40

izlazna traka, output tape, 150

linearno ograničen ~ (LOA), linear bounded ~ (LBA), 167

deterministički LOA (DLOA), deterministic LBA, 167

konačni ~, finite ~, 15

deterministički konačni ~ (DKA), deterministic finite ~ (DFA), 15

dijagram stanja konačnog ~, transition diagram of finite ~, 17

generator konačnog ~, finite ~ generator, 50

Mealyev ~, Mealy machine, 41

minimizacija ~, minimization of finite ~, 22

Mooreov ~, Moore machine, 39

nedeterministički konačni ~ (NKA), nondeterministic finite ~ (NFA), 29

nedeterministički konačni ~ s ϵ -prijelazima (ϵ -NKA), nondeterministic finite ~ with ϵ -

moves (NFA with ϵ -moves), 34

procesni konačni ~, processing machine, 17

konačni ~ s izlazom, finite ~ with output, 39

- tablica prijelaza konačnog ~a, transition table of finite ~, 17**
- konfiguracija ~a, instantaneous description of ~ (ID), 108, 130**
- potisni ~ (PA), pushdown ~ (PDA), 103**
- PA koji prihvaca praznim stogom, PDA that accepts by empty stack, 104
 - PA koji prihvaca prihvatljivim stanjem, PDA that accepts by final state, 104
 - nedeterministički PA, nondeterministic PDA, 106
 - deterministički PA (DPA), deterministic PDA, 110
- prijelaz ~a, move of ~, transition of ~, 16**
- ϵ -prijelaz, ϵ -move, ϵ -transition, 34
 - deterministički prijelaz, deterministic move, deterministic transition, 16
 - nedeterministički prijelaz, nondeterministic move, nondeterministic transition, 30
- ulazna traka, input tape, 17**
- upravljačka jedinka, finite control, 17**
- Backus-Naurov oblik (BNF oblik), Backus-Naur form (BNF), 59**
- Church-Turingova hipoteza, Church-Turing hypothesis, 158**
- Dokaz, proof, 11**
- ~ matematičkom indukcijom, inductive ~, 11
 - baza matematičke indukcije, basis, 11
 - hipoteza matematičke indukcije, inductive hypothesis, 11
 - korak matematičke indukcije, inductive step, 11
- Funkcija, function**
- parcijalno rekurzivna ~, partial recursive ~, 131
 - potpuno rekurzivna ~, total recursive ~, 131
- Graf, graph, 10**
- čvorovi ~a, vertices, nodes, 10
 - čvor prethodnik, predecessor, 10
 - čvor sljedbenik, successor, 10
 - grane ~a, edges, 10
 - put ~a, path, 10
 - usmjereni ~, directed ~, 10
 - usmjerenе grane ~a, arcs, 10
- Gramatika, grammar, 12, 56**
- desno-linearna ~, right-linear ~, 64
 - formalna ~, formal ~, 56
 - istovjetna ~, equivalent ~, 60
 - lijevo-linearna ~, left-linear ~, 64
 - lijevo rekurzivna ~, left-recursive ~, 85
 - kontekstno neovisna ~, context-free ~, 69
 - Chomskyjev normalni oblik, Chomsky normal form, 83
 - Greibachov normalni oblik, Greibach normal form, 84
 - kontekstno ovisna ~, context-sensitive ~, 165
 - LL(k) ~, LL(k) ~, 91
 - LR(k) ~, LR(k) ~, 88
 - nejednoznačna ~, ambiguous ~, 69
 - ~ neograničenih produkacija, ~ tipa 0, unrestricted ~, type 0 ~, semi-Thue, phrase structure, 152
 - pojednostavljenje ~ke, simplification of ~, 76
 - produkacija ~ke, production of ~, 14, 59
 - ϵ -produkacija, ϵ -production, 14, 59, 77, 81
 - jedinična produkacija, unit production, 77, 83
 - lijevo rekurzivna produkacija, left-recursive production, 85
 - razrješavanje nejednoznačnosti, eliminating ambiguity, 72
 - regularna ~, regular ~, 62
- Izraz, expression**
- aritmetički ~, arithmetic ~, 5, 58
 - logički ~, boolean ~, 5, 195
- Jezični procesor, compiler, 1**
- Jezik, language, 9**
- beskonačnan ~, infinite ~, 9

