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 računalom zahtijeva izgradnju algoritma primjenom programskog jezika. Danas postoji mnoštvo programskih jezika i jezičnih procesora različitih mogućnosti i područja primjene. Sastavnice viših programskih jezika, kao što su varijable, iterativni i rekurzivni algoritmi, polja, datoteke i apstraktni tipovi podataka omogućuju korištenje računala, a da korisnik pri tome ne poznaje 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.

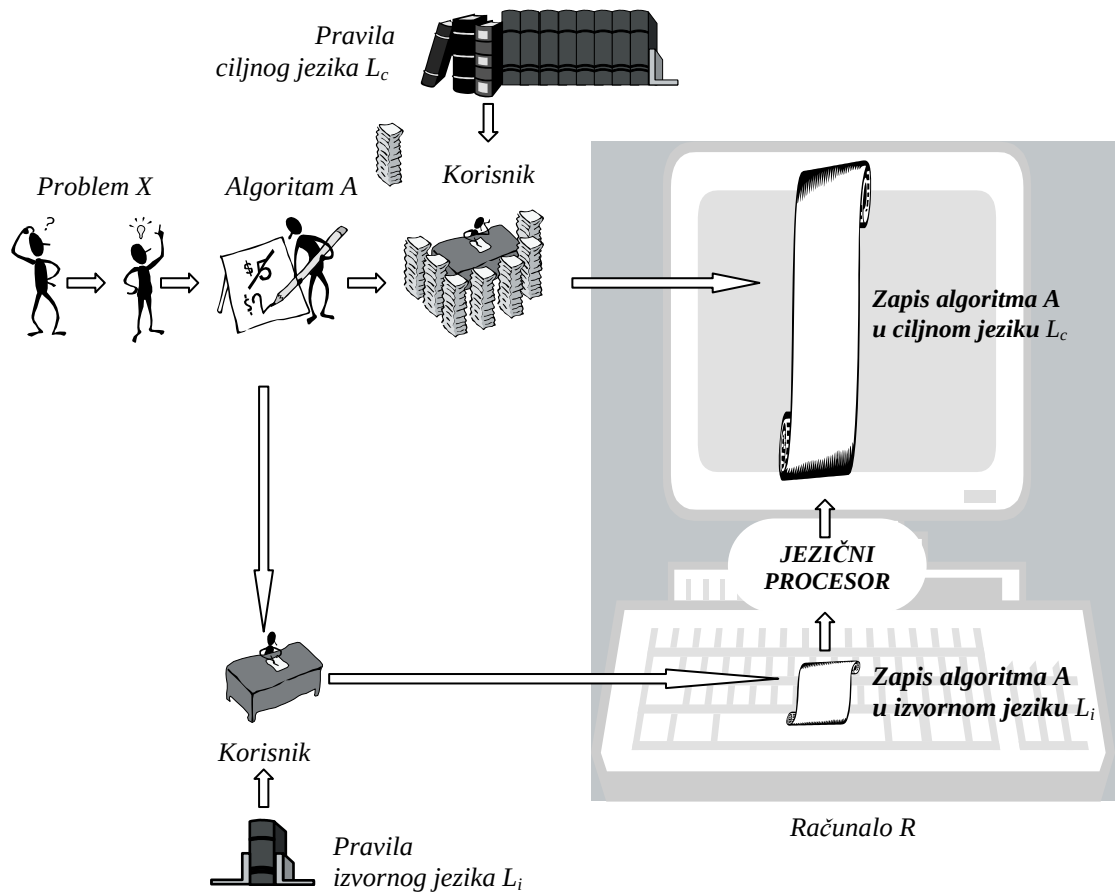
Slika 1.1 opisuje rad jezičnog procesora. Za rješavanje problema *X* želi se koristiti računalo *R*. Neka je programski jezik L_c ugrađen u računalo *R* i neka je programe zapisane tim programskim jezikom moguće izravno izvoditi na računalu *R*. *A* je algoritamski zapis rješenja problema *X*. Pretpostavimo da postoji više razloga zbog kojih programski jezik L_c nije prikladan za zapis algoritma *A*: neprilagođenost području primjene, složena pravila koja često pretpostavljaju poznavanje sklopovske građe računala, zapis algoritma *A* primjenom jezika L_c jest 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 primjenom programskog 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 zadatak jezičnog procesora je:

> *Jezični procesor* prevodi zapis algoritma iz *izvornog jezika* L_i u zapis algoritma u *ciljnom jeziku* L_c koji je moguće izvesti na zadanom 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 ciljnom 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 kojim 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. Osim tri spomenuta jezika vezana uz definiciju jezičnog procesora, razvijaju se i posebni *jezici definiranja jezika*. Jezik definiranja jezika omogućuje jednostavni i jednoznačni zapis pravila programskih 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 primjenom 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 je L_{svi} skup svih nizova koje je moguće napisati primjenom 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 daljnjem 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 $JP_{L_g}^{L_i \to L_c}$ 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, zaustavlja se daljnji proces prevođenja izvornog jezika. Jezični procesor ispisuje poruke o učinjenim pogreškama u izvornom programu. Za potrebe prihvaćanja programa (nizova) jezika L_i gradi se *formalni automat M*. Formalni automat *M* je matematički model automata koji čita ulazni niz znak po znak i koji na izlazu ispisuje je li pročitani niz znakova u zadanom jeziku (odnosno u zadanom skupu). 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, onda jezični procesor pokreće generiranje 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) jezika L_c (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 ciljnom 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_i ne generira bilo koji program jezika L_c (bilo koji niz skupa L_c), već da gramatika G_i 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, prevođenje nije moguće ostvariti jedinstvenim dvostupanjskim procesom 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 bilo kojoj razini ostvaruje se dvostupanjskim *procesom prihvaćanja* programa generiranog procesom prevođenja prethodne razine, te *generiranja programa* za proces prevođenja naredne razine. Početni proces prevođenja prihvaća izvorni program dok završni proces prevođenja generira ciljni program.