- Chomskyjeva hijerarhija ~a**, *Chomsky hierarchy of~s*, 178
ciljni ~, target ~, 1
~ definiranja ~a, meta-~, 1
dijagonalni ~, diagonal ~, 160
formalni ~, formal ~, 9
generirati ~, to generate ~, 12, 56
inherentno nejednoznačan ~, inherently ambiguous ~, 73
izračunjav ~, computable ~, 158, 160
izvorni ~, source ~, 1
~ izgradnje, implementation ~, 1
Kartezijski produkt ~a, *Cartesian product of~s*, 9
komplement ~a, complement of~, 9
kontekstno neovisan ~, context-free ~, 69
kontekstno ovisni ~, context-sensitive ~ (CSL), 165
deterministički kontekstno ovisni ~, deterministic context-sensitive ~, 172
nadovezivanje ~a, concatenation of~s, 9
nejednoznačan ~, ambiguous ~, 69
neodlučiv ~, undecidable ~, 163
neprazan ~, nonempty ~, 9
odlučiv ~, decidable ~, 163
presjek ~, intersection of~s, 9
prihvati ~, to accept ~, 12, 16
razlika ~a, difference of~s, 9
regularni ~, regular ~, regular set, 15
rekurzivan ~, recursive ~, 131
rekurzivno prebrojiv ~, recursively enumerable (r.e.) ~, 126
složenost prihvaćanja ~a, computational complexity of~, 180
hijerarhija ~a, hierarchy of~s, 190
beskonačnost hijerarhije ~a, infinity of hierarchy of~s, 191
beskonačno ubrzanje ~a, infinite speed-up of~, 913
neprekinutost (gustoća) hijerarhije ~a, density of hierarchy of~s, 191
praznine u hijerarhiji ~a, gaps in hierarchy of~s, 193
unija klasa ~a, union of classes of~s, 194
~ potpun s obzirom na (zadanu) klasu i s obzirom na (zadano) svodenje, ~ complete for (given) class with respect to (given) reduction, 198
NP –potpun, NP –complete, 198
PSPACE –potpun, PSPACE –complete, 199
~ težak s obzirom na (zadanu) klasu i s obzirom na (zadano) svodenje, hard for (given) class with respect to (given) reduction, 198
NP –težak, NP –hard, 198
PSPACE –težak, PSPACE –hard, 199
klase složenosti ~a (osnovne klase), complexity classes of~s, (basic classes), 187
DSPACE, 187
DTIME, 187
NSPACE, 187
NTIME, 187
odnosi među (osnovnim) klasama složenosti ~a, relations among (basic) complexity classes of~s, 187
polinomne klase složenosti ~a, polynomial complexity classes of~s, 194
klasa ~a determinističke polinomne vremenske složenosti P, deterministic polynomial time class P, 194
klasa ~a determinističke polinomne prostorne složenosti PSPACE, deterministic polynomial space class PSPACE, 196
klasa ~a nedeterminističke polinomne vremenske složenosti NP, nondeterministic polynomial time class NP, 194
klasa ~a nedeterminističke polinomne prostorne složenosti NSPACE, nondeterministic polynomial space class NSPACE, 196
prostorna složenost ~a, space complexity of~, 181

- prostorna hijerarhija ~a, space hierarchy of ~s**, 190
utjecaj smanjenja broja traka na prostornu složenost ~a, influence of reduction in number of tapes on space complexity, 183
vremenska složenost ~a, time complexity of ~, 182
vremenska hijerarhija ~a, time hierarchy of ~s, 190
utjecaj broja traka na vremenska složenost ~a, influence of reduction in number of tapes on time complexity, 184
strukturna složenost ~a, structural complexity of ~, 177
supstitucija ~a, substitution of ~s, 53, 121
svođenje ~a, reduction of ~, 196
 polinomno vremensko svođenje ~a, polynomial-time reduction of ~, 197
 logaritamsko prostorno svođenje ~a, log-space reduction of ~, 197
svojstva zatvorenosti ~a, closure properties of ~, 51
 konteksno neovisan ~, context-free ~, 119
 konteksno ovisni ~, context-sensitive ~, 170
 regularni ~, regular ~, regular set, 52
 rekurzivan ~, recursive ~, 156
 rekurzivno prebrojiv ~, recursively enumerable (r.e.) ~, 156
unija ~a, union of ~s, 9
univerzalni ~, universal ~, 163
- Leksička jedinka, token**, 5
Kanonski slijed, canonical order, 151
Ključna riječ, keyword, 5
Konstanta, constant, 5
Konstrukcija, construction
- ~ ε-NKA na temelju zadanih regularnih izraza, ~ of NFA with ε-moves from regular expressions, 46
 - ~ gramatike na temelju zadanog TS, ~ of grammar from Turing machine, 154
 - ~ gramatike na temelju zadanog DKA, ~ of grammar from DFA, 62
 - ~ determinističkog TS prostorne složenosti $f(n)$ na temelju determinističkog TS vremenske složenosti $f(n)$, ~ of deterministic $f(n)$ space-bounded Turing machine from deterministic $f(n)$ time-bounded Turing machine, 187
 - ~ determinističkog TS prostorne složenosti $f(n)$ na temelju nedeterminističkog TS prostorne složenosti $f(n)$, ~ of deterministic $f(n)$ space-bounded Turing machine from nondeterministic $f(n)$ space-bounded Turing machine, 189
 - ~ determinističkog TS vremenske složenosti $f(n)$ na temelju determinističkog TS prostorne složenosti $f(n)$, ~ of deterministic $f(n)$ time-bounded Turing machine from deterministic $f(n)$ space-bounded Turing machine, 187
 - ~ determinističkog TS vremenske složenosti $f(n)$ na temelju nedeterminističkog TS vremenske složenosti $f(n)$, ~ of deterministic $f(n)$ time-bounded Turing machine from nondeterministic $f(n)$ time-bounded Turing machine, 188
 - ~ DKA na temelju zadanog NKA, ~ of DFA from NFA, 32
 - ~ DKA koji prihvata komplement regularnog jezika, ~ of DFA that accepts complement of regular language, 52
 - ~ DKA koji prihvata presjek regularnih jezika, ~ of DFA that accepts intersection of regular languages, 52
 - ~ gramatike koja generira jezik koji nastaje nadovezivanjem konteksno neovisnih jezika, ~ of grammar that generates concatenation of context-free languages, 120
 - ~ gramatike koja generira jezik koji nastaje nadovezivanjem konteksno ovisnih jezika, ~ of grammar that generates concatenation of context-sensitive languages, 170
 - ~ gramatike koja generira jezik koji nastaje primjenom Kleeneovog operatora na konteksno neovisni jezik, ~ of grammar that generates Kleene closure of context-free language, 120
 - ~ gramatike koja generira jezik koji nastaje primjenom Kleeneovog operatora na konteksno ovisni jezik, ~ of grammar that generates Kleene closure of context-sensitive language, 171
 - ~ gramatike koja generira jezik koji nastaje supstitucijom konteksno neovisnih jezika, ~ of grammar that generates substitution of context-free languages, 121
 - ~ gramatike koja generira uniju konteksno neovisnih jezika, ~ of grammar for union of context-free languages, 119