Različite razine prevođenja 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 prevođenja na dvije razine. Osnovni zadatak procesa prevođenja tijekom faze analize jest pojednostavljenje procesa prihvaćanja izvornog jezika. U većini današnjih jezičnih procesora, tijekom faze sinteze ciljnog programa proces prevođenja izvodi se na tri razine. Osnovni zadatak procesa prevođenja tijekom faze sinteze jest pojednostavljenje procesa generiranja ciljnog programa.

Tijekom faze analize izvornog programa izvode se dva procesa prevođenja. Jedan proces prevođenja izvodi se tijekom *leksičke analize*, a drugi proces prevođenja 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: *varijable* (npr. niz više slova i brojaka koji započinje slovom), *ključne riječi* (npr. ključna riječ naredbe grananja jest **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) zadanog programskog jezika. Dozvoljava se da je skup pravilno napisanih leksičkih jedinki 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. Sve leksičke jedinke izvornog programa provjeravaju se primjenom automata je li su u skupu pravilno napisanih leksičkih jedinki (odnosno je li su u jeziku leksičkih jedinki). Ako su sve leksičke jedinke ispravno napisane, onda se pristupa njihovom prevođenju 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];
```

Naredbu čini četrnaest leksičkih jedinki: varijabla `PPP`, operator pridruživanja `=`, varijabla *`Jura`*, otvorena zagrada `[`, varijabla *`i`*, zatvorena zagrada `]`, otvorena zagrada `[`, varijabla *`j`*, operator zbrajnja `+`, cjelobrojna konstanta `7`, operator zbrajnja `+`, 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 prevođenju 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 varijable 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 varijable i konstante. Dok se rad sintaksnog analizatora temelji isključivo 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 sljedećeg procesa prevođenja. Proces prevođenja 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 međukôda. Sintaksna pravila definiraju: način gradnje izraza primjenom leksičkih jediniki (npr. aritmetički izrazi, logički izrazi), način gradnje ispravnih naredbi programskog jezika primjenom 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 su naredbe grananja sljedećeg oblika: **IF**<*Izraz*>**THEN**<*Naredba*>**;**. Primjer pokazuje da sintaksna pravila osim 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 primjenom 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 međukôda. Prije generiranja višeg međukôda, 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 cjelobrojne i realne varijable, onda zbroj poprima realnu vrijednost. Nadalje, semantika jezika određuje skup dozvoljenih značenja. Na primjer, neki jezici ne dozvoljavaju množenje cjelobrojnih i realnih varijabli. Potrebno je prethodno pretvoriti obje varijable u realne ili cjelobrojne. U tim jezicima operator množenja ima značenje cjelobrojnog ili realnog množenja, ali nema značenje množenja cijelih brojeva s realnima.

Tijekom procesa generiranja višeg međukôda izračunaju se konstantne vrijednosti i pojednostavi se struktura naredbe. U višem međukôdu niz `7+24` zamijeni se nizom `31` koji ima značenje cjelobrojnog 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* cjelobrojne varijable, a *Jura* je dvodimenzionalno polje cjelobrojnih vrijednosti. Označavanje dvodimenzionalnog polja se pojednostavi u cilju smanjivanja broja znakova potrebnih za zapis naredbi višeg međukôda, te u cilju učinkovitije sinteze ciljnog programa. Iako je uobičajeno da naredbe višeg međukôda 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 međukôda navedena su izvorna imena varijabli i izračunata vrijednost cjelobrojne konstante 31. Niz jedinstvenih znakova leksičkih jedinki `V←V[V][V+K+K];` prevodi se u niz znakova naredbe višeg međukôda:

```
PPP ← Jura[i,j+31]
```

Generiranjem višeg međukôda 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 prevođenja: *prevođenje višeg međukôda u srednji međukôd*, *prevođenje srednjeg međukôda u niži međukôd* i *prevođenje nižeg međukôda u ciljni program*.

Jedan od osnovnih zadataka procesa prevođenja višeg međukôda u srednji međukôd jest pretvorba složenih podatkovnih struktura, kao što su polja podataka, i složenih naredbi koje upravljaju tijekom izvođenja programa u niz naredbi koje koriste isključivo varijable i jednostavne naredbe grananja. Na primjer, naredba dohvata vrijednosti elementa dvodimenzionalnog polja prevodi se u niz naredbi koje najprije izračunaju slijednu 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 cjelobrojne 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 izrazom `(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 *AdrJura*, onda se dobiva apsolutna adresa traženog podatka u memoriji računala. Sljedeće naredbe slijedno računaju traženu adresu pomoću privremenih varijabli *v*1 do *v*6, dok zadnja

naredba dohvaća podatak koji je spremljen na adresi izračunatoj u privremenoj varijabli *v*6. Vrijednost dohvaćenog podatka pridruži se varijabli *PPP*:

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

gdje oznaka @*v*6 ima značenje dohvata podatka s adrese koja je jednaka vrijednosti varijable *v*6.

U sljedećem koraku prevodi se srednji međukôd u niži međukôd. Naredbe nižeg međukôda koriste simboličke registre r1 do r8. Simbolička imena varijabli prevode se u stvarne memorijske adrese. Varijabla *i* spremljena je u prva četiri okteta početne adrese *Početak*, varijabla *j* spremljena je u druga četiri okteta, a prvi elemenet dvodimenzionalnog polja *Jura* spremljen je na adresi *Početak*+8. Naredbe nižeg međukôda 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 međukôda. U ciljnom jeziku se umjesto simboličkih registara koriste stvarni registri procesora računala, simboličke memorijske 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 predočuje dva osnovna procesa koja su osnovica kako cjelokupnog 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 klasa 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 osim 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 elektroničkog 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. Primjeri znakova su 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 postavljenih jedan do drugog. Na primjer, nizovi 011, 1011 i 1100011 zadani su nad binarnom abecedom B={0, 1}.