- ~ **gramatike koja generira uniju konteksno ovisnih jezika**, ~ of grammar for union of context-sensitive languages, 170
 - ~ **konteksno ovisne gramatike na temelju zadanog LOA**, ~ of context-sensitive grammar from LBA, 168
 - ~ **konteksno neovisne gramatike na temelju zadanog PA koji prihvaca praznim stogom**, ~ of context-free grammar from PDA that accepts by empty stack, 115
 - ~ **lijeko-linearne gramatike na temelju zadanog NKA**, ~ of left-linear grammar from NFA, 68
 - ~ **LOA koji prihvaca presjek konteksno ovisnih jezika**, ~ of LBA that accepts intersection of context-sensitive languages, 171
 - ~ **LOA koji prihvaca komplement determinističkog konteksno ovisnog jezika**, ~ of LBA that accepts complement of deterministic context-sensitive language, 172
 - ~ **LOA na temelju zadane konteksno ovisne gramatike**, ~ of LBA from context-sensitive grammar, 168
 - ~ **Mealyevog automata na temelju zadanog Mooreovog automata**, ~ of Mealy machine from Moore machine, 41
 - ~ **Mooreovog automata na temelju zadanog Mealyevog automata**, ~ of Moore machine from Mealy machine, 42
 - ~ **NKA na temelju zadane desno-linearne gramatike**, ~ of NFA from right-linear grammar, 65
 - ~ **NKA na temelju zadane lijevo-linearne gramatike**, ~ of NFA from left-linear grammar, 67
 - ~ **NKA na temelju zadanog ϵ -NKA**, ~ of NFA from NFA with ϵ -moves, 37
 - ~ **NKA za regularni jezik zadan jednostavnom regularnom gramatikom**, ~ of NFA from simple regular grammar, 63
 - ~ **PA koji prihvaca prihvatljivim stanjem na temelju zadanog PA koji prihvaca praznim stogom**, ~ of PDA that accepts by final state from PDA that accepts by empty stack, 112
 - ~ **PA koji prihvaca praznim stogom na temelju zadanog PA koji prihvaca prihvatljivim stanjem**, ~ of PDA that accepts by empty stack from PDA that accepts by final state, 110
 - ~ **PA koji prihvaca praznim stogom jezik zadan konteksno neovisnom gramatikom**, ~ of PDA that accepts by empty stack from context-free grammar, 114
 - ~ **PA koji prihvaca presjek konteksno neovisnog jezika i regularnog jezika**, ~ of PDA that accepts intersection of context-free language and regular language, 123
 - ~ **TS koji generira nizove rekurzivnog jezika kanonskim slijedom**, ~ of Turing machine that generates words of recursive language in canonical order, 151
 - ~ **TS koji prihvaca jezik zadan gramatikom neograničenih produkcija**, ~ of Turing machine from recursively enumerable grammar, 153
 - ~ **TS koji prihvaca komplement dijagonalnog jezika**, ~ of Turing machine that accepts complement of diagonal languages, 164
 - ~ **TS koji prihvaca komplement rekurzivnog jezika**, ~ of Turing machine that accepts complement of recursive languages, 157
 - ~ **TS koji prihvaca rekurzivni jezik koji nije konteksno ovisan**, ~ of Turing machine that accepts recursive language that is not context-sensitive, 174
 - ~ **TS koji prihvaca uniju rekurzivnih jezika**, ~ of Turing machine that accepts union of recursive languages, 157
 - ~ **TS koji prihvaca uniju rekurzivno prebrojivih jezika**, ~ of Turing machine that accepts union of recursively enumerable languages, 157
 - ~ **TS koji uvijek stane na temelju zadane konteksno ovisne gramatike**, ~ of Turing machine that always halts from context-sensitive grammar, 173
 - ~ **TS koji uvijek stane na temelju zadanih Turingovih strojeva koji prihvaca jezik i njegov komplement**, ~ of Turing machine that always halts from Turing machine that accepts language and its complement, 158
 - ~ **univerzalnog TS**, ~ of universal Turing machine, 163
- Međukod**, intermediate code, intermediate representation, 6
- viši ~, high-level ~, 6
 - srednji ~, medium-level ~, 7
 - niži ~, low-level ~, 7
- Mikrokod**, microcode, 160
- Naredba**, instruction, statement, 160
- adresni dio ~be, address field of ~, 160
 - dekodiranje ~be, decoding of ~, 160