Duljina niza jednaka je broju njegovih znakova. Na primjer, niz w=1001101 jest duljine $|w|=7$. U nizu w jest 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 operacije nadovezivanja $\varepsilon w = w\varepsilon = w$.

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

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

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 jest skup nizova nad abecedom. Na temelju dane formalne definicije, primjeri jezika su: 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 $L_3=\{00, 11, 0011, 0101, 1001, 1100, ...\}$, 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: $L_4=\Sigma^*=\{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111, 0000, 0001, ...\}$.

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	$L - N$	$L \setminus N = \{w \mid w \in L \wedge w \notin N\}$
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 samim sobom). Definira se da je:

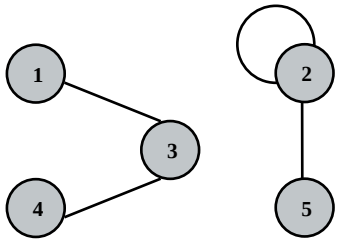
$$L^0 = \{\varepsilon\} \text{ i } L^i = L^{i-1}L.$$

Beskonačni skupovi

Kardinalni broj skupa jest broj elemenata skupa. Postoji li bijekcija između elemenata dva skupa (preslikavanje jedan u jedan), ta dva skupa imaju jednake 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, onda 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 jednake kardinalne brojeve jer postoji bijekcija $f(2i) = i$, gdje je i prirodni broj.

Nemaju svi beskonačni skupovi jednake 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 nije moguće pronaći bijektivno preslikavanje između elemenata skupa prirodnih brojeva i skupa realnih brojeva.



Slika 1.2: Primjer grafa

Graf

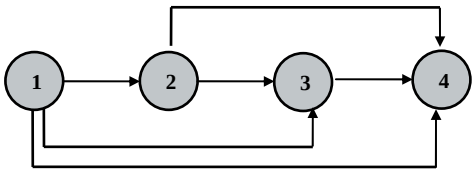
Graf $G = (V, E)$ čini konačni skup *čvorova* V i skup parova čvorova E. Parovi čvorova su *grane* grafa. Slika 1.2 prikazuje primjer grafa s čvorovima $V = \{1, 2, 3, 4, 5\}$ i granama $G = \{(1, 3), (3, 4), (2, 2), (2, 5)\}$.

Put grafa jest niz čvorova $v_1, v_2, v_3, ..., v_k$ ($k \geq 1$) 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

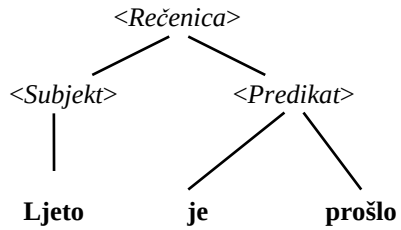
Grane *usmjerenog grafa* su uređeni parovi čvorova koje nazivamo *usmjerenim granama*. Usmjerena grana od čvora v do čvora w označava se izrazom $v \rightarrow w$. Slika 1.3 prikazuje primjer usmjerenog grafa.

Put usmjerenog grafa jest niz čvorova $v_1, v_2, v_3, ..., v_k$ ($k \geq 1$) za koji vrijedi da je $v_i \rightarrow v_{i+1}$ usmjerena 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 $w \rightarrow z$ usmjerene 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 jest usmjereni graf sljedećih svojstva:

1) Čvor nazvan *korijen* stabla nema prethodnika i od njega vodi 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 usmjerene 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) $P(n)$, 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 moguće je 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) *tranzivnosti* 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) *asimtričnosti* 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 tranzitvnosti.