- dohvat ~be, ~fetch**, 160
~ uvjetnog grananja, conditional ~, 74
izvršenje ~be, ~execution, 160
kod ~be, ~code, operation code, 160
- Niz znakova, string, word**, 8
duljina ~a ~, length of ~, 8
nadovezivanje ~ova ~, concatenation of ~s, 8
nejednoznačan ~~, ambiguous ~, 69
oznaka kraja ~a ~, endmarker, 17
podsljed ~a ~, subsequence of ~, 8
postupak generiranja ~a zamjenom krajnje desnog nezavršnog znaka, rightmost derivation of ~, 71
postupak generiranja ~a zamjenom krajnje lijevog nezavršnog znaka, leftmost derivation of ~, 71
(pravi) pod~~, (proper) sub~, 8
(pravi) prefiks ~a ~, (proper) prefix of ~, 8
(pravi) sufiks ~a ~, (proper) suffix of ~, 8
prazni ~~, empty ~, 8
- Objekt, object**, 57
- Operand, operand**, 160
dohvat ~a, ~fetch, 160
- Operator, operator**, 46, 73
desno asocijativan ~, right-associative ~, 74
Kleeneov ~ *, Kleene closure, closure, 9
Kleeneov ~ +, Kleene positive closure, positive closure, 9
lijevo asocijativan ~, left-associative ~, 73
prednost ~a, ~precedence, 46
- Parser, parser**,
LL(k) ~, LL(k) ~, 91
LR(k) ~, LR(k) ~, 98
akcija LR ~a, action of LR ~, 99
konfiguracija LR ~a, configuration of LR ~, 99
tablica novo_stanje LR ~, goto table of ~, 99
program parsiranja LR ~a, LR parsing program of LR ~, 10
tablica parsiranja LR ~a, parsing table of LR ~, 99
- Parsiranje, parsing**, 89
~ od dna prema vrhu, bottom-up ~, 89
~ od vrha prema dnu, top-down ~, 97
~ tehnikom rekurzivnog spusta, recursive-descent ~, 95
- Predikat, predicate**, 57
- Program, program**, 1
analiza izvornog ~a, analysis of source ~, front end, 1
ciljni ~, target ~, 1
generirati ~, to generate ~, 7
glavni ~, main ~, 92
izvorni ~, source ~, 1
potprogram, subroutine, 92
prihvatići ~, to accept ~, 4
sinteza ciljnog ~, synthesis of target ~, back end, 5
- Programsko brojilo, program counter (PC)**, 159
obnavljanje ~g ~la , update of ~, 160
- Računalo sa smanjenim naredbenim skupom, reduced instruction set computer (RISC)**, 159
- Redukcija, reduction**, 97
- Registar, register**, 159
memorijski adresni ~, memory address ~ (MAR), 159
radni ~, arithmetic ~, 159
- Regularni, regular**
~ izrazi, ~ expressions, 44

~ne definicije, ~definitions, 53

Relacija, relation, 11

asimetrična ~, asymmetric ~, 11

binarna ~, binary ~, 11

domena ~je, ~domain, 11

kodomena, range, 11

nerefleksivna ~, irreflexive ~, 11

refleksivna ~, reflexive ~, 11

~ekvivalencije, equivalence ~, 11

simetrična ~, symmetric ~, 11

tranzitivna ~, transitive ~, 11

tranzitivno okruženje ~je, transitive closure of ~, 12

refleksivno okruženje ~je, reflexive closure of ~, 12

Simulacija, simulation,

~apstraktnog računala zasnovanog na memoriji sa slobodnim pristupom RAM pomoću TS, ~of random access machine RAM by Turing machine, 159

~nedeterminističkog TS pomoću determinističkog TS, ~of nondeterministic Turing machine by deterministic Turing machine, 144