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:

$R^+ = \{(1, 2), (2, 2), (2, 3), \mathbf{(1, 3)}\}$,
$R^* = \{(1, 2), (2, 2), (2, 3), (1, 3), \mathbf{(1, 1), (3, 3)}\}$.

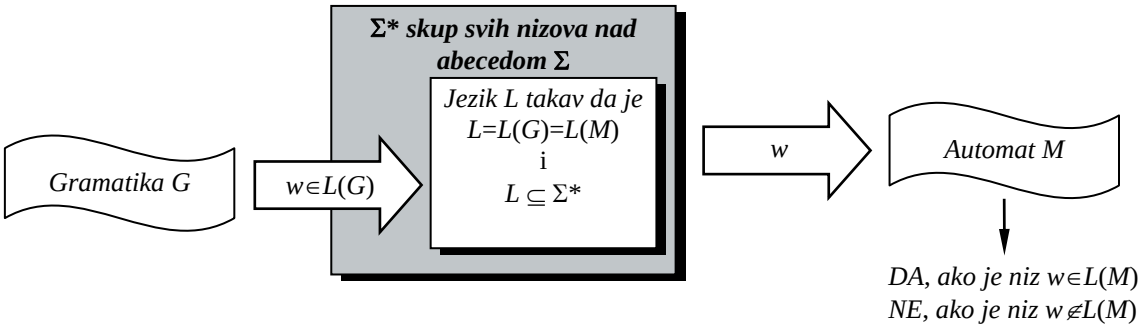
1.2 Primjer formalnog jezika, pripadajućeg automata i gramatike

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

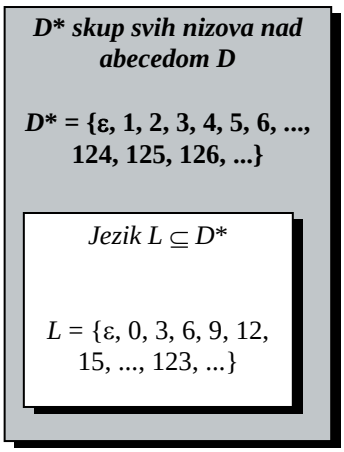
Automat jest model diskretnog matematičkog sustava koji čitanjem znak po znak odlučuje je li pročitani niz element zadanog jezika. Tijekom čitanja znakova automat mijenja *stanja*. Prije čitanja prvog znaka automat je u *početnom stanju*. Čitanjem znakova niza automat mijenja stanja. Stanja se dijele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

Gramatika jest model matematičkog sustava koji gradi, odnosno generira, nizove znakova. Nizovi se generiraju primjenom skupa produkcija. Produkcije su pravila koja određuju na koji način se grade nizovi znakova. Produkcije se zadaju *završnim* i *nezavršnim* 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 u kojem su isključivo znakovi 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 $L(G)$.

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, ..., 9}. Niz se promatra kao cjelobrojna vrijednost. Niz je u jeziku *L* ako je njegova cjelobrojna vrijednost dijeljiva s tri *L*={ε, 0, 3, 6, 9, 12, 15, ..., 123, ...}. Slika 1.6 prikazuje skup svih nizova *D** nad abecedom *D* i jezik *L*.

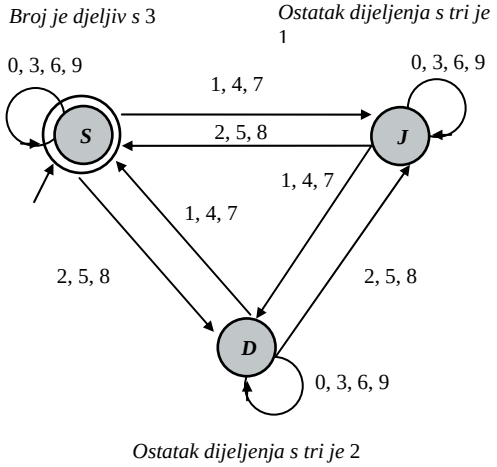
Dva su načina prikaza automata. Jedan od načina prikaza automata jest primjenom usmjerenog grafa koji se naziva dijagram stanja. Čvorovi grafa su stanja automata, a usmjerene grane označavaju prijelaze. Početno stanje automata označeno je znakom *S* i usmjerenom granom bez izvorišnog čvora. Prihvatljiva stanja su dvostruko okružena. Uz svaku granu označen je ulazni znak koji pokreće promjenu stanja automata. Prijeđe li automat u jedno od prihvatljivih stanja nakon što se pročita čitav niz, niz je element jezika.

Slika 1.7 prikazuje automat koji prihvaća 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 pokazuje se na nizu *w*=1404. Niz *w* je djeljiv s 3 i prema tome *w*∈*L*. Na početku rada automat je 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 prihvaća 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 prihvaća (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. U posljednjem stupcu jest oznaka 1 ili 0 ovisno o tome da li se stanje prihvaća ili ne. Početno stanje stavlja se u prvi redak tablice. Automat koji prepoznaje 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 je znak 4 (ulazni znak 4 određuje stupac tablice), onda automat prelazi u stanje *D*.

	Ulazni 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**
Stanja ***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. Produkcije gramatike jezika *L* prikazuju se na sljedeći način:

<*S*> → 0 <*S*> | 3 <*S*> | 6 <*S*> | 9 <*S*> | 1 <*J*> | 4 <*J*> | 7 <*J*> | 2 <*D*> | 5 <*D*> | 8 <*D*>
<*S*> → ε
<*J*> → 0 <*J*> | 3 <*J*> | 6 <*J*> | 9 <*J*> | 1 <*D*> | 4 <*D*> | 7 <*D*> | 2 <*S*> | 5 <*S*> | 8 <*S*>
<*D*> → 0 <*D*> | 3 <*D*> | 6 <*D*> | 9 <*D*> | 1 <*S*> | 4 <*S*> | 7 <*S*> | 2 <*J*> | 5 <*J*> | 8 <*J*>

U zagradama "< >" su nezavršni znakovi gramatike. <*S*> je početni nezavršni znak. Znak "→" označava da je nezavršni znak na lijevoj strani moguće zamijeniti nizom znakova na desnoj strani. Lijeva i desna strana znaka "→" su lijeva i desna strana produkcije. Oznaka "|" odijeljuje 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 ⇒ označava proces generiranja desnog međuniza znakova iz lijevog međuniza znakova primjenom točno jedne produkcije gramatike. U lijevom međunizu podcrta 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*>→ε generira prazni niz koji je element jezika *L*:

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

Niz 0 generira se primjenom dvije produkcije. Najprije se primjeni produkcija <*S*>→0<*S*>:

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

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

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

Gramatika generira niz 1404 na sljedeći način:

$\underline{<S>}$	⇒	$\underline{1\ <J>}$	(primjena produkcije <*S*>→1<*J*>)
$1\ \underline{<J>}$	⇒	$1\ \underline{4\ <D>}$	(primjena produkcije <*J*>→4<*D*>)
$1\ 4\ \underline{<D>}$	⇒	$1\ 4\ \underline{0\ <D>}$	(primjena produkcije <*D*>→0<*D*>)
$1\ 4\ 0\ \underline{<D>}$	⇒	$1\ 4\ 0\ \underline{4\ <S>}$	(primjena produkcije <*D*>→4<*S*>)
$1\ 4\ 0\ 4\ \underline{<S>}$	⇒	$1\ 4\ 0\ 4\ \underline{\varepsilon} = 1\ 4\ 0\ 4$.	(primjena produkcije <*S*>→ε)

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

<*S*> ⇒ 1 <*J*> ⇒ 1 4 <*D*> ⇒ 1 4 0 <*D*> ⇒ 1 4 0 4 <*S*> ⇒ 1 4 0 4.