~osnovnog modela TS pomoću stogovnog stroja, ~of Turing machine by multistack machine, 146

~osnovnog modela TS pomoću stroja s brojilima, ~of Turing machine by counter machine, 147

~TS s dvostranom beskonačnom trakom pomoću osnovnog modela TS, ~of Turing machine with two-way infinite tape by Turing machine with one-way infinite tape, 139

~TS s višedimenzionalnim poljem pomoću TS s jednodimenzionalnom trakom, ~of multidimensional Turing machines by Turing machines with one-dimensional tape, 144

~TS s višestrukim trakama pomoću TS s jednom trakom, ~of multitape Turing machine by Turing machine with one tape, 142

Skup, set, 9

element ~a, element of ~, member of ~, object of ~, 9

kardinalni broj ~a, cardinal number of ~, 9

neprebrojivo beskonačni ~, not countable ~, 9

partitivni ~, power ~, 9

prebrojivo beskonačni ~, countably infinite ~, countable ~, 9

~cijelih brojeva, ~of integers, 9

~izlaznih znakova, output alphabet, 39

~prirodnih brojeva, ~of natural numbers, 9

~racionalnih brojeva, ~of rationals, 9

~realnih brojeva, ~of reals, 9

~ulaznih znakova, input alphabet, 16

~znakova stoga, stack alphabet, 106

~znakova trake, tape alphabet, 127

Stablo, tree, 10

čvor dijete, son, 10

čvorovi listovi, leaves, 10

čvor predak, ancestor, 10

čvor potomak, descendant, 10

čvor roditelj, father, 10

generativno ~, derivation ~, 61

unutarnji čvorovi ~la, interior vertices of ~, 10

korijen ~la, root, 10

~parsiranja, parse ~, 89

Stanje, state, 16

istovjetno ~, equivalent ~, 22

uvjet podudarnosti, compatibility condition, 22

uvjet napredovanja, propagation condition, 22

nedohvatljivo ~, extraneous ~, 27

neprihvatljivo ~, not accepting ~, 12, 16

prihvatljivo ~, accepting state, final ~, 12, 16

- početno ~, initial ~**, 12, 16
- zapis ~nja, representing the ~**, 19
- izravan način zapisa ~nja, explicit method of representing the ~, 19
 - posredni način zaipsa ~nja, implicit method of representing the ~, 19
- višekomponentna oznaka ~nja, ~ consisting of multiple components**, 134
- upravljačka komponenta ~nja, control component of ~, 134
 - radna komponenta ~nja, storage component of ~, 134
- Stog, stack**, 103, 146
- potisni ~, last in-first out (LIFO) ~**, 103
- Turingov stroj (TS), Turing machine**, 126
- ~ koji generira jezik, language generator, 150
 - ~ koji prihvaca jezik, language recognizer, 130
 - ~ koji računa vrijednost funkcije, ~ as a computer of function, 131
 - ~ koji uvijek stane, ~ that always halts, 131
 - ~ koji ne stane uvijek, ~ that may or may not halt, 131
 - prošireni modeli ~og ~a, modification of ~**, 139
 - nedeterministički ~, nondeterministic ~, 144
 - neizravni ~, off-line ~, 146
 - ~ s dvostranom beskonačnom trakom, ~ with two-way infinite tape, 139
 - ~ s više glava za čitanje i pisanje, multihead ~, 146
 - ~ s višedimenzionalnim poljem, multidimensional ~, 144
 - ~ s višestrukim trakama, multitape ~, 142 - pojednostavljeni modeli ~og ~a, restricted ~**, 146
 - stogovni stroj, multistack machine, 146
 - stroj s brojilima, counter machine, 147
 - ~ s ograničenim brojem stanja i znakova trake, ~ with restricted number of states and with restricted tape alphabet, 149
- trag trake ~og ~a, track of ~**, 136
- donji trag trake, upper track, 140
 - gornji trag trake, lower track, 140
 - označni trag trake, checking off track, 137
 - pomoćni trag trake, auxiliary track, 136
 - ulazni trag trake, input track, 136
- univerzalni ~, universal ~**, 149, 163
- Svojstvo (lema) napuhavanja, pumping lemma**, 54, 124
- Subjekt, subject**, 57
- Tablica znakova, symbol table**, 6
- Znak, symbol**, 8
- beskoristan ~, useless ~, extraneous ~**, 76
 - dohvatljiv ~, reachable ~**, 76
 - izlazni ~, output ~**, 39
 - koristan ~, useful ~, nonextraneous ~**, 76
 - mrtav ~, dead ~**, 76
 - nedohvatljiv ~, unreachable ~**, 76
 - nezavršni ~, variable ~**, 12, 58
 - pretvorba ~ova, transliterating of ~s**, 19
 - zapis ~ova, representing the ~s**, 19
 - početni ~, start ~**, 12, 59
 - ulazni ~, input ~**, 16
 - završni ~, terminal**, 12, 58
 - ~ **stoga, stack ~**, 106
 - ~ **trake, tape ~**, 127
- višekomponentni ~ trake, ~ consisting of multiple components**, 136
- živi ~, alive ~**, 76
- Varijabla, variable, identifier**, 5

Literatura

- Aho, A.V., Sethi, R., and Ullman, J.D.: “*Compilers: Principles, Techniques, and Tools*”, Addison-Wesley Publishing Company, 1986.
- Calingaert, P.: “*Assemblers, Compilers, and Program Translations*”, Pitman Publishing Limited, 1979.
- Davis, M.D., Sigal, R., and Weyuker, Elaine J.: “*Computability, Complexity, and Languages : Fundamentals of Theoretical Computer Science (Computer Science and Applied Mathematics)*”, Academic Pr., 1994.
- Du, D.-Z., (Editor), Ko, K-I (Editor) and Du, D. (Editor): “*Advances in Algorithms, Languages, and Complexity*”, Kluwer Academic Publishers, 1997.
- Engelfriet, J.: “*Simple Program Schemes and Formal Languages*”, edited by G. Goos and J. Hartmanns, Lecture Notes in Computer Science, Springer-Verlag Berlin, 1974.
- Gojanović, D.: “*Rukopisi iz predavanja Jezični procesori*”, Elektrotehnički fakultet Sveučilišta u Zagrebu, Zagreb, šk.god. 1980/1981.
- Goos, G. and Hartmanns, J., editors: “*Design and Implementation of Programming Languages: DoD Sponsored Workshop, Ithaca 1976*”, edited by G. Goos and J. Hartmanns, Lecture Notes in Computer Science, Springer-Verlag Berlin, 1977.
- Goos, G. and Hartmanns, J., editors: “*Compiler Construction: An Advance Course*”, Lecture Notes in Computer Science, Springer-Verlag Berlin, 1974.
- Hopcroft, J.E. and Ullman, J.D.: “*Introduction to Automata Theory, Languages, and Computation*”, Addison-Wesley Publishing Company, 1979.
- Hopgood, F.R.A.: “*Compiling Techniques*”, Macdonald & Co. Ltd., American Elsevier Publishing Company, Inc., 1971.
- Howie, J.M.: “*Automata and Languages*”, Oxford University Press, 1992.
- Jurišić-Kette, W.: “*Rukopisi iz predavanja Jezični procesori*”, Elektrotehnički fakultet Sveučilišta u Zagrebu, Zagreb, šk.god. 1986/1987.
- Kelley, D.: “*Automata and Formal Languages: An Introduction*”, Prentice Hall, 1998.
- Kohavi, Z., Hamming, R.W. (Editor), Feigenbaum E.A. (Editor): “*Switching and Finite Automata Theory*”, McGraw Hill College Div., 1986.
- Kozen, D.C.: “*Automata and Computability (Undergraduate Texts in Computer Science)*”, Springer Verlag, 1997.
- Lewis, H.R., Papadimitriou, C. (Contributor): “*Elements of the Theory of Computation*”, Prentice Hall, 1997.
- Lewis II, P.M., Rosenkrantz, D.J., and Stearns, R.E.: “*Compiler Design Theory*”, Addison-Wesley Publishing Company, 1976.
- Linz, P.: “*Introduction to Formal Languages and Automata*”, Jones & Bartlett Pub., 1998.
- Maričić, B. i Vuković, D.: “*Priručnik za laboratorijske vježbe iz predmeta Jezični procesori*”, Elektrotehnički fakultet Sveučilišta u Zagrebu, Zagreb, 1985.

- Meduna, A.: "Automata and Languages : Theory and Applications", Springer Verlag, 2000.
- McNaughton, R.: "Elementary Computability, Formal Languages, and Automata", Prentice-Hall, Inc., 1982.
- Moll, R.N., Arbib, M.A. and Kfoury, A.J.: "An Introduction to Formal Language Theory", Spring-Verlag New York Inc., 1988.
- Morgan, R.: "Building an Optimizing Compiler", Butterworth-Heinemann, 1998.
- Muchnick, S.S: "Advanced Compiler Design Implementation", Morgan Kaufmann Publishers, Inc., 1997.
- Nicholls, J.E.: "The Structure and Design of Programming Languages", Addison-Wesley Publishing Company, 1975.
- Papadimitriou, C.H.: "Computational Complexity", Addison-Wesley Publishing Company, 1994.
- Paučić, D.: "Programski paketi YACC i LEX", konstrukcijski rad, Elektrotehnički fakultet Sveučilišta u Zagrebu, Zagreb, 1987.
- Pratt, T.W.: "Programming Languages: Design and Implementation", Prentice-Hall, Inc., 1975.
- Revesz, G.E.: "Introduction to Formal Languages", Dover Pubns., 1991.
- Rozenberg, G. (Editor) and Salomaa, A. (Contributor): "Handbook of Formal Languages : Beyond Words", Springer Verlag, 1997.
- Rozenberg, G. (Editor) and Salomaa, A. (Contributor): "Handbook of Formal Languages : Word, Language, Grammar", Springer Verlag, 1997.
- Sammet, J.E.: "Programming Languages: History and Fundamentals", Prentice-Hall, Inc., 1969.
- Schreiner, A.T. and Frideman Jr., H.G.: "Introduction to Compiler Construction with Unix", Prentice-Hall, Inc., 1985.
- Simon, M.: "Automata Theory", World Scientific Pub. Co., 1999.
- Simovici, D.A. and Tenney, R.: "Theory of Formal Languages With Applications", World Scientific Pub Co., 1999.
- Srbljić, S.: "Sintaksna analiza sljednih upravljačkih programa", Automatika, Vol. 32, No. 3-4, 1991, str 101-106.
- Srbljić, S.: "Izvođenje radnih programa računalom upravljanog alatnog stroja", magistarski rad, Elektrotehnički fakultet Sveučilišta u Zagrebu, Zagreb, šk.god. 1985.
- Srbljić, S.: "Rukopisi iz predavanja Jezični procesori", Elektrotehnički fakultet Sveučilišta u Zagrebu, Zagreb, šk.god. 1992/1993.
- Srbljić, S.: "Rukopisi iz predmeta Automati, formalni jezici i jezični procesori I", Fakultet elektrotehnike i računarstva Sveučilišta u Zagrebu, Zagreb, šk.god. 1996/1997.
- Srbljić, S.: "Rukopisi iz predmeta Automati, formalni jezici i jezični procesori II", Fakultet elektrotehnike i računarstva Sveučilišta u Zagrebu, Zagreb, šk.god. 1996/1997.
- Srbljić, S.: "Laboratorijske vježbe iz predmeta Jezični procesori", Elektrotehnički fakultet Sveučilišta u Zagrebu, Zagreb, 1986.
- Taylor, R.: "Models of Computation and Formal Languages", Oxford Univ Press., 1997.